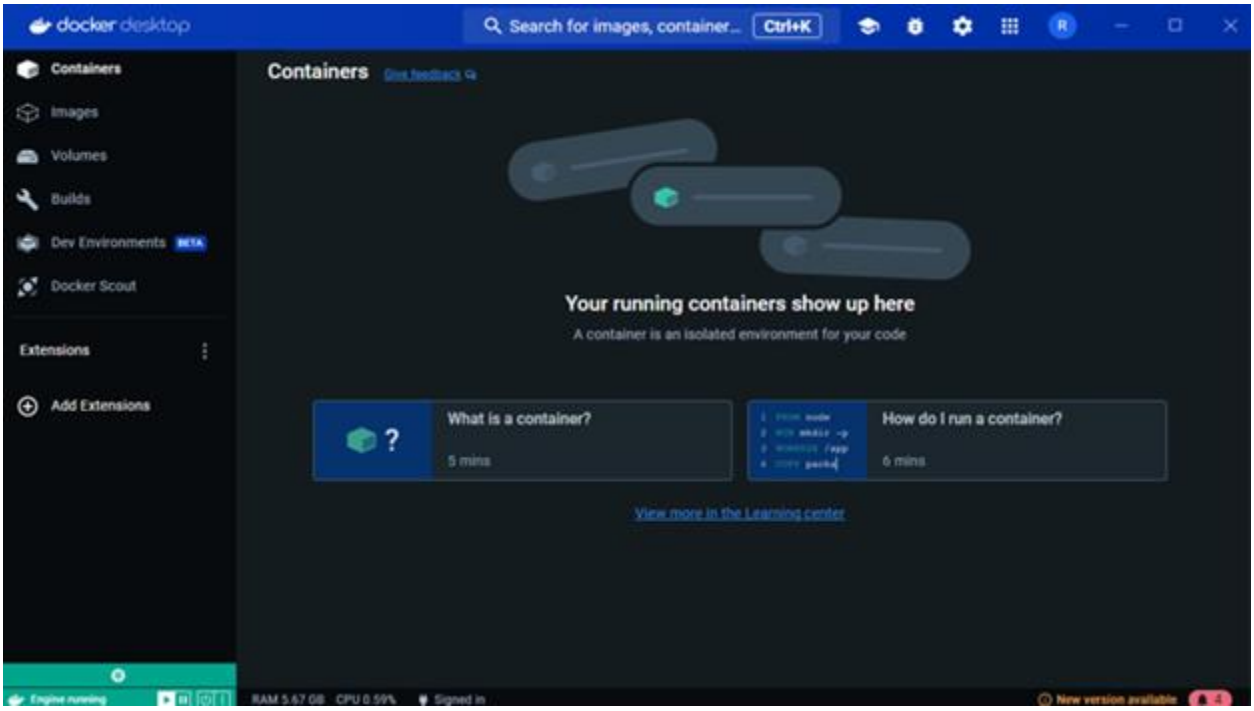
Rohan Vasantbhai Darji UID: 2001270724

ENGR-E 516: ASSIGNMENT 02

DOCKERS AND CONTAINERS

I have performed this assignment on local machine. I downloaded the Docker Desktop as it sets up the Docker daemons on my computer.



For the Server-Client communication using docker containers for both, I started with having a systematic directory tree (as shown below).

ECC_Assignment_03/

|----server/

|----|----Dockerfile

| |----server.js

|----client/

|----|----Dockerfile

| |----client.js

|----docker-compose.yml

As for the base image I have used the Node image for both server and client containers.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03> docker pull node
Using default tag: latest
latest: Pulling from library/node
1468e7ff95fc: Already exists
2cf9c2b42f41: Already exists
c4c40c3e3cdf: Already exists
c05cc1123d7e: Already exists
04f2356c02d2: Pull complete
fd0103e43e65: Pull complete
8a7c6202e2f9: Pull complete
82cce69906c7: Pull complete
Digest: sha256:bda531283f4bafd1cb41294493de89ae3c4cf55933da14710e46df970e77365e
Status: Downloaded newer image for node:latest
docker.io/library/node:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview node
PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03>
```

As per the requirements of the assignment I have created two volumes: 1. servervol and 2. clientvol

- Servervol is for the server to store data

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03> docker volume create servervol
servervol
```

- Clientvol is for the client to store data

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03> docker volume create clientvol
clientvol
```

I created the network using the following command:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03> docker network create rohan
a2dcfd7c4d09aa51917626befee1f6194da732c45ebf1026bbbbdd6e8f338f25
```

The server.js file is explained below:

1. **Module Imports:** The script begins by importing the necessary modules:
    - net for creating the TCP server
    - fs for file system operations

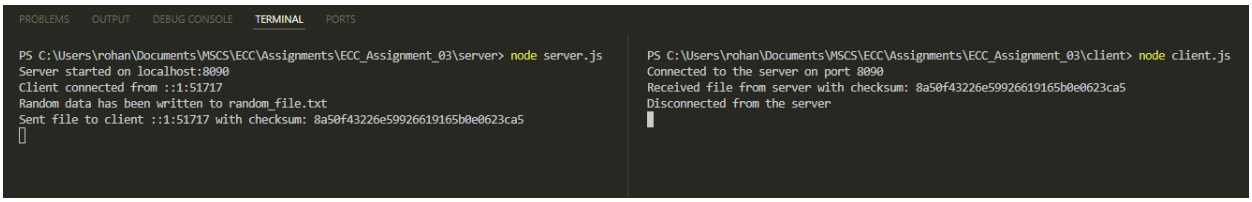
   - crypto for generating random bytes and calculating MD5 hashes
2. **Server Configuration:** The server's host and port are set based on command line arguments, with default values of 'localhost' and 8090.
3. **Random data generation:** A function generateRandomData is defined to generate random data of a specified length. This function is then used to generate a random data of size FILE_SIZE.
4. **Server Creation:** A TCP server is created using net.createServer. This server listens for client connections and performs several actions when a client connects:
   - It logs the client's address and port.
   - It calculates the MD5 hash of the random data.
   - It writes the random data to a file.
   - If writing the file is successful, it reads the file and sends the file data to the client.
   - It logs the client's disconnection and any errors.
5. **Server Error Handling:** The server listens for 'error' events and logs any errors that occur.
6. **Server Start:** Finally, the server starts listening on the specified host and port.

