构建 Tekton 管道



预计所需时间： 40 分钟

欢迎来到 **构建 Tekton 管道** 的动手实验。在本实验中，您将创建一个简单的 Tekton 管道，第一步中包含一个任务，然后在第四步中为其添加一个参数。您将学习构建 Tekton 管道项目的最佳实践，以及如何编写易于使用和参数化的 Tekton 管道和任务。您将看到 Tekton 允许您重用您的管道作为代码的工件，并且您将了解将管道和任务定义发布到 Git 存储库的实际方法。

学习目标

完成本实验后，您将能够：

- 创建一个基础管道和任务以回显消息。
- 将参数应用于任务和管道。
- 为管道应用额外参数以克隆 Git 存储库。

先决条件

您需要以下条件才能完成本实验中的练习：

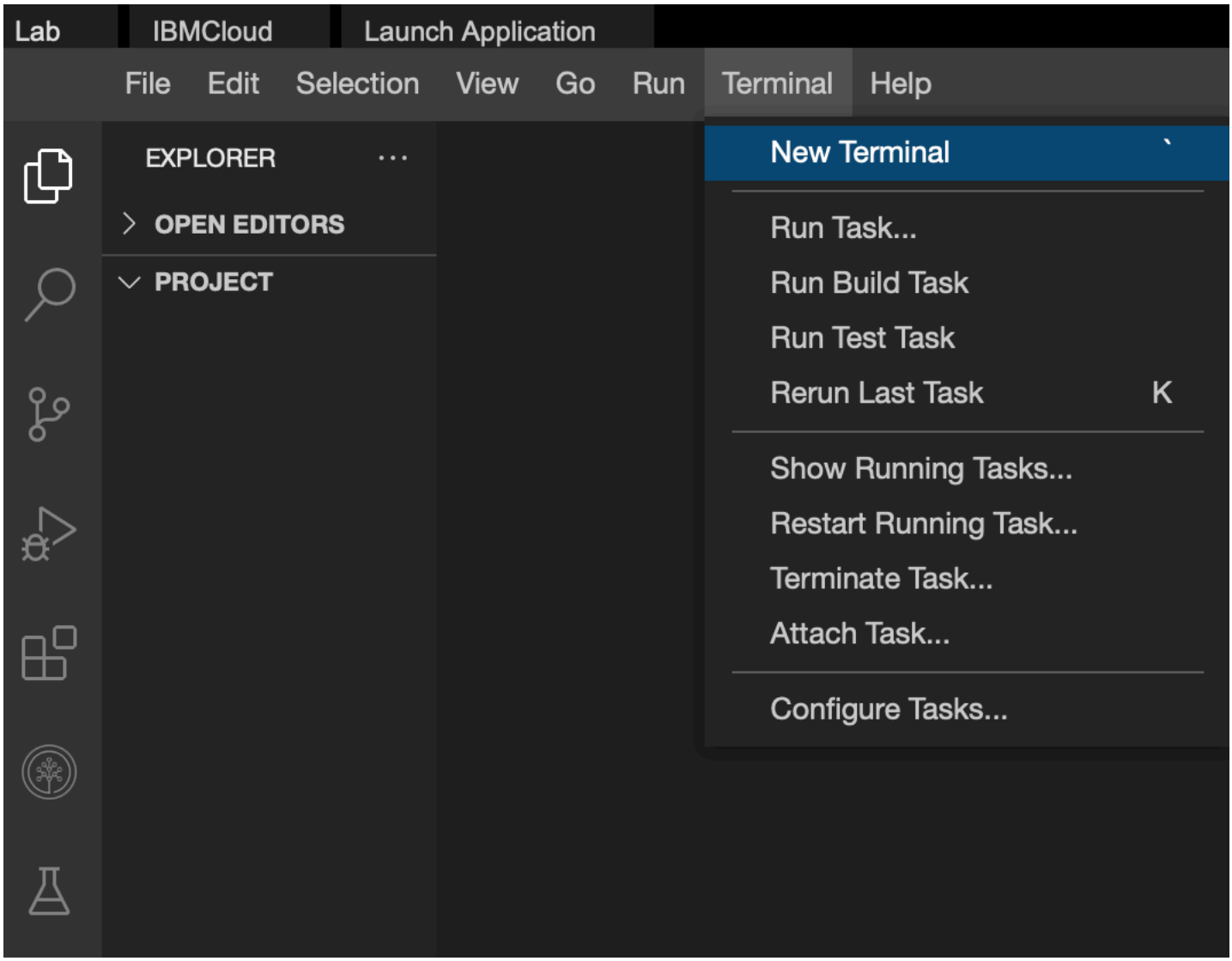
- 对 YAML 有基本了解
- 拥有一个 GitHub 账户
- 对 CLI 有中级水平的知识

