OpenShift Container Platform 4.4

Pipelines

在 OpenShift Container Platform 中配置和使用 Pipelines

Last Updated: 2021-03-11

OpenShift Container Platform 4.4 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/。

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

- 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。
- OpenShift Pipelines 是为开发基于微服务架构的非中心化团队设计的。
- 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 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. OPENSHIFT PIPELINES 概念

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。

PipelineResource

PipelineResource 是一个用于 Pipeline 任务的输入和输出的对象。例如，如果输入是 Git 存储库，输出是从该 Git 存储库构建的容器镜像，则它们都被归类为 PipelineResources。PipelineResources 目前支持 Git 资源、镜像资源、集群资源、存储资源和 CloudEvent 资源。

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。

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 OPENSHIFT PIPELINES OPERATOR

您可以使用 OpenShift Container Platform OperatorHub 中列出的 Operator 来安装 OpenShift Pipelines。安装 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 **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 **OpenShift Pipelines Operator** 页中有关 Operator 的简单描述。点 **Install**.
4. 在 **Create Operator Subscription** 页面：
   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 OpenShift Pipelines Operator 最新稳定版本的安装。
      - **preview3** 频道启用 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 **Subscribe**。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 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 OpenShift Pipelines Operator。

流程

1. 创建一个订阅对象 YAML 文件，以便为 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
```

   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 OpenShift Pipelines Operator 分为两个步骤：

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

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

3.1. 删除 OPENSHIFT PIPELINES 组件和自定义资源

删除安装 OpenShift Pipelines Operator 的过程中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 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 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 OPENSHIFT PIPELINES OPERATOR

流程

1. 在 **Operators → OperatorHub** 页面中，使用 **Filter by keyword** 复选框来搜索 **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 使用 OPENSHIFT PIPELINES 创建应用程序

使用 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。
- 创建 Pipeline 和 PipelineResources，以定义应用程序的 Pipeline。
- 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。

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 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 获得了前端 **ui-repo** 和后端 **api-repo** 的 Git 存储库的副本。
- 具有管理员访问权限来访问您的软件存储库。

4.1. 创建项目并检查 PIPELINE SERVICEACCOUNT

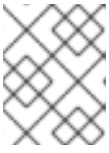
流程

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：

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
```

4.2. 关于任务

*任务（Task）*是 Pipeline 的构建块，它由带有一定顺序的执行步骤组成。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。

任务使用 **inputs** 参数（如 Git 资源），**outputs** 参数（如 registry 中的镜像）与其他任务交互。它们可以重复使用，并可用于多个 Pipelines。

这里是一个使用单一步骤构建基于 Maven 的 Java 应用程序的 Maven 任务示例。

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: maven-build
spec:
  resources:
    inputs:
    - name: workspace-git
      targetPath: /
      type: git
  steps:
  - name: build
    image: maven:3.6.0-jdk-8-slim
    command:
    - /usr/bin/mvn
    args:
    - install
```

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。

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-1/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-1/01_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
```

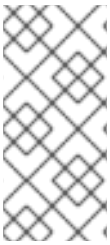
输出中列出了 Operator 安装的 ClusterTasks:

```
NAME                      DESCRIPTION   AGE
buildah                               1 day ago
buildah-v0-11-3                       1 day ago
git-clone                             1 day ago
jib-maven                             1 day ago
kn                                    1 day ago
maven                                 1 day ago
openshift-client                      1 day ago
openshift-client-v0-11-3              1 day ago
s2i                                   1 day ago
s2i-dotnet-3                          1 day ago
s2i-dotnet-3-v0-11-3                  1 day ago
s2i-go                                1 day ago
s2i-go-v0-11-3                        1 day ago
s2i-java-11                           1 day ago
s2i-java-11-v0-11-3                   1 day ago
s2i-java-8                            1 day ago
s2i-java-8-v0-11-3                    1 day ago
s2i-nodejs                            1 day ago
s2i-nodejs-v0-11-3                    1 day ago
s2i-perl                              1 day ago
s2i-perl-v0-11-3                      1 day ago
s2i-php                               1 day ago
s2i-php-v0-11-3                       1 day ago
s2i-python-3                          1 day ago
s2i-python-3-v0-11-3                  1 day ago
s2i-ruby                              1 day ago
s2i-ruby-v0-11-3                      1 day ago
s2i-v0-11-3                           1 day ago
tkn                                   1 day ago
```

4.4. 定义并创建 PIPELINERESOURCE

PipelineResources 是用作任务输入或输出的工件（artifact）。

创建任务后，创建 PipelineResources，其中包含 Pipeline 执行期间要使用的 Git 存储库和镜像 registry 的具体内容：



注意

如果您没有处于 **pipelines-tutorial** 命名空间，并且正在使用其他命名空间，请更新以下步骤中的前端和后端镜像资源以使 URL 包括正确的命名空间。例如：

```
image-registry.openshift-image-registry.svc:5000/<namespace-name>/vote-api:latest
```

流程

1. 为前端应用程序创建定义 Git 存储库的 PipelineResource:

```
$ tkn resource create
? Enter a name for a pipeline resource : ui-repo
? Select a resource type to create : git
? Enter a value for url : http://github.com/openshift-pipelines/vote-ui.git
? Enter a value for revision : release-tech-preview-1
```

检查输出来验证 **ui-repo** PipelineResource 是否已创建。

```
New git resource "ui-repo" has been created
```

2. 创建一个 PipelineResource，用于定义 OpenShift Container Platform 内部镜像 registry（您要将前端镜像推送到的位置）：

```
$ tkn resource create
? Enter a name for a pipeline resource : ui-image
? Select a resource type to create : image
? Enter a value for url : image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/ui:latest
? Enter a value for digest :
```

