Create Your First Django App and Deploy using Docker

Estimated time needed: 20 minutes

In this lab, you will create your first Django project, Django app, Django view, and a Docker image of your app.

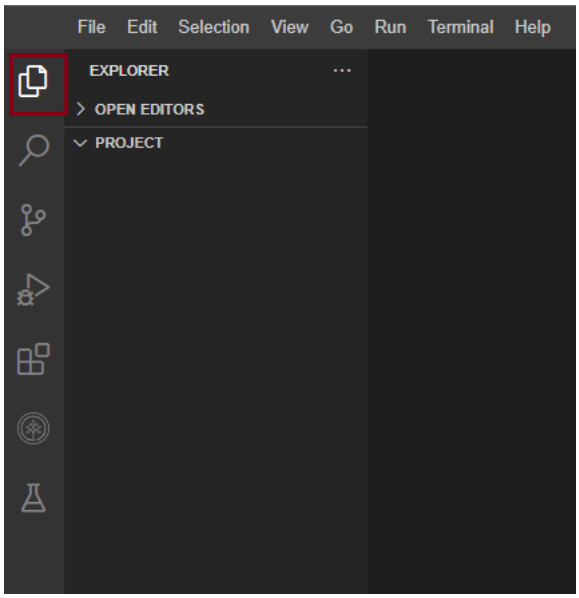
Learning Objectives

- Create your first Django project and app using command line utils
- Create your first Django view to return a simple HTML page
- Create a Docker container image of your application

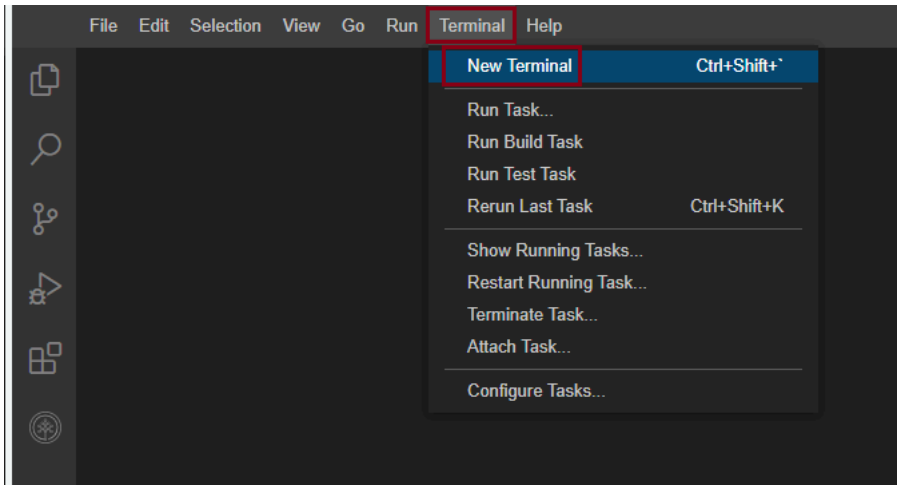
Working with files in Cloud IDE

If you are new to Cloud IDE, this section will show you how to create and edit files, which are part of your project, in Cloud IDE.

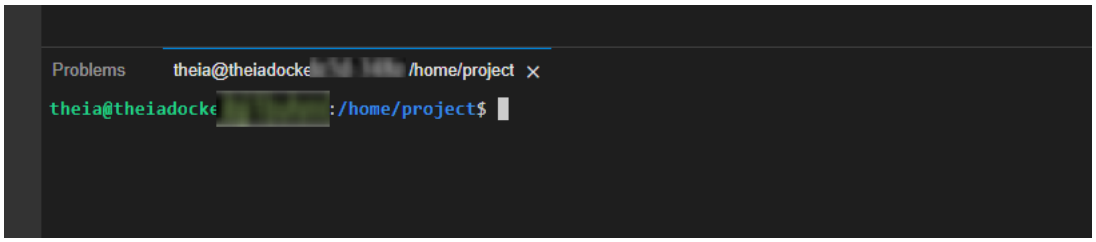
To view your files and directories inside Cloud IDE, click on this files icon to reveal it.



Click on the `Terminal` menu, and then `New Terminal`.



This will open a new terminal where you can run your commands.