The client.js file is explained below:

1. **Module Imports:** The script begins by importing the necessary modules:
   - net for creating the TCP client
   - fs for file system operations
   - crypto for calculating MD5 hashes
2. **Client Configuration:** The client's host and port are set based on command line arguments, with default values of 'localhost' and 8090.
3. **Client Creation:** A TCP client is created using new net.Socket().
4. **Client Connection:** The client connects to the server using clientSocket.connect(). Once connected, it sends a message to the server requesting the randomly generated file.
5. **Data Reception:** The client listens for 'data' events with clientSocket.on("data", ...). When data is received from the server:
   - It logs the received data.
   - It calculates the MD5 hash of the received data.
   - It writes the received data to a file.
   - It sends an acknowledgement message to the server.
6. **Connection Closure:** The client waits for 4 minutes, then closes the connection.
7. **Event Handling:** The client listens for 'end' and 'error' events and logs when the connection ends or if an error occurs.

Testing the code on local machine to check if it is working properly or not:

- Instead of running the files on docker directly, I ran the files on the local system to check if there are any errors or bugs in the code. After running I verified that the code is working properly and the file is being transferred from server to client.



- Below is the 'random_file.txt' file that the client had received from the server.

Creating dockerfiles for both server and client:

Dockerfile for server.js is given below:

```
server > Dockerfile > ...
1   # Use the official Node.js 14.17.0 image
2   FROM node:14.17.0-alpine
3
4   # Set working directory inside the container
5   WORKDIR /app
6
7   # Copy server.js to the working directory
8   COPY server.js /app/server.js
9
10  # create a dir for serverdata
11  RUN mkdir /app/serverdata
12
13  EXPOSE 8090
14
15  # Command to run the server
16  CMD ["node", "server.js" , "0.0.0.0", "8090"]
```

Dockerfile for client.js is given below:

client > Dockerfile > ...

```
1   # Use the official Node.js 14.17.0 image
2   FROM node:14.17.0-alpine
3
4   # Set working directory inside the container
5   WORKDIR /app
6
7   # Copy client.js to the working directory
8   COPY client.js /app/client.js
9
10  # Create a directory for the client
11  RUN mkdir /app/clientdata
12
13  # Command to run the client
14  CMD ["node", "client.js", "server", "8090"]
```

For building the docker image, I ran the following command in the terminal:

Server image build status:

```
PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03\server> docker build -t server -f Dockerfile .
[+] Building 1.2s (10/10) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                          0.0s
 => => transferring dockerfile: 398B                                                                          0.0s
 => [internal] load metadata for docker.io/library/node:14.17.0-alpine                                        0.8s
 => [auth] library/node:pull token for registry-1.docker.io                                                   0.0s
 => [internal] load .dockerignore                                                                             0.0s
 => => transferring context: 2B                                                                               0.0s
 => [1/4] FROM docker.io/library/node:14.17.0-alpine@sha256:f07ead757c93bc5e9e79978075217851d45a5d8e5c48eaf823e7f1  0.0s
 => [internal] load build context                                                                             0.0s
 => => transferring context: 3.26kB                                                                           0.0s
 => CACHED [2/4] WORKDIR /app                                                                                 0.0s
 => [3/4] COPY server.js /app/server.js                                                                       0.0s
 => [4/4] RUN mkdir /app/serverdata                                                                           0.3s
 => exporting to image                                                                                        0.0s
 => => exporting layers                                                                                       0.0s
 => => writing image sha256:a37c8d44af0537c0c2109a91bee8fa5b7d8a1f7fcb0b5d5bf95d72d31fca035f                  0.0s
 => => naming to docker.io/library/server                                                                     0.0s

View build details: docker-desktop://dashboard/build/default/default/b87onhhifta26gnhfhkt0q3j8

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03\server>
```

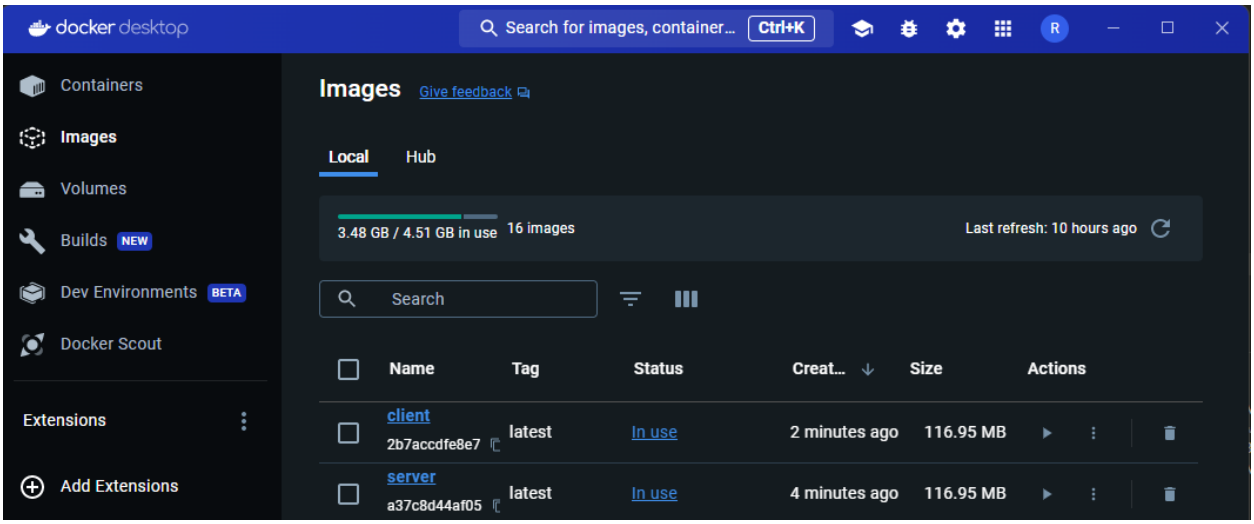
Client image build status:

```
PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03\client> docker build -t client -f Dockerfile .
2024/04/25 00:30:34 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 0.9s (9/9) FINISHED                                                                                docker:default
 => [internal] load build definition from Dockerfile                                                                      0.2s
 => => transferring dockerfile: 387B                                                                                      0.1s
 => [internal] load metadata for docker.io/library/node:14.17.0-alpine                                                    0.4s
 => [internal] load .dockerignore                                                                                         0.1s
 => => transferring context: 2B                                                                                           0.1s
 => [1/4] FROM docker.io/library/node:14.17.0-alpine@sha256:f07ead757c93bc5e9e79978075217851d45a5d8e5c48eaf823e7f12d9bbc1d  0.0s
 => [internal] load build context                                                                                         0.0s
 => => transferring context: 1.24kB                                                                                       0.0s
 => CACHED [2/4] WORKDIR /app                                                                                             0.0s
 => [3/4] COPY client.js /app/client.js                                                                                   0.0s
 => [4/4] RUN mkdir /app/clientdata                                                                                       0.2s
 => exporting to image                                                                                                    0.0s
 => => exporting layers                                                                                                   0.0s
 => => writing image sha256:2b7accdfe8e79ed779596a9ca5c472f4d31735baf67d18bc366de76944302739                              0.0s
 => => naming to docker.io/library/client                                                                                 0.0s

View build details: docker-desktop://dashboard/build/default/default/61heks5xro6o0a55kcjl0nuc9

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03\client>
```

