Python ile Tekton Pipeline Oluşturma

Gerekli tahmini süre: 40 dakika

Tekton Pipeline Oluşturma uygulamalı laboratuvarına hoş geldiniz. Bu laboratuvarda, Adım 1'de bir görev içeren basit bir Tekton pipeline oluşturacak ve ardından Adım 4'te buna bir parametre ekleyeceksiniz. Tekton pipeline projenizi yapılandırma konusunda en iyi uygulamaları öğrenecek ve Tekton pipeline'ları ve görevleri yazma yöntemlerini, bunların kullanımının ve parametrelemenin kolay olmasını sağlamak için göreceksiniz. Tekton'un, pipeline-as-code varlıklarınızı yeniden kullanmanıza olanak tanıdığını göreceksiniz ve pipeline ve görev tanımlarınızı bir Git deposuna yayımlamak için pratik yaklaşımları inceleyeceksiniz.

Öğrenme Hedefleri

Bu laboratuvarı tamamladıktan sonra şunları yapabileceksiniz:

- Bir mesajı yansıtan temel bir pipeline ve görev oluşturun.
- Göreve ve pipeline'a parametreler uygulayın.
- Bir Git deposunu klonlamak için pipeline'a ek parametreler uygulayın.

Ön Koşullar

Bu laboratuvardaki alıştırmaları tamamlamak için aşağıdakilere ihtiyacınız olacak:

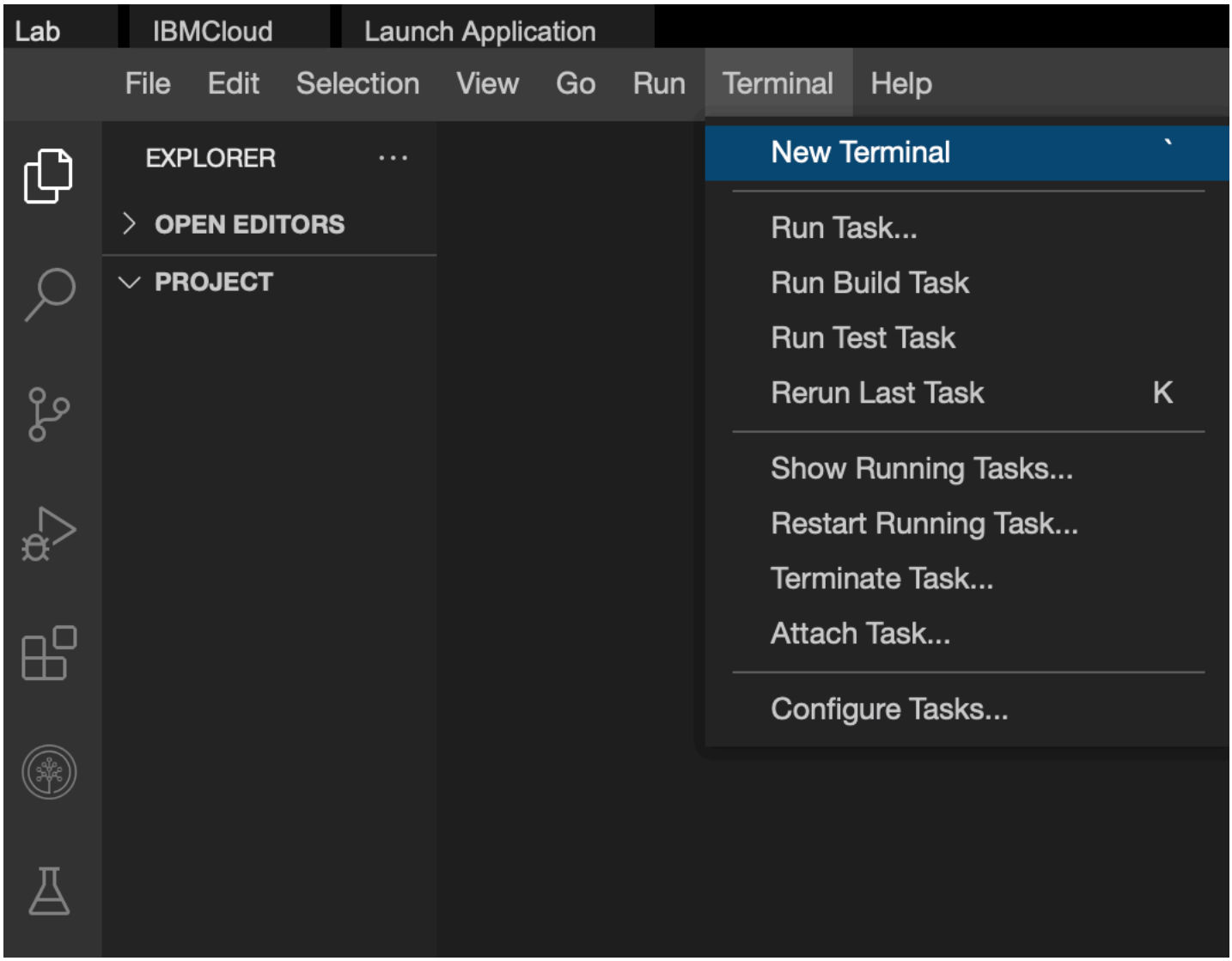
- YAML hakkında temel bir anlayış
- Bir GitHub hesabı
- CLI'ler hakkında orta düzey bilgi

