Smart Cow - React Flask NGINX Dockerize App Guide

Task #1

https://github.com/ryancomia/sc-exercise/tree/dev

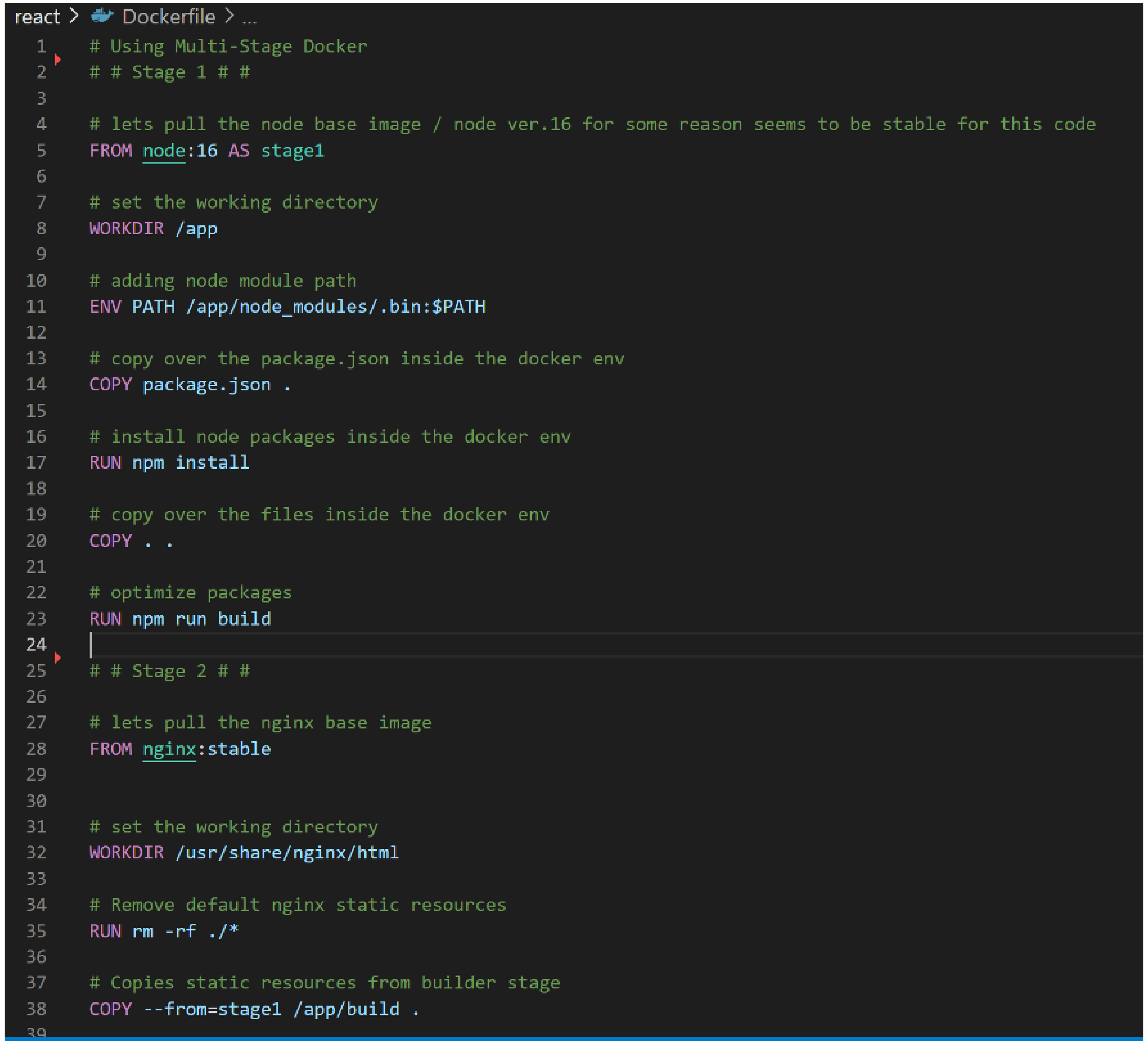
Containerize React (using multi stage docker for this build)

Stage 1
- run the image of Node version
- Initialize the react app folder
- Copy the package.json requirements
- Run npm install
- Move the files into the container
- Run npm build. (to optimize the build files)

Stage 2

- Run image of NGINX
- Set the working directory where the files would be moved
- Clear the working directory
- Copy over the app from stage 1

Note: Multi Docker build produces less build artefact (size) which takes less more time.


react > Dockerfile > ...

