Lab 2

Lab 2: Write Your First Dockerfile

Objective

We will see an example of converting our Flask app into a Docker image and learning some basic Docker commands. Flask is a micro-framework for building small web applications. We are just using it as an example of a Python project. One can use any other project in other language as well, obviously the commands to build and run the project in Dockerfile will vary.

We will also learn how to pass environment variables in docker both on build time and runtime.

Directory Structure for Dockerizing the Flask App

This is the directory structure of the Flask App

Folder Structure

```
flask-docker-demo/
├── demo.py          # Contains the Flask app code
├── Dockerfile       # Contains the instructions for creating the Docker image
└── requirements.txt   # Lists the necessary Python packages
```

Explanation of each file

- **demo.py**: This file is where the Flask application code lives. It defines the app's behavior and routes. In this example, we're keeping it simple with one route that displays a welcome message.
- **Dockerfile**: The Docker file is a set of instructions that tells Docker how to create an image for this app. It includes details like the base Python image to use, which files to include, any dependencies to install, and commands to start the app.
- **requirements.txt**: This file lists any Python libraries that the app depends on—in this case, just flask. Docker will use this to install the necessary packages when building the image.

Setup Flask and Dockerfile

1. Make Project Folder

Open your terminal and make a folder for your flask application let's say "flask-docker-demo" by executing the following commands (You can also create the folder and above files manually)

```
$mkdir flask-docker-demo
```

```
$cd flask-docker-demo #to change the directory
```

Then open the folder in any code editor like vscode.

Paste the following code into "demo.py":

```
from flask import Flask
import os
app = Flask(__name__)
# Define which environment variables to show
env_vars_to_display = ['CUSTOM_ENV_VAR1', 'CUSTOM_ENV_VAR2']  # Add more as needed
@app.route('/')
def hello():
# Filter and format the environment variables for HTML
```

```
env_vars = "<br>".join([f"{key}: {value}" for key, value in os.environ.items() if key in env_vars_to_display])
message = "Welcome to the Flask tutorials"
if env_vars:
message += "<br><br>Environment Variables:<br>" + env_vars
else:
message += "<br><br>No specific environment variables to display."
return message
if __name__ == "__main__":
app.run(host='0.0.0.0', port=5001, debug=True)
```

2. Insert the following code into the Dockerfile created earlier

Add a new file and name is as “Dockerfile”. **Don’t give any extension**. Paste the following code in it

```
FROM python:alpine3.12
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
EXPOSE 5001
ENTRYPOINT [ "python" ]
CMD [ "demo.py" ]
```

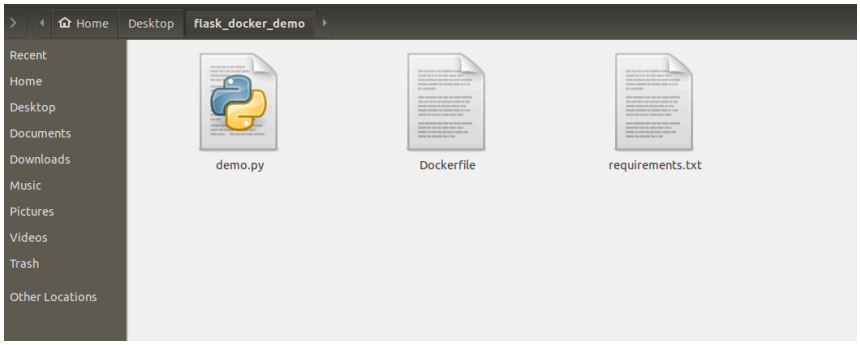
Let’s see what our Dockerfile does.***FROM*** python:alpine3.12 pulls python 3.12’s image from the docker hub, ***WORKDIR*** command sets the working directory. ***COPY*** command copies the flask app and other files present in root into the container “***RUN*** pip install -r requirements.txt” this command will install each requirement written in “requirements.txt ” file one by one bye on the host system. ***EXPOSE*** as the name says exposes port 5001 which Flask app will use to the container so that later it can be mapped with the system’s port.***Entrypoint*** and ***CMD*** together just execute the command “python demo.py” which runs this file.

3. Copy the following into “requirements.txt” file

Create the requirements.txt manually and Add the following line into it

```
flask
```

