OpenShift Container Platform 4.5

Pipelines

在 OpenShift Container Platform 中配置和使用 Pipelines

Last Updated: 2021-05-07

OpenShift Container Platform 4.5 Pipelines

在 OpenShift Container Platform 中配置和使用 Pipelines

法律通告

摘要

本文档提供在 OpenShift Container Platform 中配置和使用 Pipelines 的说明。

目录

第 1 章 了解 OPENSHIFT PIPELINES

重要

OpenShift Pipelines 目前只是一个技术预览功能。技术预览功能不被红帽产品服务等级协议 (SLA) 支持，且可能在功能方面有缺陷。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详情，请参阅 https://access.redhat.com/support/offerings/techpreview/。

Red Hat OpenShift Pipelines 是一个基于 Kubernetes 资源的云原生的持续集成和持续交付（continuous integration and continuous delivery，简称 CI/CD）的解决方案。它通过提取底层实现的详情，使用 Tekton 构建块进行跨多个平台的自动部署。Tekton 引入了多个标准自定义资源定义 (CRD)，用于定义可跨 Kubernetes 分布的 CI/CD 管道。

1.1. 主要特性

- Red Hat OpenShift Pipelines 是一个无服务器的 CI/CD 系统，它在独立的容器中运行 Pipelines，以及所有需要的依赖组件。
- Red Hat OpenShift Pipelines 是为开发基于微服务架构的非中心化团队设计的。
- Red Hat OpenShift Pipelines 使用标准 CI/CD 管道（pipeline）定义，这些定义可轻松扩展并与现有 Kubernetes 工具集成，可让您按需扩展。
- 您可以通过 Red Hat OpenShift Pipelines 使用 Kubernetes 工具（如 Source-to-Image (S2I)、Buildah、Buildpacks 和 Kaniko）构建镜像，这些工具可移植到任何 Kubernetes 平台。
- 您可以使用 OpenShift Container Platform 开发控制台（Developer Console）来创建 Tekton 资源，查看 Pipeline 运行的日志，并管理 OpenShift Container Platform 命名空间中的管道。

1.2. RED HAT OPENSHIFT PIPELINES 概念

Red Hat OpenShift Pipelines 提供一组标准自定义资源定义 (CRD)，用作构建块，您可以使用它们来为应用程序构建 CI/CD 管道。

Task

Task（任务）是在 Pipeline 中可配置的最小单元。它基本上是一个构成 Pipeline 构建的输入和输出的功能。它可以独立运行，也可以作为 Pipeline 的一部分运行。Pipeline 包含一个或多个任务，每个任务由一个或多个步骤组成。步骤（Step）是由任务顺序执行的一系列命令。

Pipeline

Pipeline（管道）由一系列任务组成，执行这些任务是为了构建能够自动化应用程序的构建、部署和交付的复杂工作流。它是 PipelineResources、参数以及一个或多个任务的集合。Pipeline 使用 PipelineResources 与外部进行交互，这些资源作为输入和输出添加到任务中。

PipelineRun

PipelineRun 是一个 Pipeline 的运行实例。PipelineRun 启动 Pipeline，并为 Pipeline 中执行的每个任务管理一个 TaskRun 的创建。

TaskRun

PipelineRun 由 Pipeline 中每个任务的 PipelineRun 自动创建。它是在 Pipeline 中运行任务实例的结果。如果某个任务在 Pipeline 之外运行，它也可以被手工创建。

Workspace

Workspace 是一个存储卷，任务（Task）在运行时需要它来接收输入或提供输出。Task 或 Pipeline 会声明 Workspace，一个TaskRun 或 PipelineRun 则会提供存储卷的实际位置，存储卷被挂载到声明的 Workspace 上。这使得任务具有灵活性、可重复使用，并允许在多个任务间共享工作区。

Trigger

Trigger（触发器）捕获外部事件，如 Git 拉取请求，并处理事件有效负载以获取关键信息。然后，提取的信息会映射到一组预定义的参数，这些参数会触发一系列可能需要创建和部署 Kubernetes 资源的任务。您可以使用 Triggers 和 Pipelines 来创建全面的 CI/CD 系统，在这些系统中，执行操作完全通过 Kubernetes 资源定义。

Condition

Condition（条件）指的是在您的 Pipeline 中运行某个任务前执行的验证或检查。Conditions 就象是 **if** 语句，用来执行逻辑测试，它的返回值为 **True** 或 **False**。如果所有 Conditions 返回 **True**，则会执行任务；但如果任何 Conditions 返回了失败状态，则会跳过这个任务以及随后的所有任务。您可以使用 Pipeline 中的条件来创建涵盖多个场景的复杂工作流。

1.3. OPENSHIFT PIPELINE 概念详情

本指南提供了对管道（Pipeline）概念的详细论述。

1.3.1. 任务（Task）

*任务（Task）*是 Pipeline 的构建块，它由带有一定顺序的执行步骤组成。任务（Task）可以重复使用，并可用于多个 Pipelines。

*步骤（Step）*是一系列实现特定目标的命令，如构建镜像。每个任务都作为 pod 运行，每个步骤都在同一个 pod 内自己的容器中运行。由于步骤在同一个 pod 中运行，所以它们可以访问同一卷来缓存文件、ConfigMap 和 Secret。

以下示例显示了 **apply-manifests** 任务。

```
apiVersion: tekton.dev/v1beta1 1
kind: Task 2
metadata:
  name: apply-manifests 3
spec: 4
  params:
  - default: k8s
    description: The directory in source that contains yaml manifests
    name: manifest_dir
    type: string
  steps:
  - args:
    - |-
      echo Applying manifests in $(inputs.params.manifest_dir) directory
      oc apply -f $(inputs.params.manifest_dir)
      echo -----------------------------------
    command:
    - /bin/bash
    - -c
    image: quay.io/openshift/origin-cli:latest
    name: apply
```

```
  workingDir: /workspace/source
 workspaces:
 - name: source
```

1. 任务 API 版本 **v1beta1**。
2. 指定 Kubernetes 对象的类型。在此例中，**Task**。
3. 此任务的唯一名称。
4. 列出任务中的参数和步骤，以及任务使用的工作区（workspace）。

此任务启动 pod，并在这个 pod 中使用 **maven:3.6.0-jdk-8-slim** 镜像运行一个容器，来运行指定的命令。它接收了一个名为 **workspace-git** 的输入目录，其中包含应用程序的源代码。

该任务仅声明了 Git 存储库的占位符，并没有指定要使用哪个 Git 存储库。这将允许此任务被重复用于多个管道和目的。

1.3.2. TaskRun

TaskRun 使用集群上的特定输入、输出和执行参数来实例化一个任务用来执行它。它可以自行调用，或作为 PipelineRun 的一部分。

任务由执行容器镜像的一个或多个步骤组成，每个容器镜像执行特定的构建工作。TaskRun 以指定顺序在任务中执行步骤，直到所有步骤都成功执行或发生失败为止。

以下示例显示了一个带有相关输入参数运行 **apply-manifests** 任务的 TaskRun:

```
apiVersion: tekton.dev/v1beta1 (1)
kind: TaskRun (2)
metadata:
  name: apply-manifests-taskrun (3)
spec: (4)
  serviceAccountName: pipeline
  taskRef: (5)
    kind: Task
    name: apply-manifests
  workspaces: (6)
  - name: source
    persistentVolumeClaim:
      claimName: source-pvc
```

1. TaskRun API 版本 **v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中，**TaskRun**。
3. 用于标识此 TaskRun 的唯一名称。
4. TaskRun 的定义。对于这个 TaskRun，指定了任务和所需的工作区。
5. 用于此 TaskRun 的任务引用的名称。此 TaskRun 会执行 **apply-manifests** 任务。
6. TaskRun 使用的工作空间。

1.3.3. Pipelines

Pipeline 是按特定顺序排列的任务集合。您可以使用包含一个或多个任务的 Pipelines 为应用程序定义 CI/CD 工作流。

Pipeline 定义由多个字段或属性组成，它们一起可让 Pipeline 实现特定目标。每个 Pipeline 定义必须至少包含一个任务，用于控制特定输入并生成特定的输出。Pipeline 定义也可以根据应用程序要求包括 Conditions、Workspaces、Parameters 或 Resources。

以下示例显示了 **build-and-deploy** Pipeline，它使用 **buildah** ClusterTask 从 Git 存储库构建应用程序镜像 :

```
apiVersion: tekton.dev/v1beta1 1
kind: Pipeline 2
metadata:
  name: build-and-deploy 3
spec: 4
  workspaces: 5
  - name: shared-workspace
  params: 6
  - 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: "release-tech-preview-2"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks: 7
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image 8
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
```

```
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - fetch-repository
  - name: apply-manifests (9)
    taskRef:
      name: apply-manifests
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter: (10)
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    workspaces:
    - name: source
      workspace: shared-workspace
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests
```