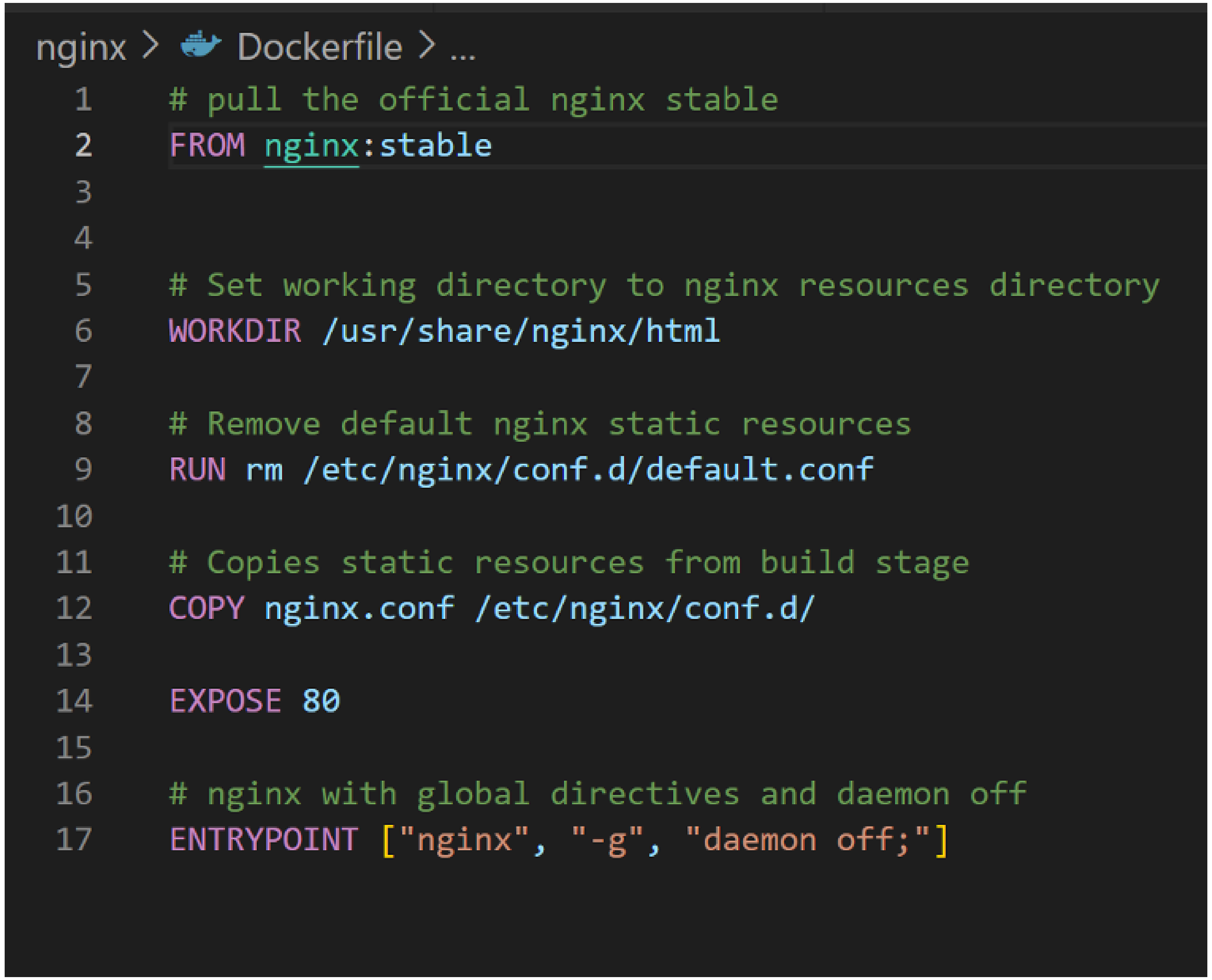
```dockerfile
# Using Multi-Stage Docker
# # Stage 1 # #

# lets pull the node base image / node ver.16 for some reason seems to be stable for this code
FROM node:16 AS stage1

# set the working directory
WORKDIR /app

# adding node module path
ENV PATH /app/node_modules/.bin:$PATH

# copy over the package.json inside the docker env
COPY package.json .

# install node packages inside the docker env
RUN npm install

# copy over the files inside the docker env
COPY . .

# optimize packages
RUN npm run build

# # Stage 2 # #

# lets pull the nginx base image
FROM nginx:stable


# set the working directory
WORKDIR /usr/share/nginx/html

# Remove default nginx static resources
RUN rm -rf ./*

# Copies static resources from builder stage
COPY --from=stage1 /app/build .
```

