

**NAME:** MULLA AFRAH AKKAS ALI

**ROLL NO.:** 612038

**BRANCH:** T.E. – I.T.

**SEMESTER:** ODD SEMESTER 5

**COURSE:** Advance DevOPs (ITL504)

**DATE:** 21-09-2022

## EXPERIMENT 9

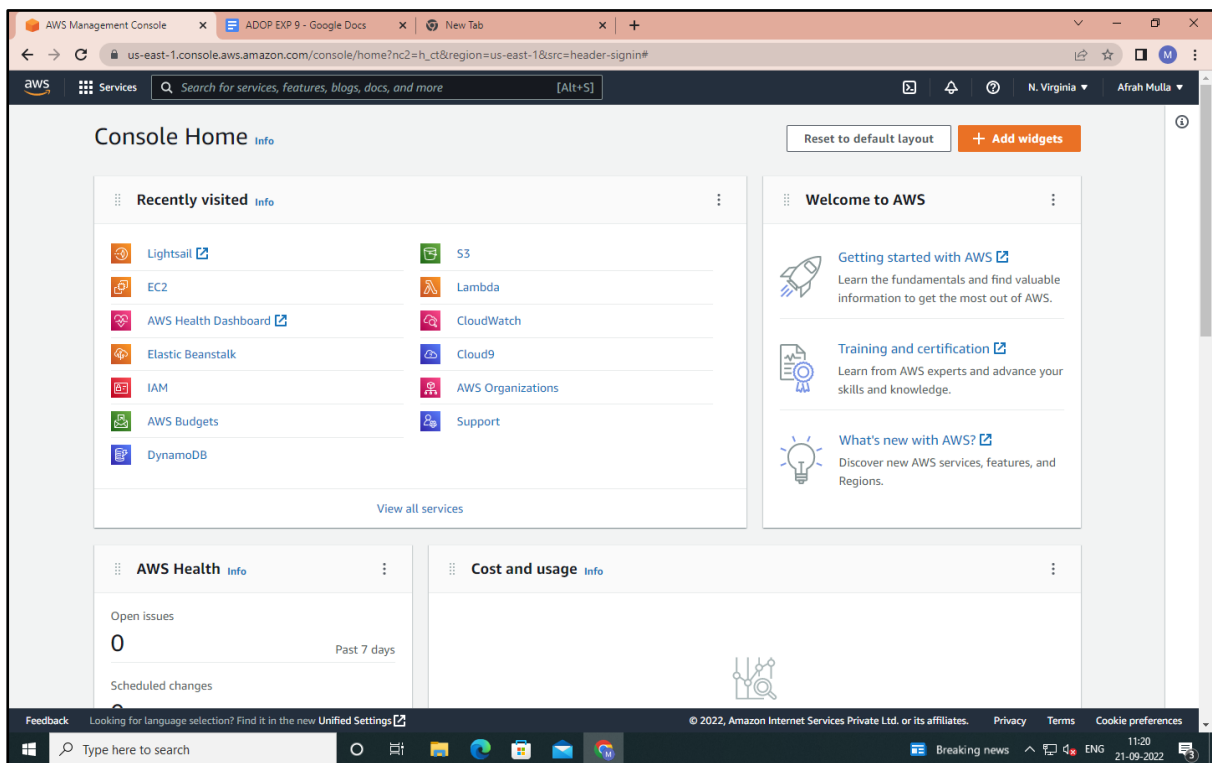
### 1. Install docker on AWS EC2 – Ubuntu by using curl

```
#curl -fsSL https://get.docker.com -o get-docker.sh
```

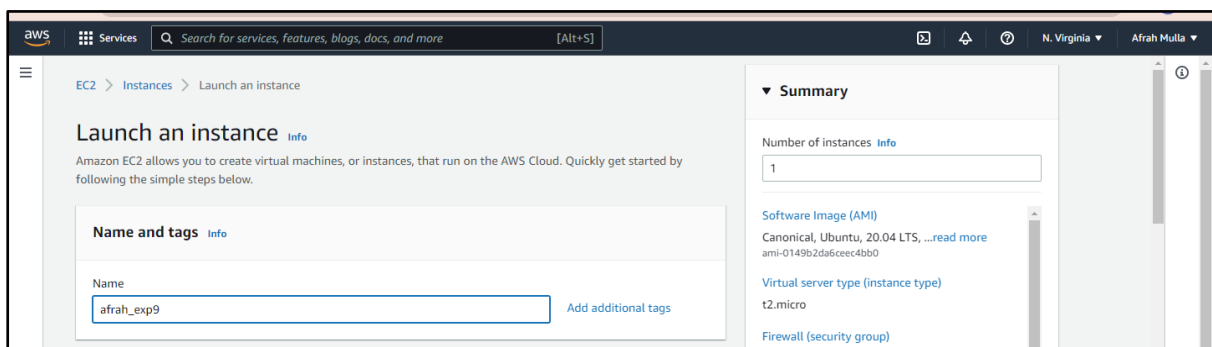
```
#sh get-docker.sh
```

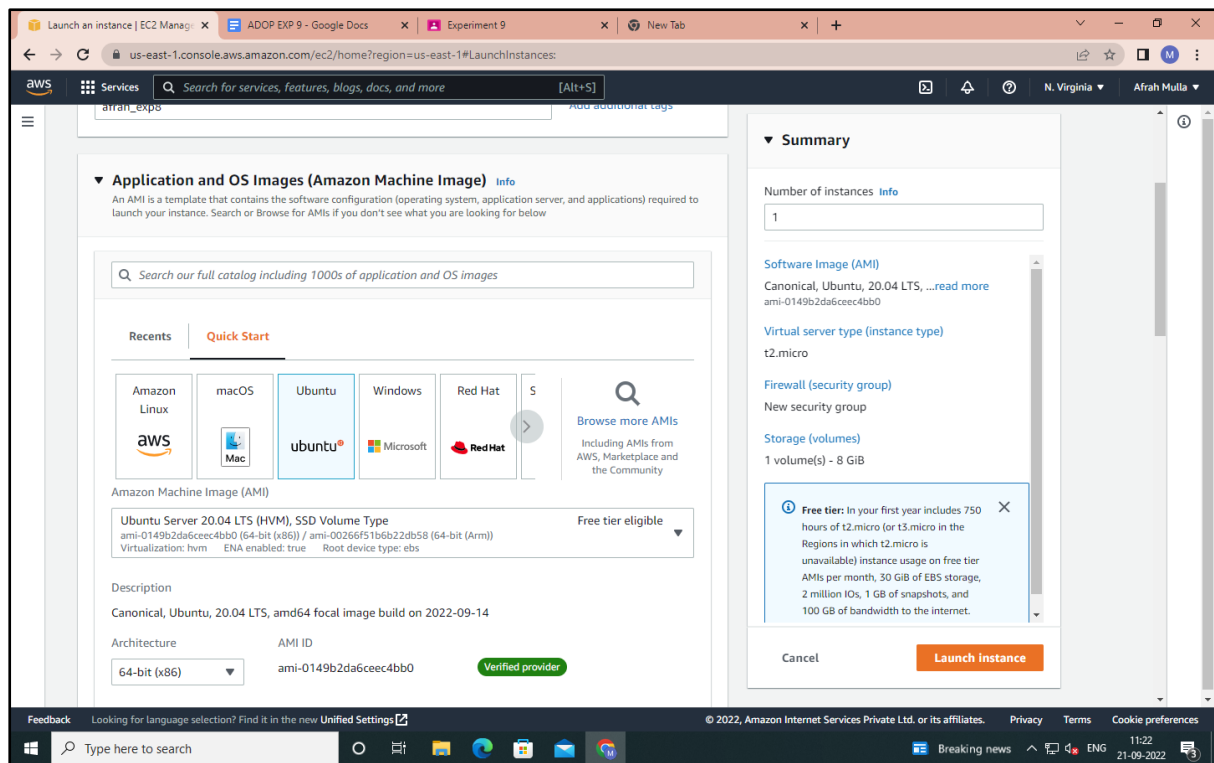
### 2. Run a Flask Application inside a Docker Container and explain the steps.