1. Pipeline API 版本 **v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中, **Pipeline**。
3. 此 Pipeline 的唯一名称。
4. 指定 Pipeline 的定义和结构。
5. Pipeline 中所有任务使用的工作区。
6. Pipeline 中所有任务使用的参数。
7. 指定 Pipeline 中使用的任务列表。
8. 任务 **build-image** 使用 **buildah** ClusterTask 从给定的 Git 仓库构建应用程序镜像。
9. 任务 **apply-manifests** 使用相同名称的用户定义的任务。
10. 指定在 Pipeline 中运行任务的顺序。在本例中，**apply-manifests** 任务仅在 **build-image** 任务完成后运行。

1.3.4. PipelineRun

PipelineRun 用集群上的特定输入、输出和执行参数来实例化 Pipeline 执行。在 PipelineRun 中为每个任务自动创建一个对应的 TaskRun。

Pipeline 中的所有任务均按定义的顺序执行，直到所有任务成功或任务失败为止。**status** 字段跟踪并存储 PipelineRun 中的每个 TaskRun 的进度，用于监控和审核目的。

以下示例显示了一个 PipelineRun，用于运行带有相关资源和参数的 **build-and-deploy** Pipeline:

```
apiVersion: tekton.dev/v1beta1 (1)
kind: PipelineRun (2)
metadata:
  name: build-deploy-api-pipelinerun (3)
spec:
  pipelineRef:
    name: build-and-deploy (4)
  params: (5)
  - name: deployment-name
    value: vote-api
  - name: git-url
    value: http://github.com/openshift-pipelines/vote-api.git
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
  workspaces: (6)
  - name: shared-workspace
    persistentvolumeclaim:
      claimName: source-pvc
```

1. PipelineRun API 版本 **v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中，**PipelineRun**。
3. 用于标识此 PipelineRun 的唯一名称。
4. 要运行的 Pipeline 的名称。在本例中，**build-and-deploy**。
5. 指定运行 Pipeline 所需的参数列表。
6. PipelineRun 使用的 Workspace。

1.3.5. Workspaces（工作区）

注意

建议您在 OpenShift Pipelines 中使用 Workspaces 而不是 PipelineResources，因为 PipelineResources 很难调试，范围有限，且不容易重复使用。

Workspace 声明 Pipeline 中的任务在运行时需要的共享存储卷。Workspaces 不指定卷的实际位置，它允许您定义运行时所需的文件系统或部分文件系统。您必须提供在 TaskRun 或 PipelineRun 中挂载到该 Workspace 中的卷的特定位置详情。这种将卷声明与运行时存储卷分开来使得任务可以被重复使用、灵活且独立于用户环境。

使用 Workspaces，您可以：

- 存储任务输入和输出
- 任务间共享数据
- 使用它作为 Secret 中持有的凭证的挂载点
- 使用它作为 ConfigMap 中保存的配置的挂载点
- 使用它作为机构共享的通用工具的挂载点
- 创建可加快作业的构建工件缓存

您可以使用以下方法在 TaskRun 或 PipelineRun 中指定 Workspaces:

- 只读 ConfigMap 或 Secret
- 与其他任务共享的现有 PersistentVolumeClaim
- 来自提供的 VolumeClaimTemplate 的 PersistentVolumeClaim
- TaskRun 完成后丢弃的 emptyDir

以下显示了 **build-and-deploy** Pipeline 的代码片段，它为任务 **build-image** 和 **apply-manifests** 声明了一个 **shared-workspace** Workspace。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces: 1
  - name: shared-workspace
  params:
...
  tasks: 2
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces: 3
    - name: source 4
      workspace: shared-workspace 5
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces: 6
    - name: source
      workspace: shared-workspace
```

```
    runAfter:
     - build-image
...
```

1. Pipeline 中定义的任务共享的 Workspace 列表。Pipeline 可以根据需要定义 Workspace。在这个示例中，只声明了一个名为 **shared-workspace** 的 Workspace。
2. Pipeline 中使用的任务定义。此片段定义了两个任务，**build-image** 和 **apply-manifests**。这两个任务共享一个 Workspace。
3. **build-image** 任务中使用的 Workspaces 列表。任务定义可以根据需要包含多个 Workspace。但建议任务最多使用一个可写 Workspace。
4. 唯一标识任务中使用的 Workspace 的名称。此任务使用一个名为 **source** 的 Workspace。
5. 任务使用的 Pipeline Workspace 的名称。请注意，Workspace **source** 使用 Pipeline Workspace **shared-workspace**。
6. **apply-manifests** 任务中使用的 Workspace 列表。请注意，此任务与 **build-image** 任务共享 **source** Workspace。

以下是 **build-deploy-api-pipelinerun** PipelineRun 的代码片段，它使用 PersistentVolumeClaim 为 **build-and-deploy** Pipeline 中使用的 **shared-workspace** Workspace。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: build-deploy-api-pipelinerun
spec:
  pipelineRef:
    name: build-and-deploy
  params:
...

  workspaces: 1
  - name: shared-workspace 2
    persistentvolumeclaim:
      claimName: source-pvc 3
```

1. 指定 Pipeline Workspaces 列表，用于在 PipelineRun 中提供卷绑定。
2. 提供卷的 Pipeline 中的 Workspace 的名称。
3. 指定预定义的 PersistentVolumeClaim 的名称，该名称将附加到 Workspace。在本例中，现有 **source-pvc** PersistentVolumeClaim 与 **shared-workspace** Workspace 附加。

1.3.6. 触发器

使用触发器（Trigger）和 Pipelines 一起创建一个完整的 CI/CD 系统，其中 Kubernetes 资源定义整个 CI/CD 执行。Pipeline 触发器捕获外部事件，并处理它们以获取关键信息。将这个事件数据映射到一组预定义的参数可触发一系列任务，然后创建和部署 Kubernetes 资源。

例如，您可以使用 Red Hat OpenShift Pipelines 为应用程序定义 CI/CD 工作流。PipelineRun 必须启动，才能在应用程序存储库中使任何新的更改生效。通过捕获和处理任何更改事件，并通过触发器部署新镜像的 PipelineRun 来自动触发这个过程。

触发器由以下主要组件组成，它们可一起组成可重复使用、分离和自力更生的 CI/CD 系统：

- *EventListeners* 提供端点（endpoint）或事件 sink，用于使用 JSON 有效负载侦听传入的基于 HTTP 的事件。EventListener 使用 Event Interceptors 在有效负载上执行轻量级事件处理，它可识别有效负载类型并进行自选修改。目前，Pipeline Triggers 支持四种拦截器：Webhook Interceptors、GitHub Interceptors、GitLab Interceptors 和 Common Expression Language (CEL) Interceptors。
- *TriggerBindings* 从事件有效负载中提取字段并将其作为参数保存。
- *TriggerTemplates* 指定如何使用 TriggerBindings 中的参数化数据。TriggerTemplate 定义了一个资源模板，它接受 TriggerBindings 的输入，然后执行一系列操作来创建新 PipelineResources 并启动新的 PipelineRun。

EventListeners 把 TriggerBindings 和 TriggerTemplates 的概念组合在一起。EventListener 侦听进入的事件，使用 Interceptors 处理基本的过滤，使用 TriggerBindings 提取数据，然后处理这些数据以使用 TriggerTemplates 创建 Kubernetes 资源。

以下是 **vote-app-binding** TriggerBinding 的代码片段，它从接收的事件有效负载中提取 Git 存储库信息：

```
apiVersion: triggers.tekton.dev/v1alpha1 (1)
kind: TriggerBinding (2)
metadata:
  name: vote-app (3)
spec:
  params: (4)
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

1. TriggerBinding API 版本 **v1alpha1**。
2. 指定 Kubernetes 对象的类型。在本例中，**TriggerBinding**。
3. 用于标识这个 TriggerBinding 的唯一名称。
4. 从接收的事件有效负载中提取并传递给 TriggerTemplate 的参数列表。在本例中，Git 仓库 URL、名称和修订版本是从事件有效负载主体中提取的。

以下是 **vote-app-template** TriggerTemplate 的代码片段，它从 TriggerBinding 提供的 Git 仓库信息创建 Pipeline 资源：

```
apiVersion: triggers.tekton.dev/v1alpha1 (1)
kind: TriggerTemplate (2)
metadata:
  name: vote-app (3)
spec:
```