检查输出来验证 **ui-image** PipelineResource 是否已创建。

```
New image resource "ui-image" has been created
```

3. 创建用于为后端应用程序定义 Git 存储库的 PipelineResource:

```
$ tkn resource create
? Enter a name for a pipeline resource : api-repo
? Select a resource type to create : git
? Enter a value for url : http://github.com/openshift-pipelines/vote-api.git
? Enter a value for revision : release-tech-preview-1
```

检查输出来验证是否已创建了 **api-repo** PipelineResource。

```
New git resource "api-repo" has been created
```

4. 创建一个 PipelineResource，用于定义 OpenShift Container Platform 内部镜像 registry（您要将后端镜像推送到的位置）：

```
$ tkn resource create
? Enter a name for a pipeline resource : api-image
? Select a resource type to create : image
? Enter a value for url : image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/api:latest
? Enter a value for digest :
```

检查输出来验证是否已创建 **api-image** PipelineResource。

```
New image resource "api-image" has been created
```

5. 查看创建的 **resources** 列表：

```
$ tkn resource list
```

输出会列出创建的所有 PipelineResource。

```
NAME       TYPE    DETAILS
api-repo   git     url: http://github.com/openshift-pipelines/vote-api.git
ui-repo    git     url: http://github.com/openshift-pipelines/vote-ui.git
api-image  image   url: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/api:latest
ui-image   image   url: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/ui:latest
```

4.5. 组装 PIPELINE

一个 Pipeline 代表一个 CI/CD 流，由要执行的任务定义。它通过 **inputs**、**outputs** 和 **runAfter** 参数来指定不同的任务间如何进行交互以及它们执行的顺序。它被设计为在多个应用程序和环境中通用且可重复使用。

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：

- 从 Git 存储库 **api-repo** 和 **ui-repo** 中克隆应用程序的源代码。
- 使用 **buildah** ClusterTask 构建容器镜像 **api-image** 和 **ui-image**。
- 将 **api-image** 和 **ui-image** 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:
  resources:
  - name: git-repo
    type: git
  - name: image
    type: image
  params:
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  tasks:
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    resources:
      inputs:
      - name: source
        resource: git-repo
      outputs:
      - name: image
        resource: image
    params:
    - name: TLSVERIFY
      value: "false"
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    resources:
      inputs:
      - name: source
        resource: git-repo
    runAfter:
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    resources:
      inputs:
      - name: image
        resource: image
    params:
    - name: deployment
      value: $(params.deployment-name)
    runAfter:
    - apply-manifests
```

请注意，Pipeline 定义提取了要在 Pipeline 执行过程中使用的 Git 源存储库和镜像 registry 的特定内容。

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-1/01_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   ---        ---       ---        ---
```

4.6. 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 -r git-repo=api-repo -r image=api-image -p deployment-name=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 -r git-repo=ui-repo -r image=ui-image -p deployment-name=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. 关于触发器

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 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。

4.8. 在 PIPELINE 中添加触发器

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-1/03_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: master
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1alpha1
    kind: PipelineResource
    metadata:
      name: $(params.git-repo-name)-git-repo-$(uid)
    spec:
      type: git
      params:
      - name: revision
        value: $(params.git-revision)
      - name: url
        value: $(params.git-repo-url)

  - apiVersion: tekton.dev/v1alpha1
    kind: PipelineResource
    metadata:
      name: $(params.git-repo-name)-image-$(uid)
    spec:
      type: image
```

```
      params:
      - name: url
        value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/$(params.git-
repo-name):latest

  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(params.git-repo-name)-$(uid)
    spec:
      serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      resources:
      - name: git-repo
        resourceRef:
          name: $(params.git-repo-name)-git-repo-$(uid)
      - name: image
        resourceRef:
          name: $(params.git-repo-name)-image-$(uid)
      params:
      - name: deployment-name
        value: $(params.git-repo-name)
```

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-1/03_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:
    - name: 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-1/03_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
```

4.9. 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**。

4.10. 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-api.git -b release-tech-preview-1
```

2. 推送空提交：

```
$ git commit -m "empty-commit" --allow-empty && git push origin release-tech-preview-1
```

3. 检查 PipelineRun 是否被触发：

```
$ tkn pipelinerun list
```

请注意，一个新的 PipelineRun 被启动。

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 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，同时在 OpenShift Container Platform 上创建应用程序。

创建 Pipelines 后，您可以查看并与您部署的 Pipelines 进行视觉交互。

<Discrete><title>先决条件</title>

- 可以访问 OpenShift Container Platform 集群并已登陆到 web 控制台。
- 有集群管理员特权来安装 Operator，并 安装了 OpenShift Pipelines Operator。
- 处于 Developer 视角。
- 您已创建了一个项目。

</Discrete>

5.1. 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。
2. 点击所需 Pipeline 查看 **Pipeline Details** 页面。这可显示 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。

第 6 章 OPENSHIFT PIPELINES 发行注记

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 OpenShift Pipelines 的概述信息，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 1.0 发行注记

6.2.1. 新功能

OpenShift Pipelines 技术预览 (TP) 1.0 现在包括在 OpenShift Container Platform 4.4 中。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。

6.2.1.1. 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。

6.2.1.2. 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。

6.2.1.3. 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。

6.2.2. 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。

6.2.3. 已知问题

- 如果您要从 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 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 OpenShift Pipelines Operator。如需了解更多详细信息，请参阅卸载 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。

6.2.4. 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。