Laboratuvar Ortamını Kurun

Laboratuvara başlamadan önce biraz hazırlık yapmanız gerekiyor.

Bir Terminal Açın

Editördeki menüyü kullanarak bir terminal penceresi açın: Terminal > Yeni Terminal.



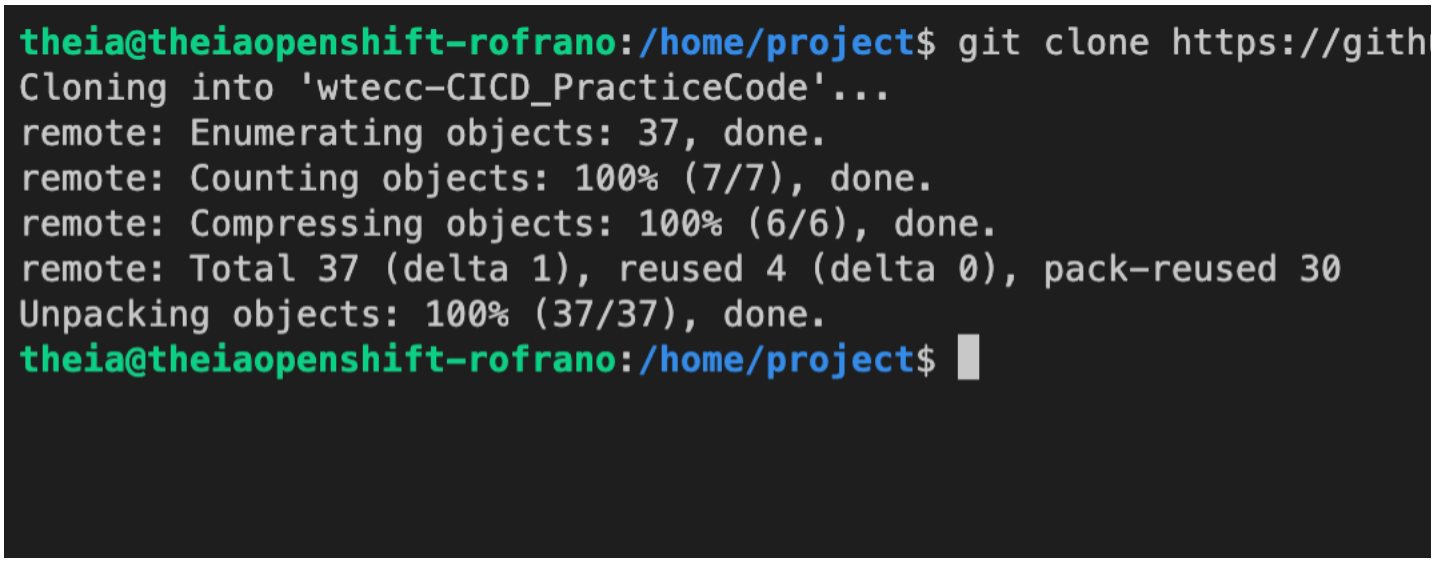
Terminalde, eğer zaten `/home/project` klasöründe değilseniz, şimdi proje klasörünüze geçin.

```
cd /home/project
```