设置实验环境

在开始实验之前，您需要进行一些准备工作。

打开终端

通过编辑器中的菜单打开终端窗口：终端 > 新建终端。



在终端中，如果您尚未在 `/home/project` 文件夹中，请现在切换到您的项目文件夹。

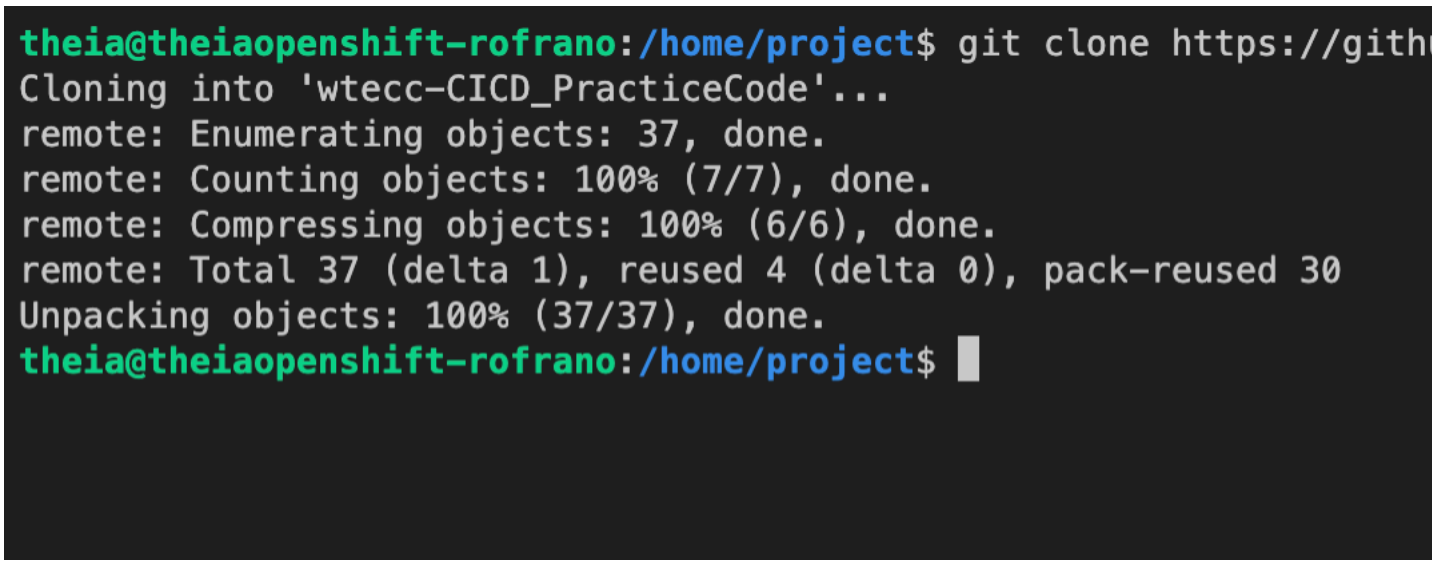
```
cd /home/project
```

克隆代码库

现在，获取您需要测试的代码。为此，请使用 `git clone` 命令克隆 Git 仓库：

```
git clone https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git
```