Concepts covered in the lab

1. `Django project`: Defines the structure of your application and configuration files.
2. `Django app`: A Django application that includes models , views , templates, URLs, and other resources like static files.
3. `View`: A function that determines what to do with incoming requests, and how to generate the corresponding response.
4. `URL or urls.py`: Serves as the central URL configuration for your application, mapping incoming URL patterns to corresponding view functions or class-based views.
5. `Template`: A file that defines the structure and presentation of the output to be rendered and displayed in a web browser.
6. `Docker`: An open-source platform that allows you to automate the deployment and management of applications within isolated environments called containers.
7. `Containerization`: An isolated environment that contains all the necessary dependencies and configurations required for an application to run reliably across different computing environments.

Create Your First Django Project

Before starting the lab, make sure your current directory is `/home/project`.

- Install these must-have packages and setup the environment.

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
1. pip install --upgrade distro-info
2. pip3 install --upgrade pip==23.2.1
3. pip install virtualenv
4. virtualenv djangoenv
5. source djangoenv/bin/activate
```

Copied! Executed!

First, we need to install Django related packages.

- Go to terminal and run:

```
1. 1
```

```
1. pip install Django
```

Copied! Executed!

- Once the installation is finished, you can create your first Django project by running:

```
1. 1
```

```
1. django-admin startproject firstproject
```

Copied! Executed!

A folder called `firstproject` will be created which is a container wrapping up settings and configurations for a Django project.

- If your current working directory is not `/home/project/firstproject`, `cd` to the project folder

```
1. 1
```

```
1. cd firstproject
```

Copied! Executed!

- and create a Django app called `firstapp` within the project

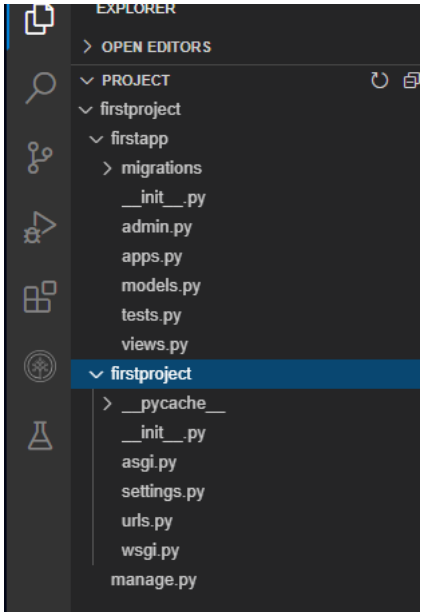
```
1. 1
```

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

Copied! Executed!

Django created a project scaffold for you containing your first `firstapp` app.

Your CloudIDE workspace should look like the following:



The scaffold contains the fundamental configuration and setting files for a Django project and app:

- For project-related files:
  - `manage.py` is a command-line interface used to interact with the Django project to perform tasks such as starting the server, migrating models, and so on.
  - `firstproject/settings.py` contains the settings and configurations information.
  - `firstproject/urls.py` contains the URL and route definitions of your Django apps within the project.
- For app-related files:
  - `firstapp/admin.py`: is for building an admin site
  - `firstapp/models.py`: contains model classes
  - `firstapp/views.py`: contains view functions/classes
  - `firstapp/urls.py`: contains URL declarations and routing for the app
  - `firstapp/apps.py`: contains configuration meta data for the app
  - `firstapp/migrations/`: model migration scripts folder
- Before starting the app, you will to perform migrations to create necessary database tables:

```
1. 1
1. python3 manage.py makemigrations
```

Copied! Executed!

- and run migration