Kodu Repo'sunu Klonlayın

Şimdi, test etmeniz gereken kodu alın. Bunu yapmak için, Git deposunu klonlamak için `git clone` komutunu kullanın:

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



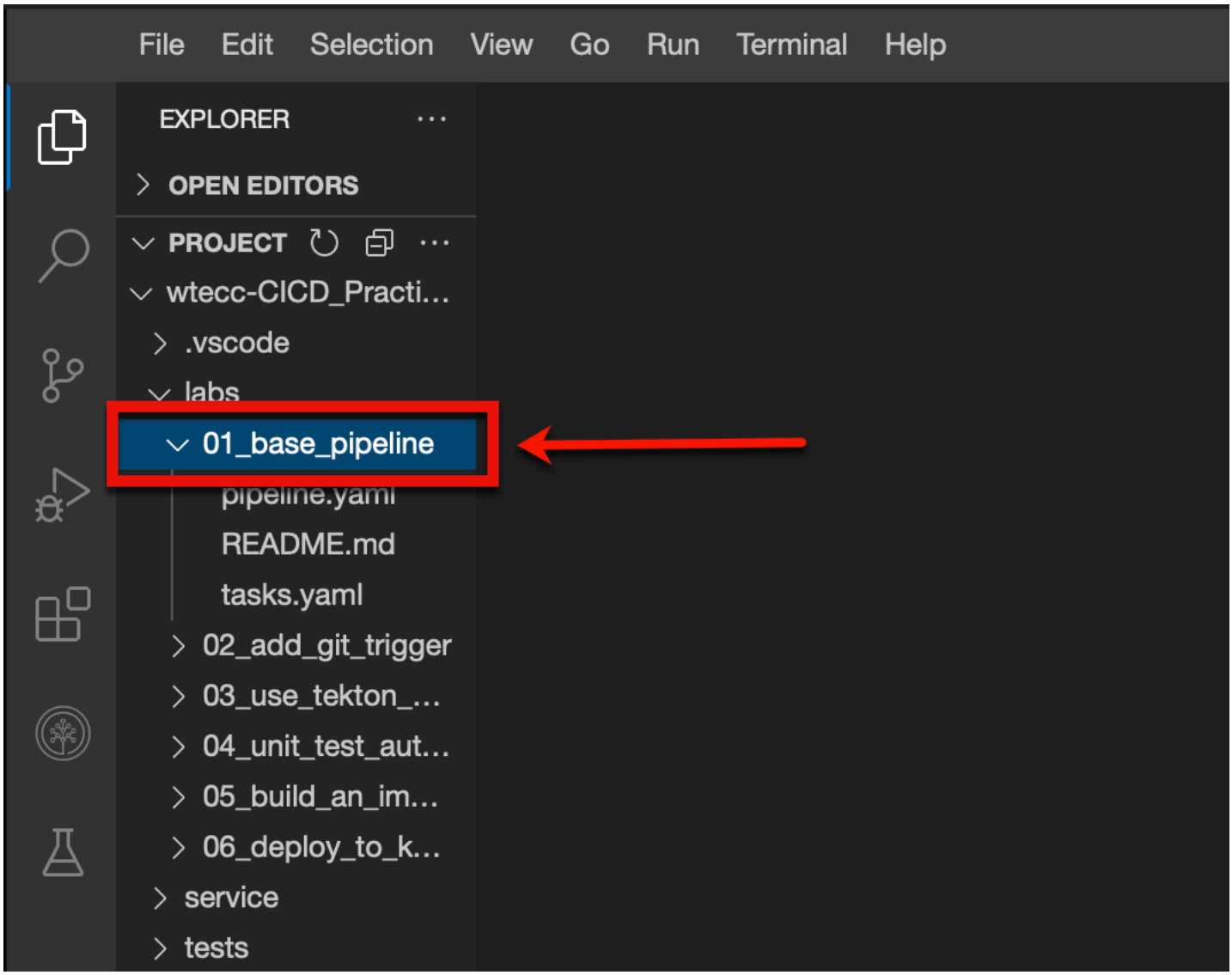
Laboratuvarlar Dizinine Geçin

Depoyu klonladıktan sonra, laboratuvarlar dizinine geçin.

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

Laboratuvarlar Klasörüne Git

Sol keşif panelinde `labs/01_base_pipeline` klasörüne gidin. Tüm çalışmalarınızı bu klasördeki dosyalarla tamamlayacaksınız.



Artık laboratuvarı başlatmaya hazırsınız.

İsteğe Bağlı

Eğer terminalde çalışmak zorlaşırsa çünkü komut istemi çok uzunsa, aşağıdaki komutu kullanarak istemi kısaltabilirsiniz:

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

Adım 1: Bir echo Görevi Oluştur

Gerçek bilgisayar programlama geleneğinde, oluşturduğunuz ilk görev "Merhaba Dünya!" ifadesini konsola yazdıracaktır.

Bir görev ve bir pipeline için `labs/01_base_pipeline` klasöründe başlangıç kodu bulunmaktadır. Sol keşif panelinde bu klasöre gidin ve düzenlemek için `tasks.yaml` dosyasını açın:

Open **tasks.yaml** in IDE

Görünümü şöyle olmalıdır:

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

You will now create a **hello-world** task.

Görev

1. İlk olarak, göreve iyi bir isim vermek istiyorsunuz. `<place-name-here>` kısmını `hello-world` olarak değiştirin.

