İlk Django Uygulamanızı Oluşturun ve Docker ile Dağıtın

Gerekli tahmini süre: 20 dakika

Bu laboratuvar çalışmasında, ilk Django projenizi, Django uygulamanızı, Django görünümünüzü ve uygulamanızın bir Docker imajını oluşturacaksınız.

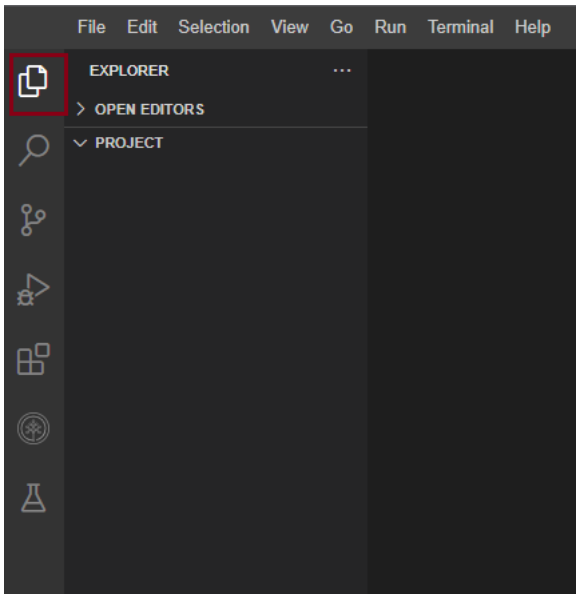
Öğrenme Hedefleri

- Komut satırı araçlarını kullanarak ilk Django projenizi ve uygulamanızı oluşturun
- Basit bir HTML sayfası döndüren ilk Django görünümünüzü oluşturun
- Uygulamanızın bir Docker konteyner imajını oluşturun

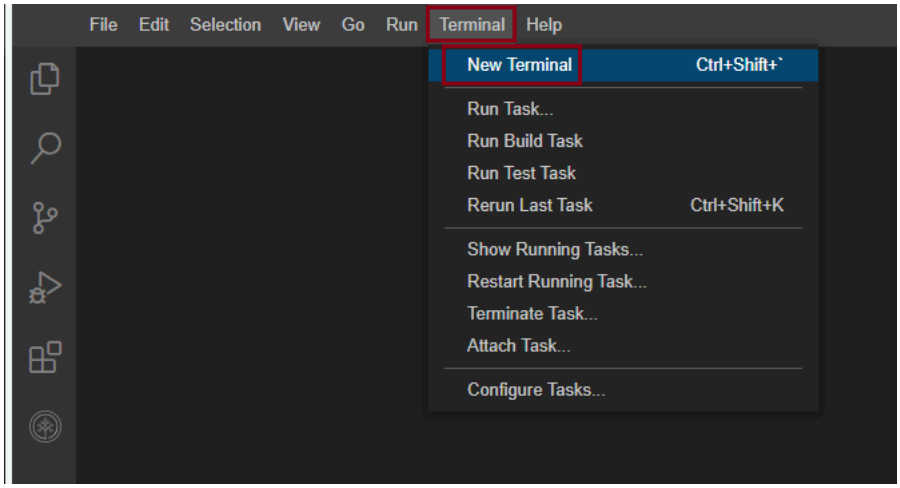
Cloud IDE'de dosyalarla çalışma

Cloud IDE'ye yeniyseniz, bu bölüm projelerinizin bir parçası olan dosyaları nasıl oluşturup düzenleyeceğinizi gösterecektir.

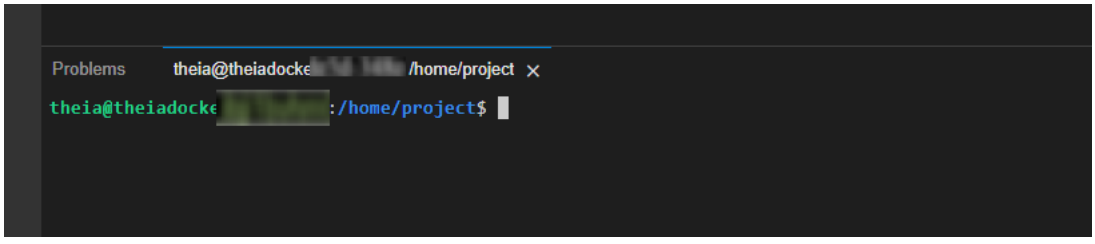
Cloud IDE içinde dosyalarınızı ve dizinlerinizi görüntülemek için bu dosya simgesine tıklayın.



`Terminal` menüsüne tıklayın, ardından `Yeni Terminal` seçeneğine tıklayın.



Bu, komutlarınızı çalıştırabileceğiniz yeni bir terminal açacaktır.



Laboratuvarında Kapsanan Kavramlar

1. `Django project`: Uygulamanızın yapısını ve yapılandırma dosyalarını tanımlar.
2. `Django app`: Modeller, görünümler, şablonlar, URL'ler ve statik dosyalar gibi diğer kaynakları içeren bir Django uygulamasıdır.
3. `View`: Gelen taleplerle ne yapılacağını ve buna karşılık gelen yanıtın nasıl oluşturulacağını belirleyen bir işlevdir.
4. `URL or urls.py`: Uygulamanız için merkezi URL yapılandırması olarak hizmet eder, gelen URL desenlerini karşılık gelen görünüm işlevlerine veya sınıf tabanlı görünümlere eşler.
5. `Template`: Bir web tarayıcısında işlenip görüntülenecek çıktının yapısını ve sunumunu tanımlayan bir dosyadır.
6. `Docker`: Uygulamaların, konteyner adı verilen izole ortamlarda otomatik olarak dağıtımını ve yönetimini sağlamanıza olanak tanıyan açık kaynaklı bir platformdur.
7. `Containerization`: Farklı bilgisayar ortamlarında uygulamanın güvenilir bir şekilde çalışması için gerekli olan tüm bağımlılıkları ve yapılandırmaları içeren izole bir ortamdır.

