💡

LS Lab 1: Containerization and Application Layer Load Balancing

Name: Emeka Michael Nzeopara

▶ Details

To install the docker engine in my ubuntu machine we will run the following command.
We need to uninstall the previous old version of the docker engine, we can achieve this using thwe following command

```
* sudo apt-get remove docker docker-engine docker.io containerd runc
```

We will set up the docker engine by first, update the apt package index and install the packages to allow apt to use a repository over https: This can be done with the following command.

```
* sudo apt-get update
* sudo apt-get install ca-certificates curl gnupg lsb-release
```

Next is to add docker's official gpg key using the following command

```
* sudo mkdir -p /etc/apt/keyrings
* curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/ap
```

Then we will set up the repository with the following command

```
* echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https:/
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

We have successfully set-up the apt repository, we will update the package index using the command

* `sudo apt-get update`

Should incase there is an error; run the command

* `sudo chmod a+r /etc/apt/keyrings/docker.gpg`
* `sudo apt-get update`

To install the docker engine, containerd and docker Compose

* `sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin`

After this has been completed, I obtained an error showing the docker service is not running properly

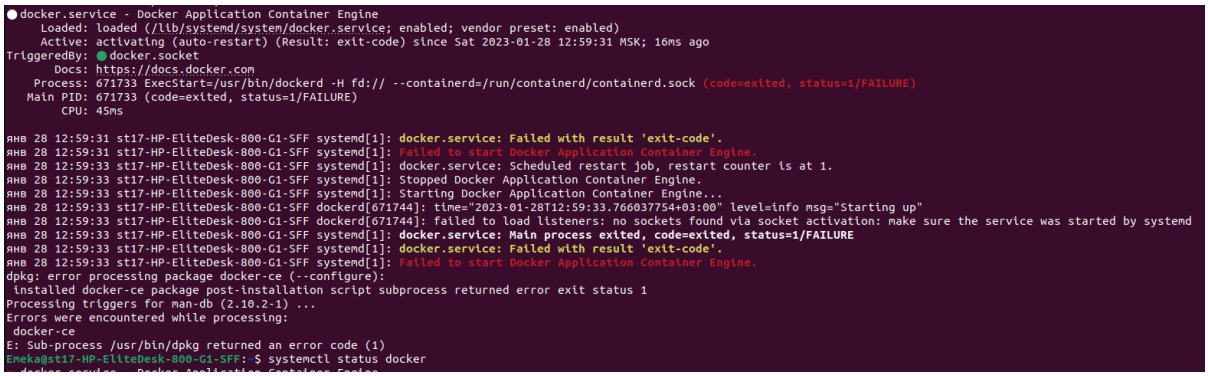

Figure 1: Error after the Installation

The way i went about resolving this problem is by runnung an upgrade command on the apt, and restarting the docker

```
* sudo apt upgrade
* systemctl restart docker
```

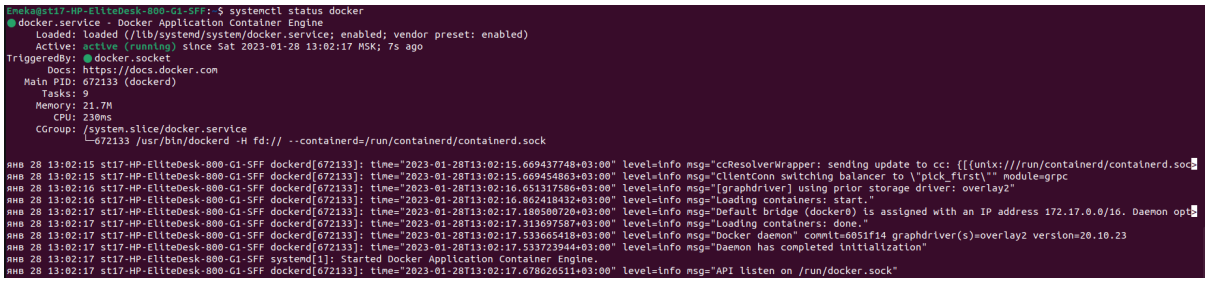

Figure 2: After correcting the Error

Task 1: Get familiar with Docker Engine

1. Pull Nginx v1.23.3 image from dockerhub registry and confirm it is listed in local images.

This can be done by running the following command

```
* docker pull nginx:1.23.3
```

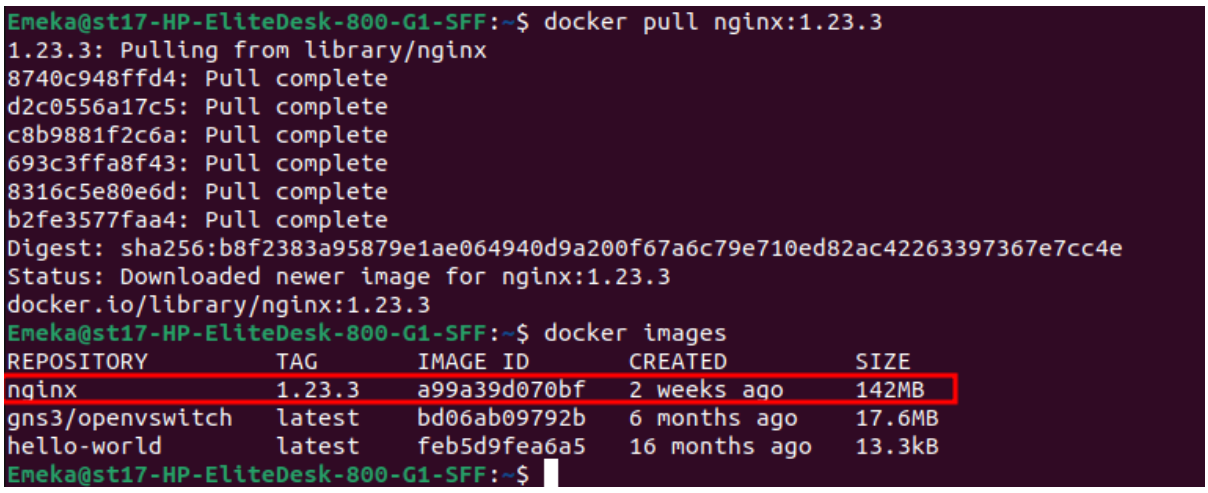

Figure 3: Pulling the Nginx Image

where the **docker pull** command will pull the specified nginx image and the **docker images** command will list the images in your local machine as shown in the picture above.

2. Run the pulled Nginx as a container with the below properties
   a) Map the port to 8080.
   b) Name the container as nginx-<stX>.
   c) Run it as daemon
   d) Access the page from your browser

To perform this task we will run the command as given thus, as given below