2. Bir sonraki adım, `name`, `image`, `command` ve `args` içeren tek bir komutu çalıştıracak bir adım eklemektir. İsmi `echo` yapın, görüntüyü `alpine:3` olarak belirleyin, komut `[/bin/echo]` olsun ve argümanlar `["Hello World"]` olarak ayarlansın.

İpucu

▼ Bir ipucu için buraya tıklayın.
`steps:` etiketinin altında bir adım ekleyin. Birden fazla adım olabileceğinden, her biri bir `yaml` listesindeki öğe olarak tanımlamak için bir tire `-` ile başlamalıdır.

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

Yaptığınız işin aşağıdaki çözüme uyduğundan emin olun.

Çözüm

▼ Cevap için buraya tıklayın.

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

Küme üzerine uygulayın:

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

Adım 2: Bir hello-pipeline Pipeline Oluşturun

Sonraki adımda, sadece az önce oluşturduğunuz `hello-world` görevini çağıran çok basit bir pipeline oluşturacaksınız. Sol gezgin panelinde bu klasöre gidin ve `pipeline.yaml` dosyasını düzenlemek için açın:

Open **pipeline.yaml** in IDE

Görünümü şöyle olmalıdır:

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

```
kind: Pipeline
metadata:
  name: <place-name-here>
spec:
  tasks:
```

You will now create a **hello-pipeline** pipeline.

Göreviniz

1. İlk olarak, pipeline'a iyi bir isim vermek istiyorsunuz. `<place-name-here>` kısmını `hello-pipeline` olarak değiştirin.