```
1. 1
1. python3 manage.py migrate
```

Copied! Executed!

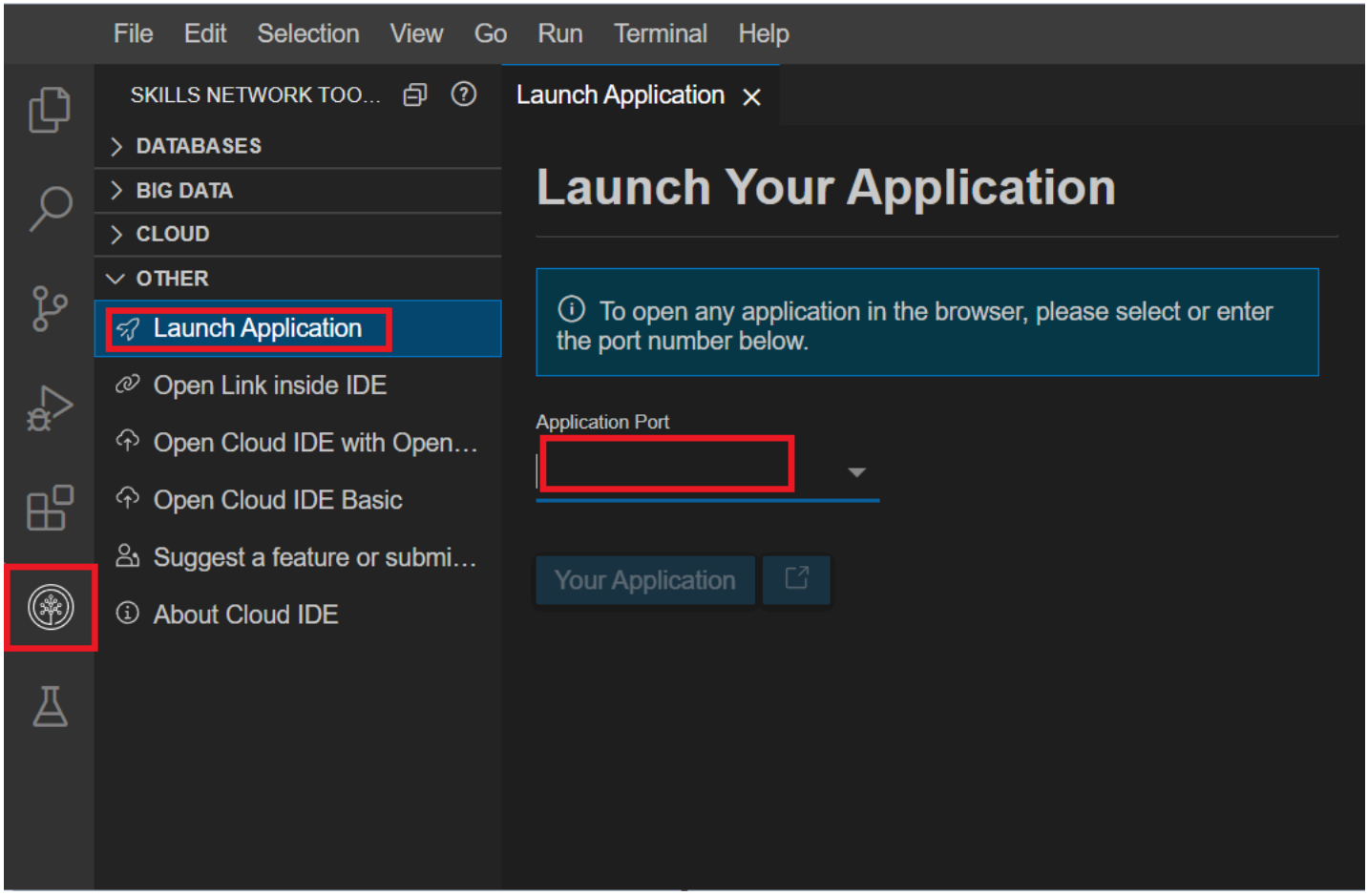
- Then start a development server hosting apps in the `firstproject`:

```
1. 1
1. python3 manage.py runserver
```

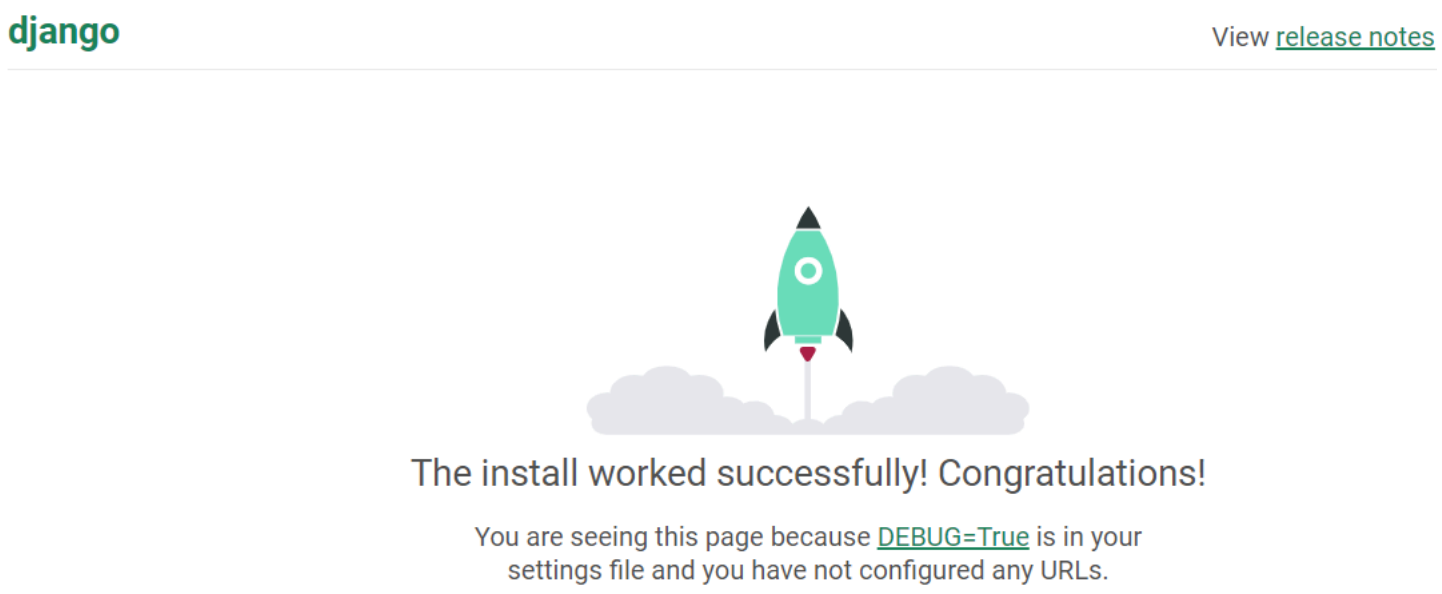
Copied! Executed!

To see your first Django app from Theia,

- Click on the Skills Network button on the left, it will open the **"Skills Network Toolbox"**. Then click the **Other** then **Launch Application**. From there you should be able to enter the port `8000` and launch.



and you should see the following welcome page:



When developing Django apps, in most cases Django will automatically load the updated files and restart the development server. However, it might be safer to restart the server manually if you add/delete files in your project.

Let's try to stop the Django server now by pressing:

- `Control + C` or `Ctrl + C` in the terminal

Add Your First View

Next, let's include your ***firstapp*** into ***firstproject***

Open ***firstproject/settings.py*** file.

Open **settings.py** in IDE

Find `INSTALLED_APPS` section, and add a new app entry as

```
1. 1
```

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

Copied!

Your `INSTALLED_APPS` should look like the following

```
# Application definition

INSTALLED_APPS = [
    'firstapp.apps.FirstappConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

You can also see some pre-installed Django apps such as ***admin*** for managing the Admin site, ***auth*** for authentication, etc.

Next, we need to add the ***urls.py*** of ***firstapp*** to ***firstproject*** so that views of ***firstapp*** can be properly routed.

- Create an empty ***urls.py*** under ***firstapp*** folder

```
1. 1
2. 2
```

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

Copied! Executed!

- Open ***firstproject/urls.py***, you can find a ***path*** function: `from django.urls import path`, has been already imported.

Open **urls.py** in IDE

Now also import an `include` method from `django.urls` package:

```
1. 1
```

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

Copied!

- Then add a new `path` entry

```
1. 1
```

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

Copied!

Your ***firstproject/urls.py*** now should look like the following:

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('firstapp/', include('firstapp.urls')),
]
```

Now you can create your first view to receive HTTPRequest and return a HTTPResponse wrapping a simple HTML page as its content.

- Open ***firstapp/views.py***, write your first view after the comment `# Create your views here.`

Open **views.py** in IDE

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
```

```
1. from django.http import HttpResponse
2.
3. def index(request):
4.     # Create a simple html page as a string
5.     template = "<html>" \
6.                 "This is your first view" \
7.                "</html>"
8.     # Return the template as content argument in HTTP response
9.     return HttpResponse(content=template)
```

Copied!

Next, configure the URL for the index view.