Images of client and server created in the docker desktop:



As we can see the images of both Server and Client have been created in the docker desktop using the commands as mentioned in the assignment.

Next thing that I did was creating a container and running on docker:

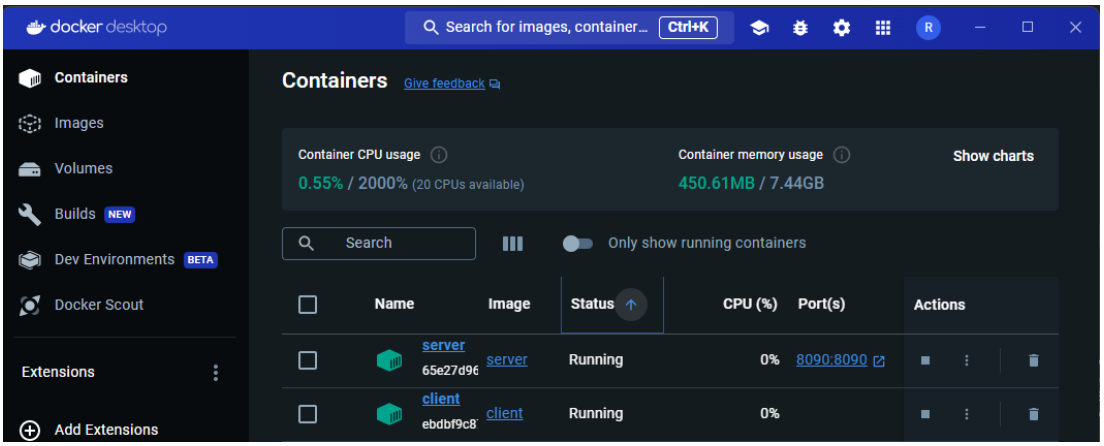
Server run command:

```
PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03\server> docker run -v
 servervol:/app/serverdata -p 8090:8090 -d --name server --network rohan server
fc128fba754368e25e66e6732a068955a7a0df8ae8ce82a31fee9bce55d0a670
```

Client run command:

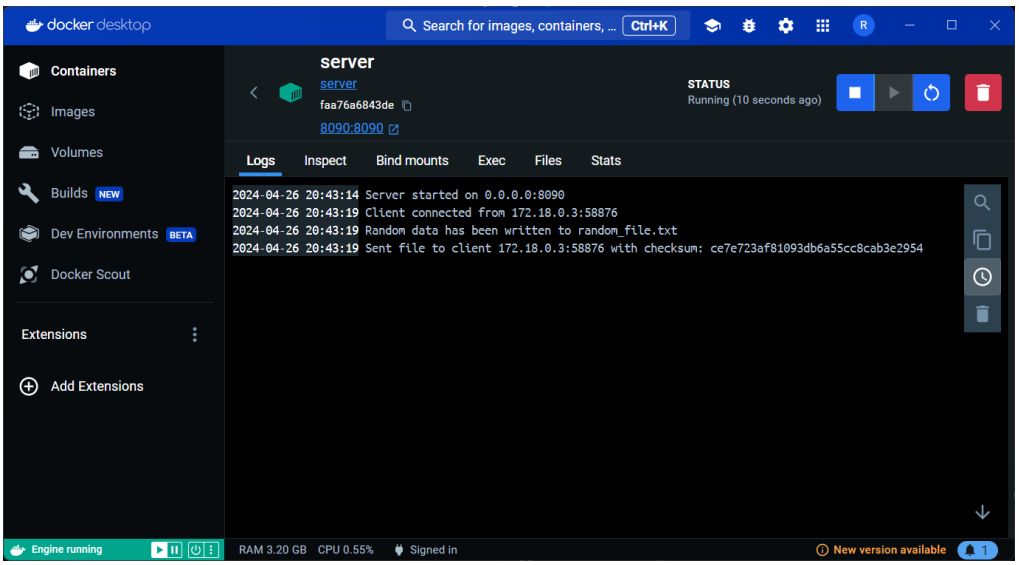
```
PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03\client> docker run -v
 clientvol:/app/clientdata -d --name client --network rohan client
89724097762df7e569cb7bfda17545a61fb6171dea2045d72a0a39f863f4cbc9
```

Running containers in docker:



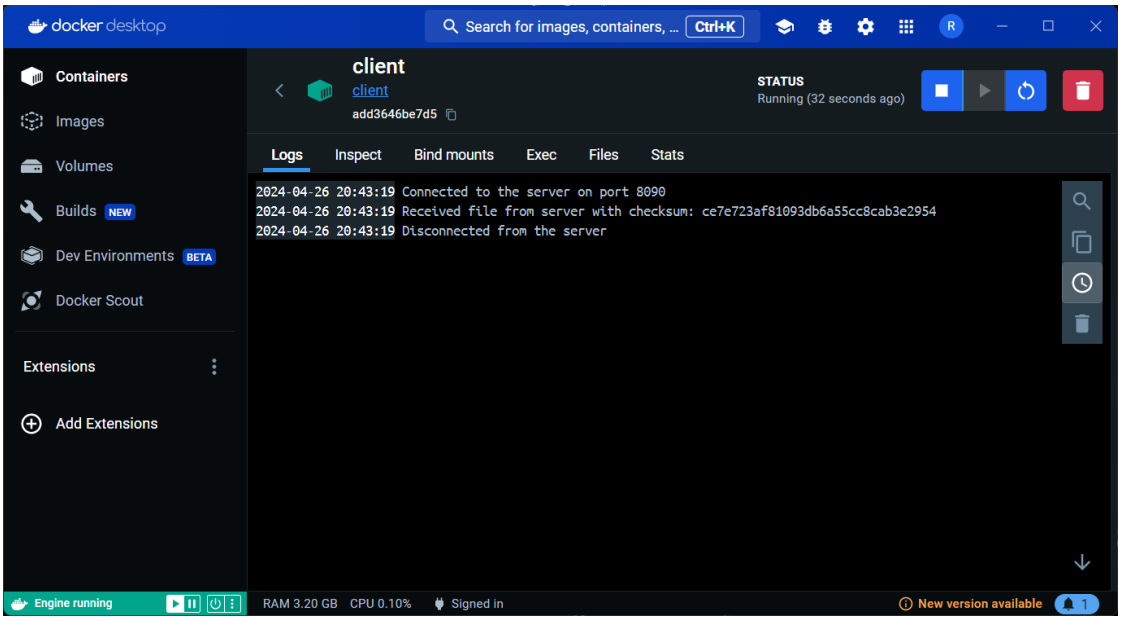
As we can see the containers of server and client have been created.

Logs of server:



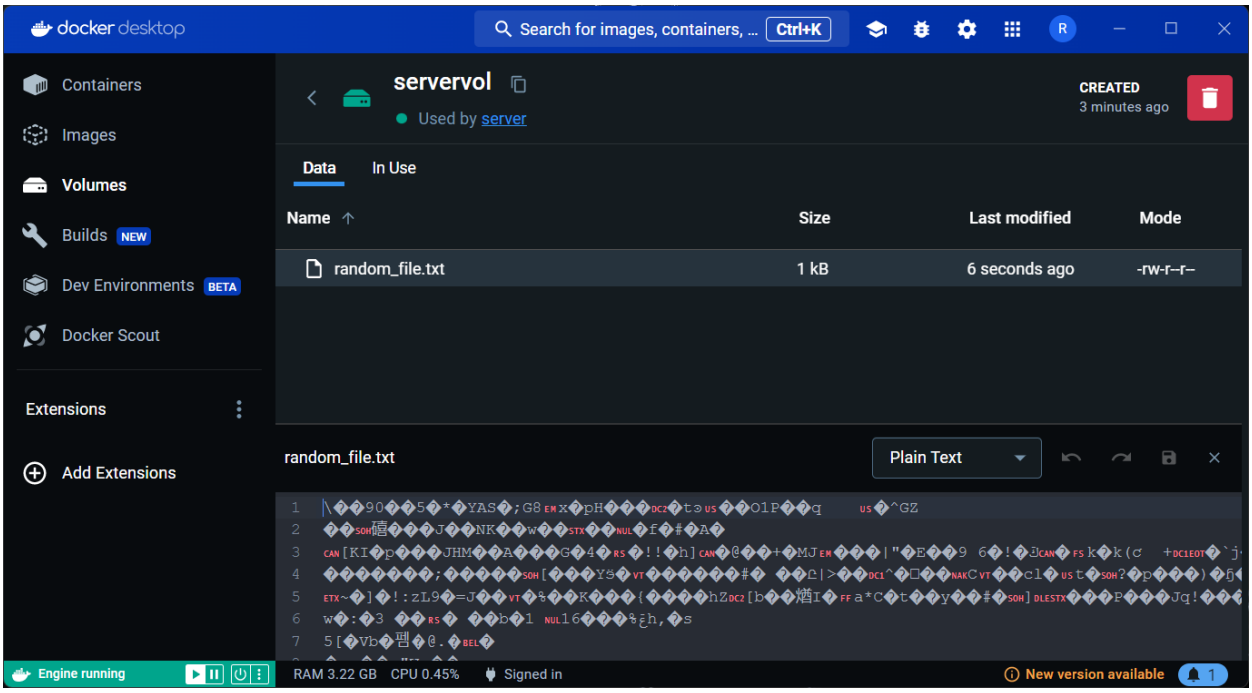
- From the logs, we can see that the file was generated and sent to the client.