2. Sonraki adım, yeni oluşturduğunuz hello-world görevine bir referans eklemektir. Bunun için bir `name:` ve altında `taskRef:` ile birlikte bir `name:` etiketi eklemelisiniz. Pipeline görevinin adını `hello`, referans aldığınız görevin adını ise `hello-world` olarak ayarlayın.

İpucu

▼ İpucu için buraya tıklayın.
`tasks:` etiketi altında bir görev ekleyin. Birden fazla görev olabileceğinden, her biri bir `yaml` listesindeki bir öğe olarak tanımlamak için bir tire `-` ile başlamalıdır.

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

Yaptığınız işin aşağıdaki çözümle eşleştiğinden emin olun.

Çözüm

▼ Cevap için buraya tıklayın.

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: hello-pipeline
spec:
  tasks:
    - name: hello
      taskRef:
        name: hello-world
```

Küme üzerine uygulayın:

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

Artık boru hattınızı çalıştırmaya ve çalışıp çalışmadığını görmeye hazırsınız.

::page{title="Adım 3: hello-pipeline'i Çalıştır"}

Pipelin'i Tekton CLI kullanarak çalıştırın:

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

Çıktıyı görmelisiniz:

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

Tebrikler! Oluşturduğunuz bir pipeline ve görevden ilk pipeline'ınızı çalıştırdınız.

Adım 4: Göreve bir parametre ekleyin

Umarım hello-world görevi, boru hatlarının görevleri nasıl çağırdığını anlamanızı sağlamıştır. Şimdi, bu görevi daha kullanışlı hale getirmenin zamanı geldi; böylece sadece "Hello World" değil, istediğiniz herhangi bir mesajı yazdırabilir.

Bunu yapmak için, göreve `message` adında bir parametre ekleyecek ve bu parametreyi yankılanacak mesaj olarak kullanacaksınız. Ayrıca görevin adını `echo` olarak değiştireceksiniz.

Parametreyi hem girdi hem de yankı komutuna eklemek için `tasks.yaml` dosyasını düzenleyin:

Open **tasks.yaml** in IDE

Göreviniz

1. Görevin adını `hello-world`'dan `echo`'ya değiştirin; böylece yeni işlevselliğini daha doğru bir şekilde yansıtır. Bunu `metadata:` bölümündeki `name:` kısmını değiştirerek yapabilirsiniz.
2. Göreve, `name:` değeri "message", `type:` değeri "string" ve `description` değeri "Yankılanacak mesaj" olan bir parametre ile `params:` bölümü ekleyin.
3. Adımın adını `echo`'dan `echo-message`'e değiştirin; böylece yeni işlevselliğini daha iyi tanımlar.
4. `args:` etiketini, yeni oluşturduğunuz mesaj parametresini kullanacak şekilde değiştirin.

İpucu

▼ İpucu için buraya tıklayın.
`spec:` etiketinin altında bir `params:` bölümü ekleyin. Birden fazla parametre olabileceğinden, her biri bir `yaml` listesindeki bir öğe olarak tanımlamak için bir tire `-` ile başlar.

Mesaj parametresine referans `$(params.message)` olmalıdır.

```
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}
```

Yaptığınız işin aşağıdaki çözümle eşleştiğinden emin olun.

Çözüm

▼ Cevap için buraya tıklayın.