Step 1: AWS management console

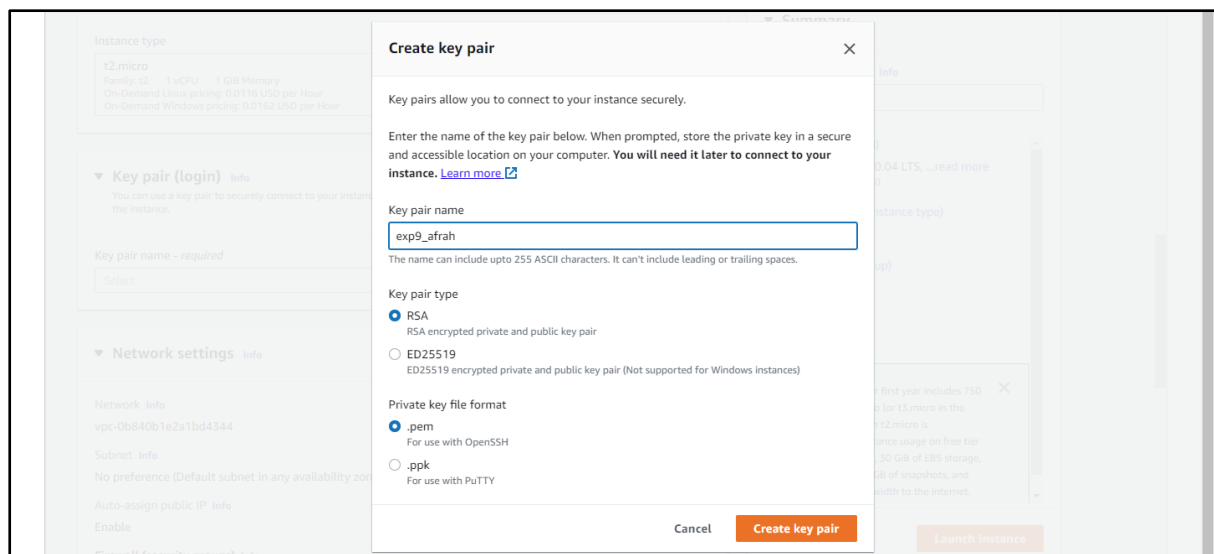


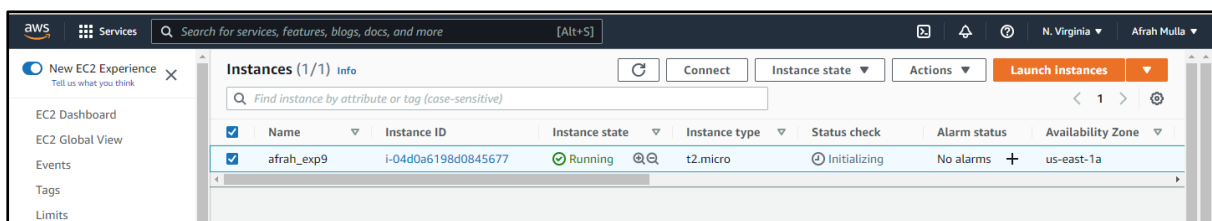
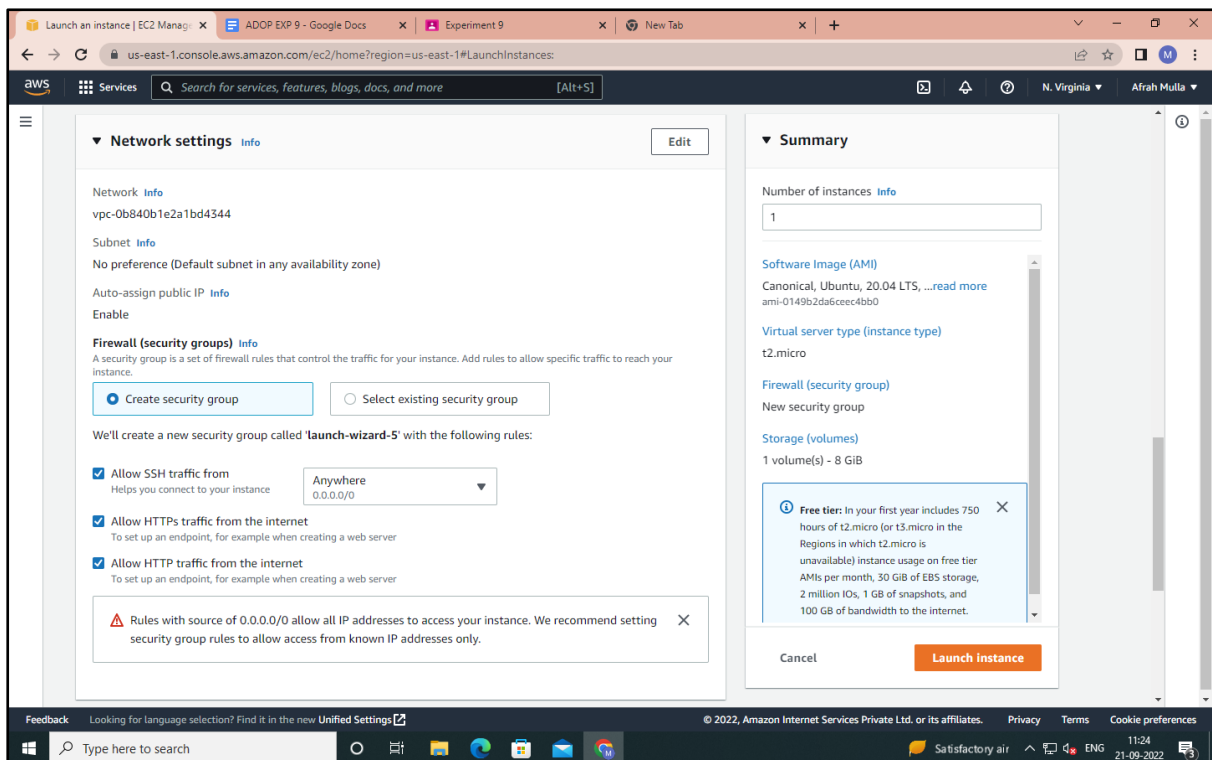
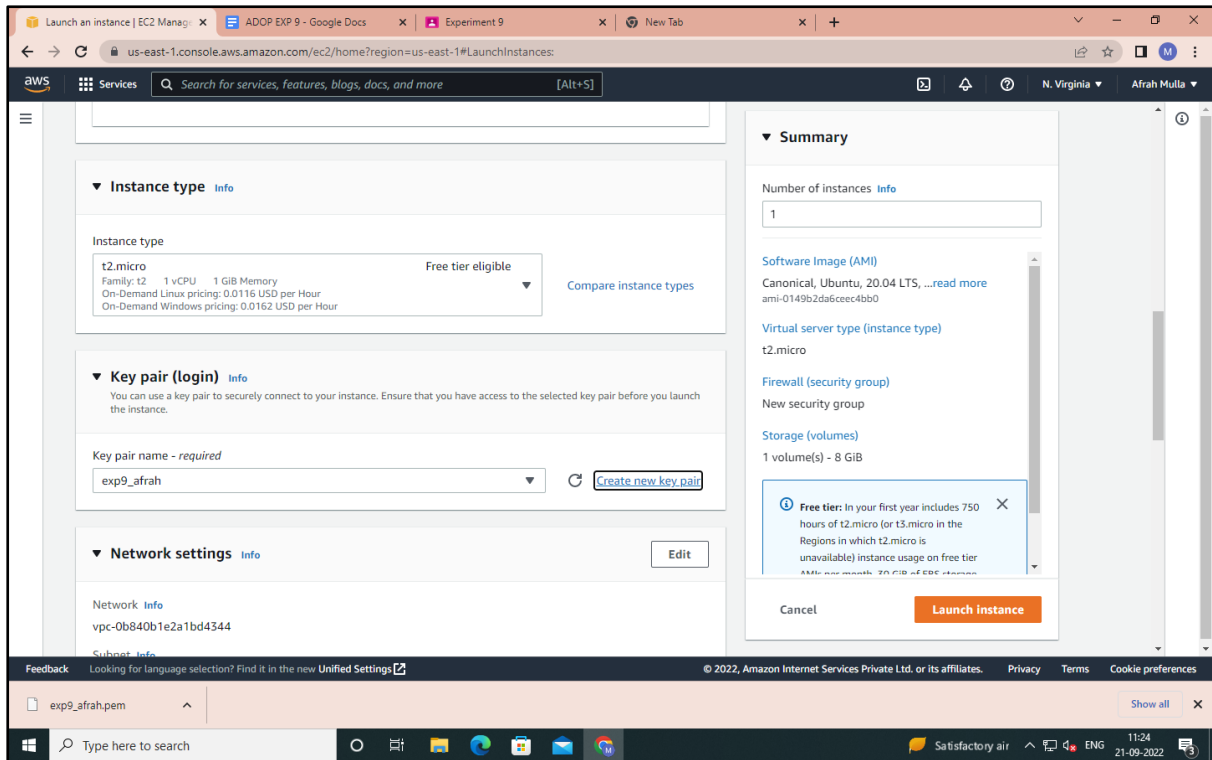
Step 2: Search for EC2 -> Select it -> Create an instance -> Name and Select Ubuntu instance with 20.04 LTS version





