Ana Uygulamayı Yayınlama

Tahmini süre: 90 dakika

Ana Uygulamayı Yayınlama laboratuvarına hoş geldiniz. Bu capstone'daki önceki modüllerden tüm kodun çalışır durumda olması gerekiyor. Laboratuvar, **Django** uygulamasını IBM Kubernetes Hizmetleri üzerinde yayınlamaya odaklanmaktadır.

Öğrenme Hedefleri:

Bu laboratuvarı tamamladıktan sonra şunları yapabileceksiniz:

1. Django uygulamalarını konteynerleştirmek için bir `Dockerfile` oluşturmak
2. `kubectl` CLI'sını kullanmak
3. Kubernetes dağıtım `yaml` dosyası ile çalışmak
4. IKS üzerinde bir Kubernetes `Deployment` oluşturmak

> **Not:** Lütfen laboratuvarı tek bir oturumda, ara vermeden tamamlayın çünkü laboratuvar çevrimdışı moda geçebilir ve hatalara neden olabilir. Laboratuvar sürecinde herhangi bir sorun/hata ile karşılaşırsanız, lütfen laboratuvar ortamından çıkış yapın. Ardından sistem önbelleğinizi ve çerezlerinizi temizleyin ve laboratuvarı tamamlamayı deneyin.

Ortamı ve komut satırı araçlarını doğrulayın

1. Eğer bir terminal açık değilse, editördeki menüyü kullanarak bir terminal penceresi açın: `Terminal > New Terminal`.

   > **Not:** Terminal zaten görünüyorsa bu adımı atlayın.

2. `kubectl` CLI'nin yüklü olduğunu doğrulayın.

   ```
   kubectl version
   ```

3. Proje klasörünüze geçin.

   ```
   cd /home/project
   ```

Alıştırma 1: Django - Github Deposu Klonlama ve Ortamı Kurma

Daha önceki laboratuvarlarda sağlanan şablondan Django uygulaması için yeni bir depo oluşturdunuz. Eğer oluşturmadıysanız, `Django ile Müzik Grubu Web Sitesi Oluştur` laboratuvarına geri dönün ve bu laboratuvarı tamamladığınızdan emin olun.

1. Eğer açık değilse, `Terminal` > `Yeni Terminal` ile bir terminal açın.
2. Ardından, GitHub hesabınızın adını içeren bir ortam değişkeni dışa aktarmak için export GITHUB_ACCOUNT komutunu kullanın.

   > **Not:** Aşağıdaki {your_github_account} yer tutucusunu kendi gerçek GitHub hesabınızla değiştirin:

   ```
   export GITHUB_ACCOUNT={your_github_account}
   ```

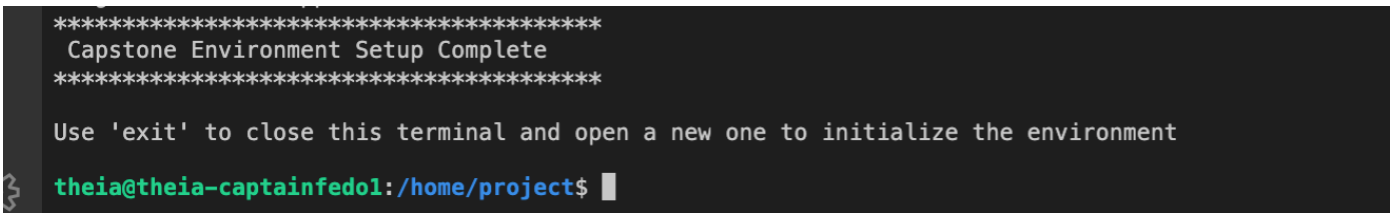
3. Sonra, deponuzu klonlamak için aşağıdaki komutları kullanın.

   ```
   git clone https://github.com/$GITHUB_ACCOUNT/Back-end-Development-Capstone.git
   ```

4. Back-end-Development-Capstone dizinine geçin ve `./bin/setup.sh` komutunu çalıştırın.

   ```
   cd Back-end-Development-Capstone
   bash ./bin/setup.sh
   ```

5. Kurulum tamamlandıktan sonra aşağıdakileri görmelisiniz:



6. Son olarak, mevcut terminali kapatmak için `exit` komutunu kullanın. Ortam, bir sonraki adımda yeni bir terminal açana kadar tam olarak aktif olmayacaktır.

Alıştırma 2: Django - Hizmetleri Bağlayın ve Yerel Olarak Çalıştırın

Bu kısmı tamamlamak için önceki laboratuvardan `pictures` ve `songs` mikro hizmetlerinin çalışıyor olması gerekir. Eğer bunlar çalışmıyorsa, lütfen önceki laboratuvarları takip ettiğinizden ve buraya doldurmak için hizmet URL'lerini not aldığınızdan emin olun. Ayrıca, buraya devam etmeden önce önceki laboratuvardaki tüm curl işlemlerini gerçekleştirebildiğinizden emin olun.

Sunucu dizinine geçin.

```
cd Back-end-Development-Capstone
```

Open `views.py` in the editor.