Containerize NGINX

This is straight forward, we just need to pull the stable version of NGINX. Initialize the directory and copy the nginx config file

We then logically expose 80 (this will not do the actual expose but rather an identifier)

Finally define nginx entrypoint



nginx > Dockerfile > ...

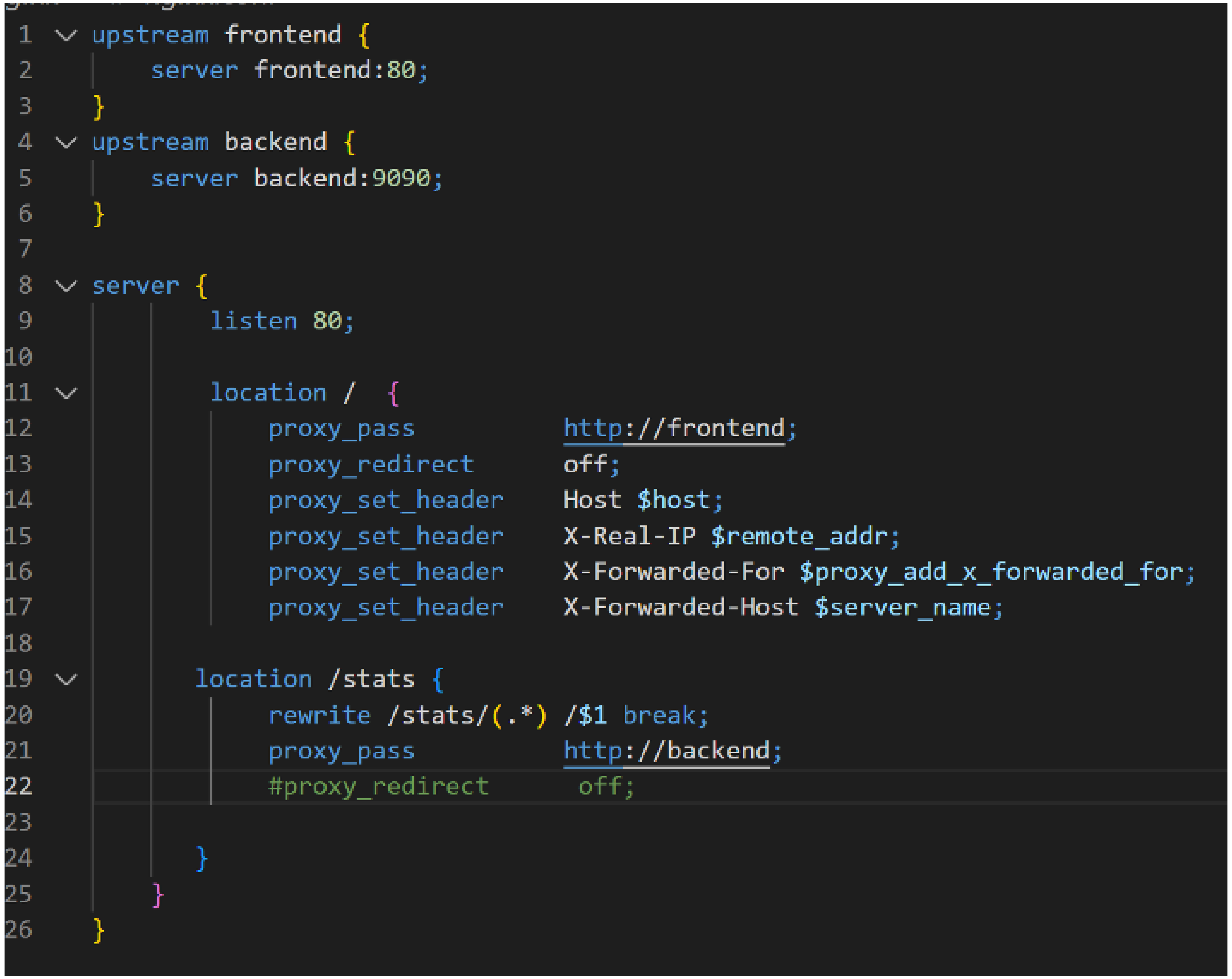
```dockerfile
# pull the official nginx stable
FROM nginx:stable


# Set working directory to nginx resources directory
WORKDIR /usr/share/nginx/html