您的输出应类似于下图所示：



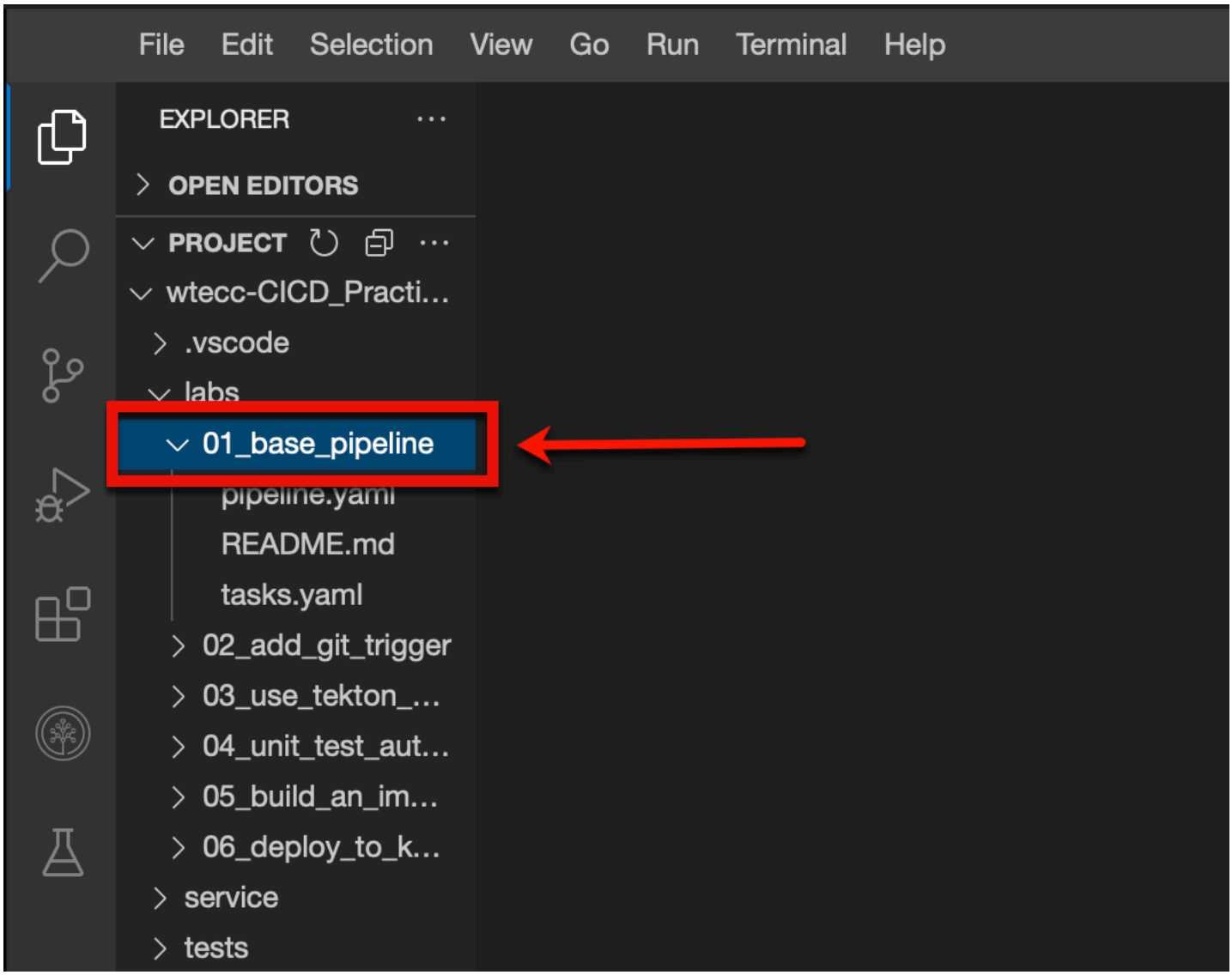
切换到实验目录

一旦您克隆了仓库，请切换到实验目录。

```
cd wtecc-CICD_PracticeCode/labs/01_base_pipeline/
```

导航到实验文件夹

在左侧资源管理器面板中导航到 `labs/01_base_pipeline` 文件夹。您所有的工作将与此文件夹中的文件完成。



您现在准备开始实验。

可选

如果在终端中工作变得困难，因为命令提示符很长，您可以使用以下命令来缩短提示符：

```
export PS1="[\[\033[01;32m\]\u\[\033[00m\]: \[\033[01;34m\]\W\[\033[00m\]]\$ "
```