Logs of client:

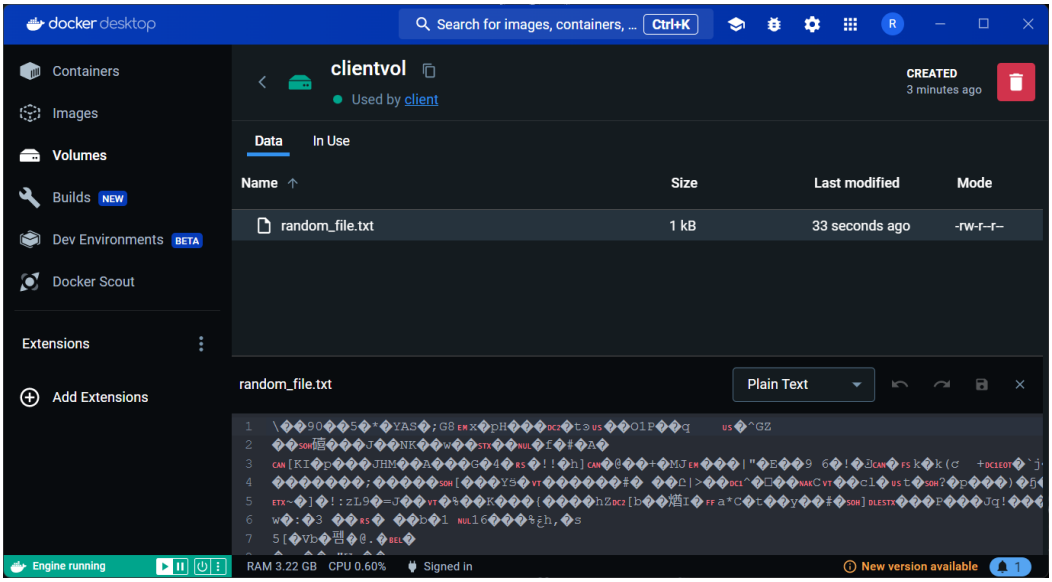


Here, we can see the file received from the server.

Checking the output file in the volumes:



Here, we can see that the output file was stored in the servervol and the ‘random_file.txt’ is also visible.



We can also see the same random file data stored in the clientvol which is received from the server.