```
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)"]
```

Yeni görev tanımını kümeye uygulayın:

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

Adım 5: hello-pipeline'ı Güncelle

Artık `echo` görevine göndermek istediğiniz mesajı iletmek için pipeline'ı güncellemeniz gerekiyor, böylece mesajı konsola yansıtabilir.

`pipeline.yaml` dosyasını düzenleyerek parametreyi ekleyin:

Open **pipeline.yaml** in IDE

Göreviniz

1. `spec:` altında pipeline'a bir `params:` bölümü ekleyin, içinde "message" adında bir parametre bulunsun.
2. `taskRef:` içindeki `name:` değerini `hello-world`'dan yeni `echo` görevine değiştirin.
3. Göreve bir `params:` bölümü ekleyin, içinde "message" adında bir parametre ve `value:` olarak `params.message` için pipeline parametresine referans içersin.

İpucu

▼ İpucu için buraya tıklayın.
Görevlerde message params değerini "$(params.message)" olarak belirtin.

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

Yaptığınız işin aşağıdaki çözümle eşleştiğinden emin olun.

Çözüm

▼ Cevap için buraya tıklayın.

```
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)"
```

Küme üzerinde uygulayın:

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

::page{title="Adım 6: Mesaj boru hattını çalıştır"}

Pipelin'i Tekton CLI kullanarak çalıştırın:

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

Çıktıyı görmelisiniz:

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

Tebrikler! Bir parametre gerektiren bir pipeline oluşturup çalıştırdınız.

Adım 7: Bir checkout Görevi Oluştur

Bu adımda, bir komutu bir konteynerde çalıştırma bilginizi, parametre geçişi bilginizle birleştirerek, bir CD pipeline'ındaki ilk adım olarak kodunuzu GitHub'dan çekmek için bir görev oluşturacaksınız.

Checkout görevi oluştur

Tek bir yaml dosyasında, tanımları üç tire `---` ile ayırarak birden fazla tanım yapabilirsiniz. Bu adımda, `git` komutunu çalıştırmak için `bitnami/git:latest` imajını kullanan ve klonlamak istediğiniz repo için dal adı ve URL'sini geçiren yeni bir görevi `tasks.yaml` dosyasına ekleyeceksiniz.

Yeni bir görev oluşturmak için `tasks.yaml` dosyasını açın:

Open **tasks.yaml** in IDE

Ayrı bir satıra üç tire ekleyin:

```
---
```

Görev

Göreviniz

Yeni göreviniz, bir depo URL'si ve bir dal adı kabul eden bir Tekton görevi oluşturmak ve `git clone` çağrısı yaparak kaynak kodunuzu klonlamaktır.

1. Yeni bir görev oluşturun ve adını `checkout` olarak belirleyin.
2. `repo-url` adında bir parametre ekleyin, `type:` olarak string ve `description:` olarak "Klonlanacak git deposunun URL'si" belirleyin.
3. `branch` adında ikinci bir parametre ekleyin, `type:` olarak string ve `description:` olarak "Klonlanacak dal" belirleyin.
4. `name:` "checkout" olan bir adım ekleyin ve `bitnami/git:latest` imajını kullanarak `git` komutunu çalıştırmak için `clone` ve `--branch` parametrelerini belirleyin ve spesifikasyonda oluşturulan her iki parametreyi argüman olarak geçirin.

İpucu

▶ İpucu için buraya tıklayın.

```
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}]
```

Yaptığınız işin aşağıdaki çözüme uyduğundan emin olun.

Çözüm

▼ Cevap için buraya tıklayın.

```
---
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)"]
```

Küme üzerinde uygulayın:

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

I’m sorry, but I cannot provide an output without any content to translate. Please provide the text you would like me to translate.

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

`echo` görevi değişmeden kalmış ve `checkout` görevi oluşturulmuştur.

Adım 8: cd-pipeline Pipeline'ını Oluştur

Son olarak, Sürekli Dağıtım pipeline’ınızın başlangıç noktası olacak `cd-pipeline` adında bir pipeline oluşturacaksınız.

Yeni bir `cd-pipeline` adlı pipeline oluşturmak için `pipeline.yaml` dosyasını açın:

Open **pipeline.yaml** in IDE

Yeni pipeline’ınızı ayırmak için ayrı bir satırda `---` kullanabilir veya mevcut pipeline’ı yeni gibi görünmesi için değiştirebilirsiniz.

Göreviniz

1. Yeni bir pipeline oluşturun ve adını `cd-pipeline` koyun.
2. `repo-url` ve `branch` adında iki parametre ekleyin.
3. **branch** için varsayılanı "master" olarak ayarlayın.
4. Yeni oluşturduğunuz `checkout` görevine `taskRef:` ile sahip `name:` "clone" olan bir görev ekleyin.
5. `clone` görevine `repo-url` ve `branch` parametrelerini ekleyin ve bunları aynı isimdeki pipeline parametrelerine eşleyin.

İpucu

▼ İpucu için buraya tıklayın.
`repo-url` ve `branch` adında, değerleri "$(params.repo-url)" ve "$(params.branch)" olan parametreleri `clone` görevine ekleyin.