步骤 1：创建一个回声任务

按照计算机编程的传统，您创建的第一个任务将向控制台回显 "Hello World!"。

在 `labs/01_base_pipeline` 文件夹中有一个任务和管道的起始代码。请在左侧资源管理器面板中导航到此文件夹，并打开 `tasks.yaml` 文件进行编辑：

Open **tasks.yaml** in IDE

它应该看起来像这样：

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: <place-name-here>
spec:
  steps:
```

你现在将创建一个 **hello-world** 任务。

你的任务

1. 首先，你需要给任务起个好名字。将 `<place-name-here>` 改为 `hello-world`。
2. 接下来，添加一个步骤，运行一个单一命令，包括 `name`、`image`、`command` 和 `args`。将名称设为 `echo`，使用镜像 `alpine:3`，命令为 `[/bin/echo]`，参数为 `["Hello World"]`。

提示

▶ 点击这里获取提示。

```
spec:
  steps:
    - name: {name here}
      image: {image here}
      command: {command here}
      args: {args here}
```

仔细检查你的工作是否与下面的解决方案匹配。

解决方案

▼ 点击这里查看答案。

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: hello-world
spec:
  steps:
    - name: echo
      image: alpine:3
      command: [/bin/echo]
      args: ["Hello World!"]
```

将其应用于集群：

```
kubectl apply -f tasks.yaml
```

步骤 2：创建 hello-pipeline 流水线

接下来，您将创建一个非常简单的流水线，仅调用您刚刚创建的 `hello-world` 任务。在左侧资源管理器面板中导航到此文件夹，并打开 `pipeline.yaml` 文件进行编辑：

Open **pipeline.yaml** in IDE

它应该看起来像这样：

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: <place-name-here>
spec:
  tasks:
```

您现在将创建一个 **hello-pipeline** 管道。

任务

1. 首先，您需要为管道命名。将 `<place-name-here>` 更改为 `hello-pipeline`。
2. 接下来，添加对您刚刚创建的 hello-world 任务的引用，该引用需要一个 `name:` 用于管道任务，以及一个 `taskRef:`，其中包含一个 `name:` 标签，设置为您引用的任务的名称。将管道任务的名称设置为 `hello`，将您引用的任务的名称设置为 `hello-world`。

提示

▼ 点击此处获取提示。
在 `tasks:` 标签下添加一个任务。由于可以有多个任务，每个任务以破折号 `-` 开头，以将其标识为 `yaml` 列表中的项目。

```
spec:
  tasks:
    - name: {name here}
      taskRef:
        name: {task name here}
```

仔细检查你的工作是否与下面的解决方案匹配。

解决方案

▼ 点击这里查看答案。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
```

```
    name: hello-pipeline
  spec:
    tasks:
      - name: hello
        taskRef:
          name: hello-world
```

将其应用于集群：

```
kubectl apply -f pipeline.yaml
```

您现在准备好运行您的管道并查看它是否有效。

步骤 3：运行 hello-pipeline

使用 Tekton CLI 运行管道：

```
tkn pipeline start --showlog hello-pipeline
```

您应该看到输出：

```
PipelineRun started: hello-pipeline-run-9vkbb
Waiting for logs to be available...
[hello : echo] Hello World!
```

恭喜你！你刚刚运行了自己创建的第一个管道和任务。

第 4 步：向任务添加参数

希望 hello-world 任务让你对管道如何调用任务有了一定的了解。现在是时候通过让任务打印你想要的任何消息，而不仅仅是“Hello World”，来使其更加实用。