İlk Django Projenizi Oluşturun

Laboratuvara başlamadan önce, mevcut dizininizin `/home/project` olduğundan emin olun.

- Bu gerekli paketleri yükleyin ve ortamı kurun.

```
pip install --upgrade distro-info
pip3 install --upgrade pip==23.2.1
pip install virtualenv
virtualenv djangoenv
source djangoenv/bin/activate
```

Öncelikle, Django ile ilgili paketleri yüklememiz gerekiyor.

- Terminali açın ve şunu çalıştırın:

```
pip install Django
```

- Kurulum tamamlandığında, aşağıdaki komutu çalıştırarak ilk Django projenizi oluşturabilirsiniz:

```
django-admin startproject firstproject
```

`firstproject` adında bir klasör oluşturulacak, bu klasör bir Django projesi için ayarları ve yapılandırmaları saracak bir kapsayıcıdır.

- Eğer mevcut çalışma dizininiz `/home/project/firstproject` değilse, proje klasörüne `cd` komutunu kullanarak geçin.

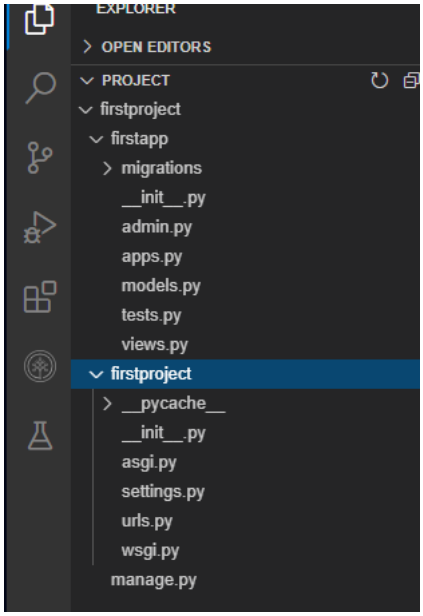
```
cd firstproject
```

- ve projede `firstapp` adında bir Django uygulaması oluşturun

  ```
  python3 manage.py startapp firstapp
  ```

Django, ilk `firstapp` uygulamanızı içeren bir proje iskeleti oluşturdu.

CloudIDE çalışma alanınız aşağıdaki gibi görünmelidir:



İskelet, bir Django projesi ve uygulaması için temel yapılandırma ve ayar dosyalarını içerir:

- Projeye ait dosyalar için:
  - `manage.py`, Django projesi ile etkileşimde bulunmak için kullanılan bir komut satırı arayüzüdür; sunucuyu başlatma, modelleri taşımak gibi görevleri yerine getirmek için kullanılır.
  - `firstproject/settings.py`, ayar ve yapılandırma bilgilerini içerir.
  - `firstproject/urls.py`, projeniz içindeki Django uygulamalarınızın URL ve yönlendirme tanımlarını içerir.
- Uygulamaya ait dosyalar için:
  - `firstapp/admin.py`: bir yönetici sitesi oluşturmak için kullanılır.
  - `firstapp/models.py`: model sınıflarını içerir.
  - `firstapp/views.py`: görünüm fonksiyonlarını/sınıflarını içerir.
  - `firstapp/urls.py`: uygulama için URL beyanları ve yönlendirmeleri içerir.
  - `firstapp/apps.py`: uygulama için yapılandırma meta verilerini içerir.
  - `firstapp/migrations/`: model göç scriptleri klasörü.
- Uygulamayı başlatmadan önce, gerekli veritabanı tablolarını oluşturmak için göç işlemlerini gerçekleştirmeniz gerekecek:

  ```
  python3 manage.py makemigrations
  ```