```
  params: 4
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: master
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates: 5
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(tt.params.git-repo-name)-$(uid)
    spec:
      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
        persistentvolumeclaim:
          claimName: source-pvc
```

1. TriggerTemplate API 版本 **v1alpha1**。
2. 指定 Kubernetes 对象的类型。在本例中，**TriggerTemplate**。
3. 用于标识此 TriggerTemplate 的唯一名称。
4. TriggerBinding 或 EventListerner 提供的参数。
5. 从 TriggerBinding 或 EventListener 中接收的参数中为 Pipeline 创建的资源模板列表。

这里是一个 EventListener 的示例，它使用 **vote-app-binding** TriggerBinding 和 **vote-app-template** TriggerTemplate 来处理传入的事件。

```
apiVersion: triggers.tekton.dev/v1alpha1 1
kind: EventListener 2
metadata:
  name: vote-app 3
spec:
  serviceAccountName: pipeline 4
  triggers:
  - bindings: 5
```

```
- ref: vote-app
template: 6
  name: vote-app
```

1. EventListener API 版本 **v1alpha1**。
2. 指定 Kubernetes 对象的类型。在本例中，**EventListener**。
3. 用于标识这个 EventListener 的唯一名称。
4. 要使用的服务帐户名称。
5. 用于这个 EventListener 的 TriggerBinding 的名称。
6. 用于这个 Eventlistener 的 Triggertemplate 的名称。

1.4. 其他资源

- 有关安装 Pipelines 的详情，请参阅 安装 OpenShift Pipelines。
- 有关创建自定义 CI/CD 解决方案的详情请参考 使用 CI/CD Pipelines 创建应用程序。

第 2 章 安装 OPENSHIFT PIPELINES

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 已安装了 **oc** CLI。
- 您已在本地系统中安装了 OpenShift Pipelines (tkn)CLI。

2.1. 在 WEB 控制台中安装 RED HAT OPENSHIFT PIPELINES OPERATOR

您可以使用 OpenShift Container Platform OperatorHub 中列出的 Operator 来安装 Red Hat OpenShift Pipelines。安装 Red Hat OpenShift Pipelines Operator 时，Pipelines 配置所需的自定义资源 (CR) 与 Operator 一起自动安装。

流程

1. 在控制台的 **Administrator** 视角中，导航到 **Operators → OperatorHub**。
2. 使用 **Filter by keyword** 复选框在目录中搜索 **Red Hat OpenShift Pipelines Operator**。点 **OpenShift Pipelines Operator**。

注意

确保您没有选择 **OpenShift Pipelines Operator** 的 **Community** 版本。

3. 参阅 **Red Hat OpenShift Pipelines Operator** 页中有关 Operator 的简单描述。点击 **Install**。
4. 在 **Install Operator** 页面中：
   a. 为 **Installation Mode** 选择 **All namespaces on the cluster (default)**。选择该项会将 Operator 安装至默认 **openshift-operators** 命名空间，这将启用 Operator 以进行监视并在集群中的所有命名空间中可用。
   b. 为 **Approval Strategy** 选择 **Automatic**。这样可确保以后对 Operator 的升级由 Operator Lifecycle Manager (OLM) 自动进行。如果您选择 **Manual** 批准策略，OLM 会创建一个更新请求。作为集群管理员，您必须手动批准 OLM 更新请求，才可将 Operator 更新至新版本。
   c. 选择一个 **Update Channel**。
      - **ocp-<4.x>** 频道将启用 Red Hat OpenShift Pipelines Operator 最新稳定版本的安装。
      - **preview3** 频道启用 Red Hat OpenShift Pipelines Operator 的最新预览版本，该版本可能包含 4.x 更新频道中还未提供的功能。
5. 点击 **Install**。您会看到 **Installed Operators** 页面中列出的 Operator。

注意

Operator 会自动安装到 **openshift-operators** 命名空间中。

6. 检查 **Status** 是否已被设置为 **Succeeded Up to date** 来确认 Red Hat OpenShift Pipelines Operator 已安装成功。

2.2. 使用 CLI 安装 OPENSHIFT PIPELINES OPERATOR

您可以使用 CLI 从 OperatorHub 安装 Red Hat OpenShift Pipelines Operator。

流程

1. 创建一个订阅对象 YAML 文件，以便为 Red Hat OpenShift Pipelines Operator 订阅一个命名空间，如 **sub.yaml**：

   订阅示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-pipelines-operator
  namespace: openshift-operators
spec:
  channel:  <channel name> 1
  name: openshift-pipelines-operator-rh 2
  source: redhat-operators 3
  sourceNamespace: openshift-marketplace 4
```

   1. 指定您要订阅 Operator 的频道名称
   2. 要订阅的 Operator 的名称。
   3. 提供 Operator 的 CatalogSource 的名称。
   4. CatalogSource 的命名空间。将 **openshift-marketplace** 用于默认的 OperatorHub CatalogSource。

2. 创建订阅对象：

```
$ oc apply -f sub.yaml
```

   Red Hat OpenShift Pipelines Operator 现在安装在默认目标命名空间 **openshift-operators** 中。

其它资源

- 您可以参阅将 Operators 添加到集群一节中的内容来了解更多有关在 OpenShift Container Platform 上安装 Operator 的信息。

第 3 章 卸载 OPENSHIFT PIPELINES

卸载 Red Hat OpenShift Pipelines Operator 分为两个步骤：

1. 删除安装 Red Hat OpenShift Pipelines Operator 时默认添加的自定义资源 (CR)。
2. 卸载 Red Hat OpenShift Pipelines Operator。

安装 Operator 时，仅卸载 Operator 不会删除默认创建的 Red Hat OpenShift Pipelines 组件。

3.1. 删除 RED HAT OPENSHIFT PIPELINES 组件和自定义资源

删除安装 Red Hat OpenShift Pipelines Operator 期间默认创建的自定义资源（CR）。

流程

1. 在 Web 控制台的 **Administrator** 视角中，导航至 **Administration → Custom Resource Definition**。
2. 在 **Filter by name** 框中键入 **config.operator.tekton.dev** 来搜索 Red Hat OpenShift Pipelines Operator CR。
3. 点击 **CRD Config** 查看 **Custom Resource Definition Details** 页面。
4. 点击 **Actions** 下拉菜单并选择 **Delete Custom Resource Definition**。

注意

删除 CR 将删除 Red Hat OpenShift Pipelines 组件，并丢失集群上的所有任务和管道。

5. 点击 **Delete** 以确认删除 CR。

3.2. 卸载 RED HAT OPENSHIFT PIPELINES OPERATOR

流程

1. 在 **Operators → OperatorHub** 页面中，使用 **Filter by keyword** 复选框来搜索 **Red Hat OpenShift Pipelines Operator**。
2. 点 **OpenShift Pipelines Operator**。Operator 标题表示已安装该 Operator。
3. 在 **OpenShift Pipelines Operator** 描述符页面中，点击 **Uninstall**。

其它资源

- 您可以参阅从集群中卸载 Operators一节中的内容来了解更多有关从 OpenShift Container Platform 上卸载 Operator 的信息。

第 4 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 Pipeline，您必须执行以下任务：

- 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 delivery Pipeline。
- 创建附加到 Workspace 的 PersistentVolumeClaim，以提供用于 Pipeline 执行的卷或文件系统。
- 创建一个 PipelineRun 来实例化并调用 Pipeline。
- 添加触发器（Trigger）来捕获源存储库中的所有事件。

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：

- 一个前端界面 **vote-ui**，它的源代码在 **ui-repo** Git 存储库中。
- 一个后端接口 **vote-api**，它的源代码在 **api-repo** Git 存储库中。
- **apply_manifest** 和 **update-deployment** Task 在 **pipelines-tutorial** Git 存储库中。

4.1. 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。
- 已使用 GitHub ID 清理前端 **ui-repo** 和后端 **api-repo-repo** 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。

4.2. 创建项目并检查 PIPELINE SERVICEACCOUNT

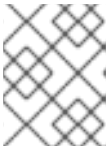
流程

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. 检查 **pipeline** ServiceAccount：
Red Hat OpenShift Pipelines Operator 添加并配置一个名为 **pipeline** 的 ServiceAccount，它有足够的权限来构建和推送镜像。这个 ServiceAccount 由 PipelineRun 使用。

```
$ oc get serviceaccount pipeline
```

4.3. 创建管道任务

流程

1. 从 **pipelines-tutorial** 存储库（它包括了一组可为管道重复使用的任务）中安装 **apply-manifests** 和 **update-deployment** 任务：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_pipeline/02_update_deployment_task.yaml
```

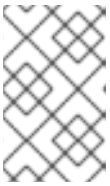
2. 使用 **tkn task list** 命令列出您创建的任务：