# Remove default nginx static resources
RUN rm /etc/nginx/conf.d/default.conf

# Copies static resources from build stage
COPY nginx.conf /etc/nginx/conf.d/

EXPOSE 80

# nginx with global directives and daemon off
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

NGINX config

The reverse proxy will serve 2 purpose
1. Proxy the react frontend container
2. Proxy the backend api app

So for that I just defined the upstream connections and then define the listener proxy redirects

Simple =)

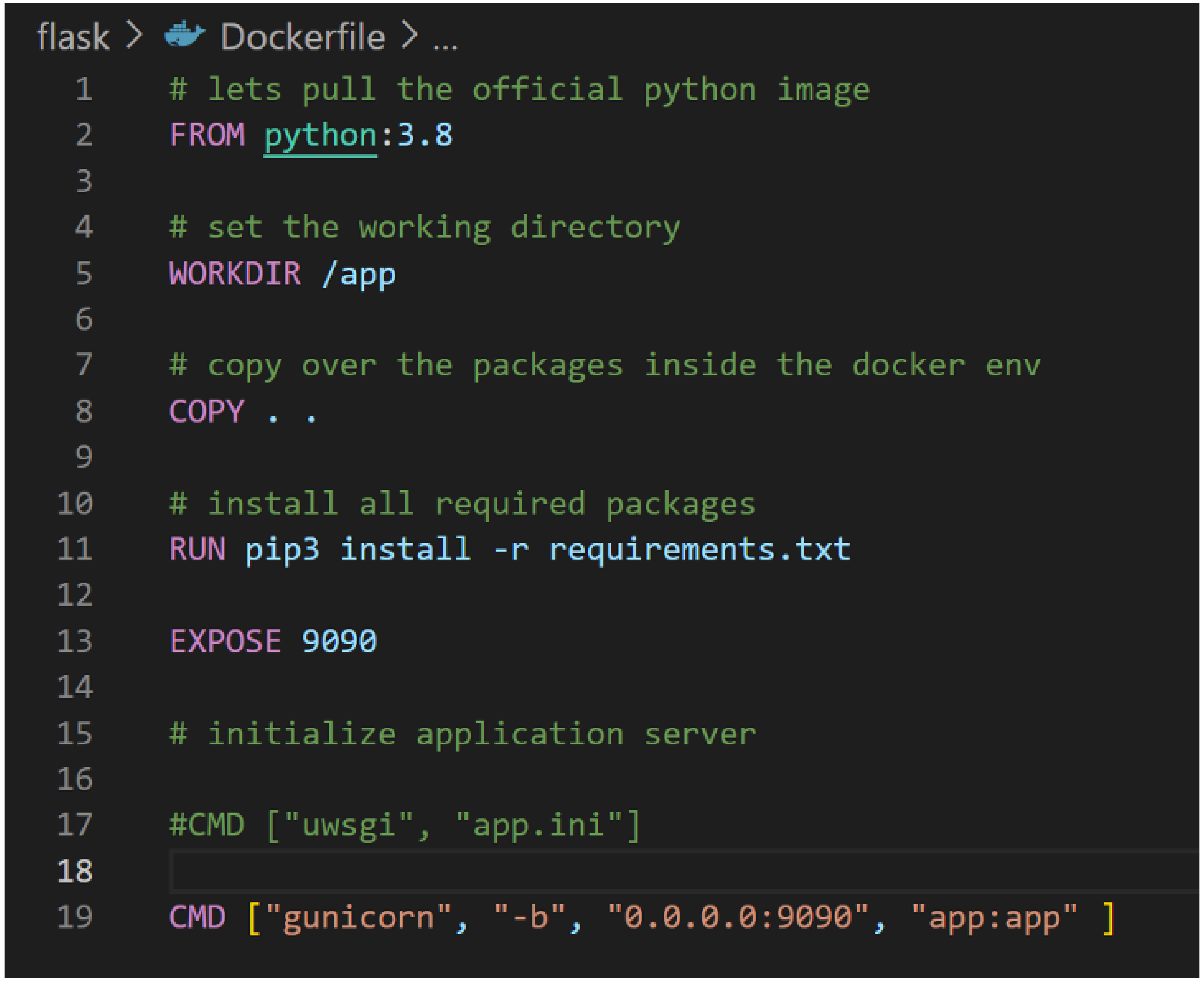