- ve göçü çalıştır

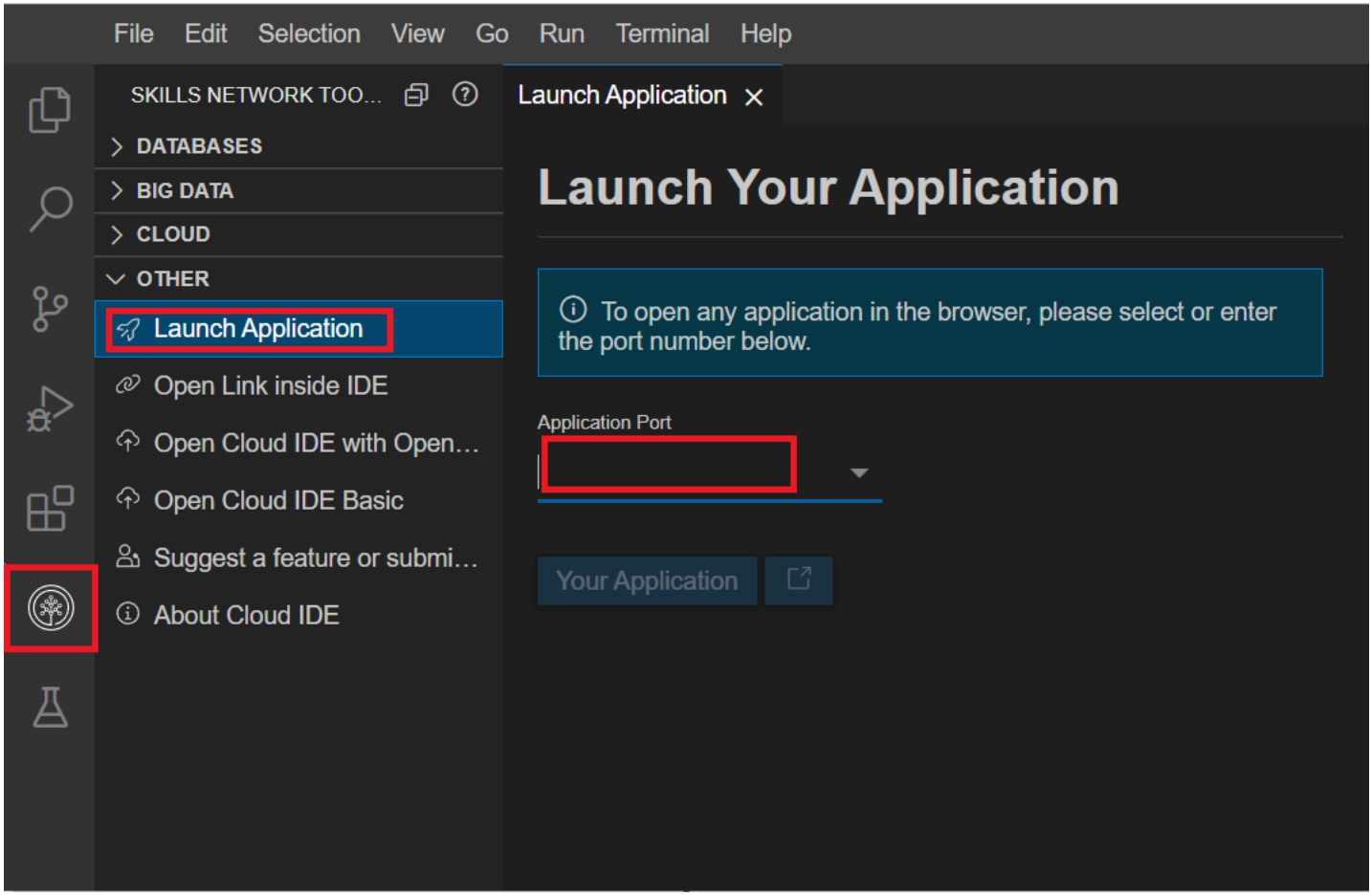
  ```
  python3 manage.py migrate
  ```

- Ardından `firstproject` içinde uygulamaları barındıran bir geliştirme sunucusu başlatın:

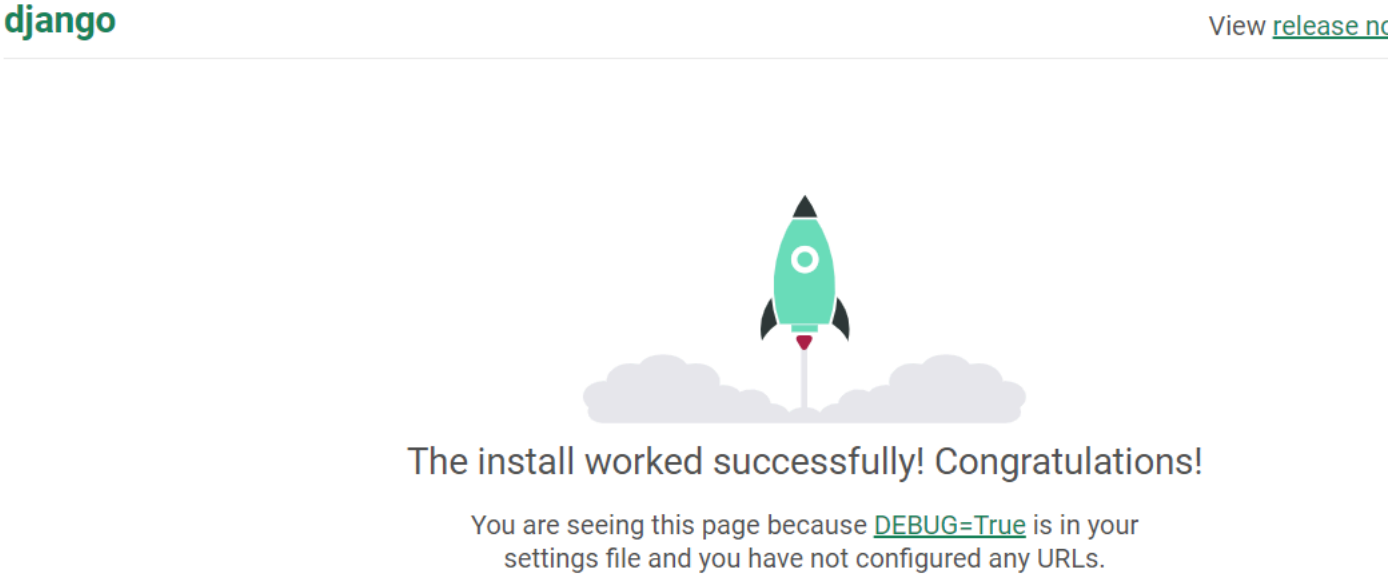
  ```
  python3 manage.py runserver
  ```

Theia'dan ilk Django uygulamanızı görmek için,

- Soldaki Skills Network butonuna tıklayın, bu **"Skills Network Toolbox"**'u açacaktır. Ardından **Diğer**'e tıklayın ve **Uygulamayı Başlat**'a tıklayın. Buradan portu `8000` girip başlatmalısınız.



ve aşağıdaki karşılama sayfasını görmelisiniz:



Django uygulamaları geliştirirken, çoğu durumda Django güncellenen dosyaları otomatik olarak yükleyecek ve geliştirme sunucusunu yeniden başlatacaktır. Ancak, projenizde dosya ekleyip/sildiğinizde sunucuyu manuel olarak yeniden başlatmak daha güvenli olabilir.