<u>Now to confirm both the containers are running on the same network i.e., rohan, we use the command ‘docker inspect rohan’</u>

```
PS C:\Users\rohan\Documents\MSCS\ECC\Assignments\Assignment_03\server> docker inspect rohan
[
    {
        "Name": "rohan",
        "Id": "a2dcfd7c4d09aa51917626befee1f6194da732c45ebf1026bbbbdd6e8f338f25",
        "Created": "2024-04-25T04:22:39.133727371Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "86848d44cde063de32fc4b813d230d640246b93f5b004a4a68d2bfa6b2a6a619": {
                "Name": "client",
                "EndpointID": "01cf444518f9c61a5b00da60f05444d8892bd7a8097881614ea082b4a6a5c6a9",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            },
            "e1d40c4b23470db2a1115c24178666e9d56b59b51203e390ff8141d647011db0": {
                "Name": "server",
                "EndpointID": "439e26bb94e22ff991dce5411d8f7d208c0e8df3dc0866e801b7a8b84148d596",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

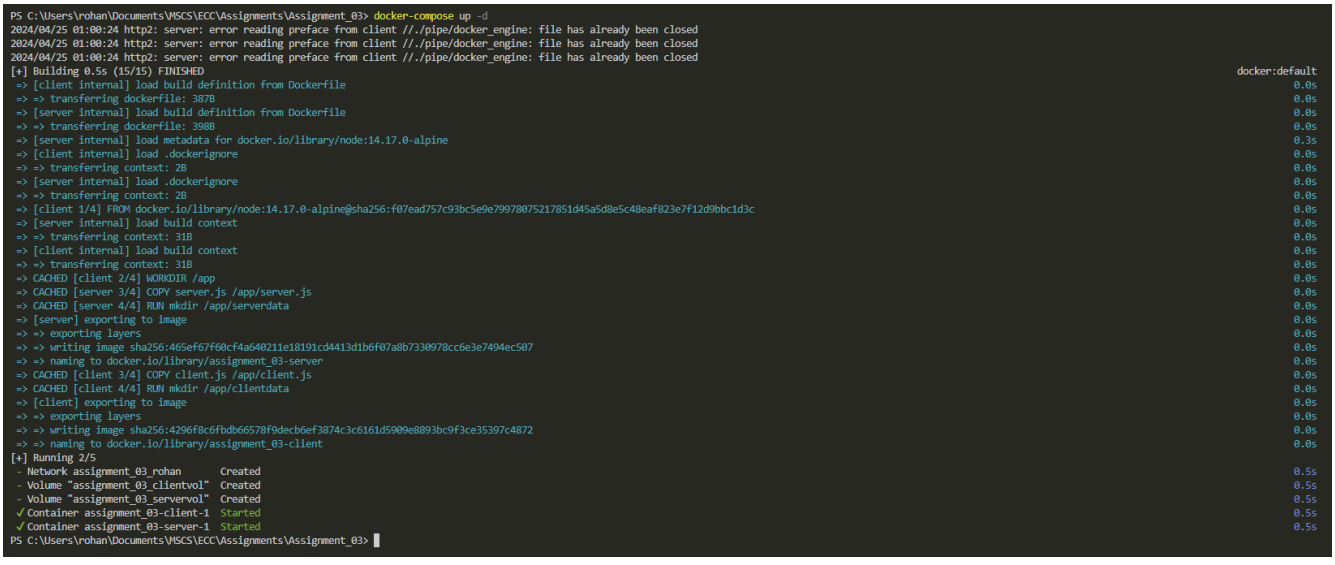
In the above image, we can see that both the containers are running on the same network rohan.

Automating the entire process of creating docker containers and running them:

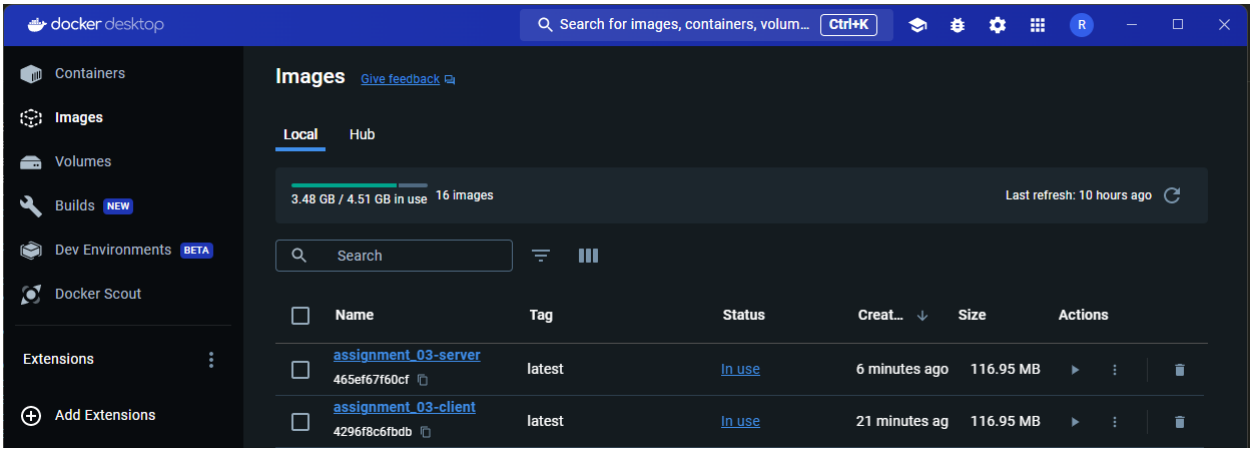
Now for the last file before building the images and running the container and getting the results was the ‘docker-compose.yml’. And for this file I referred the Docker documentation since we also had to have a user-defined network which will be necessary for the communication between the server and client containers and mentioned the following things:

- Defined Docker Compose version as '3'.
- Defined the two services: 'server' and 'client'.
- Configured the 'server' service:
    - Build using 'Dockerfile' in 'server' directory.
    - Attach to 'rohan' network.
    - Mount 'servervol' volume to '/serverdata' in the container.
- Then Configured 'client' service:
    - Build using 'Dockerfile' in 'client' directory.
    - Attach to 'rohan' network.
    - Mount 'clientvol' volume to '/clientdata' in the container.
    - Specified its dependency on 'server' service.
- Defined a 'rohan' network (user-defined network).
- Lastly, defined both the volumes (as per the assignment): 'servervol' and 'clientvol'.

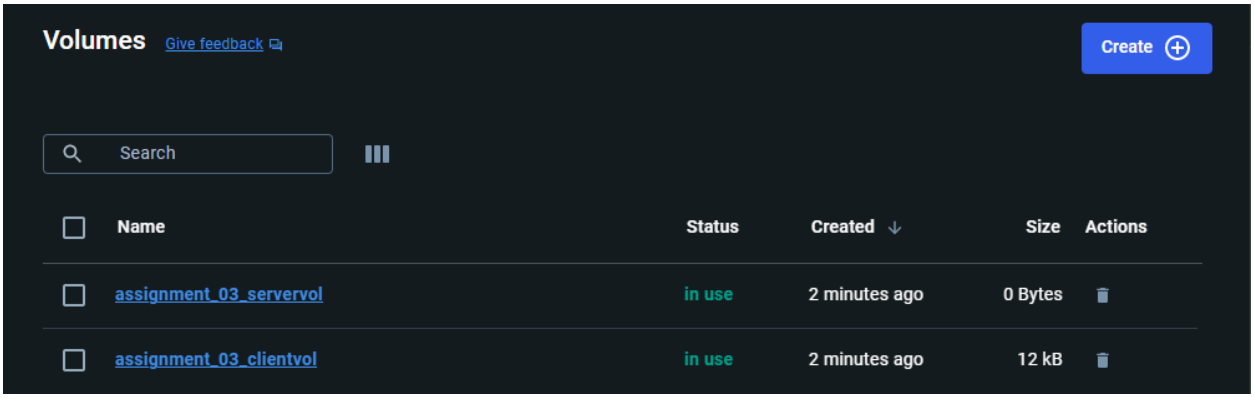