```
$ 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 ClusterTasks，如 **buildah** 和 **s2i-python-3**:



注意

您必须使用特权 Pod 容器来运行 **buildah ClusterTask**，因为它需要特权安全上下文。如需了解更多有关 pod 安全性上下文约束（SCC）的信息，请参阅附加资源部分。

```
$ tkn clustertasks list
```

输出中列出了 Operator 安装的 ClusterTasks:

```
NAME                DESCRIPTION   AGE
buildah                           1 day ago
git-clone                         1 day ago
s2i-php                           1 day ago
tkn                               1 day ago
```

4.4. 组装 PIPELINE

一个 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。

Pipeline 通过使用 **from** 和 **runAfter** 参数来指定在不同任务间如何进行交互以及它们执行的顺序。它使用 **workspaces** 字段指定 Pipeline 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 Pipeline，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。

Pipeline 为后端应用程序 **vote-api** 和前端应用程序 **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 **buildah** ClusterTask 构建容器镜像。
- 通过引用 **image** 参数将镜像推送到内部 镜像 registry。
- 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 Pipeline YAML 文件示例内容并保存：

```
apiVersion: tekton.dev/v1beta1
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: "release-tech-preview-2"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks:
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
```

```
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    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
    workspaces:
    - name: source
      workspace: shared-workspace
    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 Pipeline 被触发并执行时，这些详细信息会作为 **params** 添加。

2. 创建 Pipeline:

```
$ 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/release-tech-preview-2/01_pipeline/04_pipeline.yaml
```

3. 使用 **tkn pipeline list** 命令来验证是否在应用程序中添加了 Pipeline:

```
$ 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   ---        ---       ---        ---
```

4.5. 在工作区中指定 PERSISTENTVOLUMECLAIMS 作为 VOLUMESOURCE

工作区可帮助任务共享数据，并允许您指定 Pipeline 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，要创建一个 PersistentVolumeClaim 来提供数据存储并将其绑定到 Workspace。此 PersistentVolumeClaim 提供了 Pipeline 执行所需的卷或文件系统。

流程

1. 复制并保存以下 PersistentVolumeClaim YAML 示例文件的内容：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: source-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

2. 创建 PersistentVolumeClaim，指定您刚才创建的文件：

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

或者，从 Git 存储库直接执行 YAML 文件：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_pipeline/03_persistent_volume_claim.yaml
```

4.6. 运行 PIPELINE

PipelineRun 启动一个 Pipeline，并将其与 Git 和应用于特定调用的镜像资源相关联。它为 Pipeline 中的每个任务自动创建并启动 TaskRuns。

流程

1. 启动后端应用程序的 Pipeline：

```
$ tkn pipeline start build-and-deploy -w name=shared-workspace,claimName=source-pvc -p
deployment-name=vote-api -p git-url=http://github.com/openshift-pipelines/vote-api.git -p
IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
```

请注意命令输出返回的 PipelineRun ID。

2. 跟踪 PipelineRun 进程：

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

3. 启动前端应用程序的 Pipeline：

```
$ tkn pipeline start build-and-deploy -w name=shared-workspace,claimName=source-pvc -p
deployment-name=vote-api -p git-url=http://github.com/openshift-pipelines/vote-ui.git -p
IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-ui
```

请注意命令输出返回的 PipelineRun ID。

4. 跟踪 PipelineRun 进程：

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

5. 几分钟后，使用 **tkn pipelinerun list** 命令列出所有 PipelineRuns 来验证 Pipeline 是否成功运行：

```
$ tkn pipelinerun list
```

输出中列出了 PipelineRuns:

```
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 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. 要重新运行最后一个 PipelineRun，请使用之前 Pipeline 的 PipelineResources 和 ServiceAccount 运行：

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

4.7. 在 PIPELINE 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。在组装并启动应用程序 Pipeline 后，添加 TriggerBindings、TriggerTemplates 和 EventListener 以捕获 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/v1alpha1
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**:

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/03_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/v1alpha1
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: release-tech-preview-2
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(params.git-repo-name)-$(uid)
    spec:
      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
        persistentvolumeclaim:
          claimName: source-pvc
```

4. 创建 **TriggerTemplate**:

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-
preview-2/03_triggers/02_template.yaml
```

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

```
apiVersion: triggers.tekton.dev/v1alpha1
kind: EventListener
metadata:
  name: vote-app
spec:
  serviceAccountName: pipeline
  triggers:
  - bindings:
    - ref: vote-app
    template:
      name: vote-app
```

6. 创建 **EventListener**:

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-
preview-2/03_triggers/03_event_listener.yaml
```

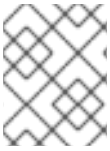
7. 将 EventListener 服务以 OpenShift Container Platform 路由的形式公开，以便公众可以访问它：

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

4.8. 创建 WEBHOOK

Webhook 是 EventListeners 在存储库中配置的事件发生时接收到的 HTTP POST 信息。然后，事件有效负载映射到 TriggerBindings，由 TriggerTemplates 进行处理。TriggerTemplates 最终会启动一个或多个 PipelineRuns，从而创建并部署 Kubernetes 资源。

在本节中，您将在 Git 存储库 **vote-ui** 和 **vote-api** 的副本中配置一个 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:

```
$ 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 **vote-ui**。
   b. 点 **Settings → Webhooks → Add Webhook**
   c. 在 **Webhooks/Add Webhook** 页面中：
      i. 在 **Payload URL** 字段中输入第一步中的 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 来设置后端存储库 **vote-api**。

4.9. 触发 PIPELINERUN

每当 Git 存储库中发生 **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 TriggerBindings 和 TriggerTemplates 对。TriggerBinding 提取参数，TriggerTemplate 使用这些参数来创建资源。这可能会重建并重新部署应用程序。

在此部分中，您将把一个空的提交推送到前端 **vote-api** 存储库，该存储库将触发 PipelineRun。

流程

1. 在终端中，克隆 Git 存储库 **vote-api** 的副本：

```
$ git clone git@github.com:<your GitHub ID>/vote-ui.git -b release-tech-preview-2
```

2. 推送空提交：

```
$ git commit -m "empty-commit" --allow-empty && git push origin release-tech-preview-2
```

3. 检查 PipelineRun 是否被触发：

