ECC Assignment 3

Dockers and Containers

Server Container

Step 1 : Build the dockerfile which contains the docker configuration

1. Base Image: python:3.8
2. Work Directory: /app
3. Dependency Installation: Using requirements.txt (Libraries: Flask, requests)
4. Port Exposed: 5000
5. Command: python server.py

Step 2: Build the docker image ‘server’

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\server>docker build -t server .
[+] Building 25.5s (10/10) FINISHED                                                                                 docker:default
 => [internal] load .dockerignore                                                                                             0.0s
 => => transferring context: 2B                                                                                               0.0s
 => [internal] load build definition from Dockerfile                                                                          0.0s
 => => transferring dockerfile: 545B                                                                                          0.0s
 => [internal] load metadata for docker.io/library/python:3.8                                                                 1.0s
 => [auth] library/python:pull token for registry-1.docker.io                                                                 0.0s
 => [1/4] FROM docker.io/library/python:3.8@sha256:7264a50439679c2868a99f71a5c9b9831cc082b1d3f05c8643788e1d7914a             20.1s
 => => resolve docker.io/library/python:3.8@sha256:7264a50439679c2868a99f71a5c9b9831cc082b1d3f05c8643788e1d7914af             0.0s
 => => sha256:7264a50439679c2868a99f71a5c9b9831cc082b1d3f05c8643788e1d7914afeb 1.86kB / 1.86kB                                0.0s
 => => sha256:3c3270c231ce82b081e2527c60734395b951c0d0c8b9e6b3a3caa515a701b73a 2.01kB / 2.01kB                                0.0s
 => => sha256:d3a767d1d12e57724b9f254794e359f3b04d4d5ad966006e5b5cda78cc382762 64.13MB / 64.13MB                              6.5s
 => => sha256:8a61cde23424d30d0050cf384b4700250bc5198ce17773c737700fd5e670a06e 7.51kB / 7.51kB                                0.0s
 => => sha256:90e5e7d8b87a34877f61c2b86d053db1c4f440b9054cf49573e3be5d6a674a47 49.58MB / 49.58MB                              6.9s
 => => sha256:27e1a8ca91d35598fbae8dee7f1c211f0f93cec529f6804a60e9301c53a604d0 24.05MB / 24.05MB                              1.4s
 => => sha256:711be5dc50448ab08ccab0b44d65962f36574d341749ab30651b78ec0d4bfd1c 211.07MB / 211.07MB                           11.9s
 => => sha256:48b2d58a56e99520540b5ecafb202ae036a53b5a982845434722691ccba6499a 6.16MB / 6.16MB                               7.1s
 => => extracting sha256:90e5e7d8b87a34877f61c2b86d053db1c4f440b9054cf49573e3be5d6a674a47                                     2.4s
 => => sha256:0a9c4786546c13ac4e6c2f770c6364aee71ca0cd483a848c1492730f334ae5ee 17.28MB / 17.28MB                              9.4s
 => => sha256:5575d20769f15dcea9755eb1473ed34381e479e294ab0cb9fc0a85d52701a388 232B / 232B                                    7.2s
 => => sha256:4711d910ffa25756544938a346f48f59be0b2b66776faa909731f4f9e205bb5e 2.85MB / 2.85MB                                7.6s
 => => extracting sha256:27e1a8ca91d35598fbae8dee7f1c211f0f93cec529f6804a60e9301c53a604d0                                     0.6s
 => => extracting sha256:d3a767d1d12e57724b9f254794e359f3b04d4d5ad966006e5b5cda78cc382762                                     2.7s
 => => extracting sha256:711be5dc50448ab08ccab0b44d65962f36574d341749ab30651b78ec0d4bfd1c                                     5.9s
 => => extracting sha256:48b2d58a56e99520540b5ecafb202ae036a53b5a982845434722691ccba6499a                                     0.3s
```

Step 3: Create the docker volume ‘servervol’

The mount point for this volume will be ‘/serverdata’.

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\server>docker volume create servervol
servervol
```

Step 4: Build the codebase ‘server.py’

Features:

- Generates 1 KB file with random text data in ‘/serverata’
- Calculate the file’s checksum
- Builds a flask application and sets routes for downloading the file and checksum over HTTP.

```
from flask import Flask, send_from_directory
import os
import random
import hashlib
import sys

app = Flask(__name__)

#Set the mount directory of the server's docker volume
datadir = "/serverdata"
# Ensure that the data directory exists
os.makedirs(datadir, exist_ok=True)

filename = "test_data.txt"
filepath = os.path.join(datadir, filename)

# Function to generate a 1KB file with random data
def generate_file():
    with open(filepath, 'w') as file:

file.write(''.join(random.choices('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',
k=1024)))
    print("File generated")

# Function to calculate file checksum
def calculate_checksum():
    hasher = hashlib.md5()
    with open(filepath, 'rb') as file:
        buffer = file.read()
        hasher.update(buffer)
    return hasher.hexdigest()


# Endpoint to download the file
```

```python
@app.route('/download')
def download_file():
    response = send_from_directory(datadir, filename, as_attachment=True)
    response.headers["Content-Disposition"] = f"attachment;
filename={filename}"
    return response

# Endpoint to get the checksum
@app.route('/checksum')
def get_checksum():
    return checksum

if __name__ == '__main__':
    port = int(sys.argv[1]) if len(sys.argv) > 1 else 9000
    generate_file()
    checksum = calculate_checksum()
    app.run(host='0.0.0.0', port=port)
```

Step 5: Run the docker container

Command: docker run -d --name server -v servervol:/servervol -p 9000:9000 server-image

`-v servervol:/serverdata`: This option creates a volume named "servervol" and mounts it to the `/serverdata` directory inside the container. This is used for persisting the data even when the container is stopped or deleted.
The host machine's port is initialized to 9000.

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\server>docker run -d --name server -v servervol:/servervol -p 9000:9000
server-image
e3e7af1a0f81648f6782bdc329f64e60fac3d7ca5664b525cbba758ecdb6a43f
```

Step 6: Check the status of your server container

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\server>docker ps
CONTAINER ID   IMAGE          COMMAND             CREATED       STATUS       PORTS                    NAMES
e3e7af1a0f81   server-image   "python server.py"   3 hours ago   Up 3 hours   0.0.0.0:9000->9000/tcp   server
```

Client Container

Step 1 : Build the dockerfile which contains the docker configuration

1. Base Image: python:3.8
2. Work Directory: /app
3. Dependency Installation: Using requirements.txt (Libraries: requests)
4. Port Exposed: 9000
5. Command: python client.py

Step 2: Build the docker image ‘client-image’

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\client>docker build -t client-image .
[+] Building 11.0s (10/10) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                           0.0s
 => => transferring dockerfile: 517B                                                                           0.0s
 => [internal] load .dockerignore                                                                              0.0s
 => => transferring context: 2B                                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.8                                                  0.7s
 => [auth] library/python:pull token for registry-1.docker.io                                                  0.0s
 => [1/4] FROM docker.io/library/python:3.8@sha256:7264a50439679c2868a99f71a5c9b9831cc082b1d3f05c8643788e1d7914afeb    0.0s
 => [internal] load build context                                                                              0.0s
 => => transferring context: 1.39kB                                                                            0.0s
 => CACHED [2/4] WORKDIR /app                                                                                  0.0s
 => [3/4] COPY . /app                                                                                          0.1s
 => [4/4] RUN pip install --no-cache-dir -r requirements.txt                                                   9.7s
 => exporting to image                                                                                         0.3s
 => => exporting layers                                                                                        0.3s
 => => writing image sha256:3f0b45463c67aee6b80c0ed53495031705a59416d2d0c7307dfd2b5dba25a64d                   0.0s
 => => naming to docker.io/library/client-image                                                                0.0s
```

Step 3: Create the docker volume ‘clientvol’

The mount point for this volume will be ‘/clientdata’.

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\client>docker volume create clientvol
clientvol
```

Step 4: Build the codebase ‘client.py’

<u>Features:</u>

- Downloads the file and checksum from the server.
- Verifies the file integrity using the checksum.

```python
import requests
import os
import hashlib

server_url = 'http://host.docker.internal:9000'
```

```
#Get the current working directory
cwd = os.getcwd()
datadir = os.path.join(cwd, 'clientdata')
os.makedirs(datadir, exist_ok=True)
filename = 'test_data.txt'
filepath = os.path.join(datadir, filename)

def download_file():
    response = requests.get(f'{server_url}/download')
    if response.status_code == 200:
        with open(filepath, 'wb') as file:
            file.write(response.content)
    else:
        print('Failed to download the file')
        exit(1)

def download_checksum():
    response = requests.get(f'{server_url}/checksum')
    if response.status_code == 200:
        return response.text
    else:
        print('Failed to download the checksum')
        exit(1)

def calculate_checksum():
    hasher = hashlib.md5()
    with open(filepath, 'rb') as file:
        buffer = file.read()
        hasher.update(buffer)
    return hasher.hexdigest()


def verify_checksum():
    download_file()
    server_checksum = download_checksum()
    calculated_checksum = calculate_checksum()
    if server_checksum == calculated_checksum:
        print('File verification using checksum is successful')
    else:
```

```
        print('File verification failed.')

if __name__ == '__main__':
    verify_checksum()
```