```
* docker run -d -p 8080:80 --name nginx-st17 nginx:1.23.3
```

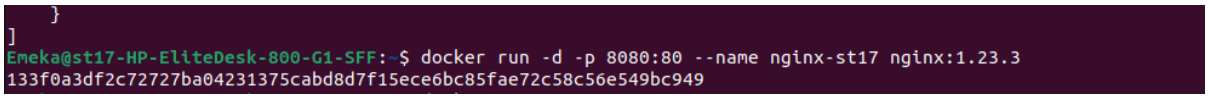

Figure 4: Nginx Command with the above properties

From the command given above they can be explained thus as:
'-d' flag is used to run the daemon in the background
'-p' is used to map port 8080 on the host to the port 80 on the container
'–name' this is used to name the image that has been created

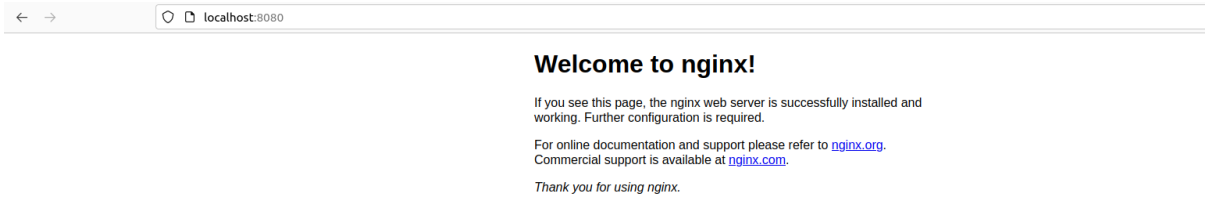

Figure 5: Accessing the Nginx page from my browser

3. Confirm port mapping.
   a) List open ports in host machine
   b) List open ports inside the running container

To run this command that will list these properties, we will run this command

```
* docker container inspect nginx-st17
```

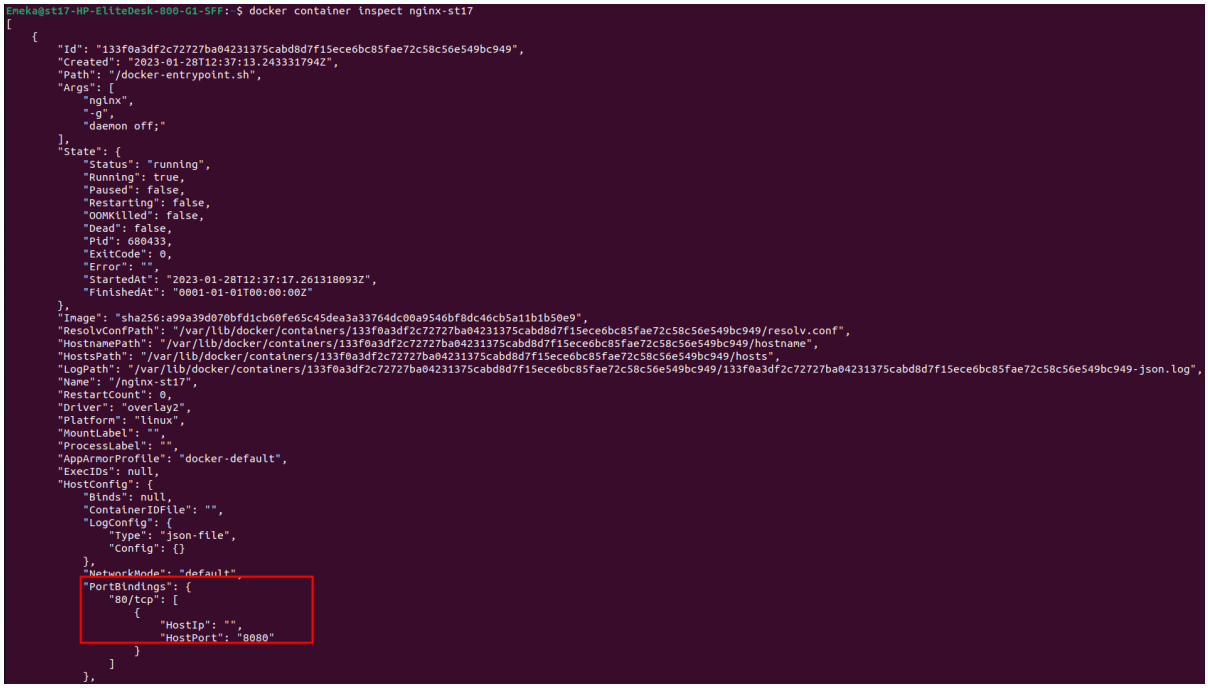

Figure 6: Inspect The Properties of the Nginx Container

The above command will output the properties of the container of Nginx, from the highlight part in the screenshot we can see that the parameters highlights both the container ports i.e. **80** to that of the host port i.e **8080**

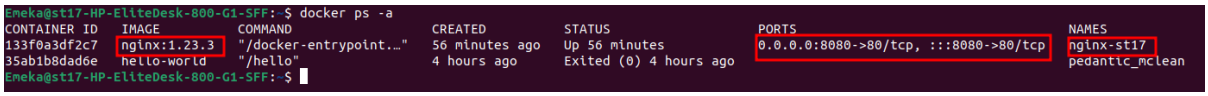

Figure 7: Alternative way to find out by running the ps command