```
upstream frontend {
    server frontend:80;
}
upstream backend {
    server backend:9090;
}

server {
        listen 80;

        location /  {
            proxy_pass          http://frontend;
            proxy_redirect      off;
            proxy_set_header    Host $host;
            proxy_set_header    X-Real-IP $remote_addr;
            proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header    X-Forwarded-Host $server_name;

       location /stats {
            rewrite /stats/(.*) /$1 break;
            proxy_pass          http://backend;
            #proxy_redirect      off;

       }
    }
}
```

Containerize Flask

For this we use a stable version of python image

- Initialize the work path
- Copy the working files
- Run the python install requirements
- Logical expose of port 9090 (this will not do the actual expose)
- Starts the application service which runs on gunicorn



flask > Dockerfile > ...

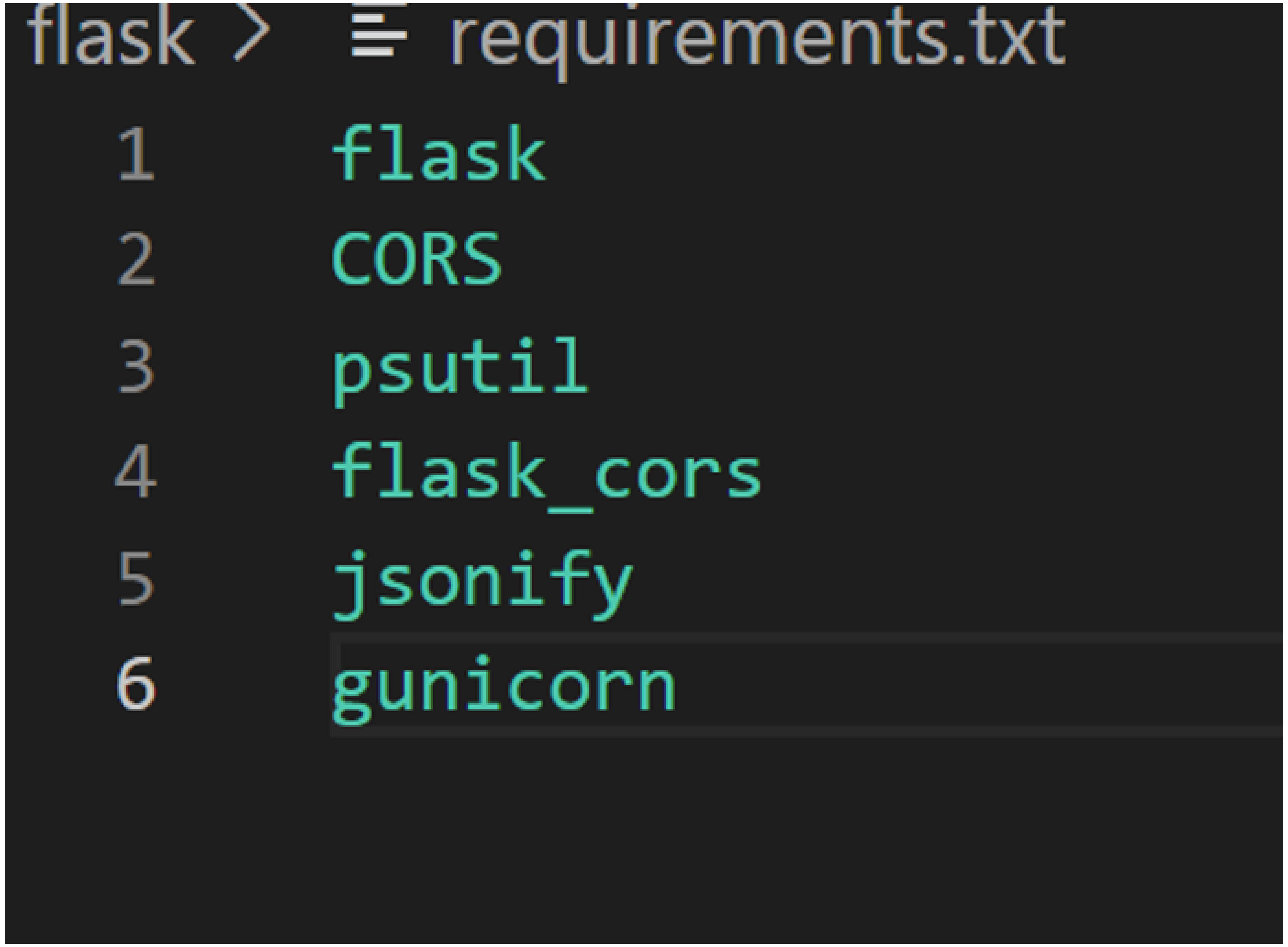
```dockerfile
# lets pull the official python image
FROM python:3.8

# set the working directory
WORKDIR /app

# copy over the packages inside the docker env
COPY . .

# install all required packages
RUN pip3 install -r requirements.txt

EXPOSE 9090

# initialize application server

#CMD ["uwsgi", "app.ini"]

CMD ["gunicorn", "-b", "0.0.0.0:9090", "app:app" ]
```

Requirement.txt

I then defined the modules required for the app. As well as the Gunicorn web app server

Note: I also tested uwgi (which explains why I have an app.ini file)

They both worked and behaved the same otherwise


flask > requirements.txt