Şimdi Django sunucusunu durdurmayı deneyelim, terminalde basarak:

- `Control + C` veya `Ctrl + C` tuşlarına basın.

İlk Görünümünüzü Ekleyin

Sonra, ***firstproject*** içine ***firstapp***'inizi ekleyelim.

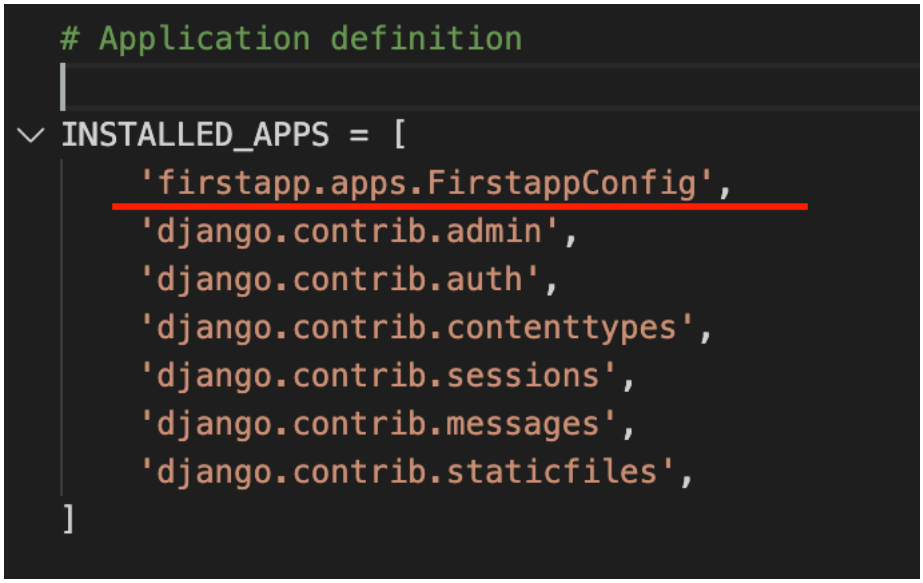
firstproject/settings.py dosyasını açın.

Open **settings.py** in IDE

`INSTALLED_APPS` bölümünü bulun ve yeni bir uygulama girişi ekleyin.

```
    'firstapp.apps.FirstappConfig',
```

`INSTALLED_APPS`'iniz aşağıdaki gibi görünmelidir



Ayrıca, Admin sitesini yönetmek için ***admin***, kimlik doğrulama için ***auth*** gibi bazı önceden yüklenmiş Django uygulamalarını da görebilirsiniz.

Sonraki adımda, ***firstapp***'in ***urls.py*** dosyasını ***firstproject***'e eklememiz gerekiyor, böylece ***firstapp***'in görünümleri doğru bir şekilde yönlendirilebilir.

- ***firstapp*** klasörü altında boş bir ***urls.py*** oluşturun.

```
cd firstapp # Make sure you are in firstapp directory
touch urls.py
```

- ***firstproject/urls.py*** dosyasını açın, zaten içe aktarılmış bir ***path*** fonksiyonu bulabilirsiniz: `from django.urls import path`.

Open **urls.py** in IDE

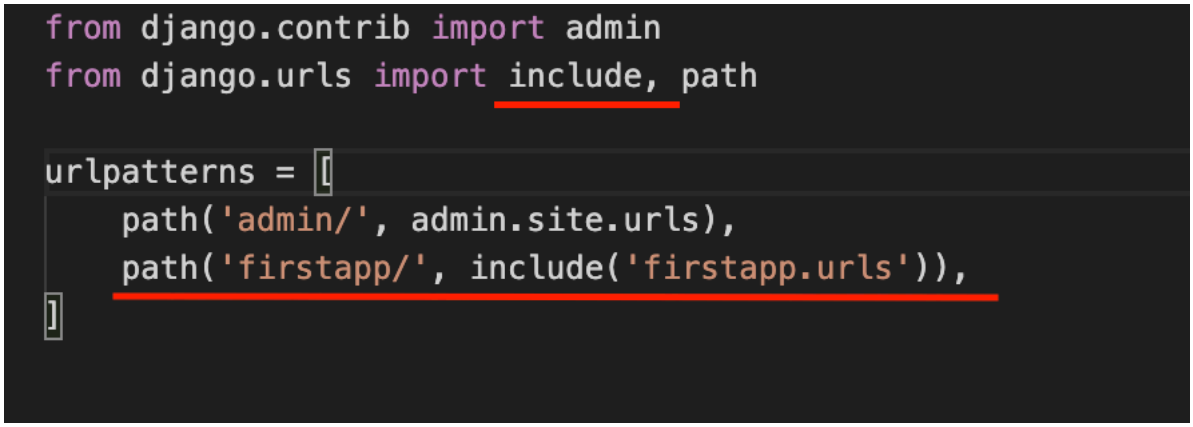
Şimdi ayrıca `django.urls` paketinden bir `include` metodunu içe aktarın:

```
from django.urls import include, path
```

- Ardından yeni bir `path` girişi ekleyin.

```
path('firstapp/', include('firstapp.urls')),
```

Sizin ***firstproject/urls.py*** dosyanız artık aşağıdaki gibi görünmelidir:



Artık HTTPRequest alacak ve içeriği basit bir HTML sayfası olarak saran bir HTTPResponse döndürecek ilk görünümünüzü oluşturabilirsiniz.

- ***firstapp/views.py*** dosyasını açın, `# Create your views here.` yorumundan sonra ilk görünümünüzü yazın.

Open **views.py** in IDE

```
from django.http import HttpResponse
def index(request):
    # Create a simple html page as a string
    template = "<html>" \
                "This is your first view" \
               "</html>"
    # Return the template as content argument in HTTP response
    return HttpResponse(content=template)
```

Sonra, indeks görünümü için URL'yi yapılandırın.

