Name: Ziad Mansour Mohamed Hassanin

ID: 1180512

Final Report – (CMPN203)

Task	Status
Pipelines:	✓
FrontEnd	✓
BackEnd	✓
Load Balancer	✓
Server Configuration	✓
Mailing	✓
Dockerization	✓
Monitoring Script ***Not Used***	✓

Pipelines

BackEnd Pipeline:

The Pipeline consist of two jobs "Test – Deploy":
Test:

- Clone Code on runner
- Run Tests

Deploy "Needs Test"**:**

- ➢ SSH to server
- ➢ Clone code on Server
- ➢ Make necessary configuration
- ➢ Reboot server

FrontEnd Pipeline:

The Pipeline consist of two jobs "Test – Build and Deploy":

Test:

- Clone Code on runner
- Run Tests

Build and Deploy "Needs Test"**:**

- ➢ Clone code on runner
- ➢ Build Code
- ➢ SSH to server
- ➢ Copy builds files from runner to server
- ➢ Make necessary configuration
- ➢ Reboot server

Load Balancer

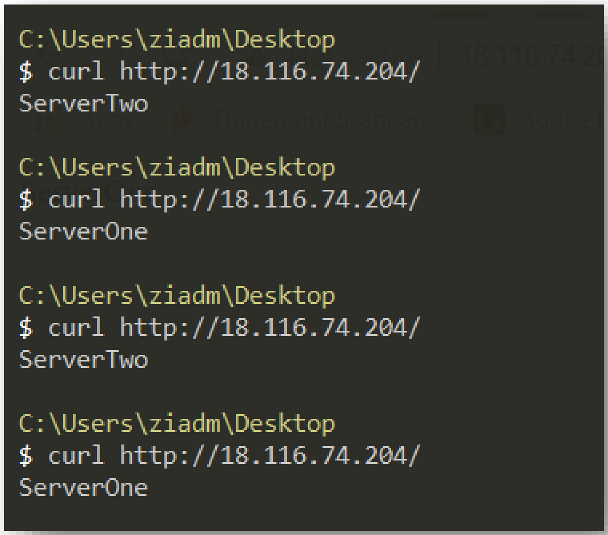
For demonstration purposes and considering the current context I choose:

- To Use only one Load Balancer and Two Servers for simplicity
- Spin only one AWS EC2-Ubuntu Instance
- Use ports 81 and 82 for the two servers
- Use port 80 for our load balancer

Steps:

- ✓ Install Nginx
- ✓ Set Up New Document Root Directories
- ✓ Create Sample Pages for Each Site
- ✓ Create Server Block Files for Each Port
- ✓ Configure Load Balancer node
- ✓ Enable, Validate, Reload
- ✓ Check Results

Results:



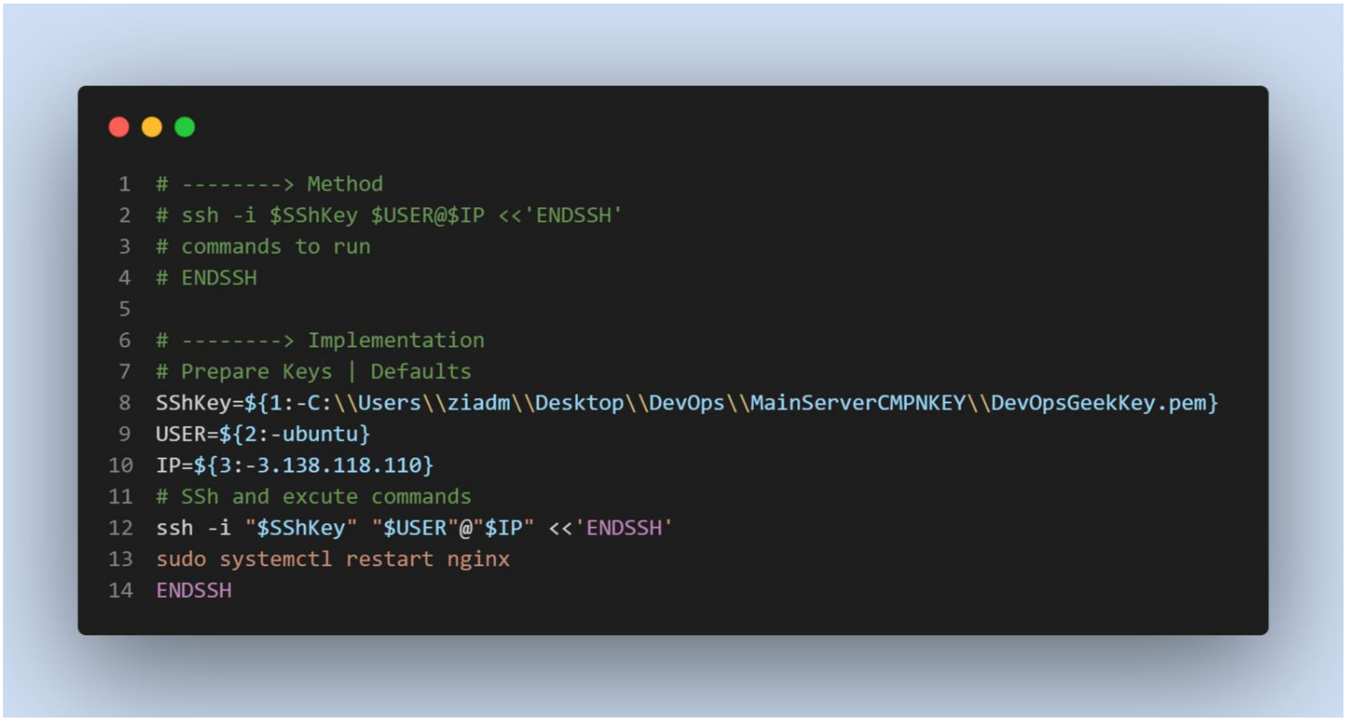
```
C:\Users\ziadm\Desktop
$ curl http://18.116.74.204/
ServerTwo

C:\Users\ziadm\Desktop
$ curl http://18.116.74.204/
ServerOne

C:\Users\ziadm\Desktop
$ curl http://18.116.74.204/
ServerTwo

C:\Users\ziadm\Desktop
$ curl http://18.116.74.204/
ServerOne
```

Server Configuration

Our goal is to automate the initial configuration of a new server so we needed to install the following packages:

1- Nginx
2- pm2
3- Node.js
4- Npm

We needed a method to help us access the server and run some commands, after some good search. I found the following, the most convenient for our case:



```
# --------> Method
# ssh -i $SShKey $USER@$IP <<'ENDSSH'
# commands to run
# ENDSSH

# --------> Implementation
# Prepare Keys | Defaults
SShKey=${1:-C:\\Users\\ziadm\\Desktop\\DevOps\\MainServerCMPNKEY\\DevOpsGeekKey.pem}
USER=${2:-ubuntu}
IP=${3:-3.138.118.110}
# SSh and excute commands
ssh -i "$SShKey" "$USER"@"$IP" <<'ENDSSH'
sudo systemctl restart nginx
ENDSSH
```