```
flask
CORS
psutil
flask_cors
jsonify
gunicorn
```

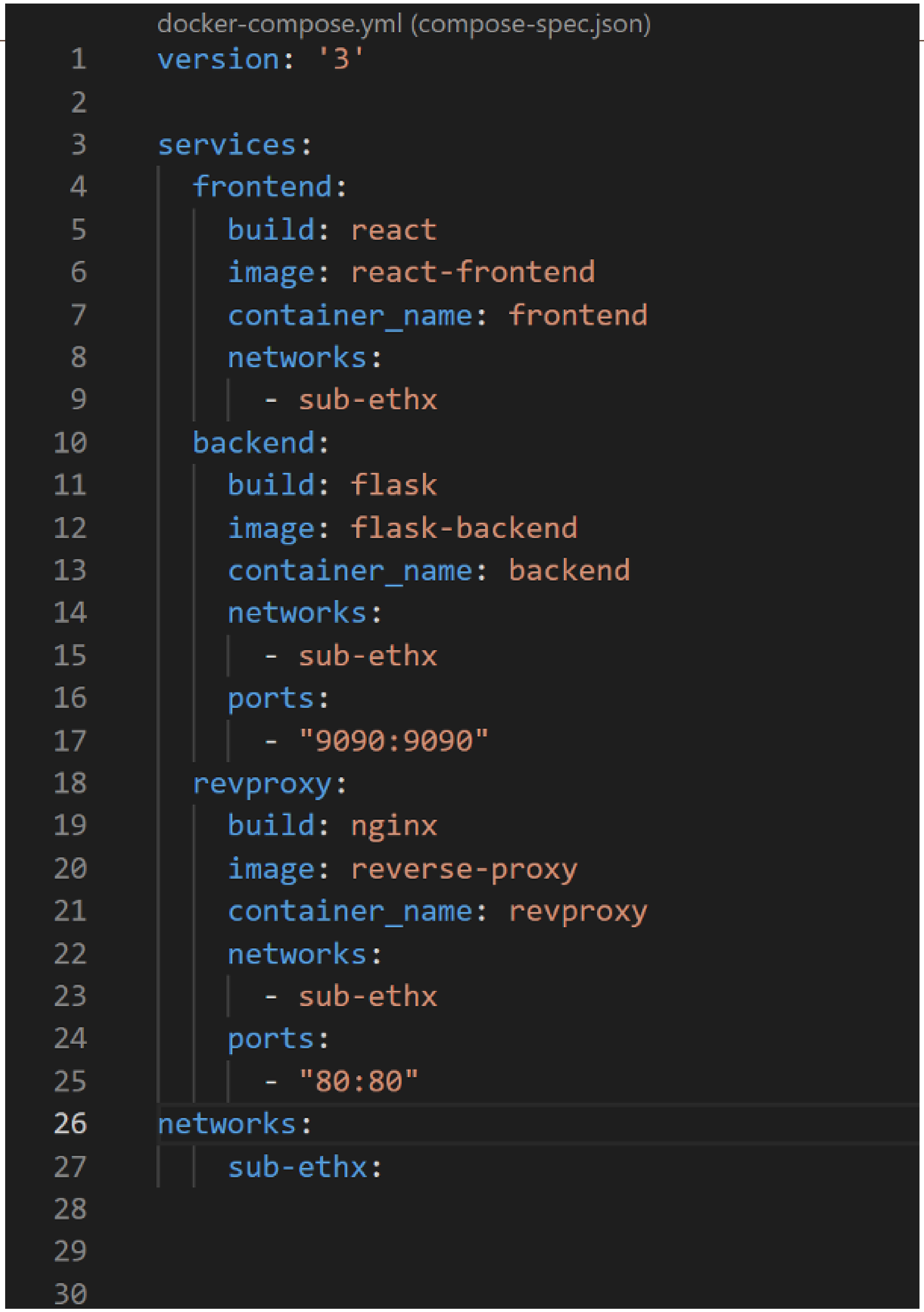
Docker Compose file

I did setup a very standard compose file which defines the docker network as sub-ethx

I then open the ports required to communicate inside the docker network to external

- backend to listen to 9090

-Reverse-proxy to listen to 80



docker-compose.yml (compose-spec.json)

```yaml
version: '3'

services:
  frontend:
    build: react
    image: react-frontend
    container_name: frontend
    networks:
      - sub-ethx
  backend:
    build: flask
    image: flask-backend
    container_name: backend
    networks:
      - sub-ethx
    ports:
      - "9090:9090"
  revproxy:
    build: nginx
    image: reverse-proxy
    container_name: revproxy
    networks:
      - sub-ethx
    ports:
      - "80:80"
networks:
    sub-ethx:
```