- ***firstapp/urls.py*** dosyasını açın, aşağıdaki kodu ekleyin:

Open **urls.py** in IDE

```
from django.urls import path
from . import views
urlpatterns = [
    # Create a path object defining the URL pattern to the index view
    path(route='', view=views.index, name='index'),
]
```

Bu kadar, şimdi ilk görünümünüzü test edelim.

- Django sunucusunu başlatmadıysanız çalıştırın:

```
cd ..
python3 manage.py runserver
```

Şimdi aşağıdaki *Uygulamayı Başlat* butonuna tıklayarak uygulamayı bir tarayıcıda açın.

Launch Application

Django, `/firstapp` ile başlayan herhangi bir HTTP isteğini `firstapp`'e yönlendirecek ve ***firstapp/urls.py*** dosyasında tanımlı yollar için eşleşmeleri arayacaktır.

Hepsi bu kadar, ilk görünümünüz tarafından döndürülen ***HTTPResponse***'u görmelisiniz; bu, içeriği `Bu sizin ilk görünümünüz` olan basit bir HTML sayfasıdır.

Kodlama Pratiği: Mevcut Tarihi Döndüren Bir Görünüm Ekle

- Bugünün tarihini döndüren bir görünüm oluşturmak için aşağıdaki kod parçasını tamamlayın ve ekleyin.

`firstapp/views.py` dosyasına bir `get_date()` görünüm fonksiyonu ekleyin (güncellenen dosyayı kaydetmeyi unutmayın):

```
from datetime import date
def get_date(request):
    today = date.today()
    template = "<html>" \
                "Today's date is {}" \
               "</html>".format(#<HINT> add today here#)
    return HttpResponse(content=#<HINT> use the template object as argument value#)
```

ve `get_date()` görünümü için `firstapp/urls.py` dosyasına bir `/date` URL yolu ekleyin:

```
urlpatterns = [
    # Create a path object defining the URL pattern to the index view
    path(route='', view=views.index, name='index'),
    # Add another path object defining the URL pattern using `/date` prefix
    path(route='#<HINT> add a date URL path here#', view=#add the get_date function name here #, name='date'),
]
```

▼ Çözümü görmek için buraya tıklayın

```
from datetime import date
def get_date(request):
    today = date.today()
    template = "<html>" \
               "Today's date is {}" \
              "</html>".format(today)
    return HttpResponse(content=template)
```

`firstapp/urls.py` dosyasını aç.

```
from django.urls import path
from . import views
urlpatterns = [
    # Create a path object defining the URL pattern to the index view
    path(route='', view=views.index, name='index'),
    # Add another path object defining the URL pattern using `/date` prefix
    path(route='date', view=views.get_date, name='date'),
]
```

Şimdi yukarıda oluşturduğunuz `/date` rotasını açmak için uygulamayı başlat butonuna tıklayın.

Launch Application

Uygulamayı Docker ile Konteynerleştirme

Docker, kodu yazdığınız makineye ve çalıştırmak istediğiniz makineye bakılmaksızın uygulamaları çalıştırmayı kolaylaştıran güçlü bir araçtır. Geliştiricilerin kodlarının çalışmadığında "Ama benim dizüstü bilgisayarımda çalışıyor!" sorununu aşmalarını sağladığı için pratikte yaygın olarak kullanılmaktadır.

Eğer yerel olarak çalışıyorsanız, önce Docker'ı bu bağlantıdan indirmeniz gerekecek.

Temel python uygulamaları için çalışma şekli şu şekildedir:

1. Bir `requirements.txt` dosyası oluşturun
2. Docker imajını nasıl oluşturacağınızı içeren talimatları barındıran bir `Dockerfile` oluşturun
3. Bir konteyner imajı oluşturmak ve çalıştırmak için `docker compose up` komutunu çalıştırın
4. İmajı bir uzak depoya gönderin ve başkalarının tam olarak sizin yapılandırdığınız gibi çalıştırabilmesi için itebilirsiniz!

Docker imajları genellikle konteynerleri yöneten bir hizmet olan Kubernetes ile birlikte kullanılır.