Our file folder structure should look like this

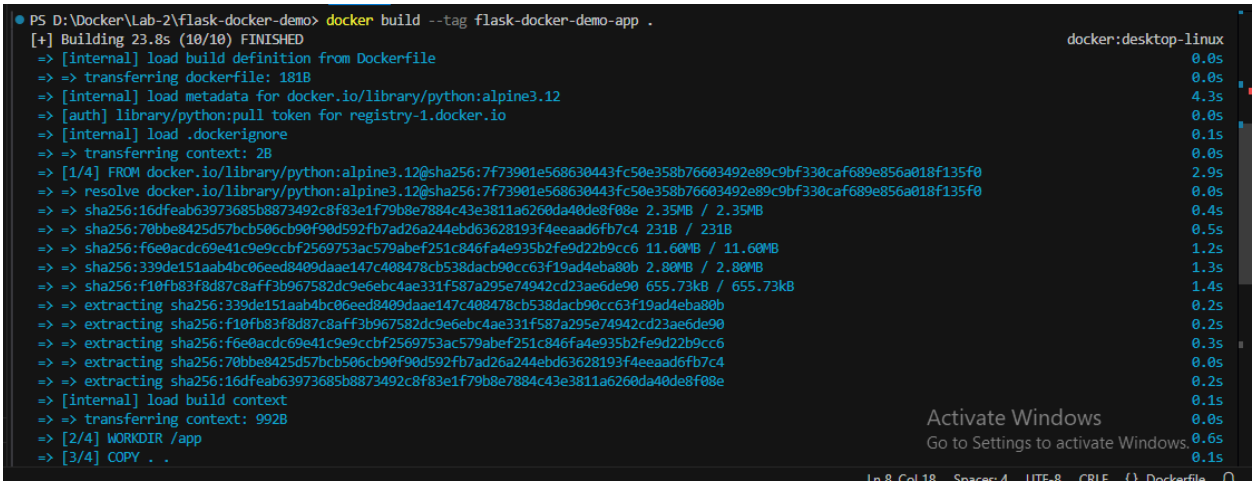


Finally, Dockerizing the Project

1. Build the Docker Image

Make sure you are in root directory of the project and run the following command.

```
docker build --tag flask-docker-demo-app .
```



In this the dot '.' denotes the context of dockerfile, which means the dockerfile will read the files in the current directory, you can set it to some other directory as well.

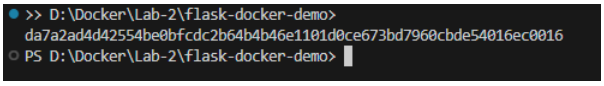
The above command will create an app with the tag flask-docker-demo-app

2. Run the docker image we just created

Run the following command:

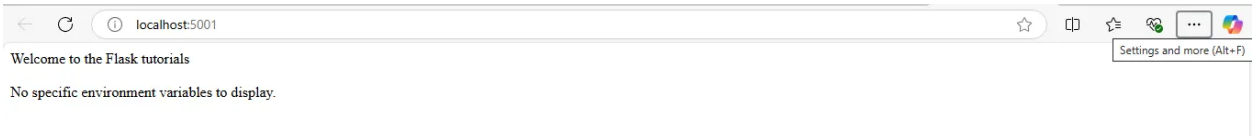
```
docker run -d --name flask-docker-demo-app -p 5001:5001 flask-docker-demo-app
```

In the above command, -d denotes detached mode
--name parameter gives name to the container,
-p parameter maps the host's (your laptops in this case) port 5001 to the container's port 5001 since the container is isolated and we need to map it in order to access it from external environment.
And at last, "flask-docker-demo-app" refers to the image to run.



3. Test the App

Open http://localhost:5001 (Links to an external site.) in the browser



Pat on your back!!! (Although I already gave you the ready-made dockerfile :))

Welcome to the Flask tutorials

No specific environment variables to display.

Now let's learn how to pass environment variables into our app

Build-time vs. Runtime Environment Variables

Build-time Environment Variables:

- **Definition**: These are variables that are set during the image build process.
- **Usage**: They are typically used to pass configuration that influences how an image is built. For example, setting proxy configuration, selecting build versions of software, or defining build-specific data, Api URL binding, etc.
- **Scope**: Only available during the build of a Docker image and are embedded into the image itself. They cannot be changed once the image is built without rebuilding the image.

Runtime Environment Variables:

- **Definition**: These variables are set when the Docker container is run from the Docker image.
- **Usage**: Ideal for passing configuration that affects the behavior of the application at runtime rather than at build. Common examples include database URLs, API keys, feature flags, or operational settings.
- **Scope**: Not included in the image itself and can be changed every time a container is started without needing to rebuild the image.

Key Differences:

1. **Modification**: Build-time variables are fixed upon image build and cannot be changed without rebuilding the image. Runtime variables can be altered each time a container is launched.
2. **Visibility**: Build-time variables can be baked into the image and thus less secure for sensitive data, as anyone who can pull the image can see them. Runtime variables are not stored in the image and can be provided securely at container launch.
3. **Flexibility**: Runtime variables offer greater flexibility for deploying the same image in different environments or configurations without the need to rebuild.