```
$ tkn pipelinerun list
```

请注意，一个新的 PipelineRun 被启动。

4.10. 其他资源

- 如需了解更多 pipelines 在 **Developer** 视角的信息，请参阅在 **Developer** 视角中使用 Pipelines 小节中的内容。
- 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 存储库。另外，您还可以在 Tekton 项目中看到 Tekton Catalog。

第 5 章 在 DEVELOPER 视角中使用 RED HAT OPENSHIFT PIPELINES

您可以使用 OpenShift Container Platform web 控制台的 **Developer** 视角为软件交付过程创建 CI/CD 管道.

在 **Developer** 视角中：

- 使用 **Add → Pipeline → Pipeline Builder** 选项为您的应用程序创建自定义管道。
- 在 OpenShift Container Platform 上创建应用程序时，通过 **Add → From Git** 选项来使用 Operator 安装的 Pipeline 模板和资源来创建 Pipelines。

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** 视图与通过 **From Git** 选项创建的 Pipelines 进行交互。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

- 您有权限访问 OpenShift Container Platform 集群，并已切换到 web 控制台中的 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。

5.1. 使用 PIPELINE BUILDER 构建管道

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

- 使用现有任务和 ClusterTask 构建管道流。在安装 OpenShift Pipelines Operator 时，它会为集群添加可重复使用的 Pipeline ClusterTask。
- 指定 Pipeline Run 所需的资源类型，如有必要，在管道中添加额外的参数。
- 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。
- 一个任务的参数会根据任务规格预先填充。如果需要，引用任务中添加至 Pipeline 的任何额外参数。

流程

1. 在 **Developer** 视角的 **Add** 视图中，点 **Pipeline** 标题查看 **Pipeline Builder** 页面。
2. 为 Pipeline 输入唯一名称。
3. 从 **Select task** 列表中选择一个任务来把它添加到 Pipeline。这个示例使用 **s2i-nodejs** 任务。
    a. 要为 Pipeline 添加后续任务，点任务右侧或左边的加号图标，从 **Select task** 列表中选择您要添加到管道的任务。在本例中，使用 **s2i-nodejs** 任务右侧的加号图标来添加 **openshift-client** 任务。
    b. 要为现有任务添加并行任务，点任务下面的加号图标，从 **Select Task** 列表中选择您要添加到管道的并行任务。

图 5.1. Pipeline Builder

4. 点 **Add Resources** 来指定 Pipeline Run 将要使用的资源的名称和类型。这些资源然后会被 Pipeline 中的任务使用作为输入和输出。在此例中：

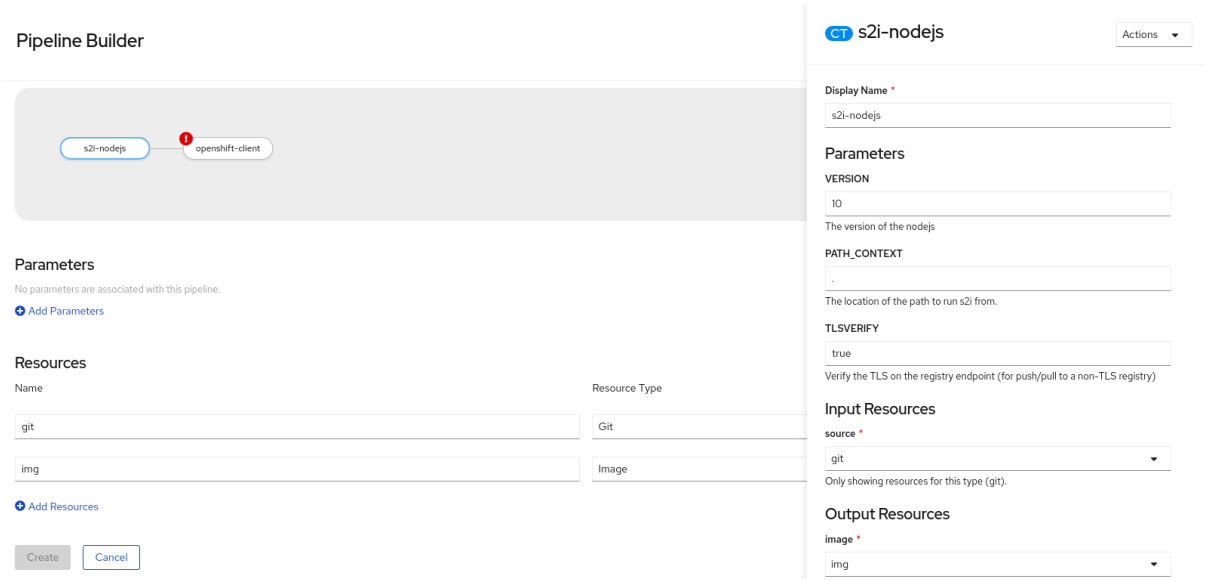
   a. 添加一个输入资源。在 **Name** 字段中输入 **Source**，从 **Resource Type** 下拉列表中选择 **Git**。

   b. 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**。

5. 一个任务的 **Parameters** 会根据任务的规格预先填充。如果需要，使用 **Add Parameters** 链接来添加额外的参数。

6. 如果没有指定任务的资源，则会在任务中显示 **Missing Resources** 警告。点 **s2i-nodejs** 任务，在侧面板中查看任务的详情。

   图 5.2. Pipelines Builder 中的任务详情



7. 在任务侧面板中为其指定资源和参数：

   a. 在 **Input Resources → Source** 部分中, **Select Resources** 下拉列表显示添加到管道中的资源。在本例中，选择 **Source**。

   b. 在 **Output Resources → Image** 部分，点 **Select Resources** 列表，然后选择 **img**。

c. 如果需要，在 **Parameters** 项中，使用 **$(params.<param-name>)** 语法在默认参数外添加更多参数。

8. 同样，为 **openshift-client** 任务添加输入资源。

9. 点 **Create** 创建管道。**Pipeline Details** 页会被显示，其中显示了创建的管道的详情。您现在可以使用 **Action** 按钮启动管道。

另外，还可以使用 **Pipeline Builder** 页面右上角的 **Edit YAML** 链接直接修改控制台中的 Pipeline YAML 文件。您还可以使用 Operator 安装的、可重复使用的代码片断和样本来创建详细的管道。

5.2. 使用 OPENSHIFT PIPELINES 创建应用程序

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。如需更多信息，请参阅使用 Developer 视角创建应用程序。

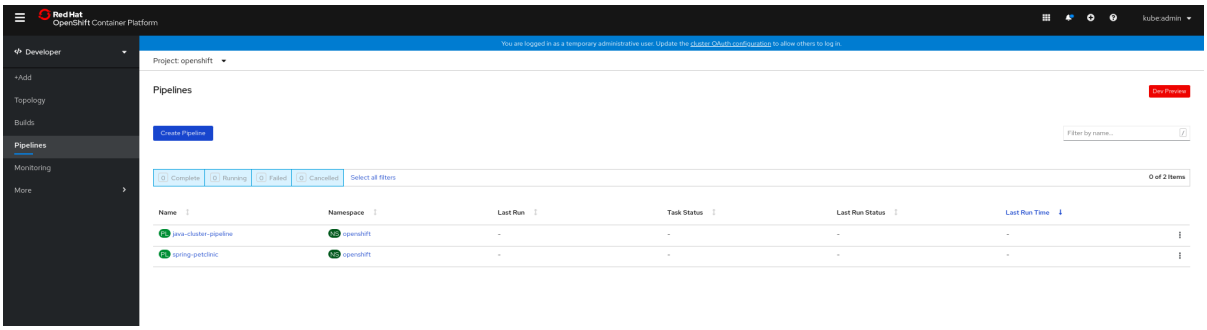
5.3. 通过 DEVELOPER 视角来使用 PIPELINES

Developer 视角 中的 **Pipelines** 视图列出了项目中的所有 Pipelines 以及详情，如创建 Pipeline 的命名空间、最后一个 PipelineRun、PipelineRun 中的 Tasks 的状态、PipelineRun 的状态以及运行所需时间。

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 Pipelines。

图 5.3. Developer 视角中的 pipelines 视图



2. 点所需 Pipeline 查看 **Pipeline Details** 页面。这个页显示 Pipeline 中所有串行和并行任务。这些任务也列在页面的右下角。您可以点击列出的 **任务**来查看具体任务的详情。

3. 另外，在 **Pipeline Details** 页面中：

- 点 **YAML** 选项卡编辑 Pipeline 的 YAML 文件。
- 点击 **Pipeline Runs** 标签页，查看完成、正在运行或运行失败的 Pipeline。您可以使用 Options 菜单 来停止正在运行的 Pipeline，使用与之前的 Pipeline 执行相同的参数和资源重新运行 Pipeline，或删除 PipelineRun。
- 点击 **Parameters** 标签，查看 Pipeline 中定义的参数。您还可以根据需要添加或者编辑附加参数。
- 点击 **Resources** 标签页，查看 Pipeline 中定义的资源。您还可以根据需要添加或编辑附加资源。

5.4. 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。您可以通过 **Pipelines** 视图、**Pipeline Details** 页或 **Topology** 视图来启动一个 Pipeline Run。

流程

使用 **Pipelines** 视图启动 Pipeline：

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。

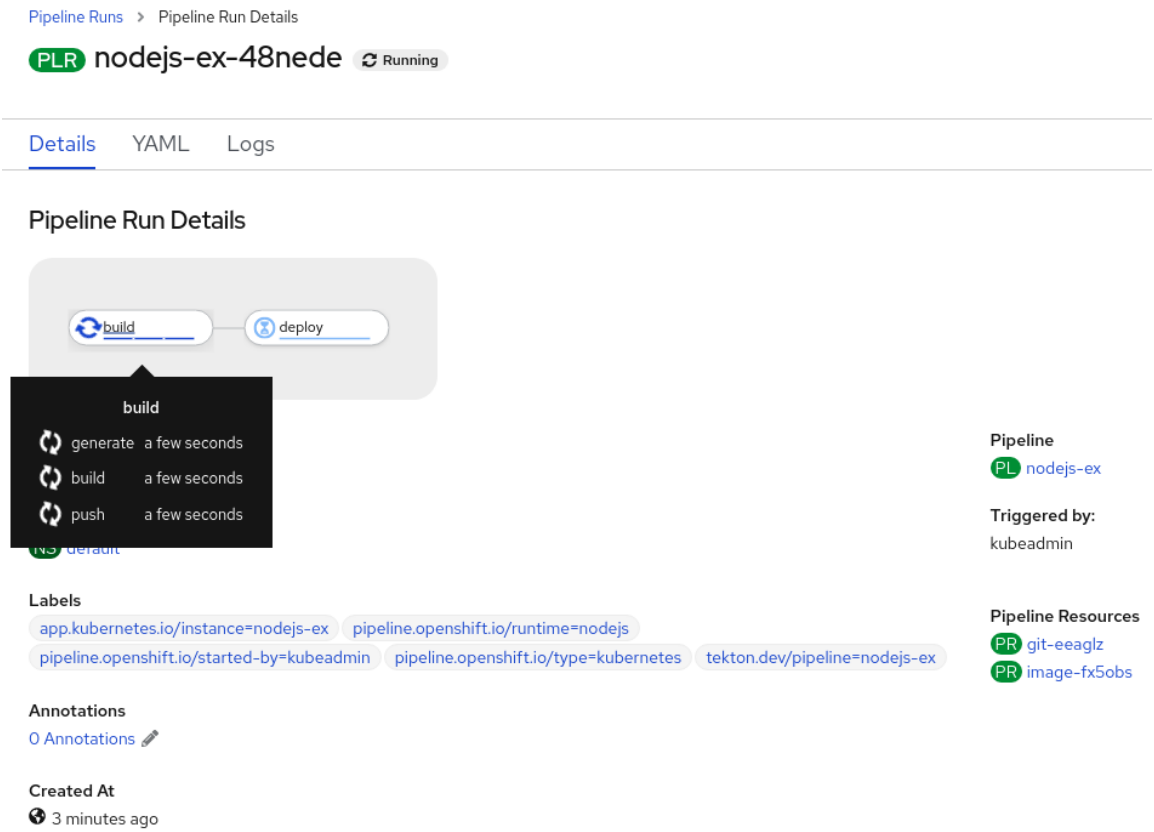
3. 可选：修改 **Advanced Options** 并添加凭证以验证指定的私有 Git 服务器或 Docker registry。
   a. 在 **Advanced Options** 下，点 **Show Credentials Options** 并选择 **Add Secret**。
   b. 在 **Create Source Secret** 部分，指定以下内容：
      i. secret 的唯一 **Secret Name**。
      ii. 在 **要被验证的指定供应商** 部分，在 **Access to** 字段中指定要验证的供应商，以及基本 **服务器 URL**。
      iii. 选择 **Authentication Type** 并提供凭证：
         - 对于 **Authentication Type Image Registry Crendentials**，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。
      iv. 选择要添加 secret 的检查标记。

   您可以根据 Pipeline 中的资源数量添加多个 secret。

4. 点 **Start** 启动 PipelineRun。
5. **Pipeline Run Details** 页面显示正在执行的 Pipeline。Pipeline 启动后，每个任务中的任务和步骤都会被执行。您可以：
   - 将鼠标悬停在任务上，以查看执行每个步骤所需时间。

- 点击一个任务来查看任务中的每个步骤的日志。
- 点 **Logs** 选项卡，根据任务的执行顺序查看日志，并使用 **Download** 按钮将日志下载到文本文件中。

图 5.4. Pipeline 运行



6. 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：

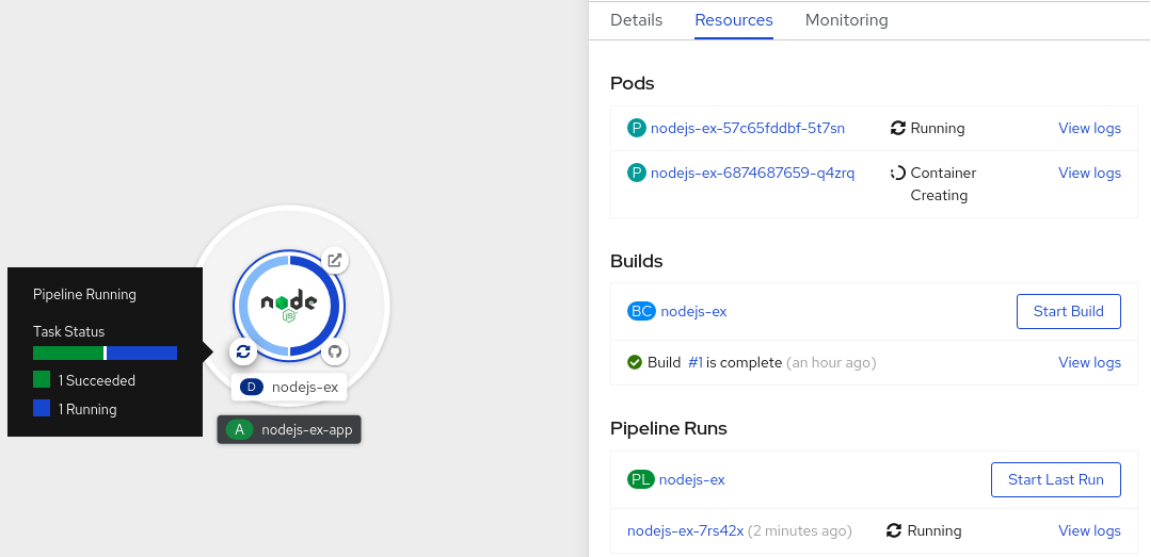
注意

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

a. 在左侧导航面板中，点 **Topology**，然后点应用程序来查看在侧面面板中列出的 Pipeline Run。

b. 在 **Pipeline Runs** 部分,点击 **Start Last Run** 来启动一个新的 Pipeline 运行，使用与前一个相同的参数和资源。如果没有启动 Pipeline Run，这个选项会被禁用。

图 5.5. Topology 视图中的管道



