Exercise Report

Author: ***Michalis Anastasiou***

Program explanation:

Below you will find the high-level design of a simple web application using the Flask framework.

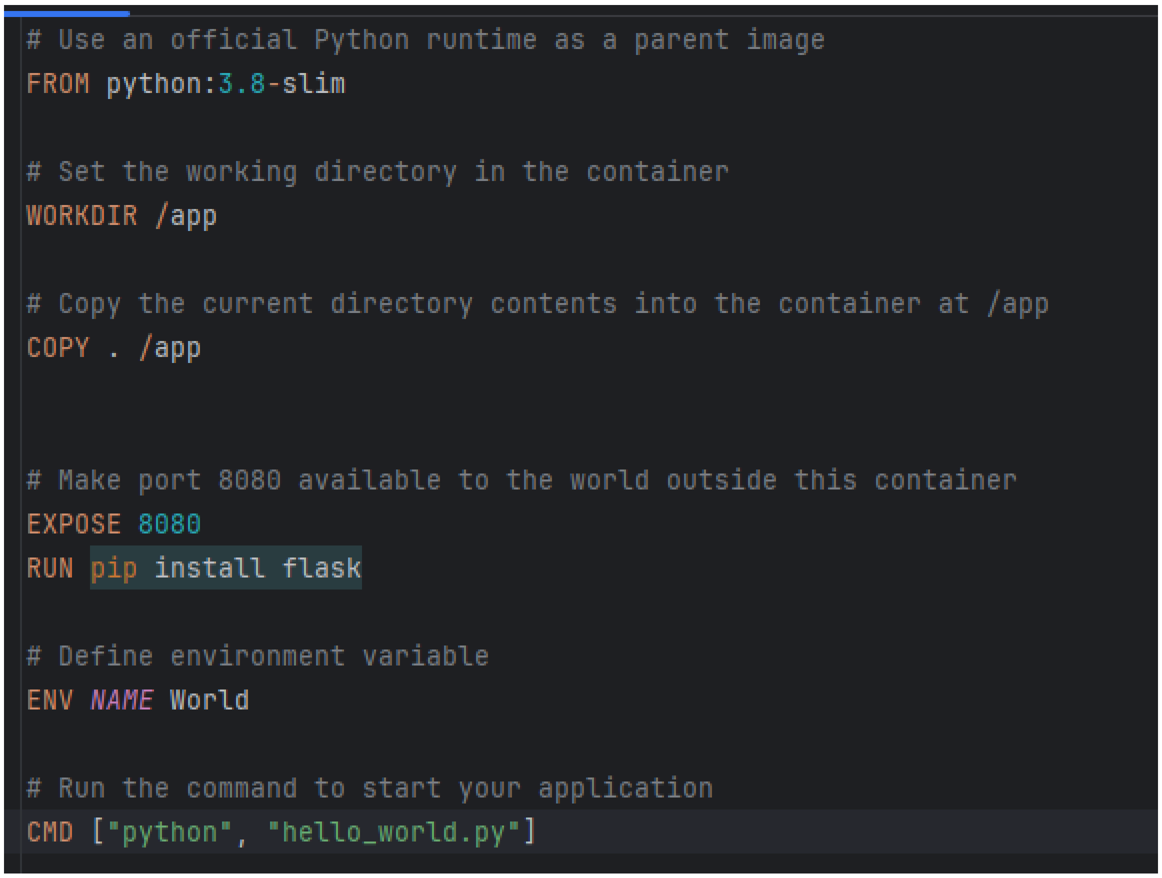
It creates a basic Web Server that listens on port 8080 and responds with "**Hello World**".

- ## Dockerize the python app:

Because this was my first time using Docker, the first step I followed was installing Docker from its' official website (windows version).

I then created a docker file in the same directory as my 'hello_world.py' script.

This file is containing instructions for building the docker image.



```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app


# Make port 8080 available to the world outside this container
EXPOSE 8080
RUN pip install flask

# Define environment variable
ENV NAME World

# Run the command to start your application
CMD ["python", "hello_world.py"]
```

After writing the above in the Dockerfile I executed the 2 following commands; in PyCharm terminal, in order to build the docker image and run the docker container.

```
docker build -t hello_world-app .
```

```
docker run -p 8080:8080 hello_world-app
```

To then test the application, I opened a web browser and navigated to localhost:8080 within the docker container.