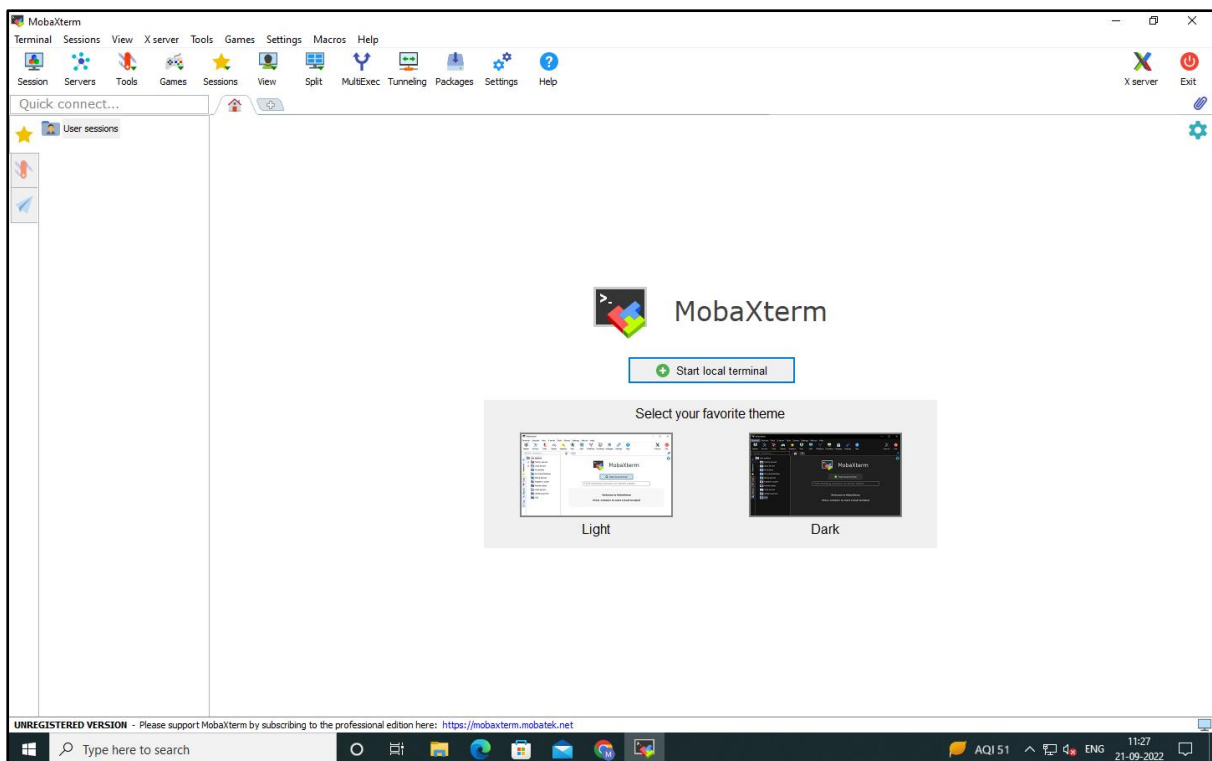
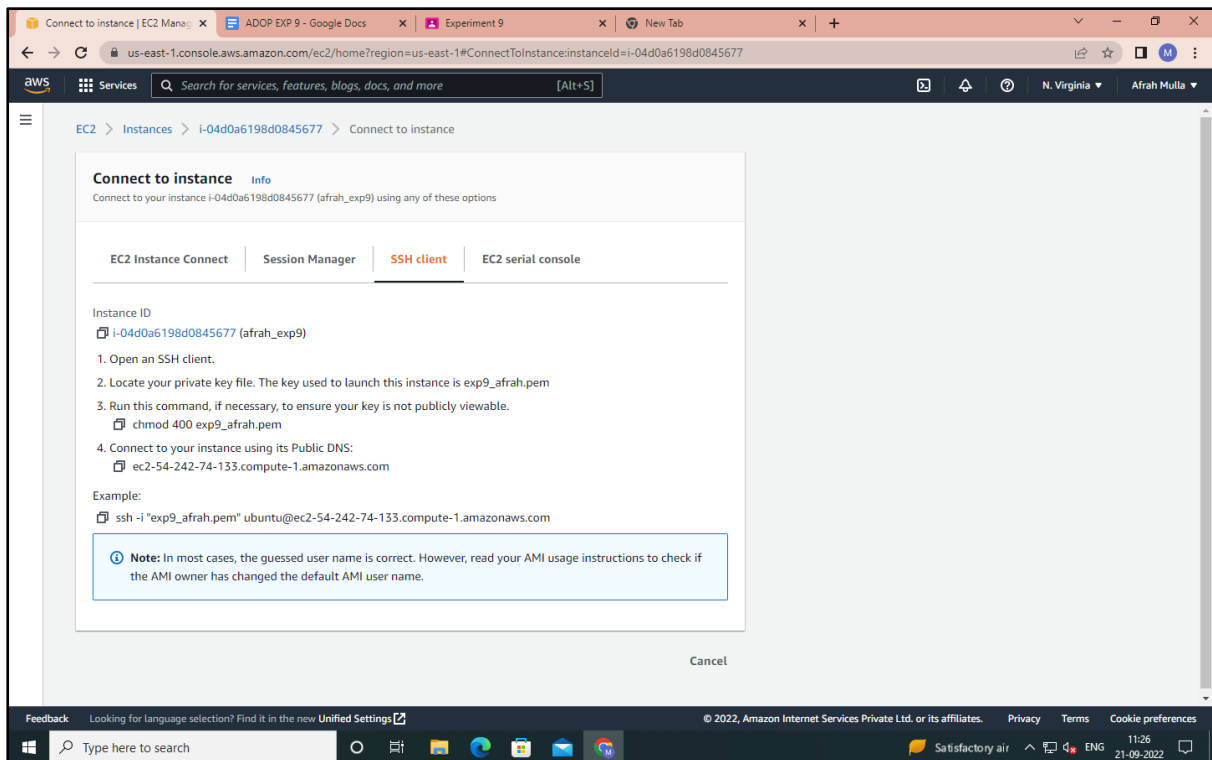
Step 3: Create a key pair. In the network settings allow the HTTPS and HTTP traffic