c. 在 **Topology** 页面中，把鼠标移到应用程序的左侧，以查看应用程序的管道运行状态。

5.5. 编辑管道

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

流程

1. 在 **Developer** 视角的 **Pipelines** 视图中，选择您要编辑的管道来查看 Pipeline 的详情。在 **Pipeline Details** 页中，点 **Actions** 并选择 **Edit Pipeline**。
2. 在 **Pipeline Builder** 页中：
   - 您可以将任务、参数或资源添加到管道。
   - 您可以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，要删除此任务，点任务，在侧面面板中点 **Actions**，并选择 **Remove Task**。
3. 点 **Save** 来保存修改的管道。

5.6. 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，点附加到 Pipeline 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。

第 6 章 RED HAT OPENSHIFT PIPELINES 发行注记

Red Hat OpenShift Pipelines 是基于 Tekton 项目的一个云原生 CI/CD 环境，它提供：

- 标准 Kubernetes 原生管道定义 (CRD)。
- 无需 CI 服务器管理开销的无服务器管道。
- 使用任何 Kubernetes 工具（如 S2I、Buildah、JIB 和 Kaniko）构建镜像。
- 不同 Kubernetes 发布系统间的可移植性。
- 用于与管道交互的强大 CLI。
- 使用 OpenShift Container Platform Web 控制台的 Developer 视角集成用户体验。

如需了解 Red Hat OpenShift Pipelines 的概述，请参阅 了解 OpenShift Pipelines。

6.1. 获取支持

如果您在执行本文档所述的某个流程时遇到问题，请访问客户门户网站以获得技术预览功能的相关支持信息。

如果您有疑问或希望提供反馈信息，请向产品团队发送邮件 pipelines-interest@redhat.com。

6.2. RED HAT OPENSHIFT PIPELINES 技术预览 1.1 发行注记

6.2.1. 新功能

Red Hat OpenShift Pipelines 技术预览（TP）1.1 现在包括在 OpenShift Container Platform 4.5 中。Red Hat OpenShift Pipelines TP 1.1 更新为支持：

- Tekton Pipelines 0.14.3
- Tekton **tkn** CLI 0.11.0
- Tekton Triggers 0.6.1
- 基于 Tekton Catalog 0.14 的 ClusterTasks

除了包括修复和稳定性改进的信息外，以下突出介绍了 OpenShift Pipelines 1.1 中的新内容。

6.2.1.1. Pipelines

- 现在可以使用 Workspaces 而不是 PipelineResources。建议您在 OpenShift Pipelines 中使用 Workspaces 而不是 PipelineResources，因为 PipelineResources 很难调试，范围有限，且不容易重复使用。如需有关 Workspaces 的更多信息，请参阅了解 OpenShift Pipelines。
- 添加了对 VolumeClaimTemplates 的工作空间支持：
  - PipelineRun 和 TaskRun 的 VolumeClaimTemplate 现在可以添加为 Workspaces 的卷源。然后，tekton-controller 使用模板创建一个 PersistentVolumeClaim（PVC），该模板被视为 Pipeline 中所有 TaskRuns 的 PVC。因此，您不需要在每次绑定多个任务的工作空间时都定义 PVC 配置。

    - 当 VolumeClaimTemplate 用作卷源时，支持使用变量替换来查找 PersistentVolumeClaim 的名称。

- 支持改进的审核：
    - **PipelineRun.Status** 字段现在包含 Pipeline 中每个 TaskRun 的状态，以及用于实例化 PipelineRun 监控 PipelineRun 的 Pipeline 规格。
    - Pipeline 结果已添加到 pipeline 规格和 **PipelineRun** 状态中。
    - **TaskRun.Status** 字段现在包含用于实例化 **TaskRun** 的具体任务规格。
- 支持为 Conditions 应用默认参数。
- 现在，通过引用 ClusterTask 创建的 TaskRun 会添加 **tekton.dev/clusterTask** 标签，而不是 **tekton.dev/task** 标签。
- **kubeconfigwriter** 现在在资源结构中添加了 **ClientKeyData** 和 **ClientCertificateData** 配置，以便使用 kubeconfig-creator 任务替换 pipeline 资源类型集群。
- 现在，**feature-flags** 和 **config-defaults** ConfigMap 的名称可以自定义。
- 现在，在 TaskRun 使用的 PodTemplate 中支持 HostNetwork。
- 现在，可以使用 Affinity Assistant 支持 TaskRuns 中共享工作空间卷的节点关联性。默认情况下，这在 OpenShift Pipelines 上被禁用。
- PodTemplate 已更新，使用 **imagePullSecrets** 指定在启动一个 pod 时，容器运行时用来拉取容器镜像的 secret。
- 如果控制器无法更新 TaskRun，则支持从 TaskRun 控制器发出警告事件。
- 在所有资源中添加了标准或者推荐的 k8s 标签，以标识属于应用程序或组件的资源。
- 现在，Entrypoint 进程被通知有信号，然后这些信号会使用一个 Entrypoint 进程的专用 PID 组来传播这些信号。
- 现在，PodTemplate 可以在运行时使用 **TaskRunSpecs** 在任务一级设置。
- 支持放出 Kubernetes 事件：
    - 控制器现在会为其他 TaskRun 生命周期事件放出事件 - **taskrun started** 和 **taskrun running**。
    - PipelineRun 控制器现在会在 Pipeline 每次启动时放出一个事件。
- 除了默认的 Kubernetes 事件外，现在还提供对 TaskRuns 的 CloudEvents 的支持。可将控制器配置为发送任何 TaskRun 事件（如创建、启动和失败）作为云事件。
- 支持使用 **$context.<task|taskRun|pipeline|pipeline|pipelineRun>.name** 变量来在 PipelineRuns 和 TaskRuns 中引用正确名称。
- 现在提供了 PipelineRun 参数的验证，以确保 PipelineRun 提供了 Pipeline 所需的所有参数。这也允许 PipelineRuns 在所需参数之外提供额外的参数。
- 现在，您可以使用 Pipeline YAML 文件中的 **finally** 字段指定 Pipeline 中的任务，这些任务会在管道退出前始终执行。

- **git-clone** ClusterTask 现在可用。

6.2.1.2. Pipelines CLI

- **tkn dlistener describe** 命令现在可以支持内嵌的 Trigger 绑定。
- 支持在使用不正确的子命令时推荐子命令并给出建议。
- 现在，如果 Pipeline 中只有一个任务存在，**tkn task describe** 命令会自动选择该任务。
- 现在您可以使用默认参数值启动 Task，方法是在 **tkn task start** 命令中指定 **--use-param-defaults** 标记。
- 现在，您可以使用 **tkn pipeline start** 或 **tkn task start** 命令的 **--workspace** 选项为 PipelineRuns 或 TaskRuns 指定 volumeClaimTemplate。
- **tkn pipelinerun logs** 命令现在会显示 **finally** 部分中列出的最终任务的日志。
- 现在，为 **tkn task start** 命令提供了交互式模式支持，并为以下 tkn 资源提供 **describe** 子命令：**pipeline**、**pipelinerun**、 **task**、**taskrun**、**clustertask** 和 **pipelineresource**。
- **tkn version** 命令现在显示集群中安装的 Triggers 版本。
- **tkn pipeline describe** 命令现在显示为 Pipeline 中使用的任务指定的参数值和超时。
- 添加了对 **tkn pipelinerun describe** 和 **tkn taskrun describe** 命令的 **--last** 选项的支持，以分别描述最新的 PipelineRun 或 TaskRun。
- **tkn pipeline describe** 命令现在显示 Pipeline 中适用于任务的条件。
- 现在，您可以在 **tkn resource list** 命令中使用 **--no-headers** 和 **--all-namespaces** 标记。

6.2.1.3. 触发器

- 现在以下通用表达式语言（CEL）功能可用：
  - **parseURL** 用来解析和提取一个 URL 的部分内容
  - **parseJSON** 用来解析嵌入在 **deployment** webhook 中的 **payload** 字段中的字符串中的 JSON 值类型
- 添加了来自 Bitbucket 的 webhook 的新拦截器。
- 现在，在使用 **kubectl get** 列出时，EventListeners 会显示 **Address URL** 和 **Available status** 作为额外的项。
- TriggerTemplate 参数现在使用 **$(tt.params.<paramName>)** 语法而不是 **$(params.<paramName>)** 来减少 TriggerTemplate 和 ResourceTemplates params 之间的混淆。
- 现在，您可以在 EventListener CRD 中添加 **tolerations（容限）**，以确保 EventListeners 的部署使用相同的配置，即使所有节点都因为安全或管理问题而产生污点也是如此。