```
* 'netstat -lntp' or 'ss -lntp' (this will show the opened port in the host)
* 'docker exec -it nginx-st17 netstat -lntp' or 'docker exec -it nginx-st17' ss -lntp
```

The command above is sort of a repitition of a shell command, the first one is the command to run when you are in a shell to obtain the open ports available where as the second row shows that we will enter the shell of the docker before the command to obtain the ports.

4. Create Dockerfile similar with below properties (let's call it container A).
   a. Image tag should be Nginx v1.23.3.
   b. Create a custom index.html file and copy it to your docker image to replace Nginx default web page.
   c. Build the image from the Dockerfile, tag it during build as nginx:<stX>, check/validate local images, and run your custom made docker image.
   d. Access via browser and validate your custom page is hosted

First I created a index.html file using the following command in a particular directory

* sudo mkdir docker && cd docker
* sudo nano index.html

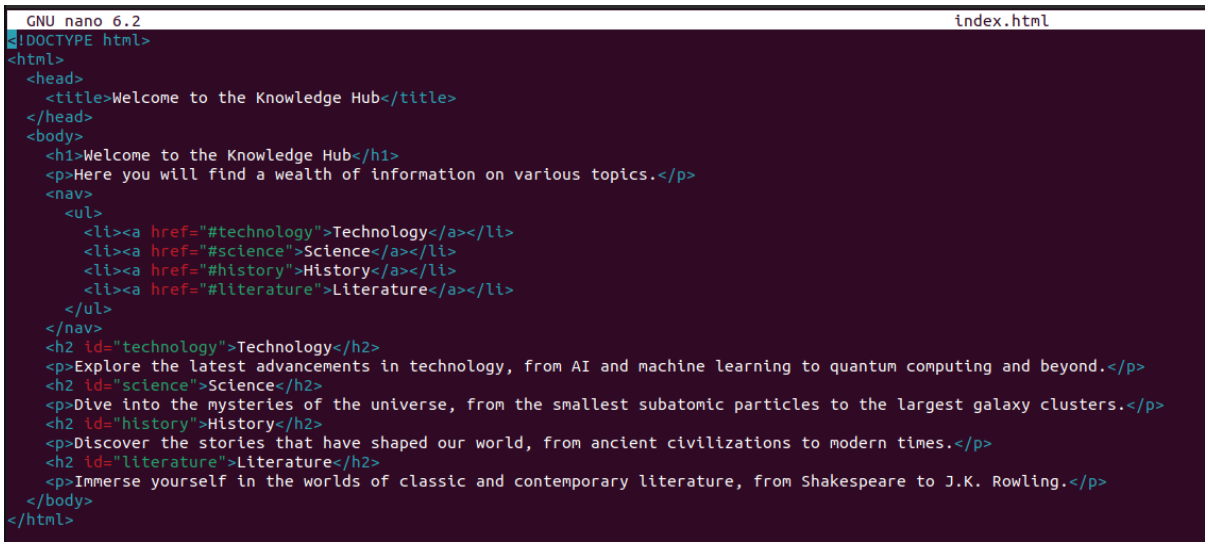

Figure 8: index.html file created

To create the Dockerfile using the above properties I created a Dockerfile as given thus

* sudo nano Dockerfile

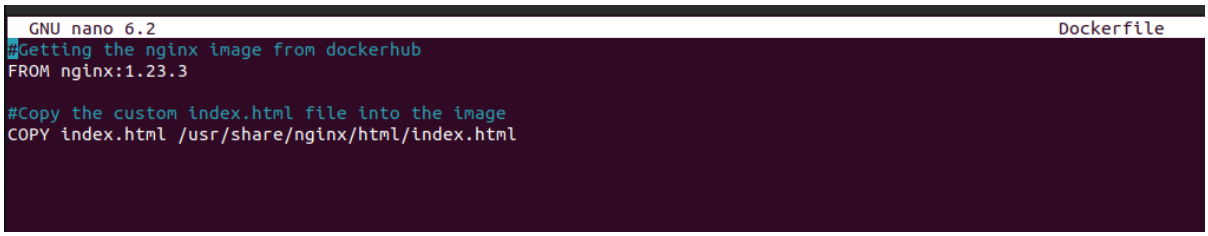

Figure 9: Dockerfile with the properties

From the Dockerfile created above the **FROM** command is used to pull the image nginx file from the dockerhub repository and in this case it is the *Nginxv1.23.3*
The **COPY** command is used to copy the custom *index.html* to that of the Nginx docker image

The next thing we would do is to build the image from the Dockerfile using the following command, running it from the directory that contains the Dockerfile.
We will also validate the command with the following command

* docker build -t nginx:stX .
* docker images

```
Emeka@st17-HP-EliteDesk-800-G1-SFF:~/docker$ docker build -t nginx:st17 .
Sending build context to Docker daemon  4.096kB
Step 1/2 : FROM nginx:1.23.3
 ---> a99a39d070bf
Step 2/2 : COPY index.html /usr/share/nginx/html/index.html
 ---> daf7f2207c67
Successfully built daf7f2207c67
Successfully tagged nginx:st17
Emeka@st17-HP-EliteDesk-800-G1-SFF:~/docker$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
nginx               st17      daf7f2207c67   2 minutes ago    142MB
nginx               1.23.3    a99a39d070bf   2 weeks ago      142MB
gns3/openvswitch    latest    bd06ab09792b   6 months ago     17.6MB
hello-world         latest    feb5d9fea6a5   16 months ago    13.3kB
Emeka@st17-HP-EliteDesk-800-G1-SFF:~/docker$
```

Figure 10: Building the image from the Dockerfile for container A

To run the custom image that has been created, we will run the following command