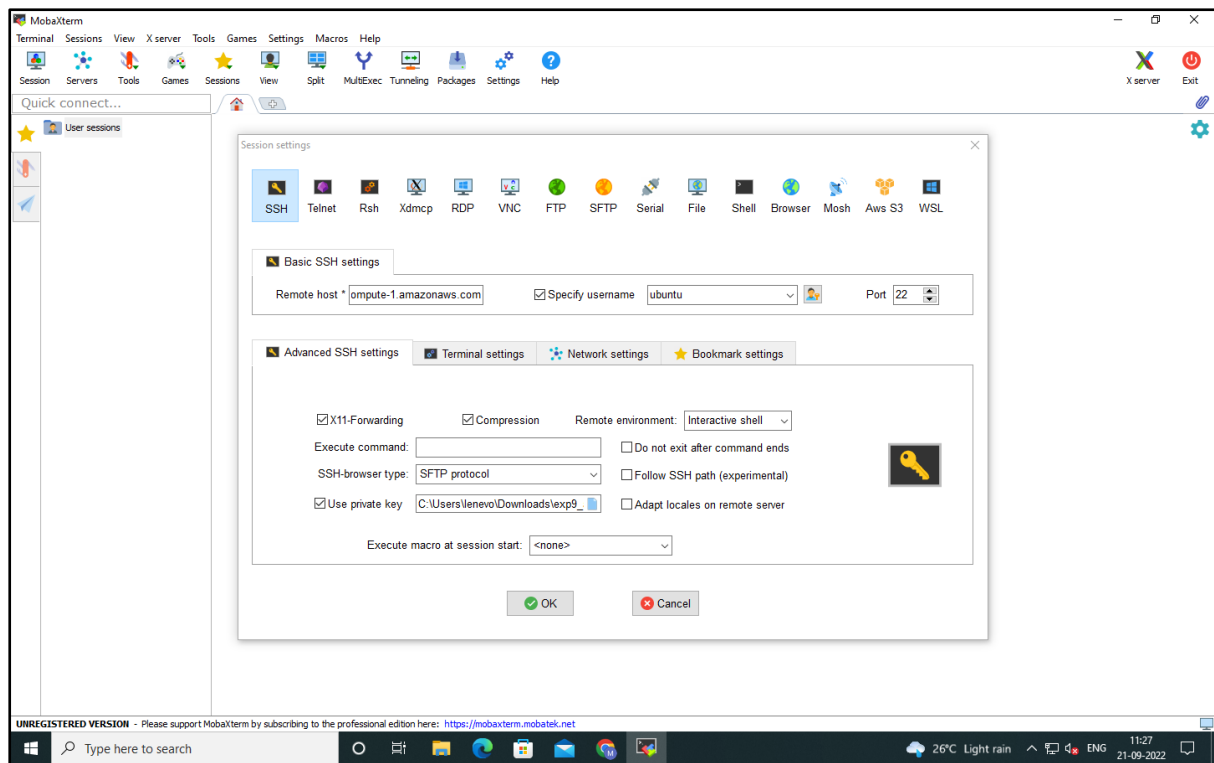




Step 4: Launch MobaXterm -> Select SSH session -> Copy the public DNS of your instance and paste it into the remote host and enter the username