- 现在，您可以在 **URL/live** 中为 EventListener Deployment 添加就绪探测（Readiness Probe）。
- 支持在 EventListener Triggers 中嵌入 TriggerBinding 规格。
- 触发器资源现在附带推荐的 **app.kubernetes.io** 标签注解。

6.2.2. 已弃用的功能

本发行版本中已弃用了以下内容：

- 所有集群范围命令（包括 **clustertask** 和 **clustertriggerbinding** 命令）的 **--namespace** 或 **-n** 标志都已弃用。它将在以后的发行版本中被删除。
- EventListener 中的 **triggers.bindings** 中的 **name** 字段已弃用。现在使用 **ref** 字段替代，并将在以后的发行版本中删除。
- 使用 **$(params)** 的 TriggerTemplates 中的变量插值已经被弃用，现在使用 **$(tt.params)** 来减少与 Pipeline 变量插入语法的混乱。在以后的发行版本中会删除 **$(params.<paramName>)** 语法。
- 在 ClusterTasks 上已弃用 **tekton.dev/task** 标签。
- **TaskRun.Status.ResourceResults.ResourceRef** 字段已弃用，并将被删除。
- **tkn pipeline create**、**tkn task create** 和 **tkn resource create -f** 子命令已被删除。
- 从 **tkn** 命令中删除了命名空间验证。
- **tkn ct start** 命令中的默认超时时间（**1h**）以及 **-t** 标志已被删除。
- **s2i** ClusterTask 已被弃用。

6.2.3. 已知问题

- 条件（Conditions）不支持 Workspace。
- **tkn clustertask start** 命令不支持 **--workspace** 选项和互动模式。
- 支持 **$(params.<paramName>)** 的向后兼容性，会强制您使用带有针对 pipeline 参数的 TriggerTemplates，因为 Triggers Webhook 无法将 Trigger 参数和 pipelines 参数区分开。
- 当针对 **tekton_taskrun_count** 和 **tekton_taskrun_duration_seconds_count** 运行一个 promQL 查询时，Pipeline metrics 会报告不正确的值。
- 当为一个 Workspace 指定了一个不存在的 PVC 名称时，PipelineRuns 和 TaskRuns 会维持在 **Running** 和 **Running(Pending)** 的状态。

6.2.4. 修复的问题

- 在以前的版本中，如果 Task 和 ClusterTask 的名称是相同的，则 **tkn task delete <name> --trs** 命令会同时删除 Task 和 ClusterTask。在这个版本中，该命令只删除任务 **<name>** 创建的 TaskRuns。
- 以前，**tkn pr delete -p <name> --keep 2** 命令会在使用 **--keep** 是忽略 **-p** 标志，并将删除除最后两个以外的所有 PipelineRuns。在这个版本中，命令只删除由 Pipeline **<name>** 创建的 PipelineRuns，但最后两个除外。
- **tkn triggertemplate describe** 输出现在以表格式而不是 YAML 格式显示 ResourceTemplates。
- 在以前的版本中，当一个新用户添加到容器时，**buildah** ClusterTask 会失败。在这个版本中，这个问题已被解决。

6.3. RED HAT OPENSHIFT PIPELINES 技术预览 1.0 发行注记

6.3.1. 新功能

Red Hat OpenShift Pipelines 技术预览（TP）1.0 现在包括在 OpenShift Container Platform 4.5 中。Red Hat OpenShift Pipelines TP 1.0 更新为支持：

- Tekton Pipelines 0.11.3
- Tekton **tkn** CLI 0.9.0
- Tekton Triggers 0.4.0
- 基于 Tekton Catalog 0.11 的 ClusterTasks

除了包括修复和稳定性改进的信息外，以下突出介绍了 OpenShift Pipelines 1.0 中的新内容。

6.3.1.1. Pipelines

- 支持 v1beta1 API 版本。
- 支持改进的 LimitRange。在以前的版本中，LimitRange 只能为 TaskRun 和 PipelineRun 指定。现在不需要显式指定 LimitRange。命名空间间使用最小 LimitRange。
- 支持使用 TaskResults 和 TaskParams 在任务间共享数据。
- 现在，管道可以被配置为不覆盖 **HOME** 环境变量和 Steps 的 **WorkDir**。
- 与任务步骤类似，**sidecar** 现在支持脚本模式。
- 现在，您可以在 TaskRun 的 **podTemplate** 中指定不同调度程序的名称 。
- 支持使用 Star Array Notation 替换变量。
- Tekton Controller 现在可以配置为监控单个命名空间。
- 现在，在 Pipeline、Task、ClusterTask、Resource 和 Condition 规格中添加了一个新的 description 字段。
- 在 Git PipelineResources 中添加代理参数。

6.3.1.2. Pipelines CLI

- 现在，为以下 **tkn** 资源添加了 **describe** 子命令：**eventlistener**、**condition**、**triggertemplate**、**clustertask** 和 **triggerbinding**。
- 在以下命令中添加 **v1beta1** 支持以及 **v1alpha1** 的向后兼容性： **clustertask**、**task**、**pipeline**、 **pipelinerun** 和 **taskrun**。
- 以下命令现在可以使用 **--all-namespaces** 标志选项列出所有命名空间的输出结果：
  - **tkn task list**
  - **tkn pipeline list**
  - **tkn taskrun list**
  - **tkn pipelinerun list**
    这些命令的输出也可以通过 **--no-headers** 选项在没有标头的情况下显示信息。

- 现在您可以使用默认参数值启动 Pipeline，方法是在 **tkn pipelines start** 命令中指定 **--use-param-defaults** 标记。
- 现在，在 **tkn pipeline start** 和 **tkn task start** 命令中增加了对 Workspace 的支持。
- 现在增加了一个新命令 **clustertriggerbinding**，它带有以下子命令：**describe**、**delete** 和 **list**。
- 现在，您可以使用本地或远程 **yaml** 文件直接启动管道运行。
- **describe** 子命令现在显示一个改进的详细输出。现在，除了新的项，如 **description**、**timeout**、**param description** 和 **sidecar status**，命令输出还提供了关于一个特定 **tkn** 资源的更详细的信息。
- 现在，如果命名空间中只有一个任务，**tkn task log** 命令会直接显示日志。

6.3.1.3. 触发器

- 现在触发器可以同时创建 **v1alpha1** 和 **v1beta1** Pipeline 资源。
- 支持新的通用表达式语言(CEL)拦截器功能 - **compareSecret**。此功能安全地将字符串与 CEL 表达式中的 secret 进行比较。
- 支持 EventListener Trigger 一级的验证和授权。

6.3.2. 已弃用的功能

本发行版本中已弃用了以下内容：

- Steps 规格中的环境变量 **$HOME**，变量 **workingDir** 已被弃用，并可能在以后的发行版本中有所变化。目前，在 Step 容器中，**HOME** 和 **workingDir** 分别被 **/tekton/home** 和 **/workspace** 覆盖。
  在以后的发行版本中，这两个字段将不会被修改，它将被设置为容器镜像和任务 YAML 中定义的值。在本发行版本中，使用标签 **disable-home-env-overwrite** 和 **disable-working-directory-overwrite** 来禁用对 **HOME** 和 **workingDir** 的覆盖。
- 以下命令已被弃用，并可能在以后的版本中被删除：
  - **tkn pipeline create**
  - **tkn task create**
- 在 **tkn resource create** 命令中使用 **-f** 标志现已弃用。以后的发行版本中可能会删除它。
- **tkn clustertask create** 命令中的 **-t** 标记和 **--timeout** 标记（使用秒格式）现已被弃用。现在只支持持续超时格式，例如 **1h30s**。这些已弃用的标记可能会在以后的版本中删除。

6.3.3. 已知问题

- 如果您要从 Red Hat OpenShift Pipelines 的旧版本升级，则必须删除您现有的部署，然后再升级到 Red Hat OpenShift Pipelines 版本 1.0。要删除现有的部署，您必须首先删除自定义资源，然后卸载 Red Hat OpenShift Pipelines Operator。如需了解更多详细信息，请参阅卸载 Red Hat OpenShift Pipelines 部分。
- 提交相同的 **v1alpha1** 任务多次会导致错误。在重新提交一个 **v1alpha1** 任务时，使用 **oc replace** 而不是使用 **oc apply**。

- 当向一个容器添加一个新用户时，**buildah** ClusterTask 并不会工作。
  当安装 Operator 时，**buildah** ClusterTask 的 **--storage-driver** 标志没有指定，因此它会被设置为默认值。在某些情况下，这会导致存储驱动程序设置不正确。当添加一个新用户时，错误的 storage-driver 会造成 **buildah** ClusterTask 失败并带有以下错误：