- Open ***firstapp/urls.py***, add the following code:

Open **urls.py** in IDE

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
1. from django.urls import path
2. from . import views
3.
4. urlpatterns = [
5.     # Create a path object defining the URL pattern to the index view
6.     path(route='', view=views.index, name='index'),
7. ]
```

Copied!

That's it, now let's test your first view.

- Run Django sever if not started:

```
1. 1
2. 2
```

```
1. cd ..
2. python3 manage.py runserver
```

Copied! Executed!

Now click the *Launch Application* button below to open the Application in a browser.

Launch Application

Django will map any HTTP requests starting with `/firstapp` to `firstapp` and search any matches for paths defined in ***firstapp/urls.py***.

That's it, you should see your the ***HTTPResponse*** returned by your first view, which is a simple HTML page with content `This is your first view`.

Coding Practice: Add a View to Return Current Date

- Complete and add the following code snippet to create a view to returning today's date.

Add the a `get_date()` view function in `firstapp/views.py` (remember to save the updated file):

```
1
2
3
4
5
6
7
8
```

```
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#)
```

Copied!

and add a `/date` URL path to `firstapp/urls.py` for `get_date()` view:

```
1
2
3
4
5
6
```

```
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'),
]
```

Copied!

▶ Click here to see solution

Now click on the launch application button to open the `/date` route you just created above

Launch Application

Containerizing the app using Docker

Docker is a powerful tool that makes it easy to run applications regardless of the machine you wrote the code on and the machine you want to run it on. It is widely used in practice as it allows developers to avoid the "But it runs on my laptop!" problem when their code doesn't work.

You will need to install Docker first from this link if you are working locally.

This is how it works for (basic) python applications:

1. Create a `requirements.txt` file
2. Create a `Dockerfile` which contains instructions on how to build a Docker image
3. Run `docker compose up` to create a container image, and run it
4. Commit and push the image to a remote repo so others can run it exactly as you've configured!

Docker images are commonly used in conjunction with Kubernetes, which is a service that manages containers.

You will be briefly introduced to how you can setup this Django application to run using Docker.

Before going into the technical stuff revolving around Docker, we just need to make one final change to our codebase. By deafult, django is configured to block all traffic from all hosts including the traffic coming from CloudIDE until we explicity define in them in `settings.py`.

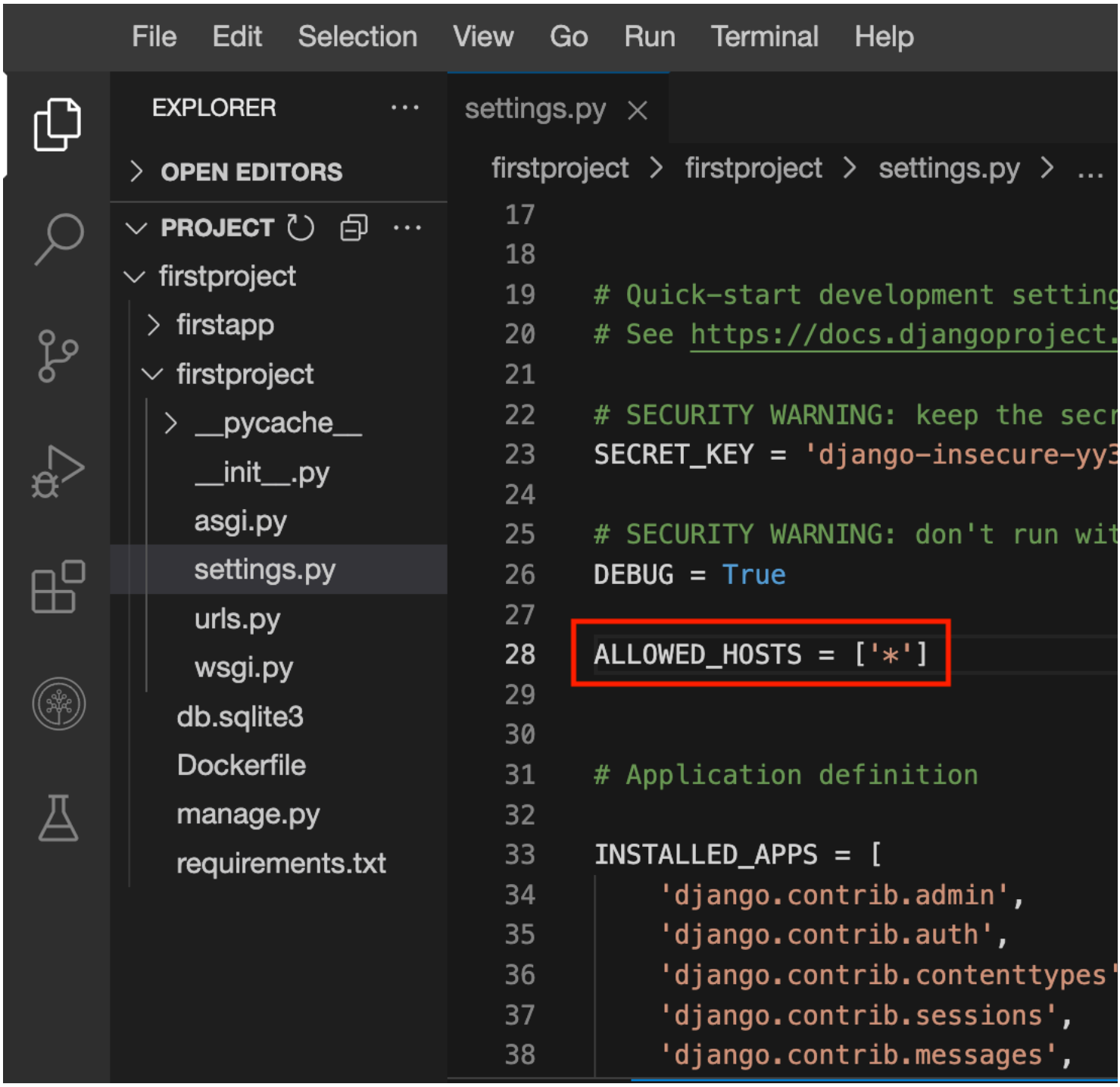
Therefore, open the setting.py file and change the `ALLOWED_HOSTS = []` code to:

```
1
```

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

Copied!

Open **settings.py** in IDE



Now lets start working with Docker.

First we need to create the `requirements.txt` file, which we use to tell Docker what python packages it needs to install. Run the following command in the main `/firstproject` folder.

```
1. 1
2. 2
```

```
1. pip install pipreqs
2. pipreqs .
```

Copied! Executed!

Next, we want to create a `Dockerfile` which instructs Docker how to build your application (in the same directory):

Run the following command to create an empty Dockerfile

```
1. 1
```

```
1. touch Dockerfile
```

Copied! Executed!

Then open the newly created Dockerfile and copy the following contents to it.

Open **Dockerfile** in IDE

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
```

```
1. # syntax=docker/dockerfile:1
2. FROM python:3
3. ENV PYTHONDONTWRITEBYTECODE=1
4. ENV PYTHONUNBUFFERED=1
5. WORKDIR /code
6. COPY requirements.txt /code/
7. RUN pip install -r requirements.txt
8. COPY . /code/
9. CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Copied!

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 use a python 3 image. You can find more details on how this code works here.

Now we can run the following command to create and run the container image:

```
1. 1
```

```
1. docker build . -t my-django-app:latest && docker run -e PYTHONUNBUFFERED=1 -p  8000:8000 my-django-app
```

Copied! Executed!

You should see something like:

```
[+] Running 1/1
 ⠿ Container firstproject-web-1  Created
Attaching to firstproject-web-1
firstproject-web-1  | Watching for file changes with StatReloader
firstproject-web-1  | Performing system checks...
firstproject-web-1  |
firstproject-web-1  | System check identified no issues (0 silenced).
firstproject-web-1  | March 07, 2023 - 14:53:31
firstproject-web-1  | Django version 4.1.7, using settings 'firstproject.settin
firstproject-web-1  | Starting development server at http://0.0.0.0:8000/
```

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. `Part 2` shows the steps to deploy for real 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 READY

1.39.6

Use Code Engine directly in your Lab environmen
Engine Projects are provided by Skills Network a

Create Project

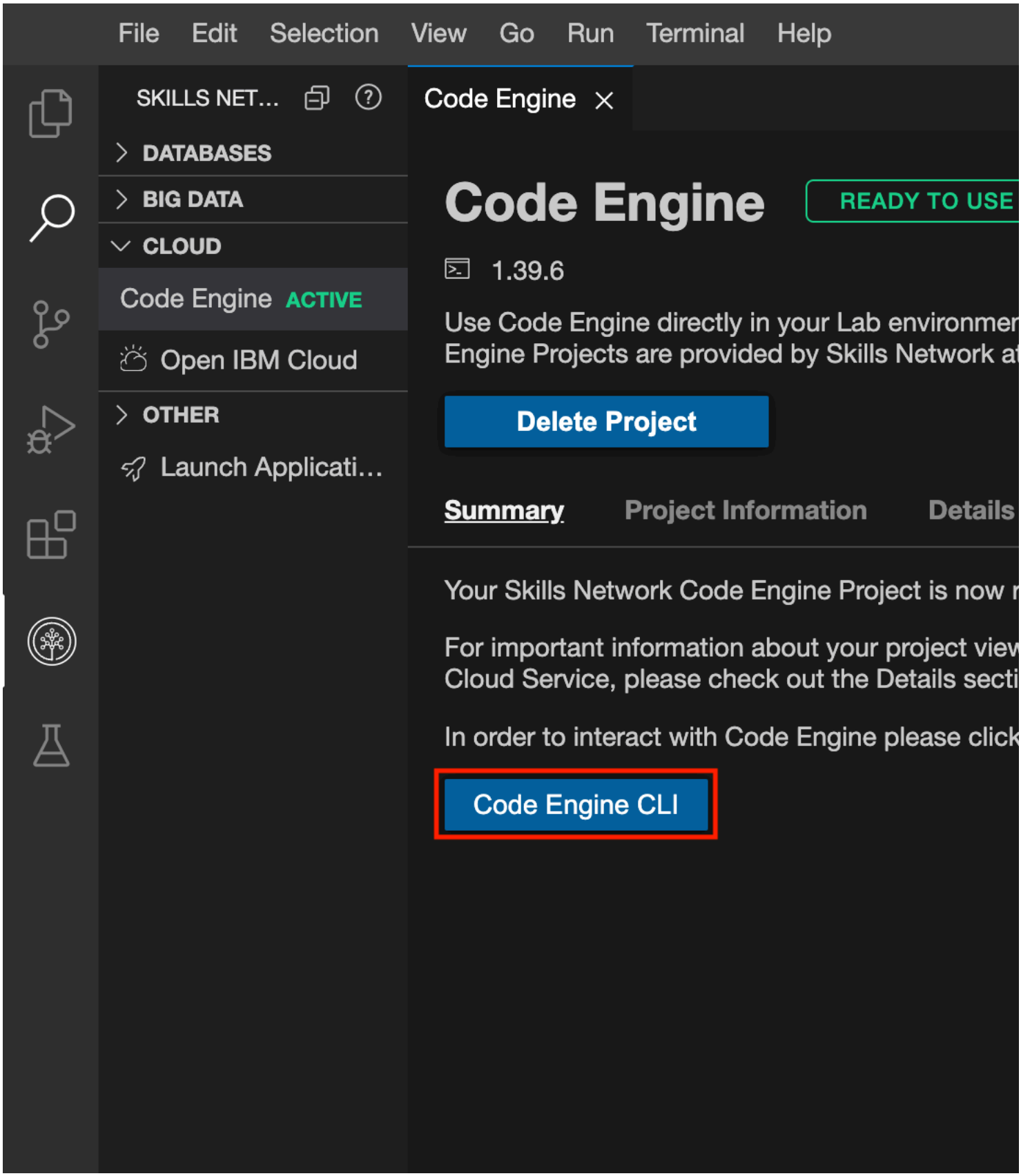
Summary Project Information Details

Get started with Code Engine the easy way with
Project, click the button above.

For more details about Code Engine as an IBM C

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

```
1. 1
```

```
1. APP_NAME=my-django-app
```

Copied! Executed!

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

```
1. 1
2. 2
3. 3
```

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

Copied! Executed!

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

```
1. 1
```

```
1. ibmcloud ce application create --name ${APP_NAME} --image ${REGISTRY}/${SN_ICR_NAMESPACE}/${APP_NAME}:latest --registry-secret icr-secret --port 8000
```

Copied! Executed!

▶ 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.

```
1. 1
```

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

Copied! Executed!

```
1. 1
```

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

Copied! Executed!

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