Use the downloaded key pair as the private key

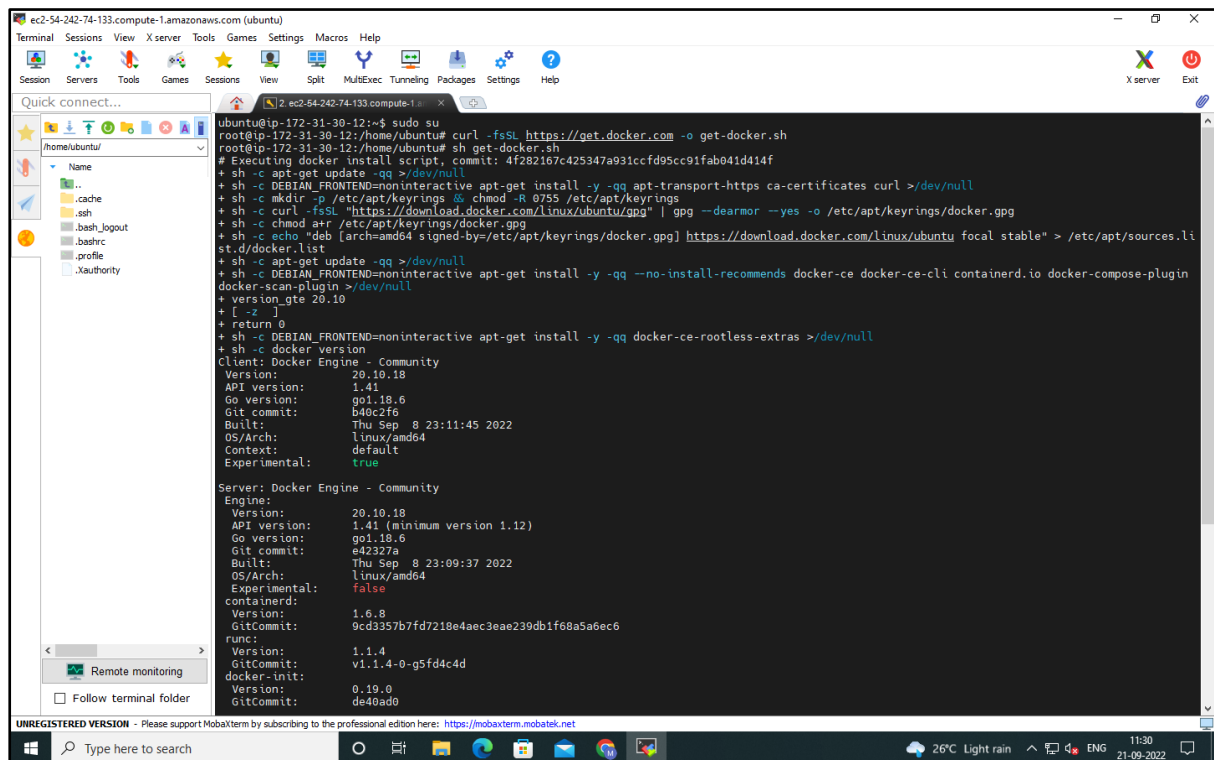


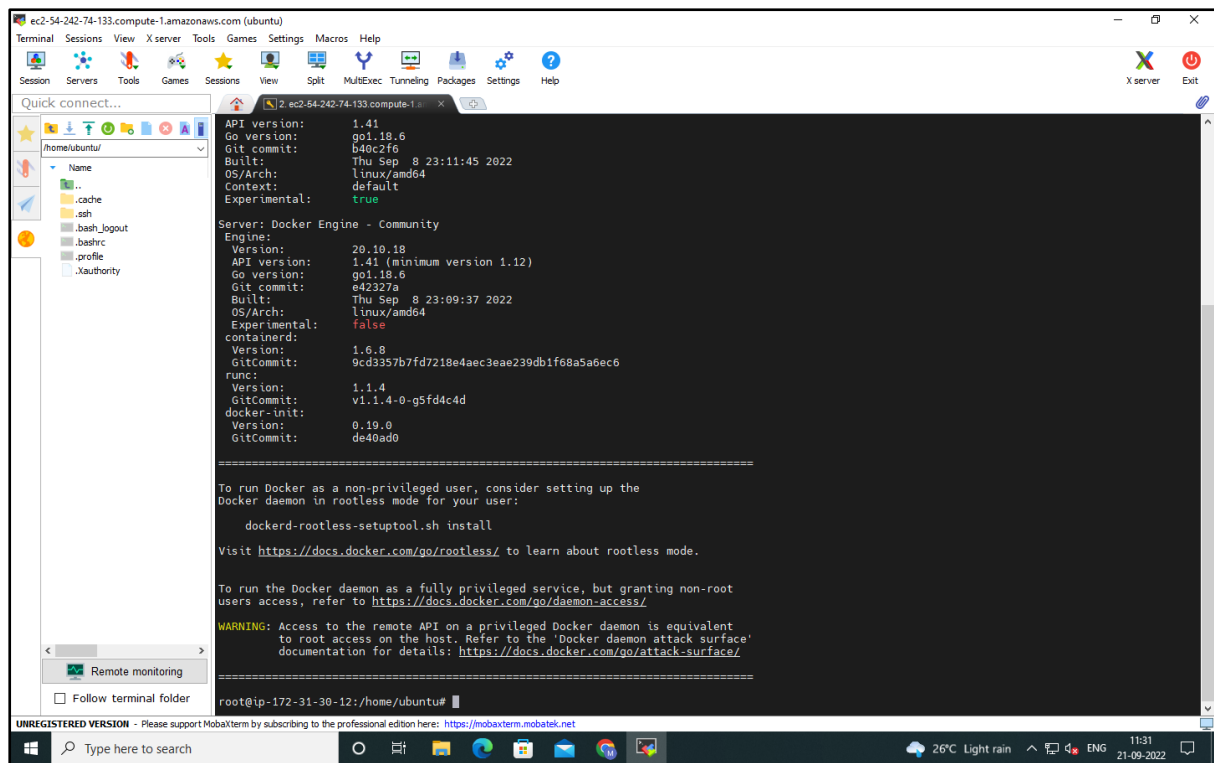


Step 5: Run the command 'sudo su' to gain root user access. Then enter commands:

`curl -fsSL https://get.docker.com -o get-docker.sh`

`sh get-docker.sh`





The screenshot shows a MobaXterm terminal window with a dark background. The title bar indicates the connection is to 'ec2-54-242-74-133.compute-1.amazonaws.com (ubuntu)'. The terminal displays the output of a Docker installation command. It shows the Docker Engine version (20.10.18) and the Docker CLI version (20.10.18). The output also includes a warning about access to the remote API on a privileged Docker daemon and instructions on how to run Docker as a non-privileged user or as a fully privileged service.

```
API version: 1.41
Go version: go1.18.6
Git commit: b40c2f6
Built: Thu Sep 8 23:11:45 2022
OS/Arch: linux/amd64
Context: default
Experimental: true

Server: Docker Engine - Community
Engine:
  Version: 20.10.18
  API version: 1.41 (minimum version 1.12)
  Go version: go1.18.6
  Git commit: e42327a
  Built: Thu Sep 8 23:09:37 2022
  OS/Arch: linux/amd64
  Experimental: false
containerd:
  Version: 1.6.8
  GitCommit: 9cd3357b7fd7218e4aec3eae239db1f68a5a6ec6
runc:
  Version: 1.1.4
  GitCommit: v1.1.4-0-g5fd4c4d
docker-init:
  Version: 0.19.0
  GitCommit: de40ad0

=====

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

  dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.

To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
to root access on the host. Refer to the 'Docker daemon attack surface'
documentation for details: https://docs.docker.com/go/attack-surface/

=====

root@ip-172-31-30-12:/home/ubuntu#
```

Step 6: Make a new directory using command

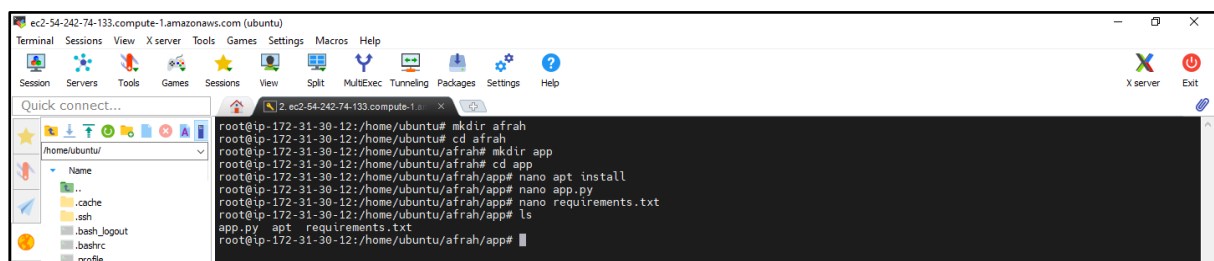
`mkdir 'directory name'`

and go into that directory using

`cd 'directory name'`

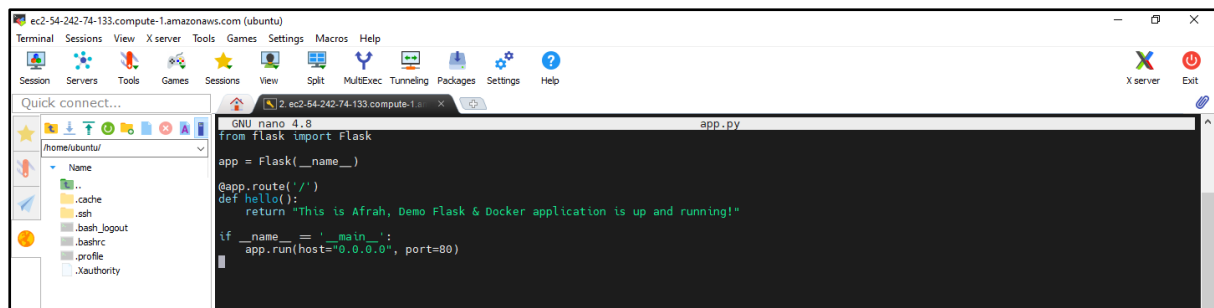
Create another director named app inside previously created directory and then go into app directory using the `cd` command.

Using the nano editor, create 2 files named `app.py` and `requirements.txt` and add the required codes in them. Then use the `ls` command to list the files inside directory



The screenshot shows a MobaXterm terminal window with a dark background. The title bar indicates the connection is to 'ec2-54-242-74-133.compute-1.amazonaws.com (ubuntu)'. The terminal displays a series of commands and their outputs, showing the creation of a directory named 'afrah', the creation of a subdirectory named 'app', and the listing of files in the 'app' directory. The files listed are 'app.py' and 'requirements.txt'.

```
root@ip-172-31-30-12:/home/ubuntu# mkdir afrah
root@ip-172-31-30-12:/home/ubuntu# cd afrah
root@ip-172-31-30-12:/home/ubuntu/afrah# mkdir app
root@ip-172-31-30-12:/home/ubuntu/afrah# cd app
root@ip-172-31-30-12:/home/ubuntu/afrah/app# nano apt install
root@ip-172-31-30-12:/home/ubuntu/afrah/app# nano app.py
root@ip-172-31-30-12:/home/ubuntu/afrah/app# nano requirements.txt
root@ip-172-31-30-12:/home/ubuntu/afrah/app# ls
app.py  apt  requirements.txt
root@ip-172-31-30-12:/home/ubuntu/afrah/app#
```