为此，你将向任务添加一个名为 `message` 的参数，并使用该参数作为它回显的消息。你还将把任务重命名为 `echo`。

编辑 `tasks.yaml` 文件，为输入和回显命令添加参数：

Open **tasks.yaml** in IDE

你的任务

1. 将任务的名称从 `hello-world` 更改为 `echo`，以更准确地反映其新功能，方法是更改 `metadata:` 部分中的 `name:`。
2. 向任务添加一个 `params:` 部分，该参数的 `name:` 为“message”，`type:` 为“string”，`description` 为“要回显的消息”。
3. 将步骤的名称从 `echo` 更改为 `echo-message`，以更好地描述其新功能。
4. 修改 `args:` 标签，以使用你刚创建的消息参数。

提示

▼ 点击这里获取提示。
在 `spec:` 标签下添加一个 `params:` 部分。由于可以有多个参数，每个参数以破折号 `-` 开头，以将其标识为 `yaml` 列表中的一个项目。

对消息参数的引用应为 `$(params.message)`。

```
metadata:
  name: {change the task name}
spec:
  params:
    - name: {name here}
      description: {description here}
      type: {type here}
  steps:
    - name: echo-message
      image: alpine:3
      command: [/bin/echo]
      args: {place parameter reference here}
```

仔细检查您的工作是否与下面的解决方案匹配。

解决方案

▼ 点击这里查看答案。

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: echo
spec:
  params:
    - name: message
      description: The message to echo
      type: string
  steps:
    - name: echo-message
```

```
      image: alpine:3
      command: [/bin/echo]
      args: ["$(params.message)"]
```

将新任务定义应用于集群：

```
kubectl apply -f tasks.yaml
```

步骤 5：更新 hello-pipeline

您现在需要更新管道，以将您想要发送的消息传递给 `echo` 任务，以便它可以将消息回显到控制台。

编辑 `pipeline.yaml` 文件以添加参数：

Open **pipeline.yaml** in IDE

你的任务

1. 在管道的 `spec:` 下添加一个 `params:` 部分，其中包含一个 `name:` 为“message”的参数。
2. 将 `taskRef:` 的 `name:` 从 `hello-world` 更改为新的 `echo` 任务。
3. 在任务中添加一个 `params:` 部分，其中包含一个 `name:` 为“message”的参数，并且 `value:` 是对管道参数 `params.message` 的引用。

提示

▼ 点击这里获取提示。
在任务中指定消息参数的值为 "$(params.message)"。

```
spec:
  params:
    - name: {parameter name here}
  tasks:
    - name: hello
      taskRef:
        name: {change parameter value here}
      params:
        - name: {task parameter name here}
          value: "$(params.message)"
```

仔细检查你的工作是否与下面的解决方案匹配。

解决方案

▼ 点击这里查看答案。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: hello-pipeline
spec:
  params:
    - name: message
  tasks:
    - name: hello
      taskRef:
        name: echo
      params:
        - name: message
          value: "$(params.message)"
```

将其应用于集群：

```
kubectl apply -f pipeline.yaml
```

第6步：运行消息管道

使用 Tekton CLI 运行管道：

```
tkn pipeline start hello-pipeline \
    --showlog  \
    -p message="Hello Tekton!"
```

您应该看到输出：

```
PipelineRun started: hello-pipeline-run-9qf42
Waiting for logs to be available...
[hello : echo-message] Hello Tekton!
```

恭喜你！你刚刚创建并运行了一个需要参数的管道。

第7步：创建一个检出任务

在这一步中，您将结合在容器中运行命令的知识与传递参数的知识，创建一个任务，从 GitHub 检出您的代码，作为 CD 流水线的第一步。

创建检出任务

您可以在单个 yaml 文件中有多个定义，通过在单行中用三个短横线 `---` 分隔它们。在这一步中，您将向 `tasks.yaml` 添加一个新任务，该任务使用 `bitnami/git:latest` 镜像运行 `git` 命令，并传入您想要克隆的分支名称和仓库 URL。