- ## Publish docker image to docker hub:

Firstly, I typed docker login in order to login in my docker account. Then I typed

```
docker tag helloworld hellsbells998/TRG:latest
```

in order to tag the docker image in the desired repository name

After tagging the docker image I typed:

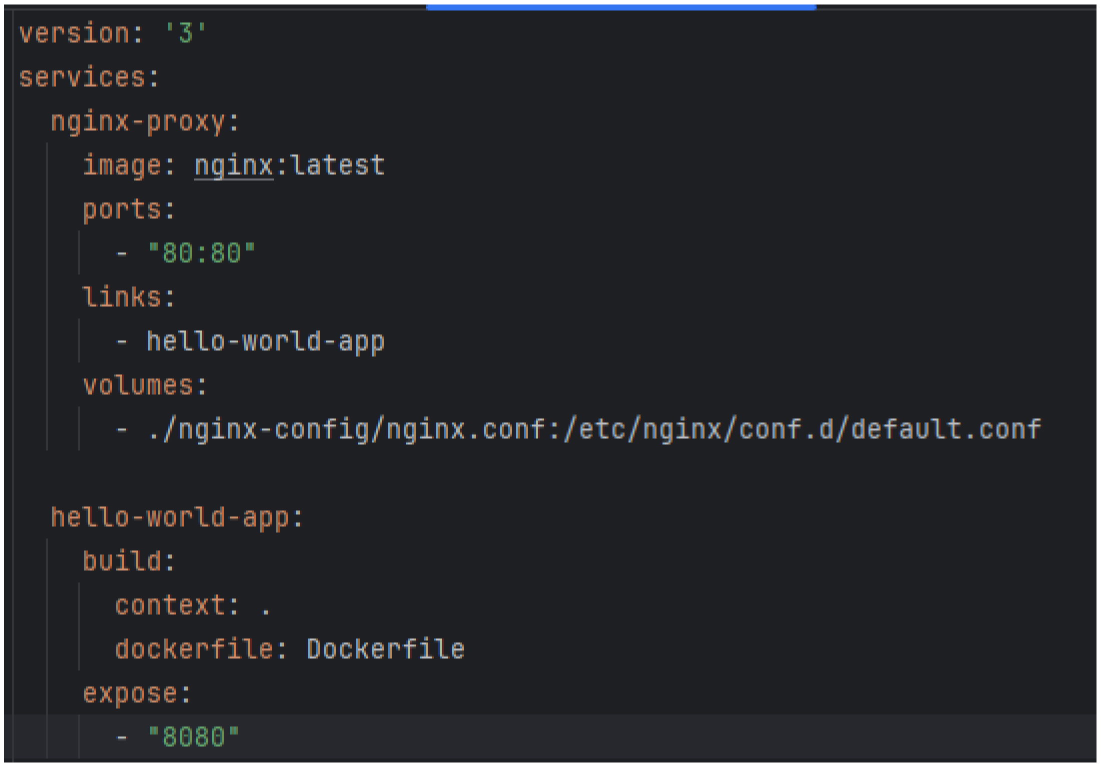
```
docker push  hellsbells998/trg
```

so that the Docker image is pushed to the docker hub.

In order for other users to pull my work, they just need to run the **docker pull** command.

- ## Create a docker compose file that will contain the image you build and an nginx proxy

Firstly, we need to create a docker-compose.yml file in the same directory with the script.



```yaml
version: '3'
services:
  nginx-proxy:
    image: nginx:latest
    ports:
      - "80:80"
    links:
      - hello-world-app
    volumes:
      - ./nginx-config/nginx.conf:/etc/nginx/conf.d/default.conf

  hello-world-app:
    build:
      context: .
      dockerfile: Dockerfile
    expose:
      - "8080"
```

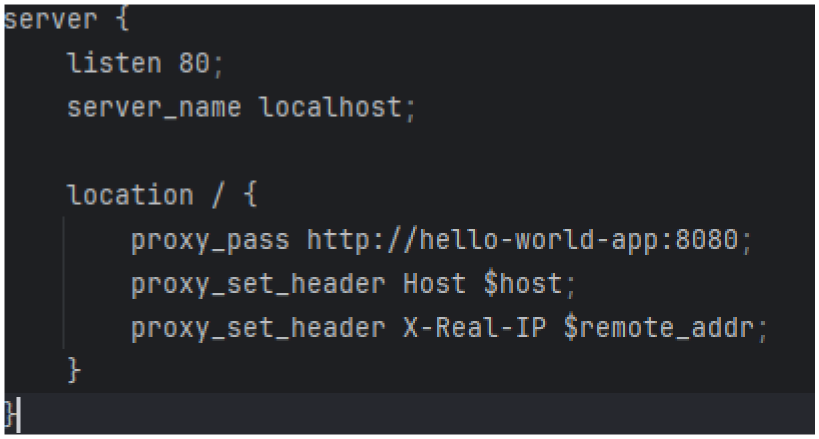
Here we define 2 services. The volume's part will be explained later.

The nginx-proxy service is using the official Nginx image and it exposes the port 80 for public access.

The hello-world-app build the docker image from the Dockerfile and exposed port 8080, but it is not publicly accessible.

By default, the Nginx container is publicly accessible on port 80, and it can proxy requests to the Flask application. In order to configure Nginx, we need a config file.

First, we need to create a directory named nginx-config and there we will place the nginx configuration file that forwards the requests to the Flask application.



```
server {
    listen 80;
    server_name localhost;

    location / {
        proxy_pass http://hello-world-app:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

So, in the docker-compose.yml we must add a volume definition, in order for the config file to be located.

Then; in order to start the docker compose configuration, we need to run the following command:

docker-compose up

Now, Nginx will proxy requests to my Flask application, making it publicly accessible while keeping the Flask application itself not.