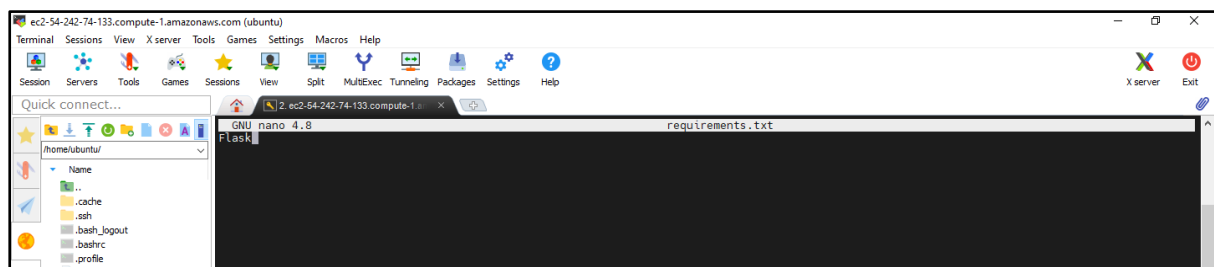


```
GNU nano 4.8 app.py
from flask import Flask

app = Flask(__name__)

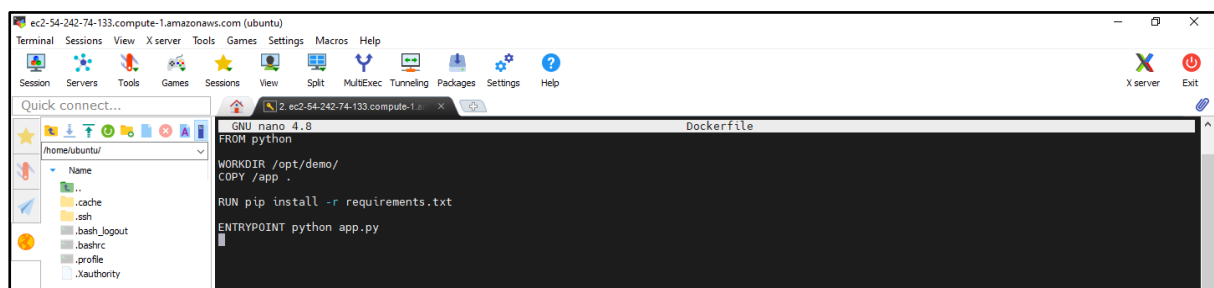
@app.route('/')
def hello():
    return "This is Afrah, Demo Flask & Docker application is up and running!"

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=80)
```



```
GNU nano 4.8 requirements.txt
Flask
```

Step 7: Use command: 'cd ..' to come back to the home directory. Create Dockerfile in home directory using nano editor. Add the required code in the Dockerfile



```
GNU nano 4.8 Dockerfile
FROM python

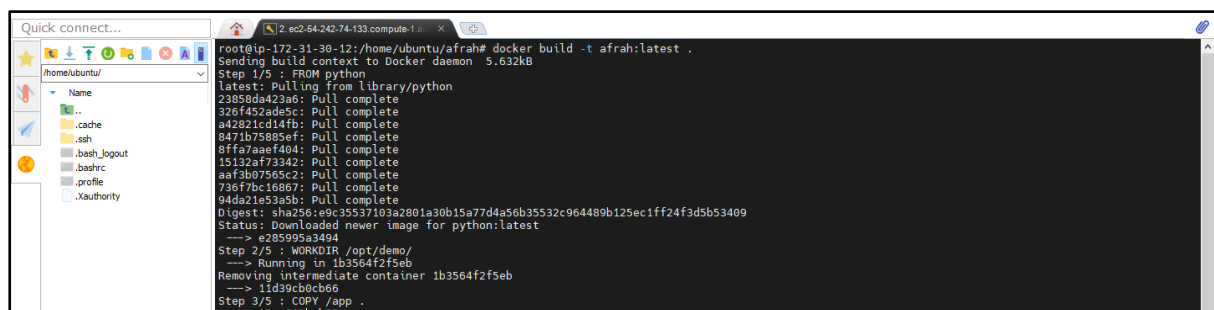
WORKDIR /opt/demo/
COPY /app .

RUN pip install -r requirements.txt

ENTRYPOINT python app.py
```

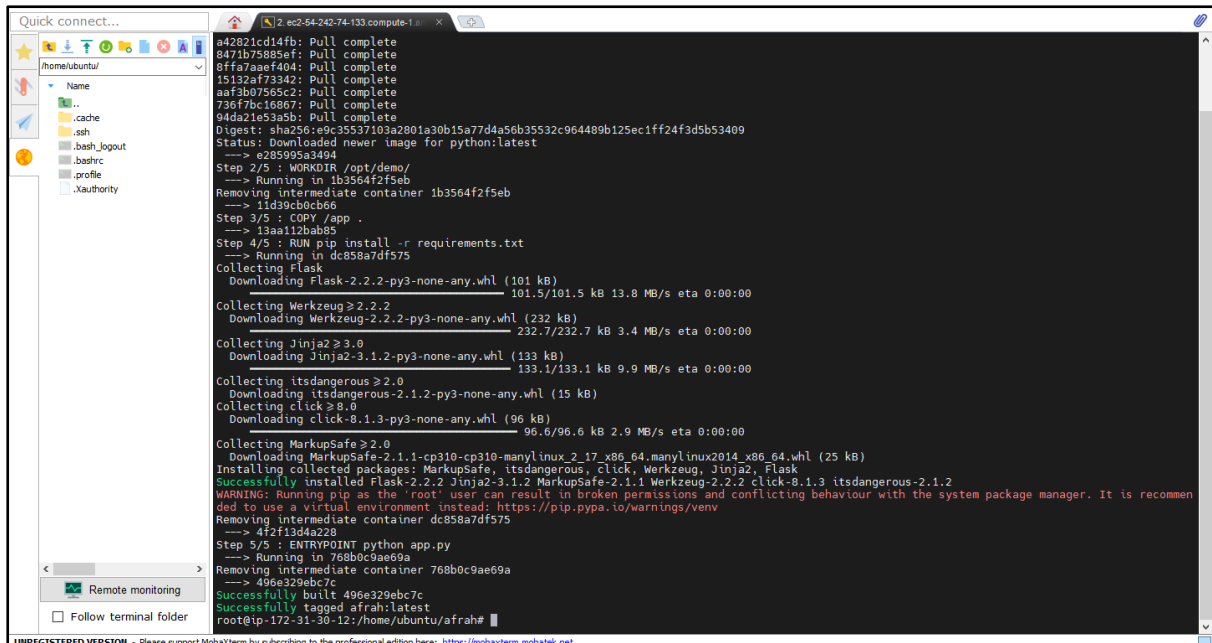
Step 8: Execute the command: 'docker build -t (directory name):latest .'

This will build an image but it will then tag the resulting image. The repository name will be 'afrah' and the tag will be latest



```
root@ip-172-31-30-12:/home/ubuntu/afrah# docker build -t afrah:latest .
Sending build context to Docker daemon 5.632kB
Step 1/5 : FROM python
latest: Pulling from library/python
23958d9423a6: Pull complete
326f452ade5c: Pull complete
a42821cd14fb: Pull complete
8471b75885ef: Pull complete
8ffa7aaef484: Pull complete
15132af73342: Pull complete
aaf3b07565c2: Pull complete
736f7bc16867: Pull complete
94da21e53a5b: Pull complete
Digest: sha256:e9c35537103a2881a30b15a77d4a56b35532c964489b125ec1ff24f3d5b53409
Status: Downloaded newer image for python:latest
--> e285995a3494
Step 2/5 : WORKDIR /opt/demo/
--> Running in 1b3564f2f5eb
Removing intermediate container 1b3564f2f5eb
--> 11d39cb0cb66
Step 3/5 : COPY /app .
--> 13aa112bab85
```