打开 `tasks.yaml` 文件以创建新任务：

Open **tasks.yaml** in IDE

在单独的一行上添加三个短横线：

```
---
```

你现在准备好添加你的新任务。

你的任务

你的新任务将创建一个 Tekton 任务，该任务接受一个代码库 URL 和一个分支名称，并调用 `git clone` 来克隆你的源代码。

1. 创建一个新任务，并命名为 `checkout`。
2. 添加一个名为 `repo-url` 的参数，`type:` 为字符串，`description:` 为“要克隆的 git 仓库的 URL”。
3. 添加第二个名为 `branch` 的参数，`type:` 为字符串，`description:` 为“要克隆的分支”。
4. 添加一个步骤，`name:` 为“checkout”，使用 `bitnami/git:latest` 镜像运行 `git` 命令，指定 `clone` 和 `--branch` 参数，并将规范中创建的两个参数作为参数传递。

提示

▼ 点击这里获取提示。
将参数传递给 checkout 任务，格式为 ["clone", "--branch", "$(params.branch)", "$(params.repo-url)"]。

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: {name of task here}
spec:
  params:
    - name: {1st parameter name here}
      description: {1st parameter name here}
      type: string
    - name: {2nd parameter name here}
      description: {2nd parameter name here}
      type: string
  steps:
    - name: {step name here}
      image: {image name here}
      command: [{command name here}]
      args: [{arguments here}]
```

仔细检查你的工作是否与下面的解决方案匹配。

解决方案

▼ 点击这里查看答案。

```
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: checkout
spec:
  params:
    - name: repo-url
      description: The URL of the git repo to clone
      type: string
    - name: branch
      description: The branch to clone
      type: string
  steps:
    - name: checkout
      image: bitnami/git:latest
      command: [git]
      args: ["clone", "--branch", "$(params.branch)", "$(params.repo-url)"]
```

将其应用于集群：

```
kubectl apply -f tasks.yaml
```

请提供需要翻译的文本。

```
task.tekton.dev/echo configured
task.tekton.dev/checkout created
```

`echo` 任务未更改，而 `checkout` 任务已创建。

步骤 8：创建 cd-pipeline 管道

最后，您将创建一个名为 `cd-pipeline` 的管道，作为您的持续交付管道的起点。

打开 `pipeline.yaml` 文件以创建一个名为 `cd-pipeline` 的新管道：

Open **pipeline.yaml** in IDE

您可以在单独的一行上使用 `---` 来分隔您的新管道，或者您可以修改现有管道使其看起来像新管道。

任务

1. 创建一个新管道并命名为 `cd-pipeline`。
2. 添加两个参数，分别命名为 `repo-url` 和 `branch`。
3. 将 **branch** 的 `default:` 设置为 "master"。
4. 添加一个名为 "clone" 的任务，该任务的 `taskRef:` 指向您刚刚创建的 `checkout` 任务。
5. 将两个参数 `repo-url` 和 `branch` 添加到任务中，并将它们映射回同名的管道参数。

提示

▼ 点击这里获取提示。
添加名称为 `repo-url` 和 `branch` 的参数，值分别为 "$(params.repo-url)" 和 "$(params.branch)" 到 `clone` 任务中。

```
spec:
  params:
    - name: {1st parameter name here}
    - name: {2nd parameter name here}
      default: {default value here}
  tasks:
    - name: {pipeline task name here}
      taskRef:
        name: {Task name here}
      params:
      - name: {1st parameter name here}
        value: "$(params.repo-url)"
      - name: {2nd parameter name here}
        value: "$(params.branch)"
```

仔细检查你的工作是否与下面的解决方案匹配。

解决方案

▼ 点击这里查看答案。

```
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: cd-pipeline
spec:
  params:
    - name: repo-url
    - name: branch
      default: "master"
  tasks:
    - name: clone
      taskRef:
        name: checkout
      params:
      - name: repo-url
        value: "$(params.repo-url)"
      - name: branch
        value: "$(params.branch)"
```

将其应用于集群：