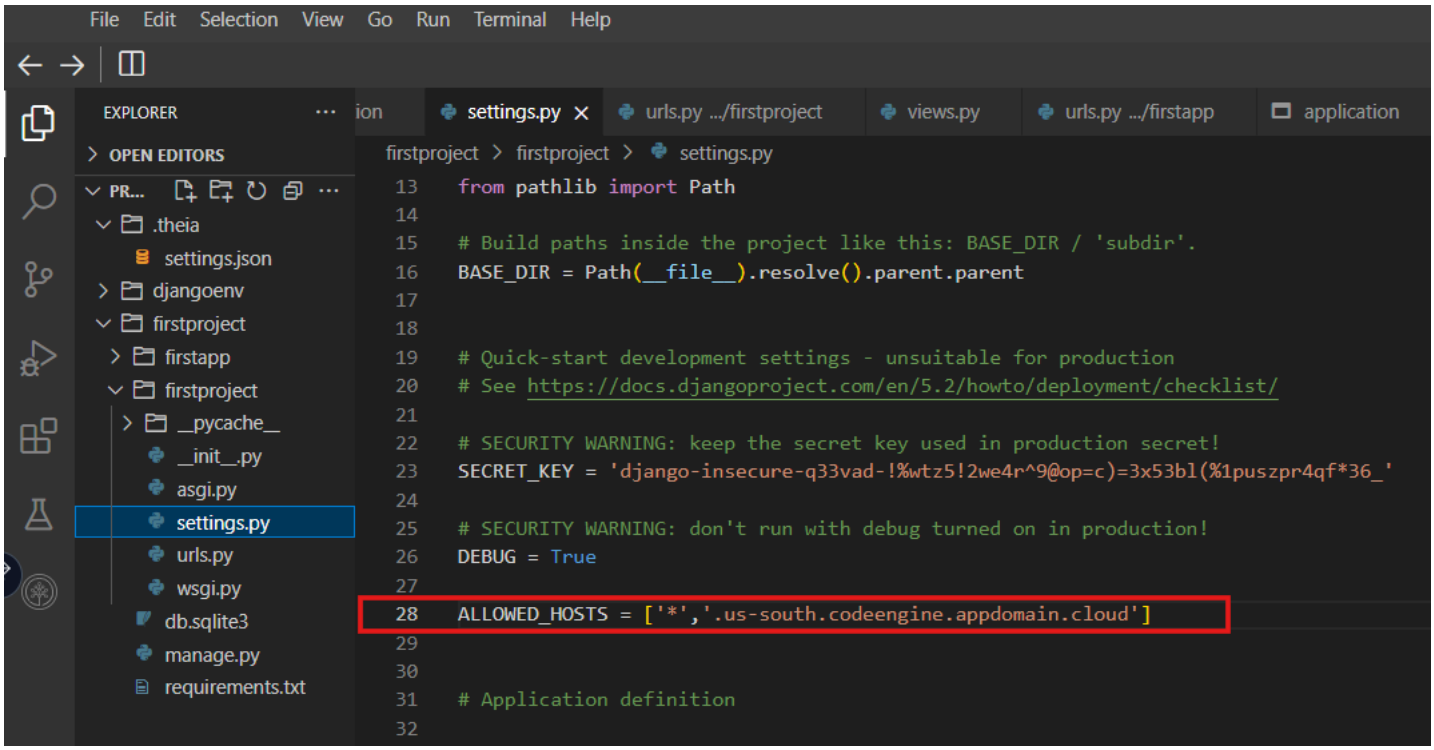
Bu Django uygulamasını Docker kullanarak çalıştırmak için nasıl ayarlayabileceğinize kısaca tanıtılacaksınız.

Docker ile ilgili teknik konulara girmeden önce, kod tabanımızda bir son değişiklik yapmamız gerekiyor. Varsayılan olarak, django, `settings.py` dosyasında açıkça tanımlanana kadar CloudIDE'den gelen trafik dahil tüm ana hostlardan gelen tüm trafiği engelleyecek şekilde yapılandırılmıştır.

Bu nedenle, setting.py dosyasını açın ve `ALLOWED_HOSTS = []` kodunu şu şekilde değiştirin:

```
ALLOWED_HOSTS = ['*','.us-south.codeengine.appdomain.cloud']
```

Open **settings.py** in IDE



Şimdi Docker ile çalışmaya başlayalım.

Öncelikle Docker'a hangi python paketlerini yüklemesi gerektiğini bildirmek için `requirements.txt` dosyasını oluşturmamız gerekiyor. Ana `/firstproject` klasöründe aşağıdaki komutu çalıştırın.

```
pip install pipreqs
pipreqs .
```

Sonraki adımda, Docker'a uygulamanızı nasıl oluşturacağını talimat veren bir `Dockerfile` oluşturmak istiyoruz (aynı dizinde):

Boş bir Dockerfile oluşturmak için aşağıdaki komutu çalıştırın:

```
touch Dockerfile
```

```
Then open the newly created Dockerfile and copy the following contents to it.
::openFile{path="/home/project/firstproject//Dockerfile"}
```

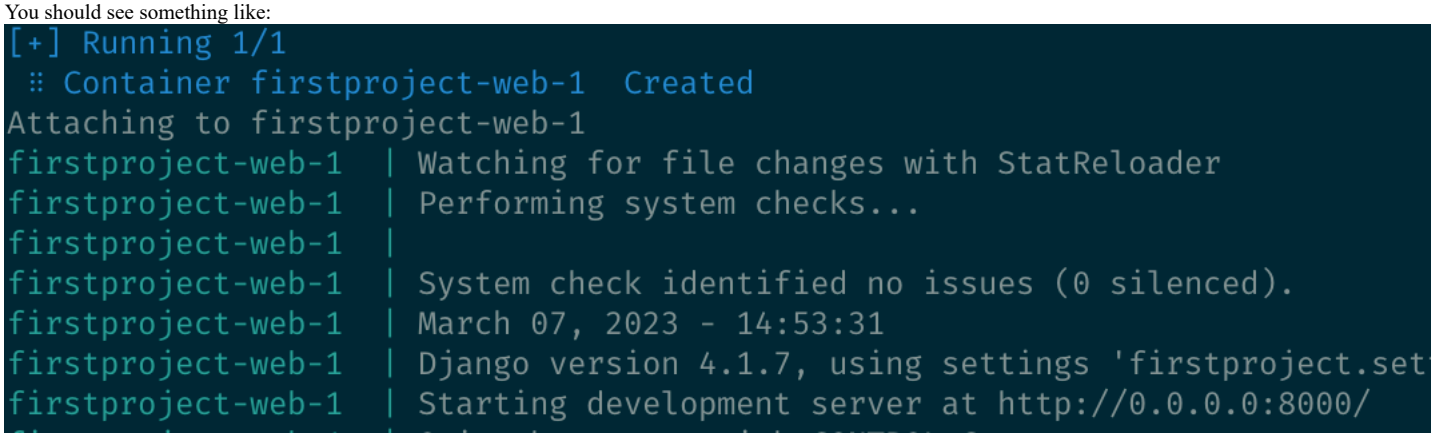
syntax=docker/dockerfile:1

FROM python:3
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
WORKDIR /code
COPY requirements.txt /code/