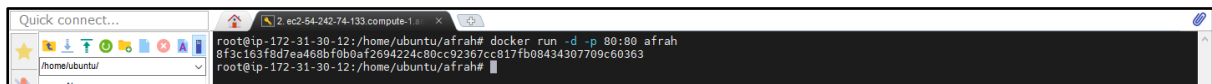


```
Quick connect...
a42821cd14fb: Pull complete
847fb75885af: Pull complete
8ff7aaef484: Pull complete
15132af73342: Pull complete
aaf3b07565c2: Pull complete
736f7bc16867: Pull complete
9ad2e1e3a5b: Pull complete
Digest: sha256:e9c35537103a2801a30b15a77d4a56b35532c964489b125ec1ff24f3d5b53409
Status: Downloaded newer image for python:latest
--> e285995a3494
Step 2/5 : WORKDIR /opt/demo/
--> Running in 1b3564f2f5eb
Removing intermediate container 1b3564f2f5eb
--> 11d39cb0cb66
Step 3/5 : COPY /app .
--> 13aa12bd885
Step 4/5 : RUN pip install -r requirements.txt
--> Running in dc858a7df575
Collecting Flask
  Downloading Flask-2.2.2-py3-none-any.whl (101 kB)
    101.5/101.5 kB 13.8 MB/s eta 0:00:00
Collecting Werkzeug>=2.2.2
  Downloading Werkzeug-2.2.2-py3-none-any.whl (232 kB)
    232.7/232.7 kB 3.4 MB/s eta 0:00:00
Collecting Jinja2>=3.0
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
    133.1/133.1 kB 9.9 MB/s eta 0:00:00
Collecting itsdangerous>=2.0
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.0
  Downloading click-8.1.3-py3-none-any.whl (96 kB)
    96.6/96.6 kB 2.9 MB/s eta 0:00:00
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.1-cp310-cp310-manylinux_2_17_x86_64_manylinux2014_x86_64.whl (25 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, Werkzeug, Jinja2, Flask
Successfully installed Flask-2.2.2 Jinja2-3.1.2 MarkupSafe-2.1.1 Werkzeug-2.2.2 click-8.1.3 itsdangerous-2.1.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Removing intermediate container dc858a7df575
--> 4f2f13d4a228
Step 5/5 : ENTRYPOINT python app.py
--> Running in 768b0c9ae69a
Removing intermediate container 768b0c9ae69a
--> 496e329ebc7c
Successfully built 496e329ebc7c
Successfully tagged afrah:latest
root@ip-172-31-30-12:/home/ubuntu/afrah#
```

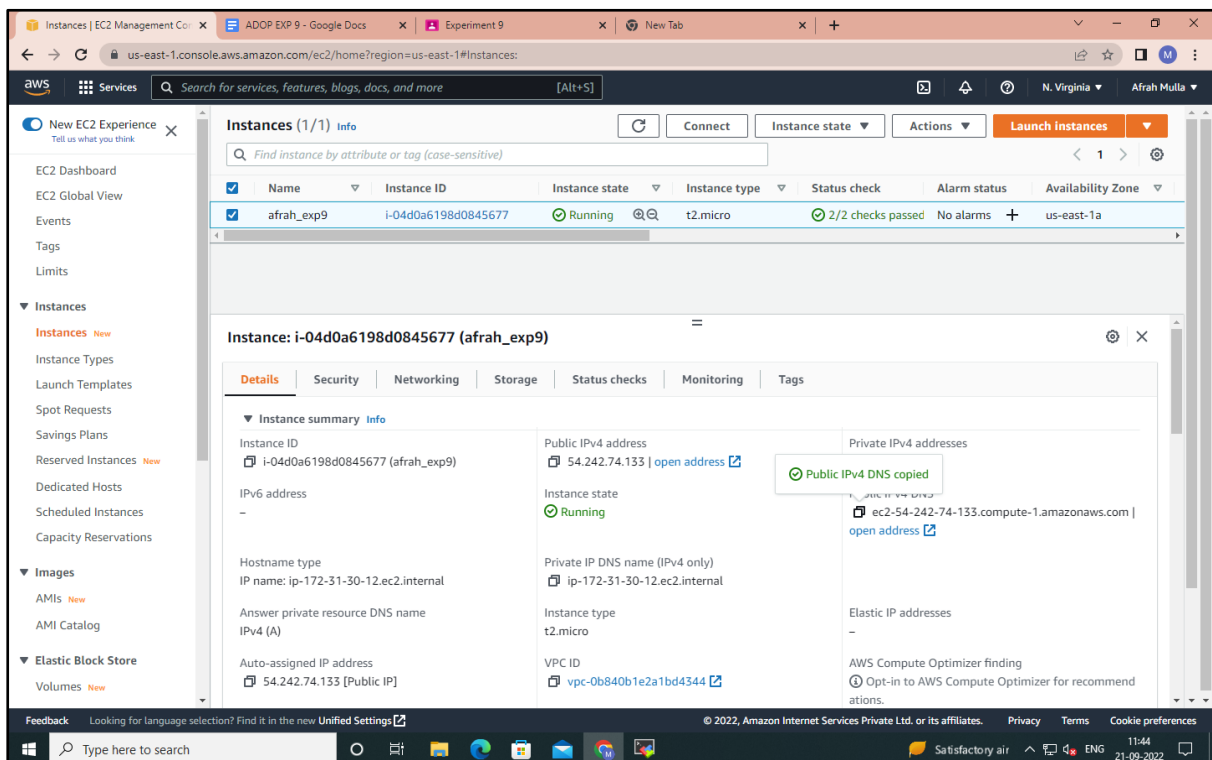
Step 9: Run command

`docker run -d -p 80:80 (name of directory)`

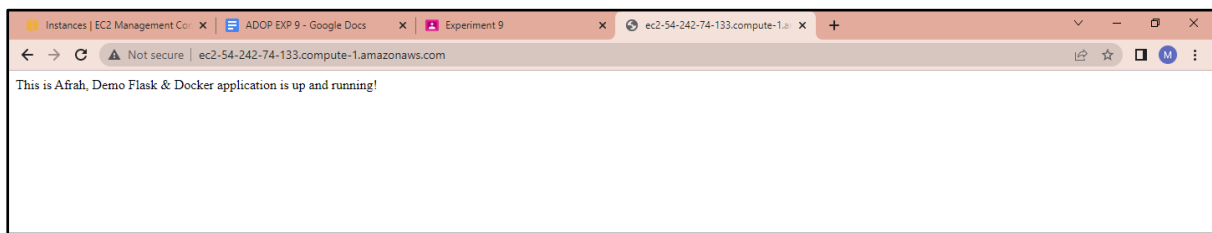
Now, to launch the web server, copy the IPv4 address from the EC2 instance details and paste it into a web browser. The web browser will display it



```
Quick connect...
root@ip-172-31-30-12:/home/ubuntu/afrah# docker run -d -p 80:80 afrah
8f3c163f8d7ea468bf0b0af2694224c80cc92367cc817fb08434307709c60363
root@ip-172-31-30-12:/home/ubuntu/afrah#
```



The screenshot displays the AWS Management Console interface. On the left, the navigation menu shows 'Instances' selected. The main panel shows the 'Instances (1/1) Info' table with one instance, 'afrah\_exp9', in a 'Running' state. Below the table, the 'Instance: i-04d0a6198d0845677 (afrah\_exp9)' details are expanded. The 'Details' tab is active, showing fields like Instance ID, Public IPv4 address (54.242.74.133), and Private IP address (ip-172-31-30-12.ec2.internal). A tooltip is visible over the Public IPv4 address, displaying the 'Public IPv4 DNS copied' message and the full DNS address: 'ec2-54-242-74-133.compute-1.amazonaws.com'.



