ENGR 516- Engineering Cloud Computing Assignment 3

Hemesh Raaja Malathi

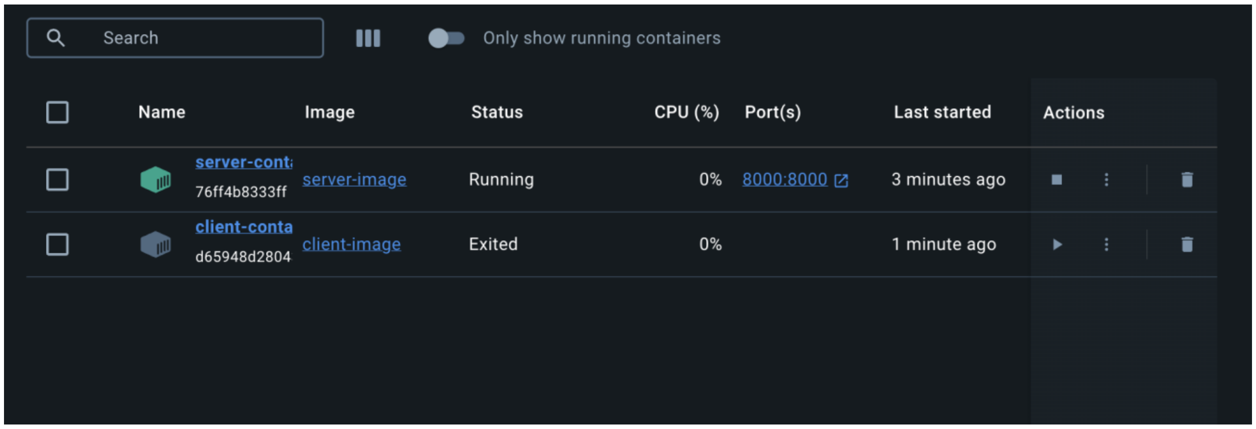
heraaj@iu.edu

Code and readme could be found in this repo along with logic behind the code

Goal is to create two docker volumes, server vol and client vol, and then configure both the vols to the same network(here I am using default network, so on additional commands needs to be executed) , so they can communicate with each other. Now send an file from the server side, it can be an random 1kb file along with check sum, and when client receives it, first check sum should be verified, if valid, and file is received it will be turned off, and we can see this process in the log. To achieve I have followed the following steps mentioned below.

1) **Install docker desktop:**

   Docker desktop could be installed from official website, choosing the appropriate OS and other config. Now that docker is up and running, lets write dockerfile and python script according to the question.



Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
server-conti 76ff4b8333ff	server-image	Running	0%	8000:8000	3 minutes ago	
client-conta d65948d2804	client-image	Exited	0%		1 minute ago	

2) **Python script logic:** (Code will be given in the uploaded in the github, and link could be found at the top of the file)

   Server-side logic:

I have written python code for server side, which creates, 1kb file with random text, which will be initially stored in the '/serverdata' and then send the file to client along with an check sum which it has created. In this line first line will be check sum of the file.

1) **Initialize a port:** Initialize an port an listen/wait for connection from client
2) **Accept Connection:** Once it receives an connection request, and client sends an message indicating to connect, it accepts it. By chance if client gets disconnected, the code will be still running, just re-executing the client would be fine.
3) **Generate and Send Data:** Now generate an random 1kb file and along with checksum being calculated, and sent to client.
4) Terminate the connection