```
* docker run -p 8081:80 --name nginx-custom -d nginx:st17
```

```
Emeka@st17-HP-EliteDesk-800-G1-SFF:~/docker$ docker run -p 8081:80 --name nginx-custom -d nginx:st17
1ef35ff7a1d87ba410183492c7ae187a1e97134e567651929a2f93cccec2c91f
Emeka@st17-HP-EliteDesk-800-G1-SFF:~/docker$
```

Figure 11: Running the custom image

To access the custom image from the browser

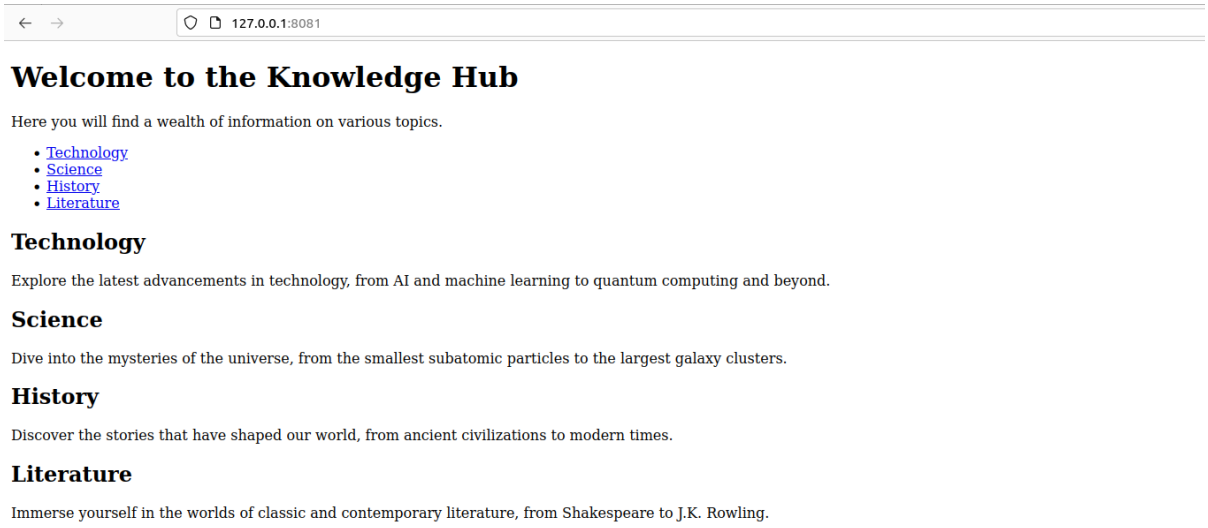

Figure 12: Accessing the custom image from Browser

Task 2: Work With Multi-Container Environment

1. Create another Dockerfile similar to step 1.4 (Let's call it container B), and an index.html with different content.

Following the same process in the previous task, the html command found is given as

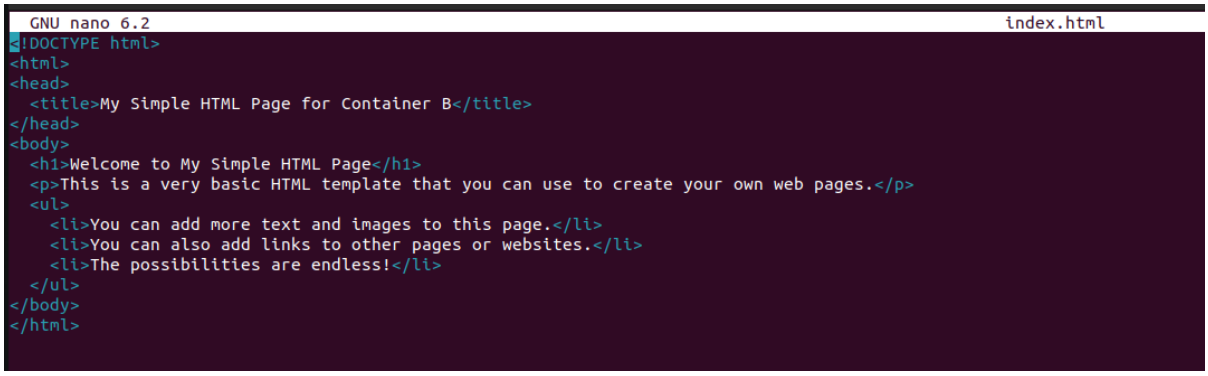

Figure 12: Index of html page for container B

The Dockerfile for container B is similar to that of container A, it is shown below as

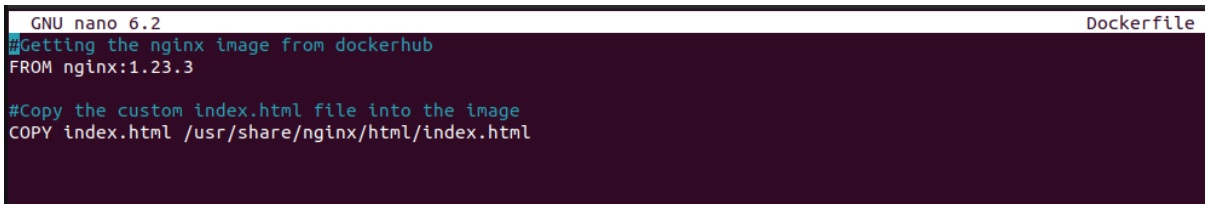

Figure 13: Dockerfile for container B

2. Write a docker-compose file with below properties
   a) Multi-build: Builds both Dockerfiles and run both images.
   b) Port mapping: Container A should listen to port 8080 and container B should listen to port 9090. (They host two different web pages)
   c) Volumes: Mount (bind) a directory from the host file system to Nginx containers to replace the default Nginx web page with the two index.html files created in Steps 1.4.b and 2.1 .

The docker compose file for the properties given above is thus:

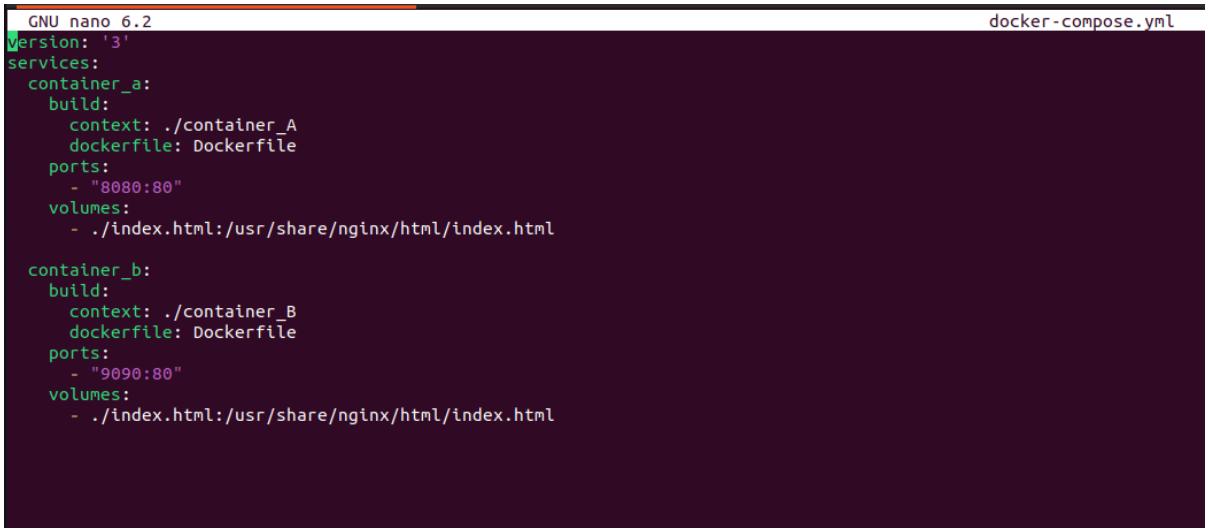

Figure 14: docker-compose file

The above file states the docker-compose file with the properties stated in this task.
The first 'container' is used to describe the services; the 'build' indicates the building section of the docker, the 'context' indicates the directory to find the Dockerfile used to build that docker, and the 'ports and voulume' indicates the two ports to access each of the service from the web browser where the latter indicates the directory from the host file to replace the Nginx containers default page.

3. Run the docker compose file and validate you have access to both Nginx web pages in your browser via their respective ports.

Firstly, I noticed that there is no docker-compose package to run the command in my ubuntu machine. So I need to install the docker-compose using the command:

* `sudo apt install docker-compose`

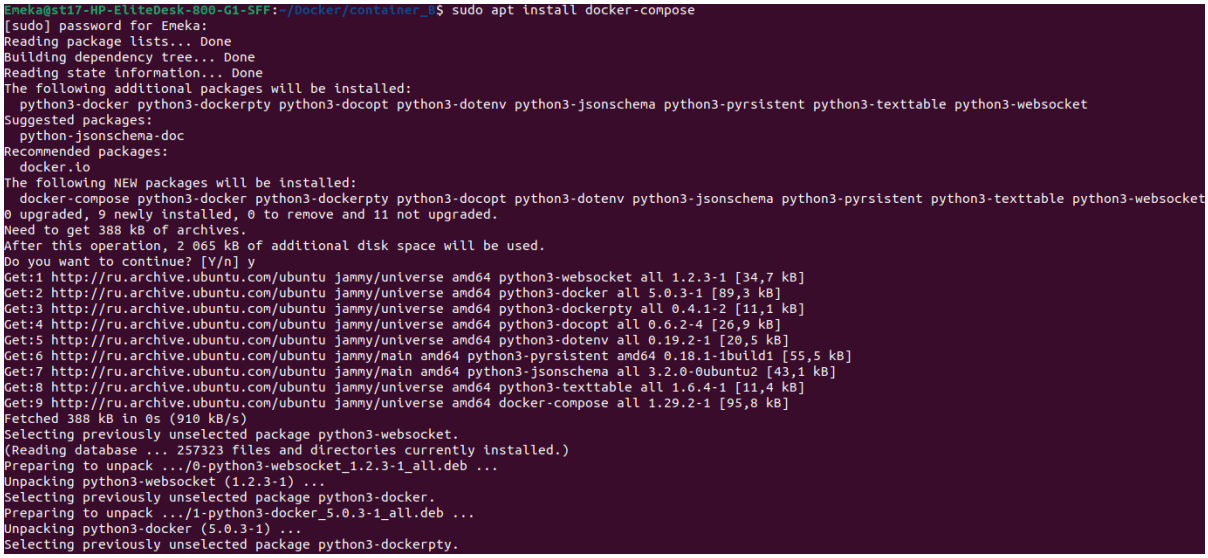

Figure 15: Install the docker-compose package

Since we have docker-compose install in the machine now, we need to build the containers using docker-compose and the command goes thus:

* `docker-compose up`

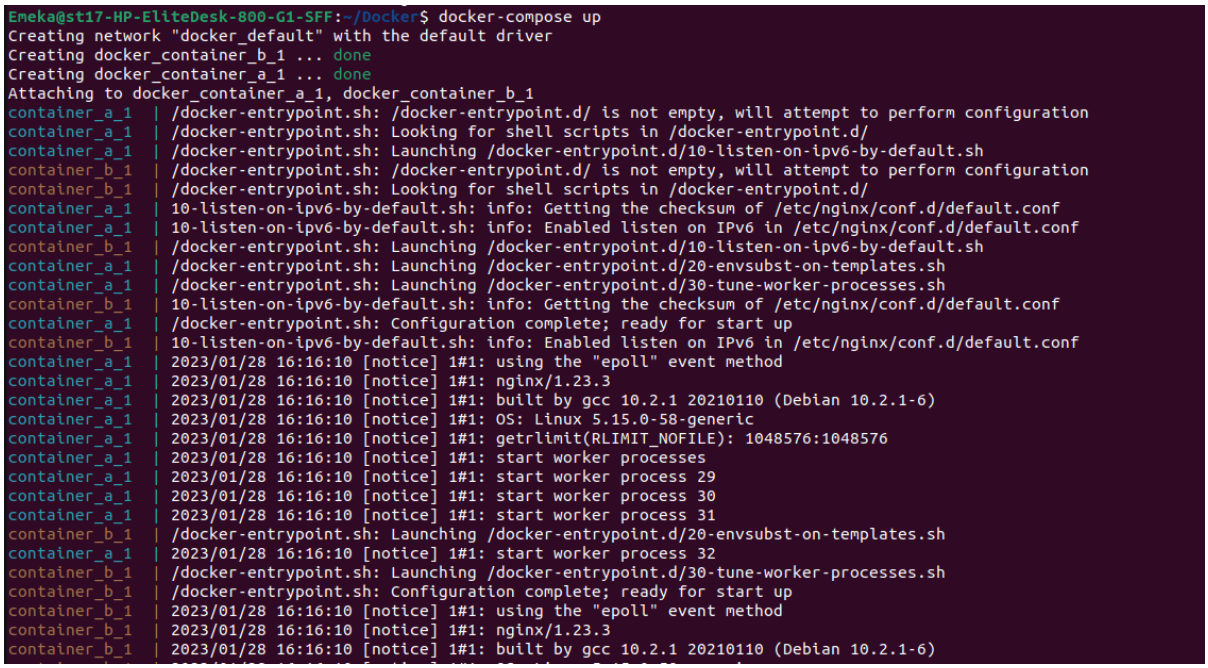

Figure 16: Running Docker Compose

The web page for both service can be accessed from the web browsers as shown below

For the Container_A:

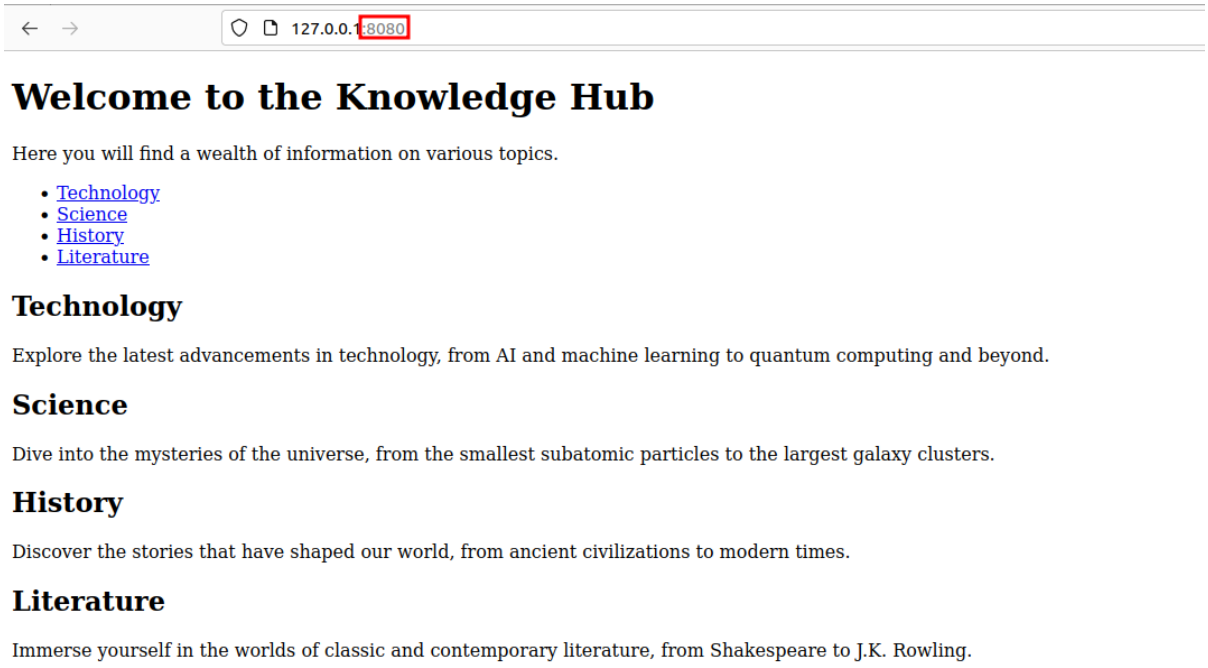

Figure 17: Webpage of Container A

For the Container B:

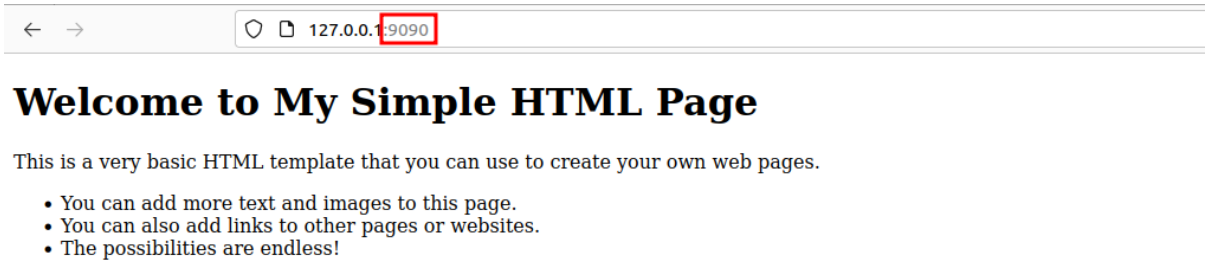