Running the docker-compose.yml using the command ‘docker-compose up -d’:



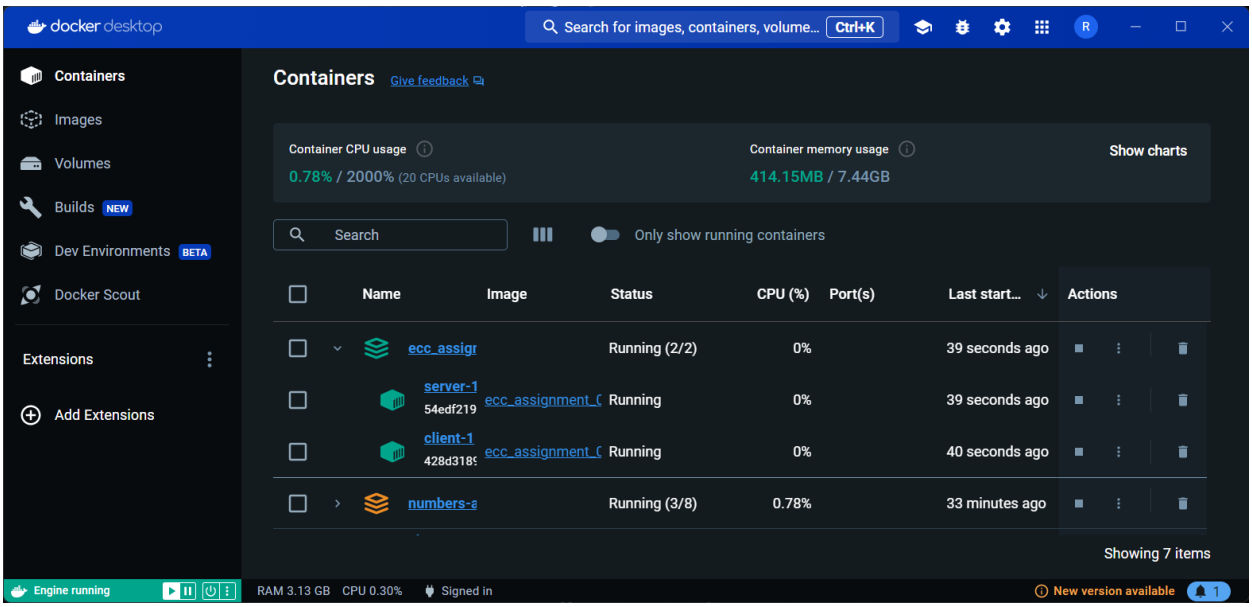
After running the above command we can see the server and client images created in the docker desktop:



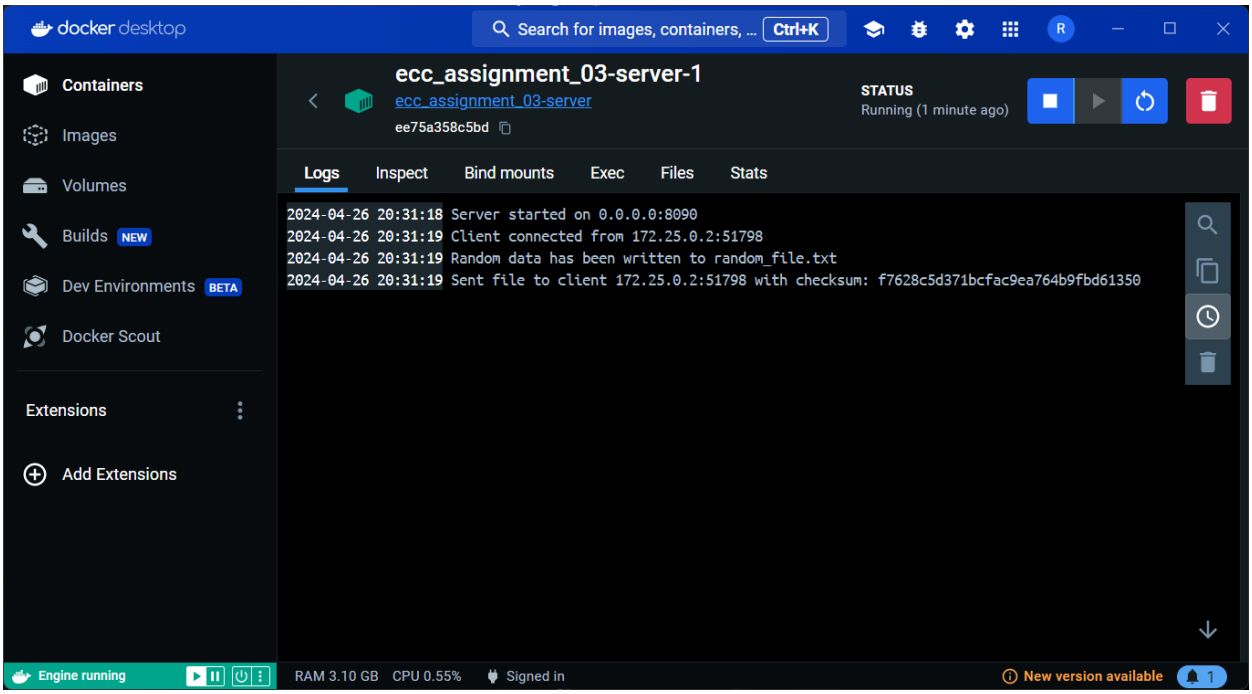
We can also see the volumes of client and server created in the docker desktop:



We can also see the containers created under the assignment_03:

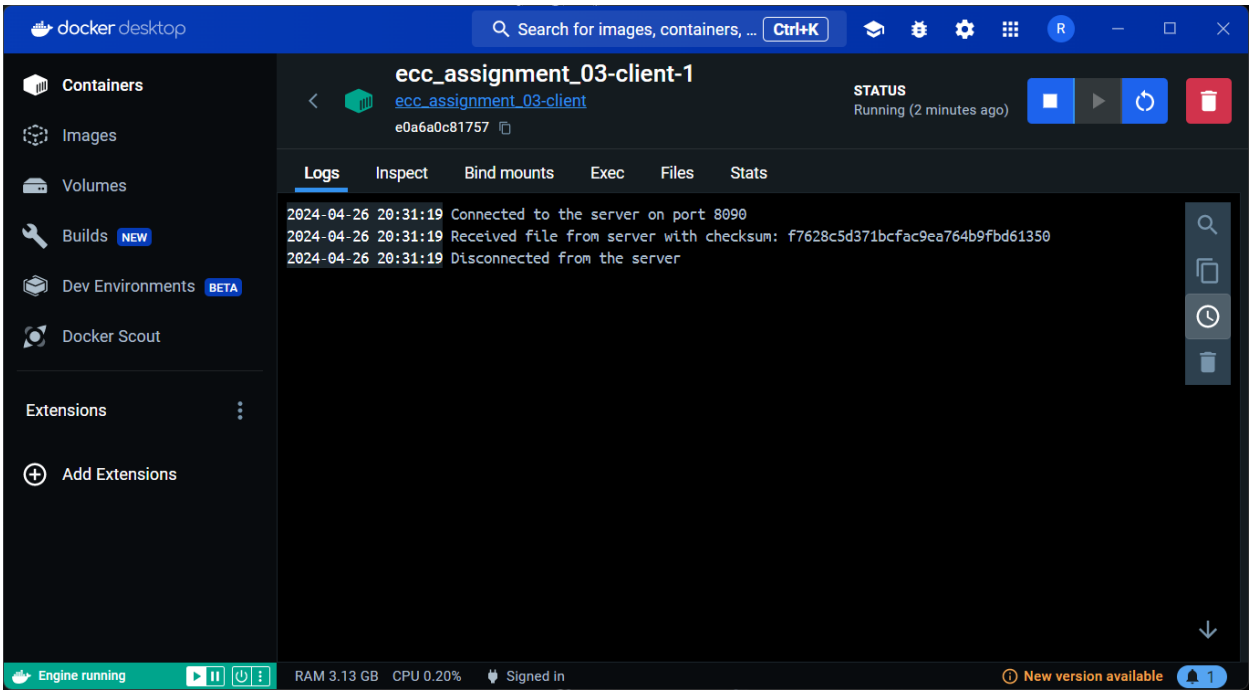


Logs of server:



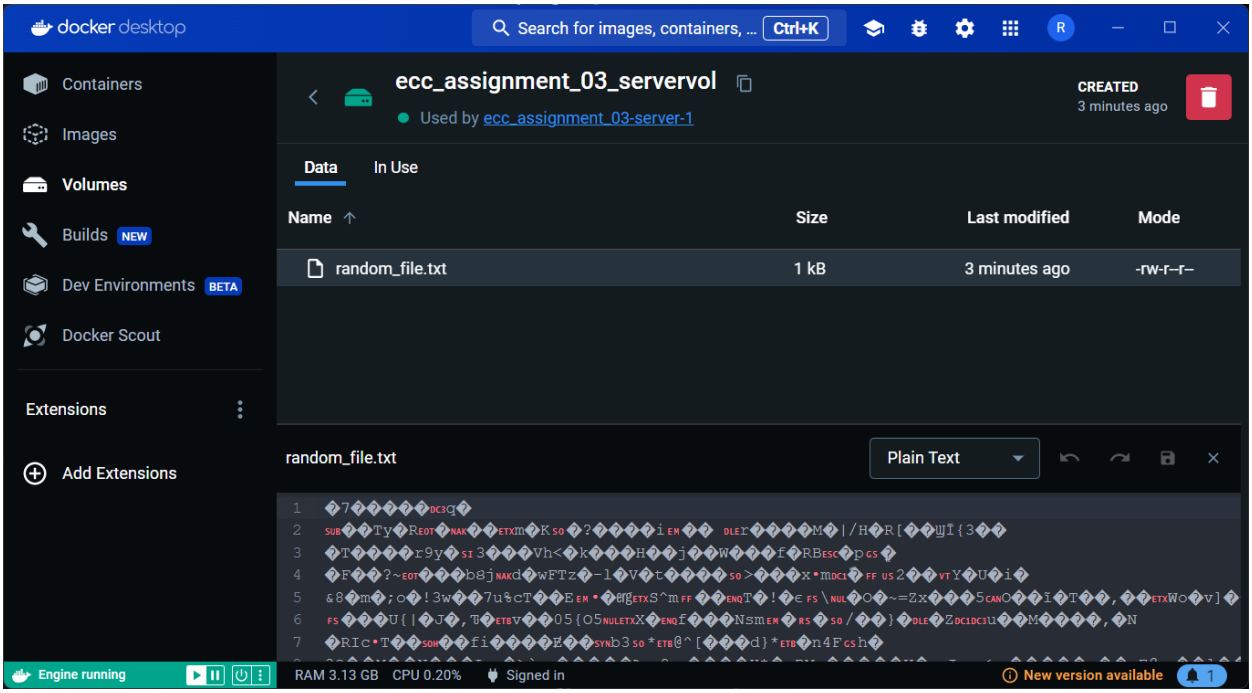
Here, we can see the random_file.txt being generated and sent to the client.

Logs of client:

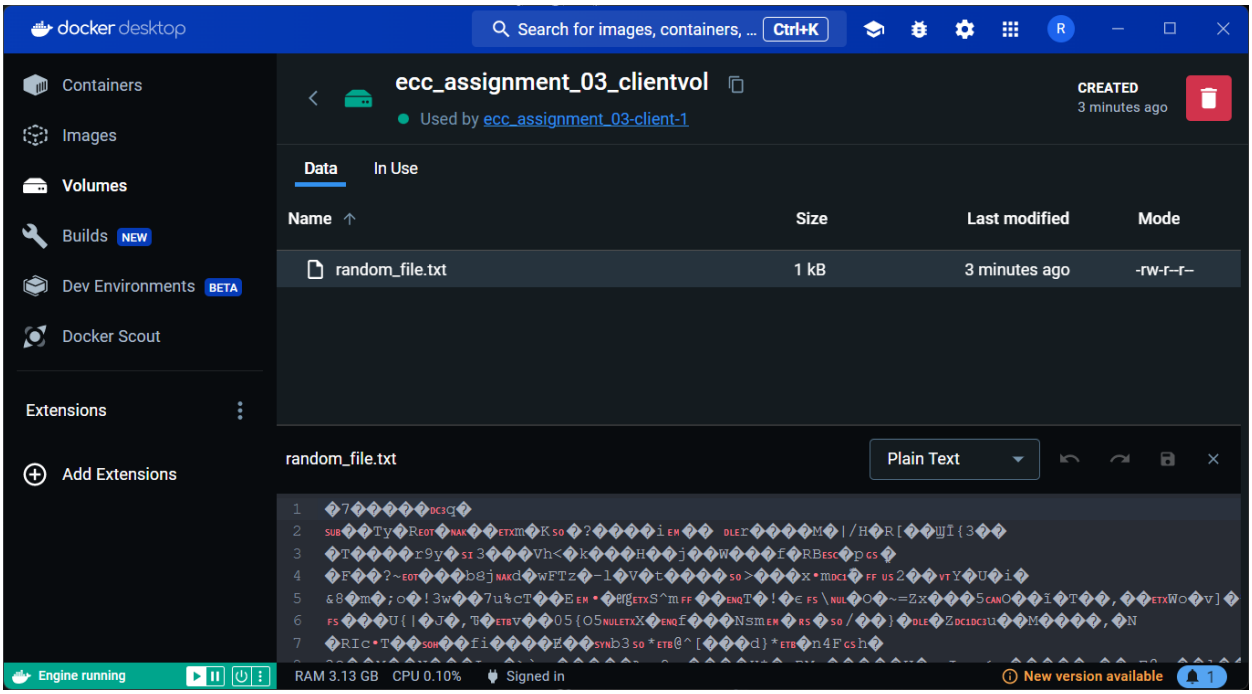


In the above image we can see the file sent from the server is being received by the client.

Output file random_file.txt which was sent from server to client stored in servervol:



Output file random_file.txt sent from server to client stored in clientvol:



References:

1. docker run | Docker Docs
2. Volumes | Docker Docs
3. How to make a client and server communicate with each other in a user defined network in docker? - General - Docker Community Forums
4. How do I calculate the MD5 checksum of a file in Python? - Stack Overflow