Step 5: Run the docker container

Command: docker run -d –name client -v clientvol:/clientdata client-image

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\client>docker run -d --name client -v clientvol:/clientdata client-image
                                                                                                                          c
f0fa2676874a1be2113418a26b54c28f976308525d325a428acdd9e1afba8c9
```

Communication between Client and Server Container

Step 1: Check the containers' status.

It can be observed that the server container is still running and client exited after running successfully.

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\client>docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED       STATUS                   PORTS
              NAMES
cf0fa2676874   client-image                   "python client.py"       3 hours ago   Exited (0) 3 hours ago
              client
e3e7af1a0f81   server-image                   "python server.py"       3 hours ago   Up 3 hours               0.0.0.0:900
0->9000/tcp   server
153617d8928b   docker/welcome-to-docker:latest   "/docker-entrypoint.…"   8 hours ago   Exited (0) 8 hours ago
              welcome-to-docker
```

Step 2: Check the server logs

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\server>docker logs e3e7af1a0f81648f6782bdc329f64e60fac3d7ca5664b525cbba7
58ecdb6a43f
File generated
['server.py']
 * Serving Flask app 'server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:9000
 * Running on http://172.17.0.2:9000
Press CTRL+C to quit
172.17.0.1 - - [12/Dec/2023 01:25:58] "GET / HTTP/1.1" 404 -
172.17.0.1 - - [12/Dec/2023 01:26:05] "GET /download HTTP/1.1" 200 -
172.17.0.1 - - [12/Dec/2023 01:30:38] "GET / HTTP/1.1" 404 -
172.17.0.1 - - [12/Dec/2023 01:30:50] "GET /download HTTP/1.1" 200 -
172.17.0.1 - - [12/Dec/2023 01:35:12] "GET /download HTTP/1.1" 200 -
172.17.0.1 - - [12/Dec/2023 01:35:12] "GET /checksum HTTP/1.1" 200 -
```

We can see in the last two requests that there were successful calls (Response 200) to '/download' and '/checksum' endpoints from the client.

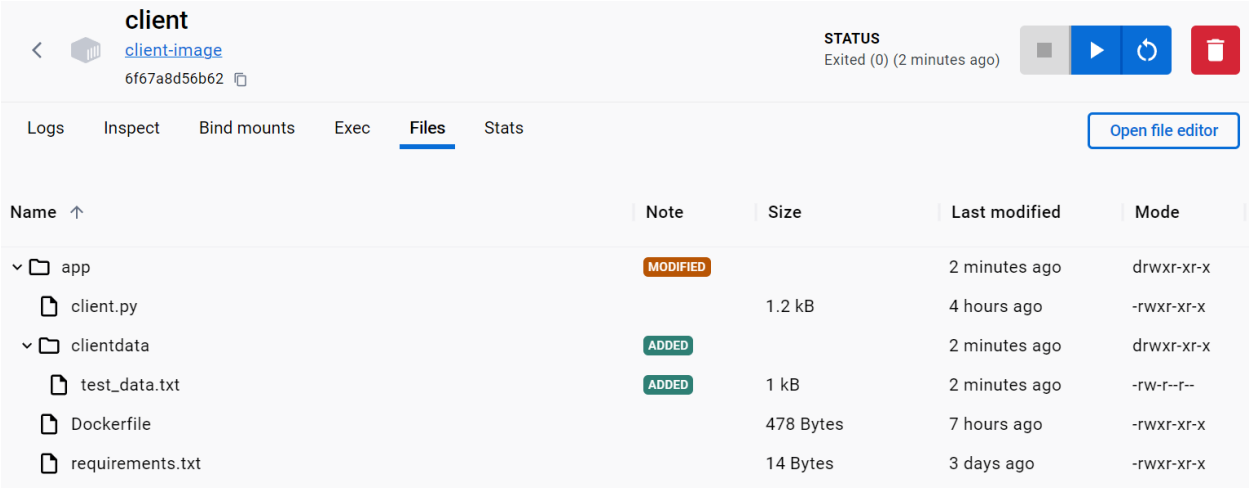
Step 3: Check the client logs

```
C:\Users\sabad\OneDrive\Documents\Docker_Assignment\server>docker logs cf0fa2676874a1be2113418a26b54c28f976308525d325a428acd
d9e1afba8c9
File verification using checksum is successful

C:\Users\sabad\OneDrive\Documents\Docker_Assignment\server>
```

From the client container logs, we can see that the verification between checksum received from server over HTTP and checksum calculated by the client has been successful. Hence, client and server have successfully communicated with each other.

Step 4: Verify downloaded file through Docker UI



Verification through Interactive console

Here we have run the client container in interactive mode (-it) which allows us to access the container's directories.
However, we have to manually run 'python client.py' to download our test file.