RUN pip install -r requirements.txt
COPY . /code/
CMD [“python”, “manage.py”, “runserver”, “0.0.0.0:8000”]

```
The code above will be run line by line. The `FROM` line indicates what base container image we want to build on, and in this case we want to us
Now we can run the following command to create and run the container image:
```shell
`docker build . -t my-django-app:latest && docker run -e PYTHONUNBUFFERED=1 -p  8000:8000 my-django-app`
```

You should see something like:



You can launch the application the same way you have previously in this project as well, through `Launch Application` and specifying port `8000`.

Alternatively, you can launch the application directly by clicking on this button.

Web Application

With this, you can easily share the Docker image by following these instructions.

Deploying to Code Engine

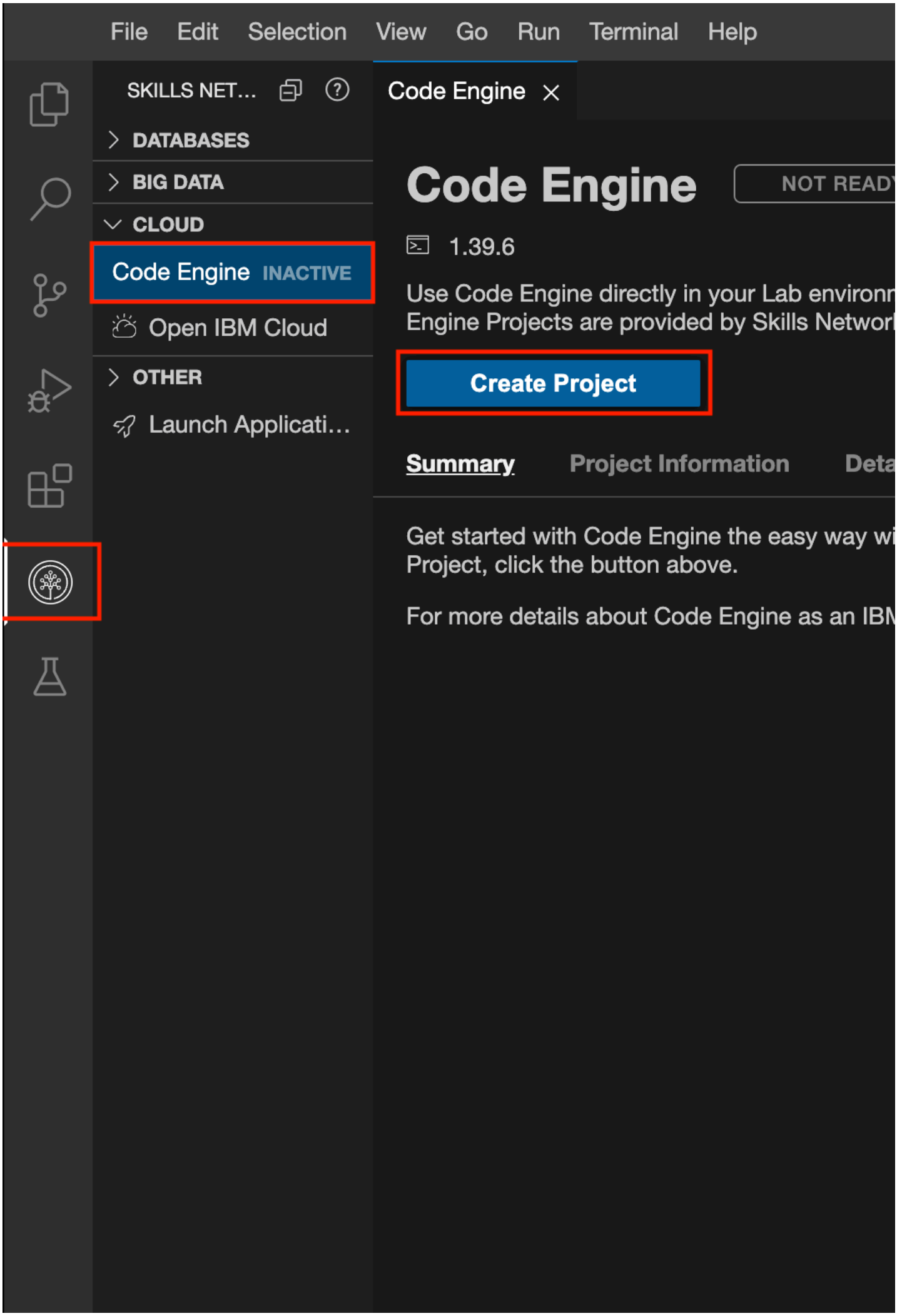
If you would like to host your application and have it be available for anyone to use, you can follow these steps in order to deploy it. The deployment will be to IBM Cloud’s Code Engine. IBM Cloud Code Engine is a fully managed, cloud-native service for running containerized workloads on IBM Cloud. It allows developers to deploy and run code in a secure, scalable and serverless environment, without having to worry about the underlying infrastructure.

The following steps in `Part 1` allow you to test deploy to a IBM Skills Network Code Engine environment to test if everything is working just fine, which is deleted after a few days. Furthermore, you can proceed with a permanent deployment to your own account.

Part 1: Deploying to Skills Network Code Engine

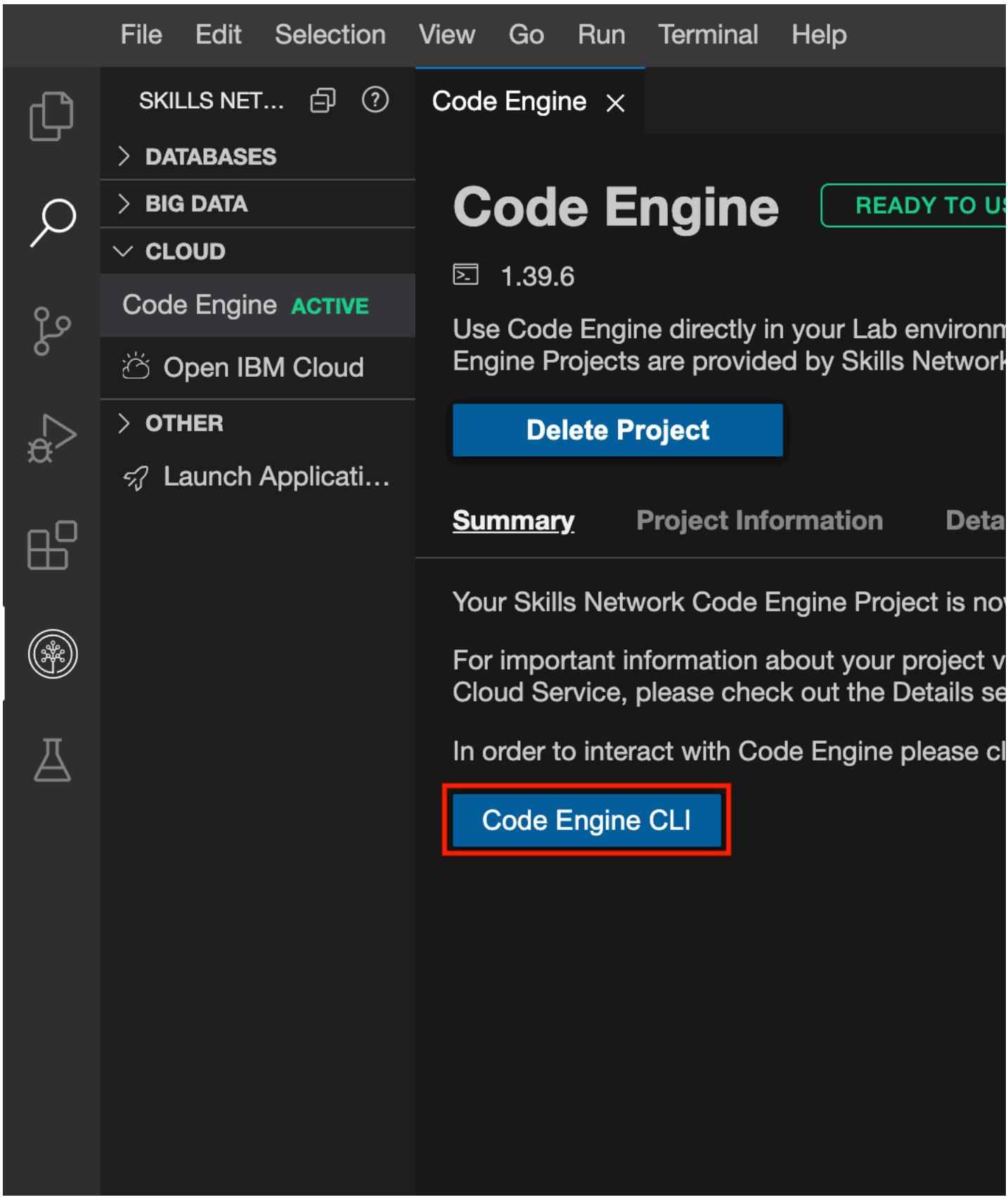
Step 1. Create Code Engine Project

In the left hand navigation pannel, there is an option for the Skills Network Toolbox. Simply open that and that expand the *CLOUD* section and then click on *Code Engine*. Finally cick on Create Project
```


