Week 4: Deployment on Flask

https://hammerhead-app-k5sqf.ondigitalocean.app/

Name: Alexis Collier
Batch Code: LISUM14
Date: 24 October 2022
Submitted to: Data Glacier

To complete this Flack, I needed the following:

A GitHub account.
Python3
Visual Studio Code

Step 1: Create a Python Virtual Environment for the Project

Run the following commands:

Mkdir flask-app
Cd flask-app

Next, create a directory to store all virtual environments:

Mkdir ~/.venvs

Now create a virtual environment using Python:

Python3 -m venv ~/.venvs/flask

This creates a directory called flask within the .venvs directory. Inside, it installs a local version of Python and a local version of pip. Can use this to install and configure an isolated Python environment for the project.

Before installing the project's Python requirements, I activated the virtual environment.

Using the following command:

Source ~/.venvs/flask/bin/activate

With the virtual environment active, install flask and gunicorn using the local instance of pip:

Pip install Flask gunicorn

Now with the flask package installed, save this requirement and its dependencies so the App Platform can install them later.

Do this now using pip and then saving the information to a requirements.txt file:

Pip freeze > requirements.txt

With all of the software needed to start a Flask app, we are almost ready to deploy.

Step 2: Creating a Minimal Flask App

In this step, we will build a standard Hello Sammy! Flask application.

Using a text editor, I used Visual Studio Code to *open a file named app.py:*

Nano app.py

Now add the following code to the file:

```
From flask import Flask
App = Flask(__name__)

@app.route('/')
Def hello_world():
  Return 'Hello Sammy!'
```

This code is the standard Hello World example for a flask with a slight modification to say hello to an example name.

Now that I have the written application code. Now will configure the Gunicorn server.

Step 3: Setting Up Gunicorn Configuration

Gunicorn is a Python WSGI HTTP server many developers use to deploy their Python applications. This WSGI (Web Server Gateway Interface) is necessary because traditional web servers do not understand how to run Python applications. For purposes, a WSGI allows the deployment of Python applications consistently. In this example, we will make the application accessible on port 8080, the standard App Platform port. We will also configure two worker threads to serve the application.

Open a file named gunicorn_config.py:

Nano gunicorn_config.py

Now add the following code to the file:

Bind = “0.0.0.0:8080”
Workers = 2

This is all that is needed to run the app on the App Platform using Gunicorn. Next, commit the code to GitHub and then deploy it.

Step 4: Pushing the Site to GitHub

This means we must get the site in a git repository and then push that repository to GitHub.

First, initialize the project directory containing files as a git repository:

Git init

When working on the Flask app locally, certain files that are unnecessary for deployment get added. Exclude those files using Git's ignore list. Create a new file called .gitignore:

Nano .gitignore

Add the following code to the file:

*.pyc

Save and close the file.

Now execute the following command to add files to the repository:

Git add app.py gunicorn_config.py requirements.txt .gitignore

Make initial commit:

Git commit -m "Initial Flask App"

Files commit:

[secondary_label Output]
[master (root-commit) aa78a20] Initial Flask App
4 files changed, 18 insertions(+)
Create mode 100644 .gitignore
Create mode 100644 app.py
Create mode 100644 gunicorn_config.py
Create mode 100644 requirements.txt

Open the browser, navigate to GitHub, log in with profile, and create a new repository called flask-app. Next, create an empty repository without a README or license file.

Once the repository is created, return to the command line and push local files to GitHub.

First, add GitHub as a remote repository:

Git remote add origin https://github.com/_username/flask-app

Next, rename the default branch main to match what GitHub expects:

Git branch -M main

Finally, push the main branch to github's main branch:

Git push -u origin main

Files transfer:

[secondary_label Output]
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 1.20 kib | 77.00 kib/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To github.com:masonegger/flask-app.git
 * [new branch]   main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

Enter GitHub credentials when prompted to push code.

The code is now on GitHub and accessible through a web browser. Now will deploy the app platform.

Step 5: Deploying with an App Platform

Once the code is pushed, visit the App platform of your choice and launch; I chose DigitalOcean.
Prompt requests that connect GitHub account:

Connect the GitHub account and allow access to repositories.

Click Install and Authorize Github.

Once the GitHub account is connected, select the _account/flask-app repository and click Next.

Next, provide the app's name, choose a region, and ensure the main branch is selected. Then ensure that Autodeploy code changes are checked. Click Next to continue to populate a partial

Run command automatically.

App configuration

Click the Edit link next to the Build and Run commands to complete the build command. The completed build command needs to reference the project's WSGI file.

Replace the existing command with the following:

Gunicorn --worker-tmp-dir /dev/shm app:app

Click Next to Finalize and Launch screen. The app will build and deploy.

Once the app finishes deploying, click on the link to the app provided. This link takes you to the Hello Sammy page.

Deploy website in browser 😊

Summary

- Created a simple Flask app.
- Configured a Gunicorn HTTP server.
- Deployed the app to an App Platform using the Gunicorn server.

Reference

https://docs.digitalocean.com/tutorials/app-deploy-flask-app/