```
useradd: /etc/passwd.8: lock file already used
useradd: cannot lock /etc/passwd; try again later.
```

  作为临时解决方案，在 **buildah-task.yaml** 文件中手工把 **--storage-driver** 标识的值设置为 **overlay**：

  1. 以 **cluster-admin** 身份登录到集群:

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

  2. 使用 **oc edit** 命令编辑 **buildah** ClusterTask:

```
$ oc edit clustertask buildah
```

     buildah clustertask YAML 文件的最新版本会在由 **EDITOR** 环境变量指定的编辑器中打开。

  3. 在 **steps** 字段中找到以下 **command** 字段：

```
 command: ['buildah', 'bud', '--format=$(params.FORMAT)', '--tls-
verify=$(params.TLSVERIFY)', '--layers', '-f', '$(params.DOCKERFILE)', '-t',
'$(resources.outputs.image.url)', '$(params.CONTEXT)']
```

  4. 使用以下内容替换 **command** 字段：

```
 command: ['buildah', '--storage-driver=overlay', 'bud', '--format=$(params.FORMAT)', '--
tls-verify=$(params.TLSVERIFY)', '--no-cache', '-f', '$(params.DOCKERFILE)', '-t',
'$(params.IMAGE)', '$(params.CONTEXT)']
```

  5. 保存文件并退出。

  另外，您还可以直接在 web 控制台中直接修改 **buildah** ClusterTask YAML 文件。导航到 **Pipelines → Cluster Tasks → buildah**。从 **Actions** 菜单中选择 **Edit Cluster Task**，如前所示替换 **command** 项。

6.3.4. 修复的问题

- 在以前的版本中，即使镜像构建已在进行中，**DeploymentConfig** 任务也会触发新的部署构建。这会导致 Pipeline 的部署失败。在这个版本中，**deploy task** 命令被 **oc rollout status** 命令替代，它会等待正在进行中的部署完成。

- 现在在 Pipeline 模板中添加了对 **APP_NAME** 参数的支持。

- 在以前的版本中，Java S2I 的 Pipeline 模板无法在 registry 中查询镜像。在这个版本中，使用现有镜像 PipelineResources 而不是用户提供的 **IMAGE_NAME** 参数来查找镜像。

- 所有 OpenShift Pipelines 镜像现在都基于 Red Hat Universal Base Images（UBI）。

- 在以前的版本中，当 Pipeline 在 **tekton-pipelines** 以外的命名空间中安装时，**tkn version** 命令会将 Pipeline 版本显示为 **unknown**。在这个版本中，**tkn version** 命令会在任意命名空间中显示正确的 Pipeline 版本。
- **tkn version** 命令不再支持 **-c** 标志。
- 非管理员用户现在可以列出 ClusterTriggerBindings。
- 现在为 CEL 拦截器修复了 EventListener CompareSecret 功能。
- 现在，**task** 和 **clustertask** 的 **list**、**describe** 和 **start** 子命令在 Task 和 ClusterTask 有相同名称时可以正确地显示输出。
- 在以前的版本中，OpenShift Pipelines Operator 修改了特权安全性上下文约束 (SCC)，这会在集群升级过程中造成错误。这个错误现已解决。
- 在 **tekton-pipelines** 命名空间中，现在将所有 TaskRuns 和 PipelineRuns 的超时设置为使用 ConfigMap 的 **default-timeout-minutes** 字段。
- 在以前的版本中，Web 控制台中的 Pipelines 部分没有为非管理员用户显示。这个问题现已解决。