File Edit Selection View Go Run Terminal Help
SKILLS NET...
DATABASES
BIG DATA
CLOUD
Code Engine INACTIVE
Open IBM Cloud
OTHER
Launch Applicati...
Code Engine
Code Engine
NOT READ
1.39.6
Use Code Engine directly in your Lab environ
Engine Projects are provided by Skills Networ
Create Project
Summary
Project Information
Deta
Get started with Code Engine the easy way wi
Project, click the button above.
For more details about Code Engine as an IBM

Step 2. Click on Code Engine CLI Button

From the same page simply click on Code Engine CLI button. This will open a new terminal and will login to a code engine project with everything alraedy set up for you.



Step 3. Deploy Your App

First, give a name to your Code Engine application, we will call it `my-django-app` as default

```
APP_NAME=my-django-app
```

Then tag and push the built image to IBM's icr registry:

```
REGISTRY=us.icr.io
docker tag ${APP_NAME}:latest ${REGISTRY}/${SN_ICR_NAMESPACE}/${APP_NAME}:latest
docker push ${REGISTRY}/${SN_ICR_NAMESPACE}/${APP_NAME}:latest
```

Finally, from the same terminal window run the following command to deploy your app to Code Engine.

```
ibmcloud ce application create --name ${APP_NAME} --image ${REGISTRY}/${SN_ICR_NAMESPACE}/${APP_NAME}:latest --registry-secret icr-secret --port
```

▶ Troubleshooting

Once the app is deployed, you should see a url outputted in the terminal.

Append `/firstapp` to the url and simply copy paste it to your preferred browser.

Enjoy your app deployed live on the web thanks to Code Engine.

You can check the status, logs and events of the application with the following commands.

```
ibmcloud ce app logs --application ${APP_NAME}
```

```
ibmcloud ce app events --application ${APP_NAME}
```

Summary

In this lab, you have created your first Django project, first Django app, and first Django view to return a simple HTML page. You also learned to create a Docker image of your app, which makes it simple to share and run it on any computer.

Author(s)

Yan Luo
Richard Ye
Talha Siddiqui