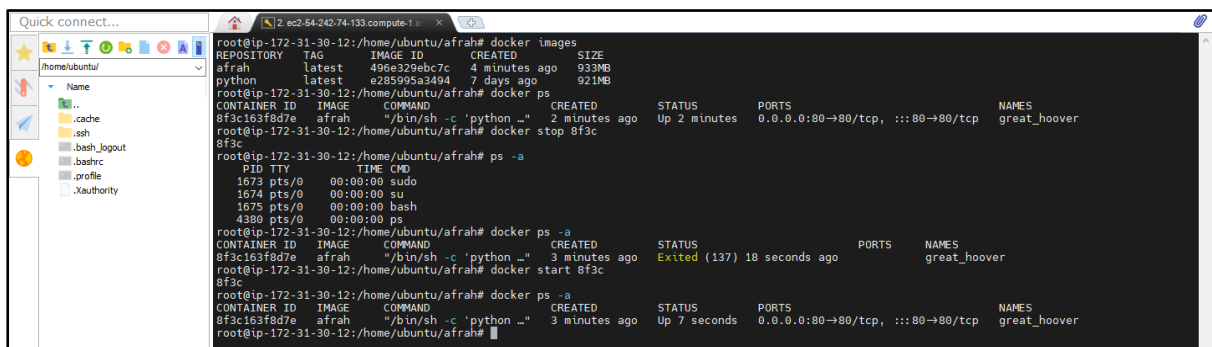
Step 10: Execute command 'docker images' to see installed images

Then execute 'docker ps' to list the running containers

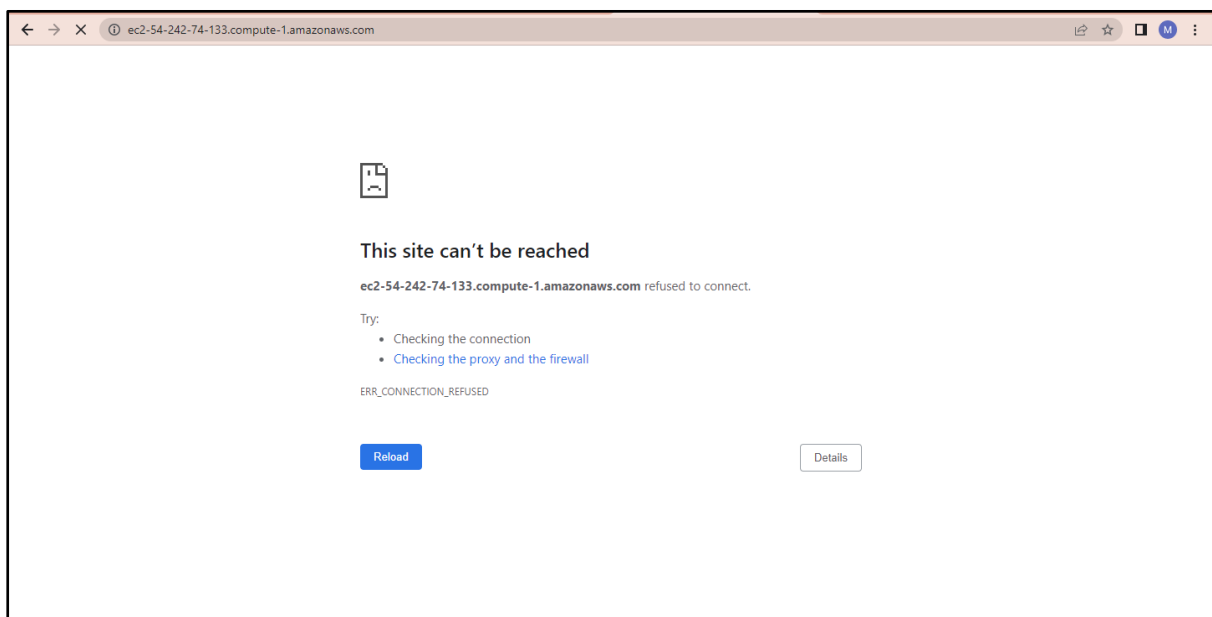
To stop a running container use command 'docker stop (container id)'

To restart an exited container use command 'docker start (container id)'

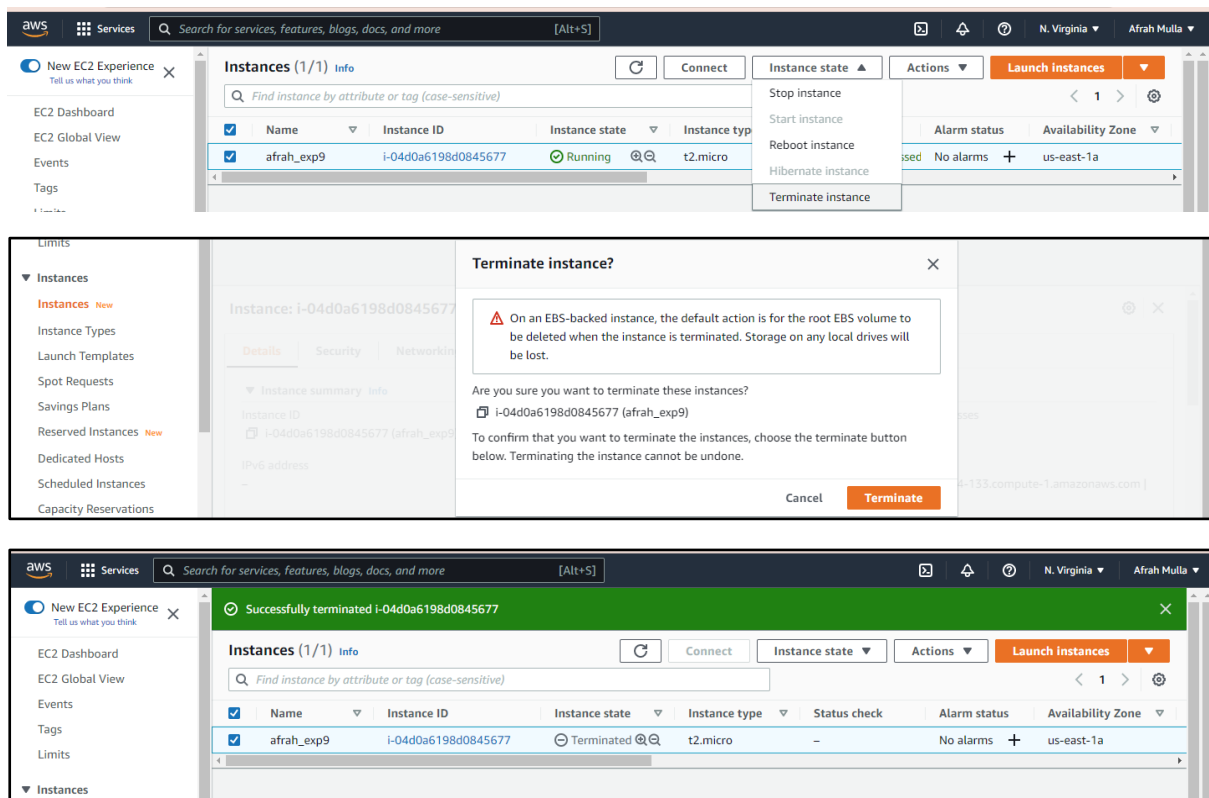
Execute 'docker ps -a' to list all running and exited containers.



When the container is not running, the web browser will not display anything



Step 11: Lastly, quit MobaXterm and delete your instance.



### 3. What is Dockerfile? Explain all lines of your Dockerfile.

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

Explanation of code:

- **FROM python** – It specifies the base image.
- **WORKDIR /opt/demo** – It sets the working directory for all the subsequent instructions to be followed.
- **COPY /app .** – It copies all the files from 'app' directory to the current directory.
- **RUN pip install -r requirements.txt** – It will install all the dependencies from requirements.txt file.
- **ENTRYPOINT python app.py** – It will run app.py in our container.