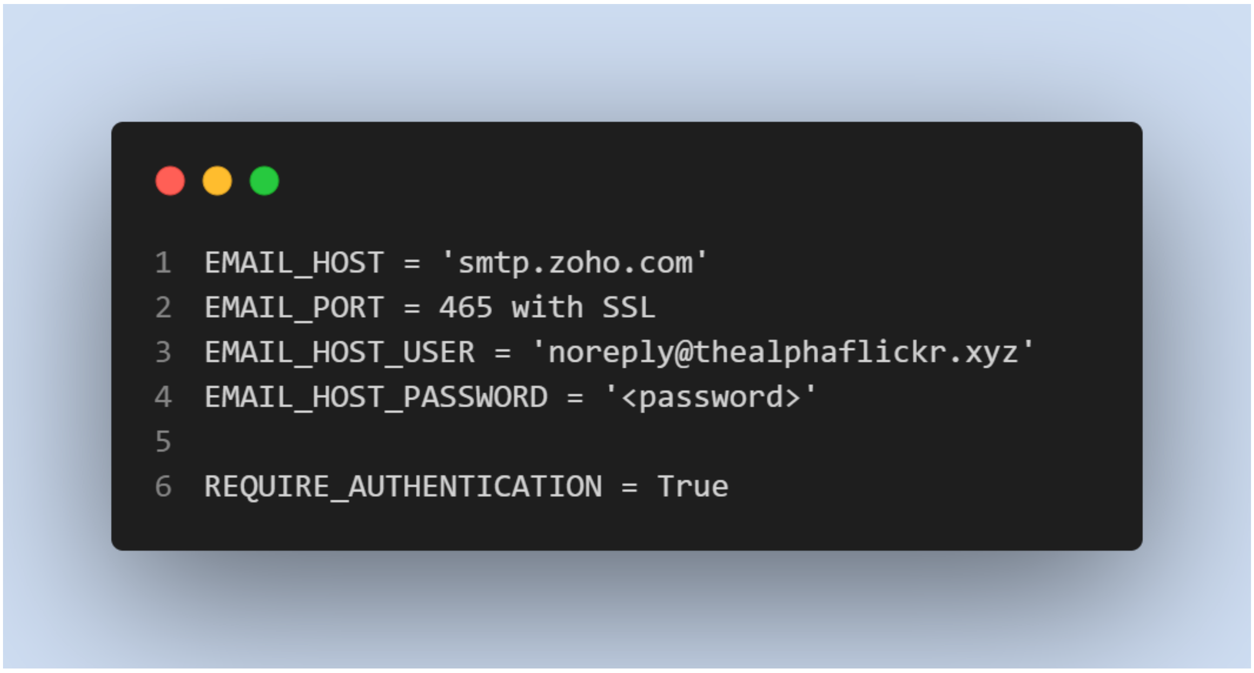
It allowed us to customize it and pass our own parameters or use the default, adding to this you can pipe as many commands as you would like to add.

Mailing

At the begging I was planning to host a complete postman mail server on an ubuntu server, but I struggled building it so in return we decided to use **Zoho** as we found that it suits our case, as we had great features to enjoy and an amazing free trial. We created some emails like:

- noreply@thealphaflickr.xyz
- admin@thealphaflickr.xyz

The following configuration were enough to start up and running:



```
EMAIL_HOST = 'smtp.zoho.com'
EMAIL_PORT = 465 with SSL
EMAIL_HOST_USER = 'noreply@thealphaflickr.xyz'
EMAIL_HOST_PASSWORD = '<password>'

REQUIRE_AUTHENTICATION = True
```