Figure 18: Webpage of Container B

4. Configure L7 Loadbalaner
   a) Install Nginx in the host machine, and configure it in front of two containers in a manner that it should distribute the load in RR approach

An L7 loadbalancer means a layer 7 load balancer i.e application layer of the OSI model. What this means is that the loadbalancer directs traffic based on the contents of the application layer protocol, such as HTTP, HTTPS or FTP. This is quite impressive because it can lead to better traffic management and routing decision.

Configuring the server in front means that we should configure Nginx server as a proxy server, and place the two running containers at the back of it.
The RR approach means that the Load balancing technique should be round robin technique i.e the request are evenly distributed between the servers.

To install the Nginx Server, we will run the command:

* `sudo apt install nginx`

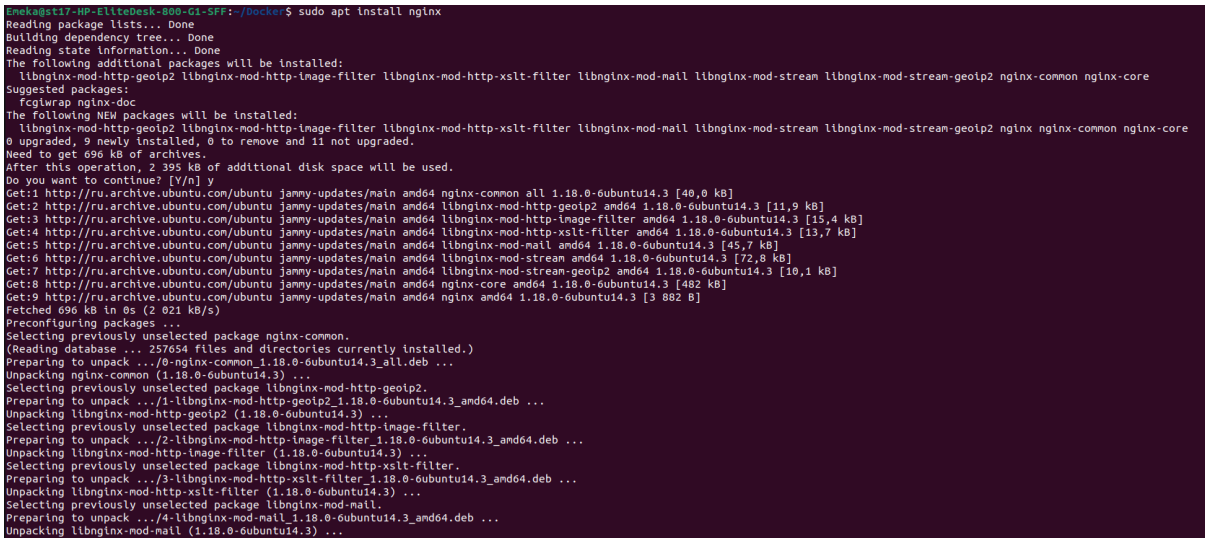

Figure 19: Installed Nginx Webserver

To configure the Nginx loadbalancer, I will access the configuration file of Nginx webserver through the command

* `sudo nano /etc/nginx/sites-available/default`

And the configuration for the server goes thus in the screenshot

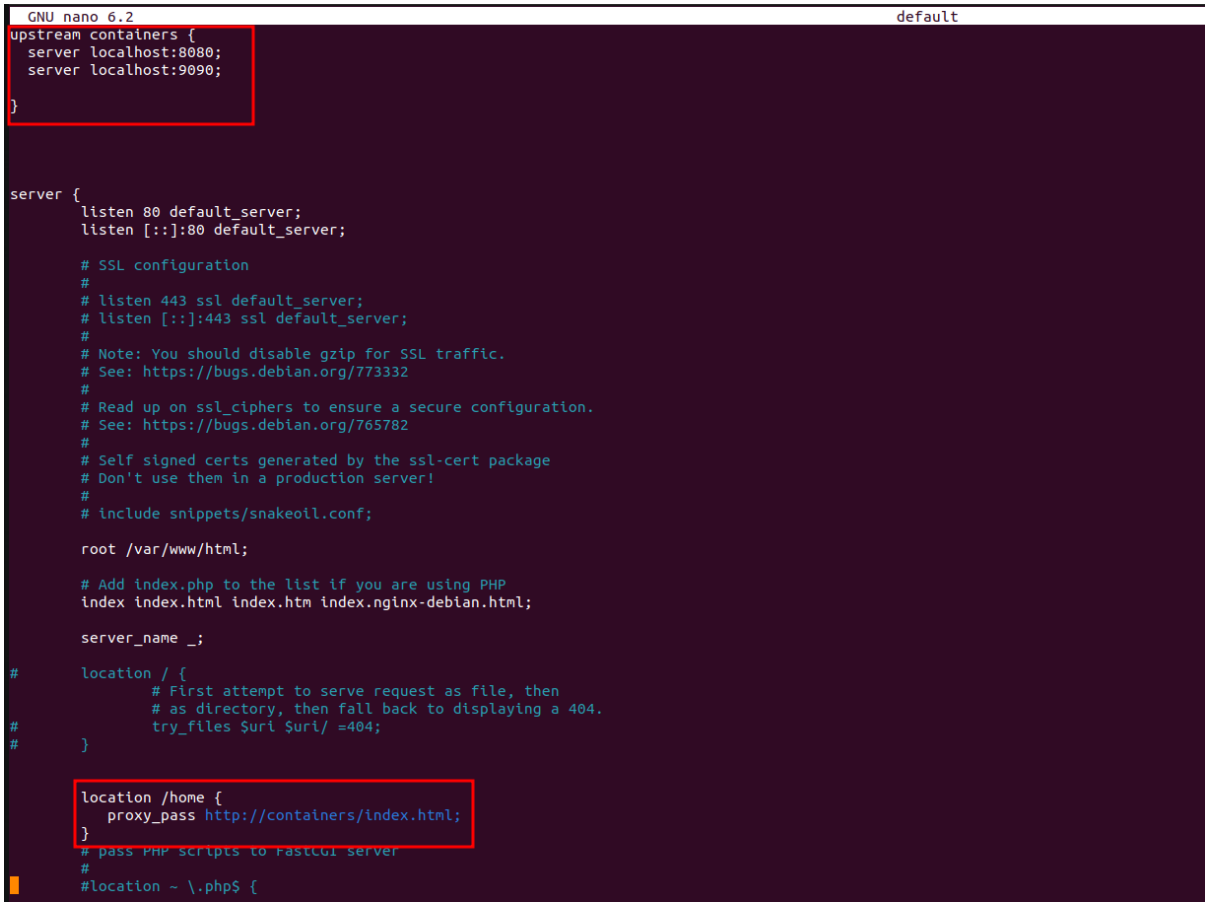

Figure 20: Loadbalancer Configurations

After this configuration has been update, we will restart the nginx webserver to activate the configuration using the command as given thus