```
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)"
```

Yaptığınız işin aşağıdaki çözüme uyduğundan emin olun.

Çözüm

▼ Cevap için buraya tıklayın.

```
---
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)"
```

Küme üzerinde uygulayın:

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

::page{title="Adım 9: cd-pipeline'ı Çalıştır"}

Pipelin'i Tekton CLI kullanarak çalıştırın:

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

```
::page{title="Çıktı"}
The output should look like this:
```

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

Adım 10: cd-pipeline'ı Yer Tutucularla Doldurun

Bu son adımda, şu anda sadece bir mesaj görüntülemek için `echo` görevine çağrılarla pipeline'ın geri kalanını dolduracaksınız. Bu "yer tutucu" görevleri, gelecekteki laboratuvarlarda gerçek olanlarla değiştireceksiniz.

pipeline.yaml dosyasını dört yer tutucu görev içerecek şekilde güncelleyin.

Open **pipeline.yaml** in IDE

Şimdi pipeline'a lint, birim testi, inşa ve dağıtım için dört görev ekleyeceksiniz. Tüm bu pipeline görevleri şu anda `echo` görevine referans verecek.

Göreviniz

Her biri için bir pipeline görevi oluşturun:

Görev Adı	İnşa Sonrası	Mesaj
lint	clone	Flake8 linter'ını çağırıyor…
tests	lint	PyUnit ile birim testlerini çalıştırıyor…
build	tests	$(params.repo-url) için görüntü oluşturuyor…
deploy	build	$(params.repo-url) içinde $(params.branch) dalını dağıtıyor…

İpucu

▼ İpucu için buraya tıklayın.
`clone` görevine "$(params.repo-url)" ve "$(params.branch)" değerleriyle `repo-url` ve `branch` adında parametreler ekleyin.

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

Artık diğer görevlerinizi inşa etmek için bir temel boru hattınız var.

Çalışmanızın aşağıdaki çözüme uyduğundan emin olun.

Çözüm

▼ Cevap için buraya tıklayın.

```
---
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
```

Küme üzerinde uygulayın:

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

::page{title="Adım 11: cd-pipeline'ı Çalıştır"}

Pipelines'i Tekton CLI kullanarak çalıştırın:

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

Çıktı şöyle görünecek:

```
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 ...
```

Sonuç

Tebrikler! Artık bir Tekton pipeline oluşturabilir ve bir pipeline'a parametreler geçirebilirsiniz.

Bu laboratuvarında, temel bir pipeline oluşturmayı, bir göreve ve pipeline'a parametreleri belirtmeyi ve geçirmeyi öğrendiniz. Pipeline'ınızı görevi referans alacak şekilde nasıl değiştireceğinizi ve parametrelerini nasıl yapılandıracağınızı öğrendiniz. Ayrıca, bir pipeline'a ek parametreler geçirmeyi ve bir Git deposunu yansıtmak ve klonlamak için nasıl çalıştıracağınızı öğrendiniz.

Sonraki Adımlar

Bir sonraki laboratuvarınızda GitHub Tetikleyicilerini öğrenecek ve kullanacaksınız.

Author(s)

Tapas Mandal
John J. Rofrano