Part 2 : Creating a docker image of a simple flask app, containerising it and running it on a localhost.

1. Creating an image of the app.

```
C:\Users\hasan\272>docker build -t myapp .
[+] Building 12.5s (12/12) FINISHED                                                                    docker:default
 => [internal] load .dockerignore                                                                                0.0s
 => => transferring context: 2B                                                                                  0.0s
 => [internal] load build definition from Dockerfile                                                             0.0s
 => => transferring dockerfile: 216B                                                                             0.0s
 => [internal] load metadata for docker.io/library/python:3                                                      1.1s
 => [auth] library/python:pull token for registry-1.docker.io                                                    0.0s
 => CACHED [1/6] FROM docker.io/library/python:3@sha256:cc7372fe4746ca323f18c6bd0d45dadf22d192756abc5f73e39f9c7f1  0.0s
 => [internal] load build context                                                                                0.0s
 => => transferring context: 712B                                                                                0.0s
 => [2/6] RUN apt update                                                                                         2.3s
 => [3/6] RUN apt install python3-pip -y                                                                         5.2s
 => [4/6] RUN pip3 install Flask                                                                                 3.4s
 => [5/6] WORKDIR /app                                                                                           0.0s
 => [6/6] COPY . .                                                                                               0.0s
 => exporting to image                                                                                           0.5s
 => => exporting layers                                                                                          0.5s
 => => writing image sha256:15e9c9d58a1fadb1b0390ccd90581c3269a67e584283ea7f9203aabf30a32d36                     0.0s
 => => naming to docker.io/library/myapp                                                                         0.0s

What's Next?
  View summary of image vulnerabilities and recommendations → docker scout quickview

C:\Users\hasan\272>docker run -d -p 5000:5000 myapp
e418ce50ecaf66f9e05e008da24c0065a4ff6958ee51fe144de3b06d786d0834

C:\Users\hasan\272>
```

2. Verifying it by listing the created images on the terminal.

```
C:\Users\hasan\272> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
myapp         latest    15e9c9d58a1f   28 hours ago   1.1GB
```

myapp 15e9c9d58a1f	latest	In use	1 minute ago	1.1 GB	

3. Creating and running a container for the same.

```
C:\Users\hasan\272>docker run -d -p 5000:5000 myapp
0467e41871ba0adab0d718e0d8e45f1dc7de0ddc9715d680c1676a4b1267144d
```

Container CPU usage
0.01% / 800% (8 cores allocated)

Container memory usage
28.87MB / 9.39GB

Show charts

Search

Only show running containers

Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
silly_morse e418ce50ecaf	myapp	Running	0.01%	5000:5000	5 minutes ago	

4. Running the container so as to obtain the app on the localhost:5000

Following are some screenshots to my dockerfile as well as my flask app.

```
FROM python:3

RUN apt update
RUN apt install python3-pip -y
RUN pip3 install Flask

WORKDIR /app

COPY . .

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

C: > Users > hasan > 272 > app.py > ...

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello, World!'

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

C: > Users > hasan > 272 > test_app.py > ...

```
import pytest
from app import app

@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
        yield client

def test_index(client):
    response = client.get('/')
    assert response.data == b'Hello, World!'
    assert response.status_code == 200
```

Part 3 : Hosting the same flask app using vagrant and virtual box. Hosting the same application on localhost and also comparing various metrics for docker container and VM.

1. Setting up a vagrant environment by the vagrant init command. This particular command creates a vagrant file in the directory the app is. We can then edit the file to host an application according to the needs.

```
PS C:\Users\hasan\272> vagrant init
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

2. Then firing up the virtual machine using the command vagrant up. This commands downloads all the requirements for the VM.

```
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'ubuntu/bionic64' could not be found. Attempting to find and install...
    default: Box Provider: virtualbox
    default: Box Version: >= 0
==> default: Loading metadata for box 'ubuntu/bionic64'
    default: URL: https://vagrantcloud.com/ubuntu/bionic64
==> default: Adding box 'ubuntu/bionic64' (v20230607.0.0) for provider: virtualbox
    default: Downloading: https://vagrantcloud.com/ubuntu/boxes/bionic64/versions/20230607.0.0/providers/virtualbox.box
Download redirected to host: cloud-images.ubuntu.com
    default:
==> default: Successfully added box 'ubuntu/bionic64' (v20230607.0.0) for 'virtualbox'!
==> default: Importing base box 'ubuntu/bionic64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'ubuntu/bionic64' version '20230607.0.0' is up to date...
==> default: Setting the name of the VM: 272_default_1694724800648_86747
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 5000 (guest) => 5000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
```

3. Then the vagrant ssh command provides to execute the same virtual environment and enter into it.

```
PS C:\Users\hasan\272> vagrant ssh
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-212-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Sep 14 20:59:44 UTC 2023

  System load:  0.85              Processes:             114
  Usage of /:   4.2% of 38.70GB   Users logged in:       0
  Memory usage: 15%               IP address for enp0s3: 10.0.2.15
  Swap usage:   0%
```

4. Going ahead, providing a command to run the flask app on the localhost.

```
^Cvagrant@ubuntu-bionic:/vagrant$ python3 app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://10.0.2.15:5000/ (Press CTRL+C to quit)
```

5. The app that was hosted.

Hello, World!

Following are the comparisons between the docker container and the VM.

1. **Docker Container**
   - Start Up time : 10s of milliseconds to containerize and host the app.
   - Memory Consumed : 30.45MB
   - CPU Utilization : 0.02% out of 8 allocated cores
2. **VM using vagrant**
   - Start Up time : 3-4 minutes to boot up vagrant up
   - Memory consumed : 1024M