```
* systemctl restart nginx
```

To verify that the Load balancer is working appropriately, we will check nginx server running on my host computer while specifying the path as shown in the snapshot

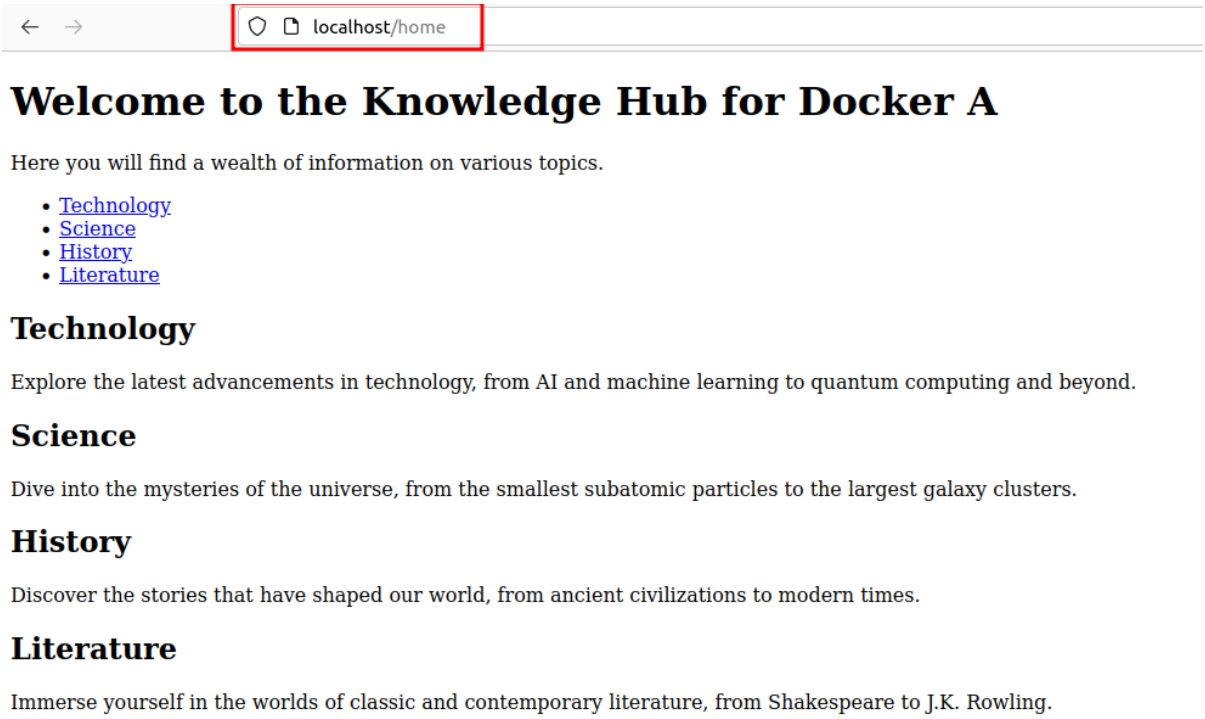

Figure 21: Proxy and Load Balancing Nginx

To verify that the application is distributed among the containers, I will check the logs of the Nginx server, as given in the snapshot below

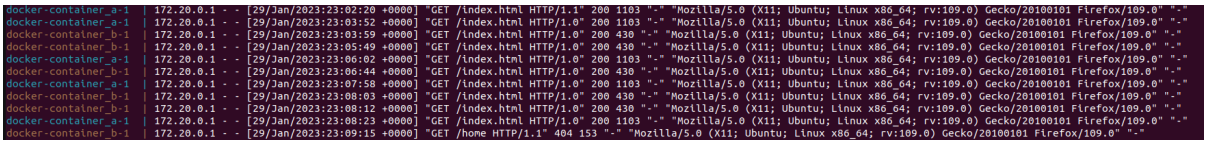

Figure 22: Log Output of the Host Nginx Server

We can see from the logs that anytime the server is accessed, we could hit the two containers periodically.

5. Automate everything in a Bash script (Optional)

- Automate all of the process in a bash script
- Push your code to Version Control Systems (VCS)

To write a basic automation script that will run this process automatically, we can check out this code below

