Tasnuva Zaman

152 Followers About Follow

Dockerize Your Flask Application

Tasnuva Zaman Apr 8, 2019 · 3 min read

In this blog, we will write a Dockerfile to dockerize a simple python flask app.

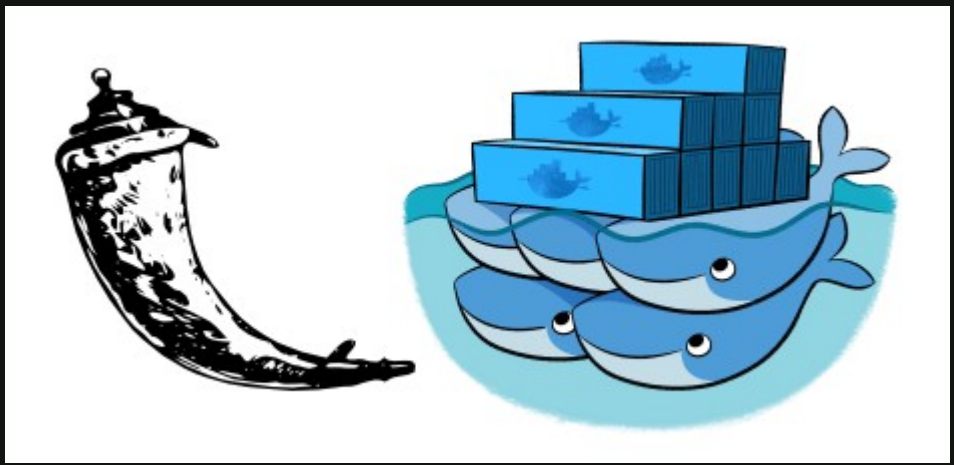
Prerequisites:

1. Python 3.6
2. Flask
3. docker

What Is Docker?

Docker is a tool which helps to create, deploy, and run applications by using containers. **Docker** provides developers and operators with a friendly interface to build, ship, and run containers on any environment. To install docker follow the link https://www.docker.com/get-started

Containers are a group of processes run in isolation, these processes are running in a shared kernel.



Step 1|| *Creating a flask app*

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

@app.route('/')
def hello_world():
    return render_template('index.html')

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

Folder Structure:

```
src
  ____ app.py
  ____ templates
         ____ index.html
  ____ requirements.txt
  ____ Dockerfile
```

requirements.txt

```
Flask==0.12
```

templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>dockerize flask app</title>
</head>
<body>
    <h1> Hello World! </h1>
</body>
</html>
```

Step 2 || Creating a Dockerfile

```
1. FROM python:3.6.1-alpine
2. WORKDIR /project
3. ADD . /project
4. RUN pip install -r requirements.txt
5. CMD ["python","app.py"]
```

> Note: Each of these lines of Dockerfile is a layer and each layer contains only the delta, or changes from the layers before it.

Line by line description:

```
1. FROM python:3.6.5-alpine
```

> *Note: **FROM** allows us to initialize the build over a base image. In our case, we are using a python:3.6.5-alpine image. (*Alpine is a small Linux Distribution (~ 5MB).*) It is heavily used by Docker apps because of its small size. In short, we are using a Linux environment with python 3.6.5 for our app.*

```
2. WORKDIR /project
```

> *Note: We create a work directory called* project *where the "**present working directory**" will be set.*

```
3. ADD . /project
```

> *Copy everything in the current directory (our server code and configurations) into the* **project** *directory.*

```
4. RUN pip install -r requirements.txt
```

> *permissions.*

```
5. CMD ["python","app.py"]
```

`CMD` is the command that is executed when you start a container. Here, you are using `CMD` to run your Python applcation. There can be only one `CMD` per Dockerfile. If you specify more than one `CMD`, then the last `CMD` will take effect.

> *Note: Layers that change frequently, such as* ***copying source code into the image****, should be placed near the bottom of the file to take full advantage of the Docker layer cache.*

Step 3 || *Build the docker image*

> Run the following command to create the docker image from `src` directory, *Pass in the* `-t` *parameter to name your image python-hello-world.*

```
$ docker image build -t python-hello-world .
```

> *Verify that your image shows in your image list:*

```
$ docker image ls
```

Step 4 || *Run the docker container*

```
$ docker run -p 5001:5000 -d python-hello-world
```

> *The* `-p` *flag maps a port running inside the container to your host. In this case, we're mapping the Python app running on port 5000 inside the container to port 5001 on your host.*

Step 5 || *Access The application from the host machine*

```
Navigate to http://localhost:5001 in a browser to see the results.
```

 # Check the log output of the container

If you want to see logs from your application, you can use the `docker container logs` command. By default, `docker container logs` prints out what is sent to standard out by your application

```
$ docker container ls

You should see your container id by running this command.

$ docker container logs [container id]
output should be look like :

* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
172.17.0.1 - - [28/Jun/2017 19:35:33] "GET / HTTP/1.1" 200 -
```

 # Stop a docker container

```
$ docker container stop [container id]

Tip: You need to enter only enough digits of the ID to be unique.
Three digits is typically adequate.
```

> *You can also use the names that you specified before to stop a container*

 # Remove the stopped containers

The following command removes any stopped containers, unused volumes and networks, and dangling images:

```
$ docker system prune
```

hooa your application is dockerized!!

Docker Python Flask DevOps