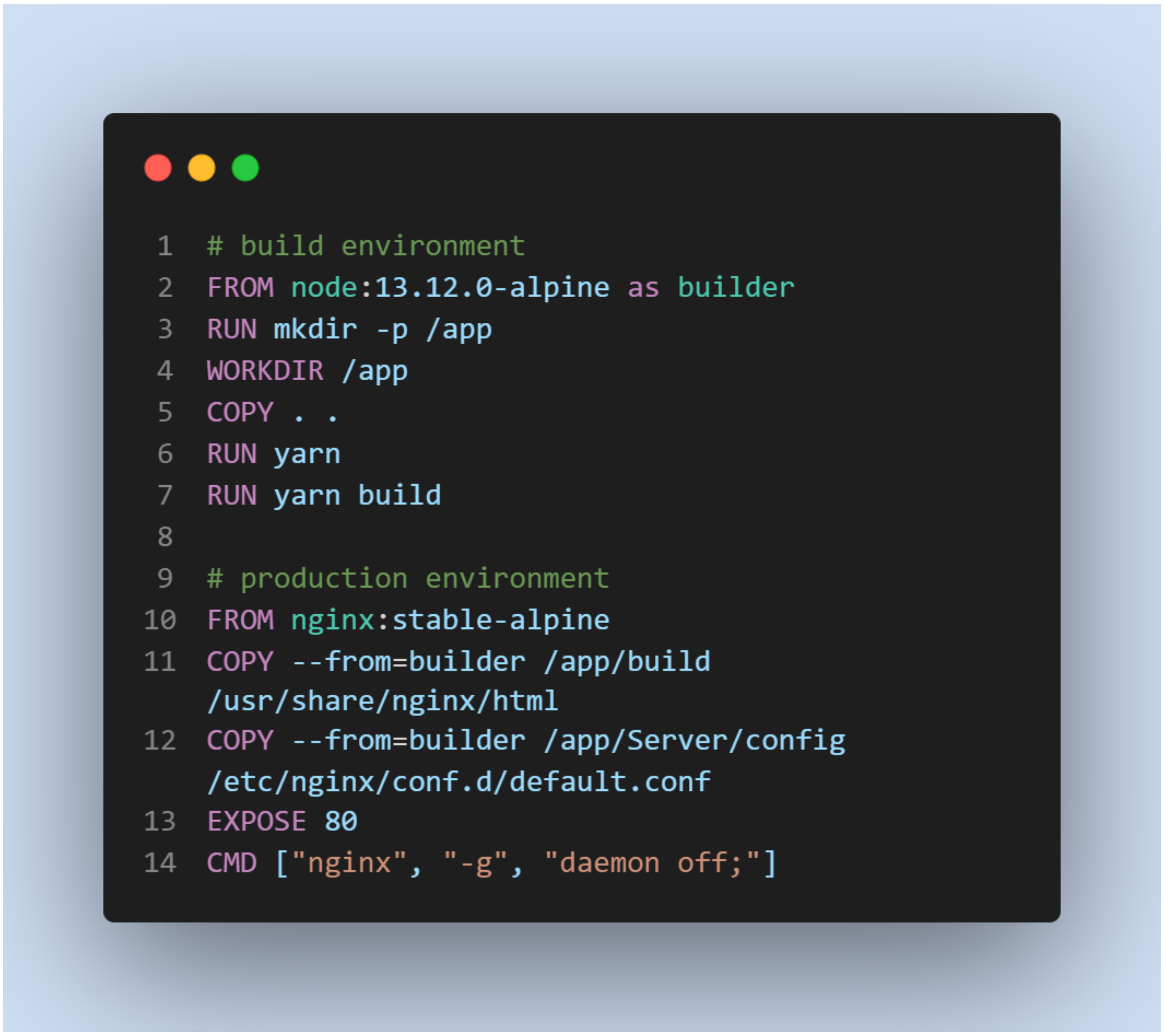
Dockerization

FrontEnd Image:

For the frontend image we needed to:

1- have only Nginx installed and a copy of the build files that is planned to be served. No need for the whole code base.
2- Provide some more level of security, so the application will have the least possible privileges to run the app properly. Which is given to a user called node
3- Light weight Linux image, no need for all the features that is usually exits in Linux, so I chose nginx:stable-alpine image.