```
#!/bin/bash

#This bash script is used to automate the flow of creating the
#docker-compose file  and tearing it down

#Change directory to where the compose file is located
cd ~/Docker

# Build the containers
docker-compose build

# Start the containers in detached mode
docker-compose up -d

#check the webpages of the containers
curl http://127.0.0.1:8080
curl http://127.0.0.1:9090

# Stop the containers
docker-compose stop

# Remove the containers
docker-compose down
```

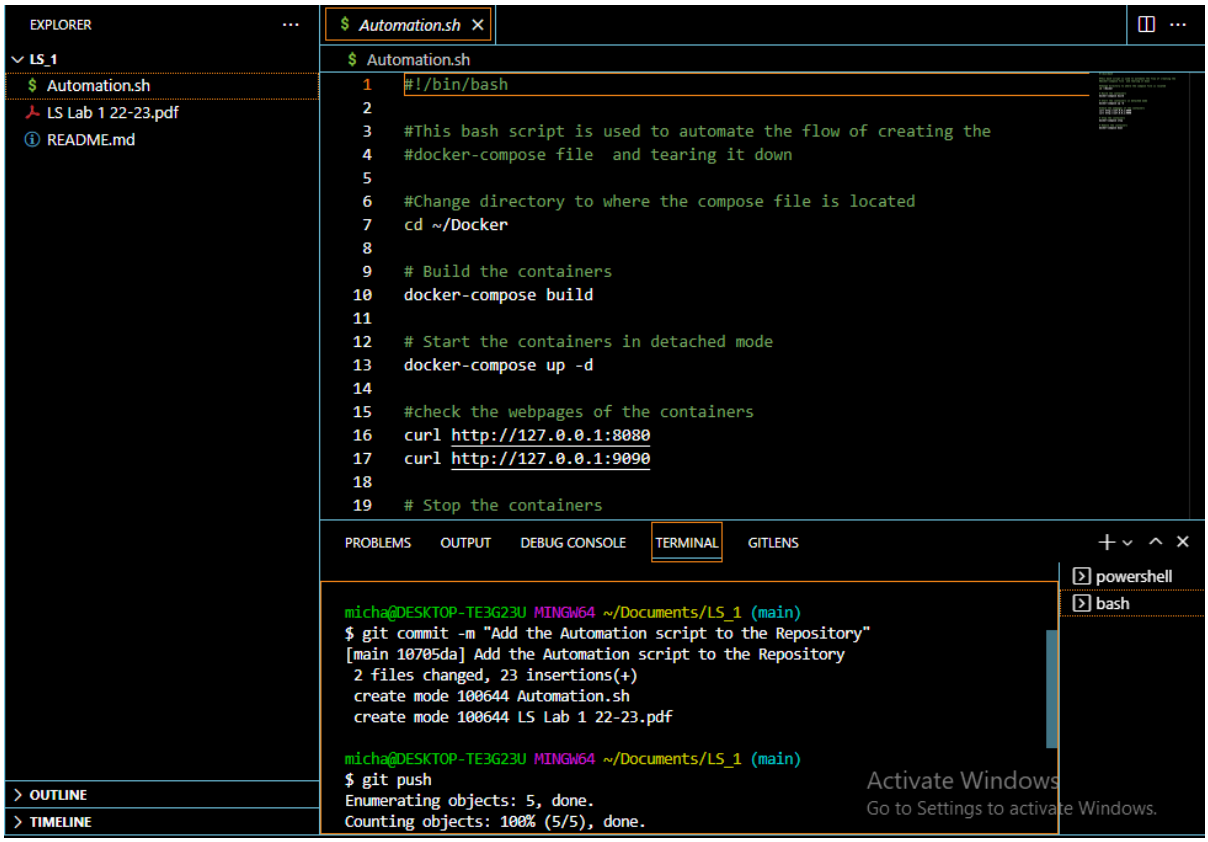

Figure 23:Pushing Automation Script to Git

To verify my git repository

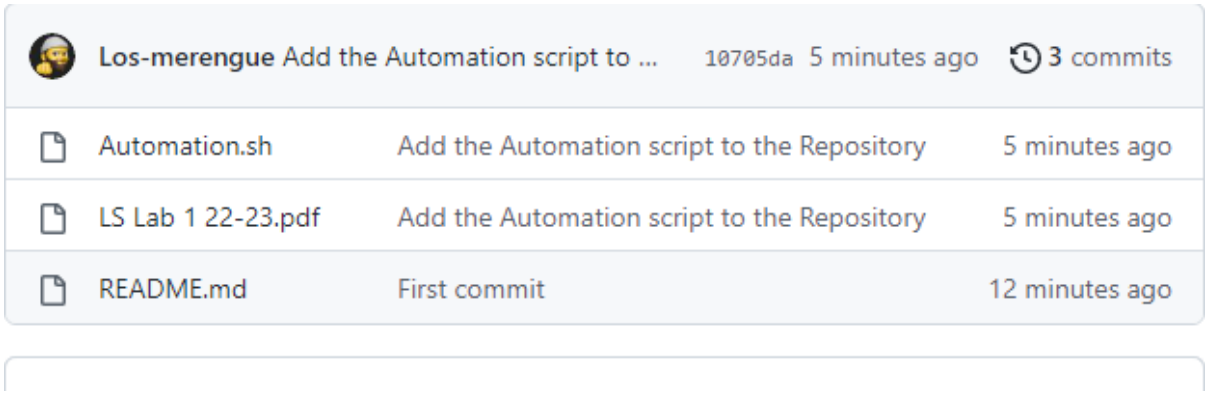

Figure 24: GitHub Repository

Reference

1. Docker Installation in Ubuntu (https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository)
2. Learning Docker (https://docker-curriculum.com/#prerequisites)
3. Nginx Documentation (https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/)
4. Cloud Infra Services (https://cloudinfrastructureservices.co.uk/install-nginx-with-docker-compose/)
5. Nginx Load Balancer (https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-setup-an-Nginx-load-balancer-example)
6. Docker-compose documentation (https://docs.docker.com/compose/features-uses/)