Webapp + Nginx and SSL in Docker Compose

A simple example in Python+Flask that can be easily adapted to different contexts.

Photo by Christina @ wocintechchat.com on Unsplash

This tutorial will explain how to create a simple web app, a proxy with Nginx, and wire them together with Docker Compose. The web app will be in Python + Flask, but it is easy to replace it with another technology.

The web app

The web app is nothing more than a simple example linking `/` to `hello()` to print “hello world!” when the user calls it.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "hello world!"
```

Docker

Docker is a virtualization technology that packages an application and its dependencies in a platform-independent *container*. I suggest installing Docker Desktop that includes all the necessary components.

A `Dockerfile` is a set of instructions that Docker uses to build an `image`, and an image is a read-only package that Docker uses to create a container.

This is the Dockerfile for this tutorial:

```
FROM python:3.9
COPY setup.py /home/
COPY app/* /home/
WORKDIR /home
RUN pip3 install -e .
```

In a nutshell, all Dockerfiles start from a base image (python 3.9 in this case), copy some files, and run one or more commands to set up and start the main process. `setup.py` contains the dependencies of the python application and `pip3` performs the setup. The last line starts Flask.

Nginx

Nginx is a web server that is often used as a load balancer or proxy. In this case, it will be the HTTPS-enabled proxy that will encrypt the communications with the clients.

```
server {
    listen                  443 ssl;
    listen                  [::]:443 ssl;
    server_name             localhost;
    ssl_certificate         /root/ssl/cert.pem;
    ssl_certificate_key     /root/ssl/key.pem;

    location / {
        proxy_pass "http://app:5000/";
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
```

This configuration enables SSL and configures the forwarding of `/` to `http://app:5000` . The certificates and the url `http://app` will be defined later.

Docker Compose

Docker Compose is a tool to define multi-container applications. In this case, it will connect Nginx and Flask in a virtual network exposing only the port of the proxy.

```
version: "3.9"
services:
  app:
    image: my_app:latest
  nginx:
    image: nginx
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
      - ./key.pem:/root/ssl/key.pem
      - ./cert.pem:/root/ssl/cert.pem
```

This configuration creates two containers, `app` and `nginx` . `app` is the image of the `flask` microservice and `nginx` is an instance of Nginx with three volumes to map the configuration mentioned above, and the cert/key for the encryption. Please note that `app` defined this way will be reachable with `http://app` within the virtual network.

Run.sh

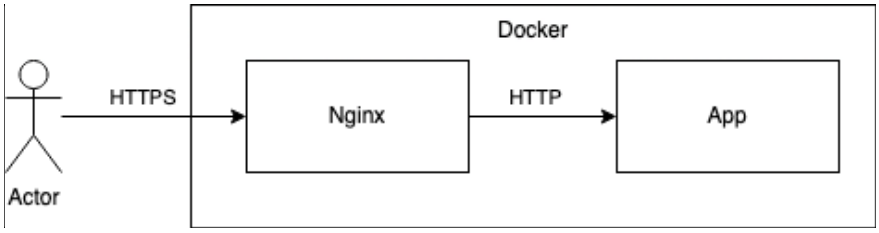
The only parts missing are the TLS certificate and the build of the Docker image. This can be automated with a script that also starts the application.

```
#!/bin/sh

openssl req -x509 -nodes -newkey rsa:2048 -keyout key.p
    -subj "/C=GB/ST=London/L=London/O=Alros/OU=IT Depa

docker build . -t my_app
```

The first line generates a self-signed certificate. This can be replaced by a static couple certificate/key. The second line builds the docker image described in `Dockerfile` and the last line starts the Docker Compose described in `docker-compose.yml` .



The complete system

The diagram above describes the system. The user connects on `https://localhost` reaching Nginx, Nginx offloads the encryption and forwards the unencrypted call to the Flask microservice.

Test

Opening https://localhost may give errors on most of browsers since the certificate is self signed.

In alternative `curl --insecure https://localhost` will simply return “hello world!”. The verbose output shows the usage of encryption.

```
$ curl --insecure https://localhost -v
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/cert.pem
  CApath: none
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: C=GB; ST=London; L=London; O=Alros; OU=IT
Department; CN=localhost
*  start date: Mar 13 18:51:40 2022 GMT
*  expire date: Mar 13 18:51:40 2023 GMT
*  issuer: C=GB; ST=London; L=London; O=Alros; OU=IT
Department; CN=localhost
*  SSL certificate verify result: self signed certificate
(18), continuing anyway.
> GET / HTTP/1.1
> Host: localhost
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.21.6
< Date: Sun, 13 Mar 2022 18:53:47 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 12
< Connection: keep-alive
<
* Connection #0 to host localhost left intact
hello world!* Closing connection 0
```

Code

The full code is available on GitHub.