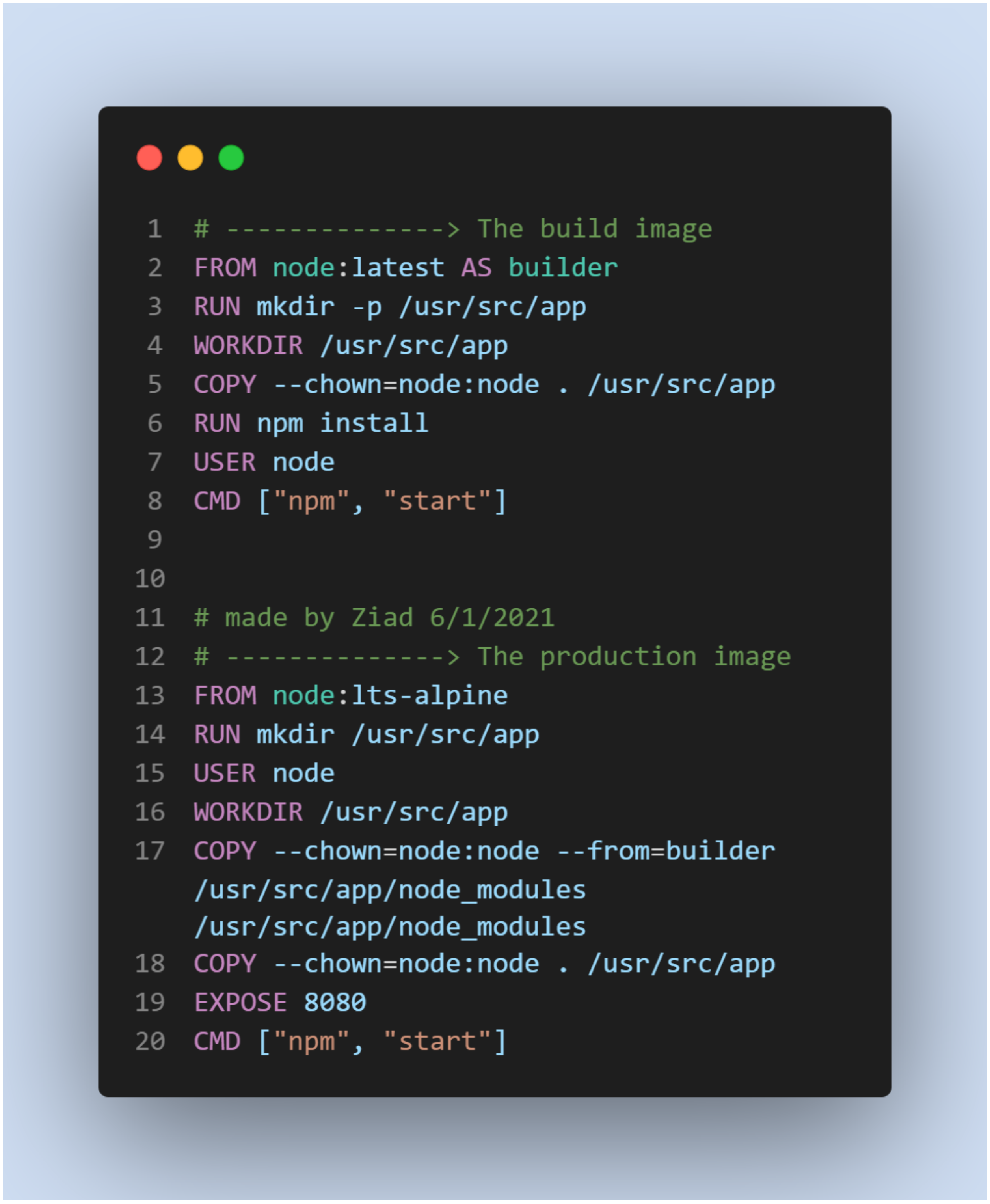
Decision: I decided to make the Dockerization in to steps the first one called builder and the second one for production, as recommended in this video.



```dockerfile
# build environment
FROM node:13.12.0-alpine as builder
RUN mkdir -p /app
WORKDIR /app
COPY . .
RUN yarn
RUN yarn build

# production environment
FROM nginx:stable-alpine
COPY --from=builder /app/build /usr/share/nginx/html
COPY --from=builder /app/Server/config /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Backend Image:

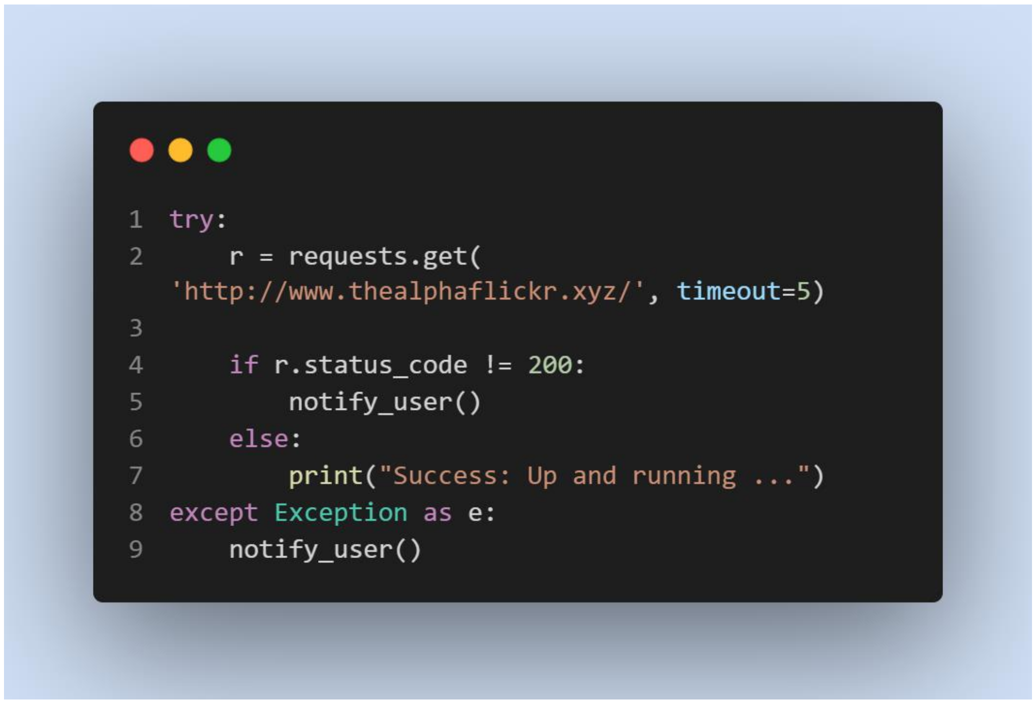
The backend image had exactly the same conditions as the frontend one but with minor tweaks



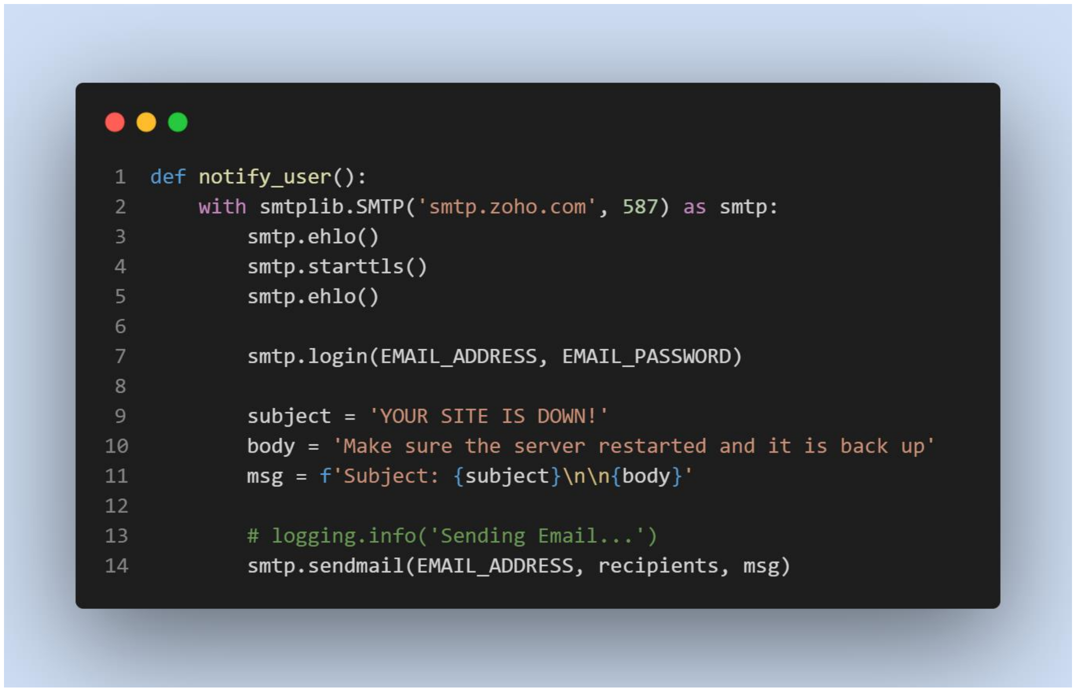
```dockerfile
# --------------> The build image
FROM node:latest AS builder
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY --chown=node:node . /usr/src/app
RUN npm install
USER node
CMD ["npm", "start"]