```
kubectl apply -f pipeline.yaml
```

步骤 9：运行 cd-pipeline

使用 Tekton CLI 运行管道：

```
tkn pipeline start cd-pipeline \
    --showlog \
    -p repo-url="https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git" \
    -p branch="main"
```

请提供您希望翻译的文本片段。

```
PipelineRun started: cd-pipeline-run-rf6zp
Waiting for logs to be available...
[clone : checkout] Cloning into 'wtecc-CICD_PracticeCode'...
```

步骤 10：用占位符填写 cd-pipeline

在最后一步中，您将用调用 `echo` 任务来填写其余的管道，以简单地显示一条消息。您将在未来的实验中将这些“占位符”任务替换为真实任务。

更新 pipeline.yaml 文件以包含四个占位符任务。

Open **pipeline.yaml** in IDE

现在您将向管道添加四个任务，以进行代码检查、单元测试、构建和部署。所有这些管道任务目前都将引用 `echo` 任务。

你的任务

为每个任务创建一个管道任务：

任务名称	构建后	消息
lint	clone	调用 Flake8 代码检查器...
tests	lint	使用 PyUnit 运行单元测试...
build	tests	为 $(params.repo-url) 构建镜像...
deploy	build	部署 $(params.repo-url) 的 $(params.branch) 分支...

提示

▼ 点击这里获取提示。
向 `clone` 任务添加名称为 `repo-url` 和 `branch` 的参数，值为 "$(params.repo-url)" 和 "$(params.branch)"。

```
spec:
  tasks:
    - name: {pipeline task name here}
      taskRef:
        name: echo
      params:
      - name: message
        value: {message to display here}
      runAfter:
        - {name of previous task}
    ...
```

您现在拥有一个基础管道，可以将其余任务构建到其中。

仔细检查您的工作是否与下面的解决方案匹配。

解决方案

▼ 点击这里查看答案。

```
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: cd-pipeline
spec:
  params:
    - name: repo-url
    - name: branch
      default: "master"
  tasks:
    - name: clone
      taskRef:
        name: checkout
      params:
      - name: repo-url
        value: "$(params.repo-url)"
      - name: branch
        value: "$(params.branch)"
    - name: lint
      taskRef:
        name: echo
      params:
      - name: message
        value: "Calling Flake8 linter..."
      runAfter:
        - clone
    - name: tests
      taskRef:
        name: echo
      params:
      - name: message
        value: "Running unit tests with PyUnit..."
      runAfter:
        - lint
    - name: build
      taskRef:
        name: echo
      params:
      - name: message
```

```
      value: "Building image for $(params.repo-url) ..."
    runAfter:
      - tests
  - name: deploy
    taskRef:
      name: echo
    params:
    - name: message
      value: "Deploying $(params.branch) branch of $(params.repo-url) ..."
    runAfter:
      - build
```

将其应用于集群：

```
kubectl apply -f pipeline.yaml
```

步骤 11：运行 cd-pipeline

使用 Tekton CLI 运行管道：

```
tkn pipeline start cd-pipeline \
    --showlog \
    -p repo-url="https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git" \
    -p branch="main"
```

输出将如下所示：

```
PipelineRun started: cd-pipeline-run-wvfzx
Waiting for logs to be available...
[clone : checkout] Cloning into 'wtecc-CICD_PracticeCode'...
[lint : echo-message] Calling Flake8 linter...
[tests : echo-message] Running unit tests with PyUnit...
[build : echo-message] Building image for https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git ...
[deploy : echo-message] Deploying main branch of https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git ...
```

结论

恭喜你！你现在能够创建一个 Tekton 管道并向管道传递参数。

在这个实验中，你学习了如何创建一个基础管道，指定并传递参数给任务和管道。你学习了如何修改管道以引用任务并配置其参数。你还学习了如何向管道传递额外参数以及如何运行它以回显和克隆 Git 仓库。

下一步

在下一个实验中，你将学习和使用 GitHub 触发器。

Author(s)

Tapas Mandal
John J. Rofrano