Slightly modified from https://homework.adhoc.team/containerize/

Containerize a Python application

You are part of a multi-agency team team who is developing a Python web application. Lately, your peers have been complaining about Python version differences causing the app to behave differently or not work at all. The problems are not consistent across all team members and you see "Works on my machine!" in Teams a lot more now.

The first "production" launch of the app is a low traffic private beta that is scheduled for next week. To support the launch, you need to build a production ready setup. Since the application is still in heavy development you don't want to make the version differences worse by adding yet another environment. You therefore decide to use Docker to build a setup that can be used for development and production.

The Task

Using Docker and docker-compose, containerize a Python application fronted by an nginx reverse proxy and show us your understanding about how to deploy an application securely while still allowing for low friction development.

Specifically we will be looking for the following in your submission:

- Secure and performant settings in your nginx configuration, including SSL/TLS specifics
- Usage of the principle of least privilege/surprise
- Docker best practices including security, configuration, and image size

Part One

Complete the provided `nginx/nginx.conf` by writing a `server` directive(s) that proxies to the upstream application.

Requirements:

- Nginx should accept requests on ports 80 and 443
- All HTTP requests should permanently redirect to their HTTPS equivalent
- Use the provided SSL keypair found in `nginx/files/localhost.crt` and `nginx/files/localhost.key` for your SSL configuration
- Ensure the SSL keypair is available to nginx, but is not baked in to the container image
- The SSL configuration should use modern and secure protocols and ciphers
- Nginx should proxy requests to the application using an `upstream` directive

- Pass headers `X-Forwarded-For`, `X-Real-IP`, and `X-Forwarded-Proto` to the upstream application with appropriate values

Part Two

Complete `app/Dockerfile`, `nginx/Dockerfile` and `docker-compose.yml` to produce a production ready image that can be also be used for development with Compose.

Requirements:

- The app found in `./app/src` is securely built, installed, and configured into a container.
- When run by itself, the app container should start the app, serving traffic with a production quality server, on port `8000` without any extra configuration.
- Running `docker-compose up` should:
    - Start the app container in development mode listening on port `8000`
    - Allow local edits of the app source to be reflected in the running app container without restart (eg: hot code reload)
    - Start an nginx container configured with the files from Part One

    - The `app` service should not reachable directly from the host and can only be accessed through the `nginx` service.

Tying it all together

After running `docker-compose up` a command such as `curl -k https://localhost/` should return output similar to:

```
It's easier to ask forgiveness than it is to get permission.
X-Forwarded-For: 172.20.0.1
X-Real-IP: 172.20.0.1
X-Forwarded-Proto: https
```

You may also run the included validation tool to further test your work. This script will stop and remove running and built containers, so *please be careful.* You can run the tool like so:

```
# Stop, remove, rebuild containers, and run tests (see -h for help)
./validate.sh
```

Tips & Guidance

Tips

- Windows 10 Home users will need to use Docker toolbox or will need to do the exercise in a Linux VM / cloud instance since Docker Desktop for Windows requires Hyper-V, a Pro only feature.
- The Python app is a Flask app and can be configured with environment variables

- You can use gunicorn for the production server service. Since we're not testing a full production config in this exercise, you don't need to tune it. Simply show that the app image boots and serves traffic using a real server. If you are more familiar with a different standalone WSGI server, feel free to substitute it and discuss the change in your `COMMENTS.md` file.

Dos

- Do add notes on running your solution and why you choose your particular solution in a `COMMENTS.md` file. Remember, you are working with multiple teams across departments. Written communication is important!
- Do feel free to offer suggestions or feedback on this exercise

Do Nots

- Do not worry about data persistence, scaling, or OCSP stapling
- Do not alter `validate.sh` or the SSL key pair in `nginx/files`.
- Do not modify the `app` and `nginx` service names in `docker-compose.yml`.
- Do not modify the `app` and `nginx` container names in `docker-compose.yml`.
- Do not create any additional Dockerfile or docker-compose files.

Included files

The files for this assessment are in BackEndAndDevOps.zip