

Create Your First Django App and Deploy using Docker



Estimated time needed: 20 minutes

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

Create Your First Django Project

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

First, we need to install Django related packages.

- Go to terminal and run:

```
1. 1
```

```
1. python3 -m 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

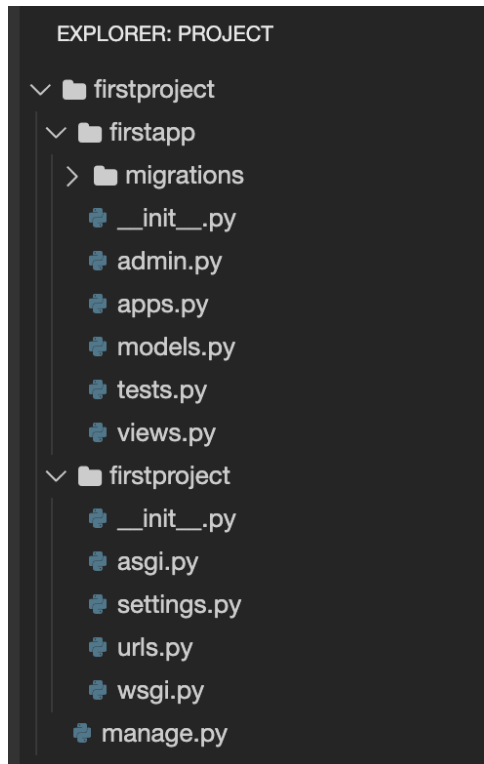
```
1. 1
```

```
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:

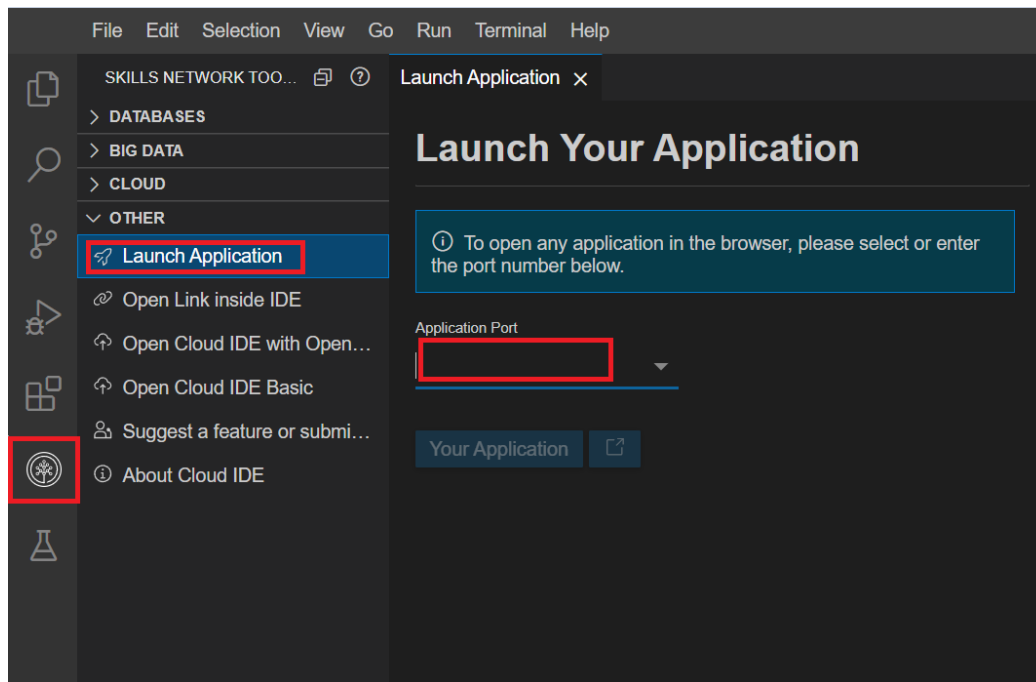
1. 1

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:



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

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

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

Copied!

Open `settings.py` in IDE

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

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

1. python3 manage.py runserver
```

Copied! Executed!

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

Launch Application

After a new browser tab is opened, add a `/firstapp` path to your full URL, note that the `userid` is a place holder and should be replaced by your real id shown after clicking *Launch Application*

`https://userid-8000.theiadocker-1.proxy.cognitiveclass.ai/firstapp/`

Basically, 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. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8

1. from datetime import date
2.
3. def get_date(request):
4.     today = date.today()
5.     template = "<html>" \
6.         "Today's date is {}" \
7.         "</html>".format(#<HINT> add today here#)
8.     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. 1
2. 2
3. 3
4. 4
5. 5
6. 6

1. urlpatterns = [
2.     # Create a path object defining the URL pattern to the index view
3.     path(route='', view=views.index, name='index'),
4.     # Add another path object defining the URL pattern using `/date` prefix
5.     path(route='#<HINT> add a date URL path here#', view=#add the get_date function name here #, name='date'),
6. ]
```

Copied!

▼ Click here to see solution

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

1. from datetime import date
2.
3. def get_date(request):
4.     today = date.today()
5.     template = "<html>" \
6.         "Today's date is {}" \
7.         "</html>".format(today)
8.     return HttpResponse(content=template)
```

Copied!

Open `firstapp/urls.py`

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

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.     # Add another path object defining the URL pattern using `/date` prefix
8.     path(route='date', view=views.get_date, name='date'),
9. ]
```

Copied!

Go to following URL and the `get_date()` view should return current date:

```
1. 1

1. `https://userid-8000.theiadocker-1.proxy.cognitiveclass.ai/firstapp/date/`
```

Copied!

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 default, django is configured to block all traffic from all hosts including the traffic coming from CloudIDE until we explicitly define in them in `settings.py`.


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

```
1. 1

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

Copied!

Open **settings.py** in IDE

 `django-allowed-hosts`

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):

```
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.settings'
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

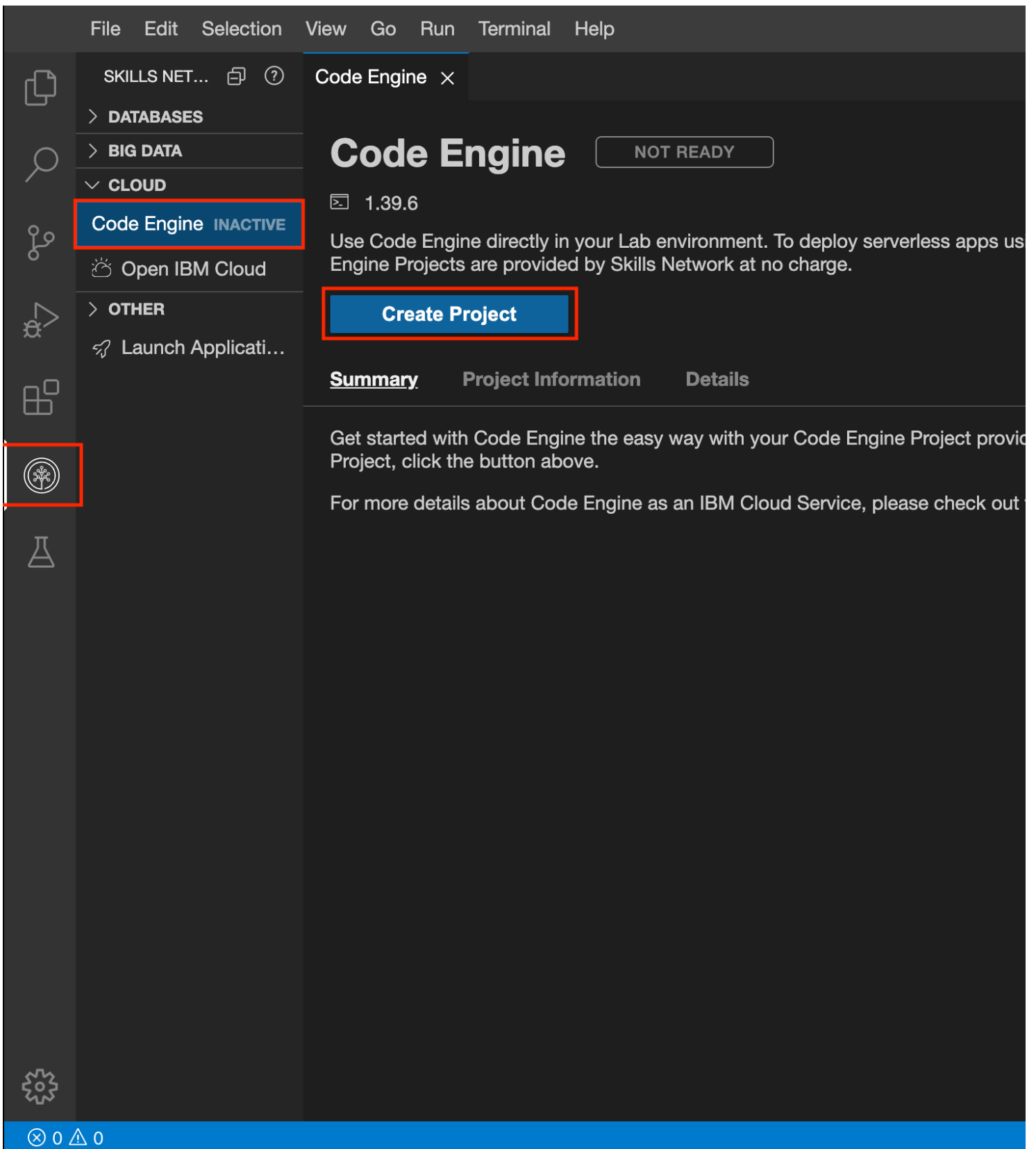
If you would like to host your application and have it be available for anyone to use, you can follow these steps in order to deploy it. The deployment will be to IBM Cloud's Code Engine. IBM Cloud Code Engine is a fully managed, cloud-native service for running containerized workloads on IBM Cloud. 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



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 already set up for you.

SKILLS NET...

> DATABASES

> BIG DATA

> CLOUD

Code Engine **ACTIVE**

Open IBM Cloud

> OTHER

Launch Applicati...

Code Engine

Code Engine

READY TO USE

1.39.6

Use Code Engine directly in your Lab environment. To deploy serverless apps using Code Engine Projects are provided by Skills Network at no charge.

Delete Project

Summary

Project Information

Details

Your Skills Network Code Engine Project is now ready to use. You can now create and deploy applications to your project.

For important information about your project view the Project Information section. If you are using Code Engine on the IBM Cloud Service, please check out the Details section.

In order to interact with Code Engine please click the following button:

Code Engine CLI

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!

If you get the message that the "my-django-app already exists....", please use the below command to update the app:

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

Copied! Executed!

Once the app is deployed, open the url outputted in the terminal, and 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!

Part 2: [Optional] - Deploying to your own account!

This deployment has some prerequisites but ensures that your deployment is permanent until you delete it yourself.

Pre-Requisites

For deploying to your own environment of Code Engine, you will need an IBM cloud account with billing enabled. You can [try it at no charge](#) and receive USD\$200 in cloud credits.

Step 1. Log in to your IBM Cloud account.

Using the `ibmcloud login` command log into your own IBM Cloud account. Remember to replace `USERNAME` with your IBM Cloud account email and then enter your password when prompted to.

```
1. 1
1. ibmcloud login -u USERNAME
```

Copied! Executed!

Use `ibmcloud login --sso` command to login, if you have a federated ID.

Then target any specific resource group in your account. By default, if you've completed the sign up process for your IBM Cloud account, you can use the `Default` resource group.

```
1. 1
1. ibmcloud target -g Default
```

Copied! Executed!

Step 2. Target a region and a resource group

Choose the region closest to you and/or your target users. Picking a region closer to you or your users makes the browser extension faster. The further the region the longer the request to the model has to travel.

You can choose any region from this list:

Americas

- `us-south` - Dallas
- `br-sao` - Sao Paulo
- `ca-tor` - Toronto
- `us-east` - Washington DC

Europe

- `eu-de` - Frankfurt
- `eu-gb` - London

Asia Pacific

- `au-syd` - Sydney
- `jp-tok` - Tokyo

Use the following commands to target Dallas as the region and the `Default` resource group.

```
1. 1
2. 2
1. REGION=us-south
2. RESOURCE_GROUP=Default
```

Copied! Executed!

```
1. 1
1. ibmcloud target -r ${REGION} -g ${RESOURCE_GROUP}
```

Copied! Executed!

Step 3. Create a namespace and log in to ICR

You'll need to create a namespace before you can upload your images, and make sure you're targeting the ICR region you want, which right now is `global`.

Choose a name for your namespace, specified as `${NAMESPACE}`, and create the namespace. Currently, it's set to `my-namespace`, you can choose to rename it to anything you choose.

```
1. 1
1. NAMESPACE=my-namespace
1. 1
2. 2
3. 3
1. ibmcloud cr region-set global
2. ibmcloud cr namespace-add ${NAMESPACE}
3. ibmcloud cr login
```

Copied! Executed!

Step 4. Create and Select a new Code Engine project

Now we will create a new project on Code Engine to deploy our Docker App image to it.

However, please note these commands are assuming you have already added a credit card to your IBM Cloud Account and will fail if one has not already been added. Though, you won't be charged until you reach a certain amount of usage. To learn more about IBM Code Engine's free tier, click [here](#).

In this example, a project named `django-app-project` will be create in the resource group set by the previous command.

```
1. 1
2. 2
1. ibmcloud ce project create --name django-app-project
2. ibmcloud ce project select --name django-app-project
```

Copied! Executed!

Step 5. 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 the namespace you created earlier:

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

Copied! Executed!

Then, execute the following command to create and deploy a new application to Code Engine

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
1. ibmcloud ce application create \
2. --name ${APP_NAME} \
3. --port 8000 \
4. --min-scale 1 --max-scale 2 \
5. --cpu 1 --memory 4G \
6. --image private.${REGISTRY}/${NAMESPACE}/${APP_NAME}:latest \
7. --registry-secret ce-auto-icr-private-${REGION} \
8. --env ACCEPT_LICENSE=true
```

Copied! Executed!

It may take a few minutes to complete the deployment. If the deployment is successful, you'll get the URL of the application's public endpoint from the command output.

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

Author(s)

[Yan Luo](#)
[Richard Ye](#)
[Talha Siddiqui](#)

Changelog

Date	Version	Changed by	Change Description
30-Nov-2020	1.0	Yan Luo	Initial version created
07-Mar-2023	1.1	Richard Ye	Deploy using Docker
08-Mar-2023	1.2	Talha Siddiqui	Deploy to CodeEngine
17-Mar-2023	1.3	K Sundararajan	App launch instructions updated
21-Mar-2023	1.4	K Sundararajan	Instructions updated

© IBM Corporation 2023. All rights reserved.