# made by Ziad 6/1/2021
# --------------> The production image
FROM node:lts-alpine
RUN mkdir /usr/src/app
USER node
WORKDIR /usr/src/app
COPY --chown=node:node --from=builder
/usr/src/app/node_modules
/usr/src/app/node_modules
COPY --chown=node:node . /usr/src/app
EXPOSE 8080
CMD ["npm", "start"]
```

Monitoring Script

I planned the monitoring script to be a GitHub action Workflow that is scheduled to trigger every 5 minutes, that will run a python file to hit the server and notify the user if anything happened.



```python
try:
    r = requests.get(
'http://www.thealphaflickr.xyz/', timeout=5)

    if r.status_code != 200:
        notify_user()
    else:
        print("Success: Up and running ...")
except Exception as e:
    notify_user()
```



```python
def notify_user():
    with smtplib.SMTP('smtp.zoho.com', 587) as smtp:
        smtp.ehlo()
        smtp.starttls()
        smtp.ehlo()

        smtp.login(EMAIL_ADDRESS, EMAIL_PASSWORD)

        subject = 'YOUR SITE IS DOWN!'
        body = 'Make sure the server restarted and it is back up'
        msg = f'Subject: {subject}\n\n{body}'

        # logging.info('Sending Email...')
        smtp.sendmail(EMAIL_ADDRESS, recipients, msg)
```