Open **views.py** in IDE

1. `def songs(request):` metodu şu anda bir dizi şarkı döndürüyor. Kodu aşağıdaki gibi değiştirin ve `SONG_URL`'yi önceki laboratuvarlarda dağıtılan şarkılar mikroservisi ile değiştirin.

```
def songs(request):
    songs = req.get("SONGS_URL/song").json()
    return render(request, "songs.html", {"songs": songs["songs"]})
```

2. `def photos(request):` metodu şu anda bir dizi şarkı döndürüyor. Kodu aşağıdaki gibi değiştirin ve `PHOTO_URL`'yi önceki laboratuvarlarda dağıtılan resimler mikroservisi ile değiştirin.

```
def photos(request):
    photos = req.get("PHOTO_URL/picture").json()
    return render(request, "photos.html", {"photos": photos})
```

3. Gerekli tabloları oluşturmak için göç işlemlerini gerçekleştirin.

```
python3 manage.py makemigrations
```

4. Uygulama için modelleri etkinleştirmek üzere göç işlemini çalıştırın.

```
python3 manage.py migrate
```

Göç komutunu çalıştırdıktan sonra terminalin ekran görüntüsünü alın ve `django-songs-photos-migrate.jpg` (veya .png) olarak kaydedin.

5. Yerel geliştirme sunucusunu başlatın.

```
python3 manage.py runserver
```

Eğer uygulama yerel olarak çalışmıyorsa, modül 3'e geri dönmeli ve son laboratuvarı tamamladığınızdan emin olmalısınız. Uygulama, IBM Code Engine ve RedHat OpenShift üzerinde çalışan **şarkılar** ve **resimler** servislerine bağlanmalıdır.

> **Not:** Uygulamayı yerel olarak başlattıktan ve /song yoluna gittiğinizde, şarkı sözleri görüntülenmiyorsa, lütfen Docker dağıtımına devam edin ve şarkı sözlerinin doğru bir şekilde render edilip edilmediğini kontrol edin.

Egzersiz 3: Django - Dockerfile'ı Tamamla

Görevleriniz

Diğer hizmetlerin aksine, ana Django uygulamasını IBM Kubernetes Servisi'nde `source-to-image` yöntemiyle dağıtma seçeneği yoktur. Görüntüyü oluşturmak için bir Dockerfile yazmalısınız. Bu egzersizdeki göreviniz, eksik Dockerfile'ı doldurmaktır.

Editörde `Back-end-Development-Capstone/Dockerfile` dosyasını açın.

Open **Dockerfile** in IDE

1. Bir python tabanlı görüntü kullanmak için kod ekleyin.
   Unutmayın, görüntünüz için bir temel görüntü belirtmenin yolu Dockerfile'da `FROM` ifadesidir. Temel görüntü olarak `FROM python:3.9.16-slim` belirtmek için `FROM` ifadesini kullanın.

   ▼ İpucu için buraya tıklayın.

   ```
   FROM python:3.8.2
   ```

2. Çalışma dizinini değiştirmek için kod ekleyin.
   Dockerfile'da `WORKDIR` ifadesinin, sonraki dosyalar için çalışma dizinini ayarlamak için kullanıldığını hatırlayın.

   ▼ İpucu için buraya tıklayın.

   ```
   WORKDIR $APP
   ```

3. requirements.txt dosyasını $APP'ye kopyalamak için kod ekleyin.
   Django projesinin çalışması için gereken tüm Python bağımlılıklarını listeleyen `requirements.txt` metin dosyasını kopyalamanız gerektiğini hatırlayın.

   ▼ İpucu için buraya tıklayın.

   ```
   COPY requirements.txt $APP
   ```

4. Sonraki adım, `RUN` komut sözdizimini kullanarak `requirements.txt` dosyasındaki tüm bağımlılıkları `pip` aracı ile yüklemek için konteynere talimat vermektir.

   ▼ İpucu için buraya tıklayın.

   ```
   RUN pip3 install -r requirements.txt
   ```

5. Gereksinimler yüklendikten sonra, `COPY` komutunu tekrar kullanarak kaynak kodun geri kalanını görüntüye kopyalayabilirsiniz.

   ▼ İpucu için buraya tıklayın.

   ```
   COPY . $APP
   ```

6. Bir sonraki adım, `EXPOSE` komutunu kullanarak 8000 numaralı portu açmaktır. Bu komut, konteynerin çalışma zamanında belirtilen ağ portlarında dinleyeceğini Docker'a bildirir. Gerçekten portu ana makineye yayınlamaz, daha ziyade kullanıcıların konteynerle etkileşim kurmasını anlamalarına yardımcı olmak için meta veriler sağlar.

   ▼ İpucu için buraya tıklayın.

   ```
   EXPOSE 8000
   ```

7. Harika! Neredeyse bitirdiniz. Dockerfile'daki son satır, Docker'a konteyneri nasıl çalıştıracağını söylemelidir. Konteyner başladığında `python manage.py runserver 0.0.0.0:8000` komutunu çalıştırmak istiyoruz. Sonundaki `0.0.0.0:8000`'ye dikkat edin. 0.0.0.0:8000 argümanı olmadan, sunucu yalnızca döngü arayüzünde (127.0.0.1) dinleyecek ve konteynerin dışından erişilemeyecektir.

▼ İpucu için buraya tıklayın.

```
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Nihai Çözüm

Kodunuzun bu şekilde göründüğünden emin olun: