How to Dockerize an Existing Flask Application

Run Flask applications with less effort

Mahbub Zaman Apr 14 · 3 min read ★

Photo by Dean Pugh on Unsplash

In this post, I'll explain how to Dockerize an existing Flask application. I'm going to use one of my Python projects for demonstration purposes. By using a Dockerized Flask application, other developers can easily run the project without any environment management, which is great for saving time and effort. Moreover, developers can focus on development. If you're new to Docker, then read the following post where I've covered some basics.

How to Mount a Directory Inside a Docker Container
Focus on writing code without worrying about environment management
towardsdatascience.com

Setup

First, you will need to install Docker and download a git repository from GitHub. For this setup, I'm using macOS.

Now, I'll create a Docker image that contains Python, and the web application framework Flask as a dependency. Let's break down the individual ingredients of the **Dockerfile** file.

```
FROM python:3.9.1
ADD . /python-flask
WORKDIR /python-flask
RUN pip install -r requirements.txt
```

To define the parent image, we need to use the `From` command. Here we working directory to **python-flask** using the `WORKDIR` command. Finally, using pip, we are going to install our application's dependency: Flask.

Now, I'll create a Docker compose file to run a Docker container using the Docker image we just created. Let's break down the individual ingredients of the **docker-compose.yml** file.

```
version: "3.8"
services:
  app:
    build: .
    command: python main.py
    ports:
      - "5000:5000"
    volumes:
      - .:/python-flask
```

Inside the **docker-compose.yml** file, we have the version, services, app, build, command, ports, and volumes tag. The **version** tag is used to define the Compose file format. Have a look at the documentation to learn more. The **services** tag is used to define the services we want to use for our application. For our application, we only have one service called **app**. The **build** command will build our Docker image using the Dockerfile we created earlier. The **command** tag is used to run the main.py file. The **ports** tag is used to define both host and container ports. It maps port 5000 on the host to port 5000 on the container. Because by default, Flask runs on port 5000. Finally, the **volumes** tag is used to mount a folder from the host machine to the container.

Now we are going to run the following command from the same directory where the **docker-compose.yml** file is located. The `docker compose up` command will start and run the entire app.

```
docker compose up
```

Congratulations! We are now successfully running a Flask application inside a Docker container. You can now access the Flask application via your favorite web browser by vising the URL http://localhost:5000/.

Now run `docker ps` to see all the running containers. Here, I have one.

docker ps

Finally, let's break down the individual ingredients of the **main.py** file.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
        return 'Hello World'

if __name__ == '__main__':
        app.run(host="0.0.0.0", debug=True)
```

First, we imported the Flask class and then created an instance of that class. After that, using `route()` decorator, we are telling Flask to trigger the `hello_world()` function based on the URL. Finally, we have the entry point of our code, where we are running the Flask application using the `run()` method. The first argument is `host="0.0.0.0"` which makes the application available on all public IPs. The second argument is `debug=True` will enable debug mode. You can read more here about a Flask application and about debugging from here

Now you know how to run a Flask application inside a Docker container. I hope this will help you to get started with Flask and Docker. Happy coding!

Related Post

How To Run a Python Script Using a Docker Container
Run Python with less effort
towardsdatascience.com

Data Science Python Software Engineering Docker Programming