Option A: Passing Environment Variables at Runtime Individually

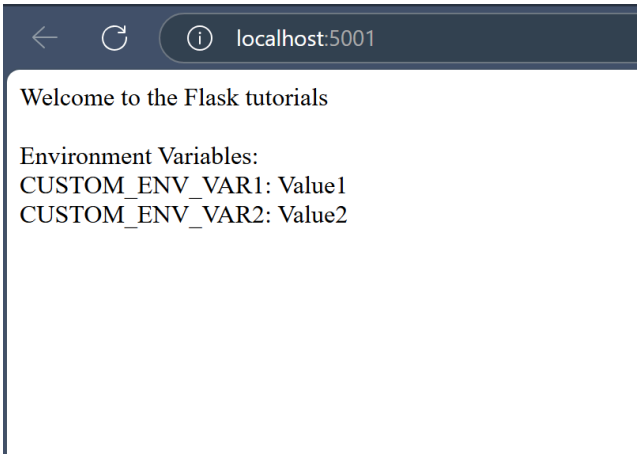
Build the app

docker build --tag flask-docker-demo-app .

Pass the variables during runtime

docker run -d --name flask-docker-demo-app -p 5001:5001 -e CUSTOM_ENV_VAR1=Value1 -e CUSTOM_ENV_VAR2=Value2 flask-docker-demo-app

Open http://localhost:5001 (Links to an external site.) in the browser



Option B: Passing Environment Variables at Runtime Using a file

Build the app

docker build --tag flask-docker-demo-app .

Create a file with .env extension like - myfile.env

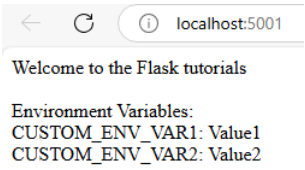
Add all the variables and their values like this

CUSTOM_ENV_VAR1: Value1

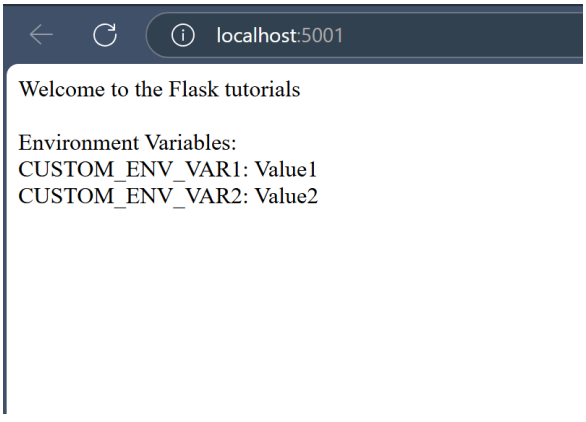
CUSTOM_ENV_VAR2: Value2

Pass the variables during runtime through a file

docker run -d --name flask-docker-demo-app -p 5001:5001 --env-file myfile.env flask-docker-demo-app



Open http://localhost:5001 (Links to an external site.)



Option C: Passing Environment Variables at Build Time

For passing the environment variables during build time we pass them as arguments during the build process. We also need to modify the dockerfile in such a way that the value of the arguments should be captured from outside and passed as an environment variable.

Let me show you how it is doneModify the Dockerfile in this way

FROM python:alpine3.12

WORKDIR /app

COPY . .

ARG CUSTOM_ENV_VAR1

ENV CUSTOM_ENV_VAR1=${CUSTOM_ENV_VAR1}

ARG CUSTOM_ENV_VAR2

ENV CUSTOM_ENV_VAR2=${CUSTOM_ENV_VAR2}

RUN pip install -r requirements.txt

EXPOSE 5001

ENTRYPOINT ["python"]

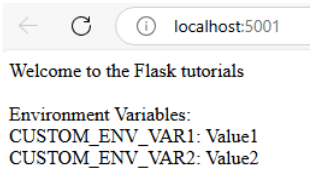
CMD ["demo.py"]

Pass the environment variables during the build time

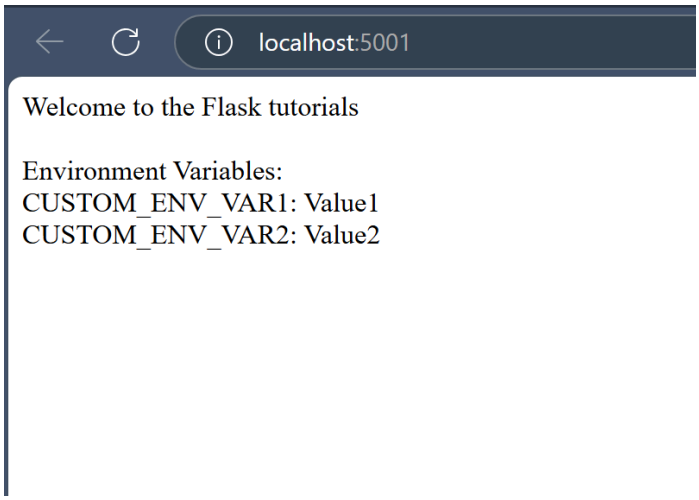
docker build --tag flask-docker-demo-app --build-arg CUSTOM_ENV_VAR1=Value1 --build-arg CUSTOM_ENV_VAR2=Value2 .

Start the container

docker run --name flask-docker-demo-app -p 5001:5001 flask-docker-demo-app



Open http://localhost:5001 (Links to an external site.)



This lab is finished, have some rest...

Just kidding move on to the next lab