Docker Tutorial

This tutorial assumes that you already have a working python Flask web application and want to run it in a Docker container.

Install some tools

- Install Docker Desktop your local laptop. Launch Docker Desktop once you have finished installing it.
- Open an IDE like VSCode

Docker

- Create a text file and name it "requirements.txt" and place any module dependencies in that file that your application requires to run. For instance, you might need Flask.
    - You can use the command "pip3 freeze > requirements.txt" to generate the file from all installed node modules. That will work, but you will also potentially get a LOT of extra modules you do not need for your application. So you will need to delete all the lines of the generated requirements.txt file for modules you are not explicitly importing in your python code!
- Create an empty text file with the name "Dockerfile". (Note that "D" is capitalized and there is no file extension. This is the default file name that Docker looks for. If you name it something else, you will need to supply additional parameters to your Docker commands. So don't do that for this tutorial.)
- Enter the following contents into Dockerfile and save.

```
# syntax=docker/dockerfile:1
FROM python:3.11-slim-bookworm
WORKDIR /python-docker
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . .
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

Here is a quick explanation of what those lines accomplish:

`# syntax`	Syntax and location of images in Dockerhub
`FROM`	Name of base image. (Are you curious why we are using this image as our base image?)
`WORKDIR`	Specify the working dir within the container

`COPY`	Copy files from the local host to the container. Path is relative and optional.
`RUN`	Runs the given command in the container. Useful for installing things in the container.
`COPY`	The dot dot parameters above indicate to copy all files from the local folder to the container.
`CMD`	Start the app when the container is started. The -m switch starts it as a module. The host command will expose the container externally. E.g. to the browser.

- Build the docker image with this terminal command. (Replace <imagename> with the name you wish to give your image. You should use a name that is meaningful to you and the homework assignment):

  `docker build --tag <imagename> .`

  (Note: don't forget the dot at the end of the previous command!)

- Verify that your image was built by using the following terminal command:

  `docker images`

- Run your docker container:

  `docker run <imagename>`

- Note the server URL (e.g. http://172.17.0.2:5000, or localhost:5000) that is echoed to the console when the docker run command is used. You can put that address in a browser. Did it work? If you got an error, do you know why?

- Quit your previous docker run command (e.g. Ctrl-C).

- Let's retry running the container. This time, we will run the container in "detached mode" by using the -d switch (which runs the container in the background and releases the command line to be used for additional commands) and map the required port (e.g. 5000:5000). The first number is the docker host port number which is the port known by the world outside the container. The second number is the port that the application is using inside the container. Enter the following command:

```
docker run -d -p 5001:5000 dockerlab
```

(Note: if you get any errors about port 5001 already being taken, run "docker ps" and note which container is running on that port. Then use the "docker stop <image name>" command to terminate the running container that is using port 5000. Then you should be able to rerun the "docker run" command above.)

- Is your container running? Check by entering "docker ps" into the terminal. You should see a list of all running containers.

- Open a browser and enter "localhost:5001". You should see your application running in the browser.

- You can access your container by creating a shell connection with the following command:

  docker exec -it <running container id> bash

Cleanup

Docker images can be large. Even if your code is small, the image is built from a base image which can be quite large. These images files can be hard to find using traditional File Explorers when trying to free up space on a local machine. The following commands will help you cleanup your Docker environment.

- Stop your container:

  ```
  docker stop <containerid>
  ```

- Remove all stopped containers:

  ```
  docker container prune
  ```

```
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
2e5c897f8a3481282d9fcd65ceeb85b791fb0c06ff58433f80d85c6be77c562a
3cf129262945464b735e5f9cadc163df45bc908d91e67db4cc50305c7d4e56f5
940a19c91a2c940554d6bd11a36c1680e08f5ab62859fcb3a0283015e41a2068
0d511334ca507266bb4a7d5acf3e8e3fb3e255762ddb2154309d7959aa020ec5
fde34edec41341356340aea3263bf1d5896cf9c2c99efe92ad4c8966a2864ff1
eebe347614da03c00740aaa4ab8cf494a0507bc3f02a56a2ec80fe63feb1648a
00fcba69fa195af78f2f2dad4236e9d4c2a67e7835d9023410cc11dc1e253bb0

Total reclaimed space: 486.3kB
```

- Remove the docker image

```
docker image rm <image id>
```

This tutorial was based on a tutorial on FreeCodeCamp.org. Feel free to read that tutorial for additional explanation on the steps above.