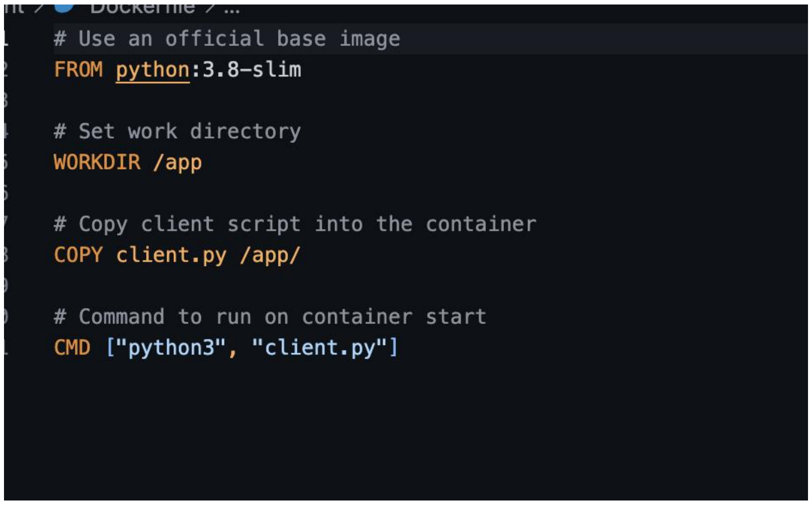
Client-side logic:
First client and server needs to configured to the same network before communicating (default network has been used), and once client establishes connection to server. The server will send file, upon receiving it, client will first check the check sum from the first line of the file, and if the value is correct, it will store the received file in the '/clientdata'.

1) **Establish Connection:** One socket will be created and connection with server will be established using specified port number and getting message would be sent;.
2) **Receive Data:** Gets the checksum from server, hash values will be used to indicate the end of it.
3) **Get data and write in file:** As connection being established server sends one kb file in small chunks and client starts writing this random text in the client_side.txt
4) **Verify File Integrity:** After writing the content, client now calculates the checksum of the file, and verifies both the server side checksum which was initially received to be equal or not.
5) If there is an mismatch stop the process

Note: You could say that let this loop run infinitely, until it matches, but it is an bad idea as we don't what breaks the code, and it ends up using the all the resources, leading to total shutdown of mac, so connection would be ended, based on log values, the error could be identified, and rerun the code.

3. **Docker file for server and client:**
Client side logic:

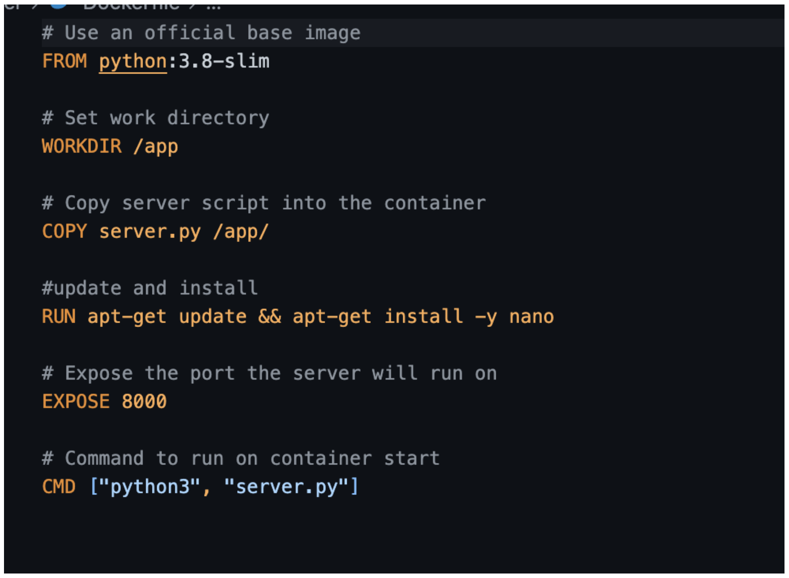
Basic docker file which uses python slim image as base, since it is an smaller application, and create work directory and copy the code and execute it the code.



```dockerfile
# Use an official base image
FROM python:3.8-slim

# Set work directory
WORKDIR /app

# Copy client script into the container
COPY client.py /app/

# Command to run on container start
CMD ["python3", "client.py"]
```

Server Side Logic:

Basic docker file which uses python slim image as base similar to the cleint, since it is an smaller application, and create work directory and copy the code. Now install required files, and set an port which could be used for communication. And run the code.

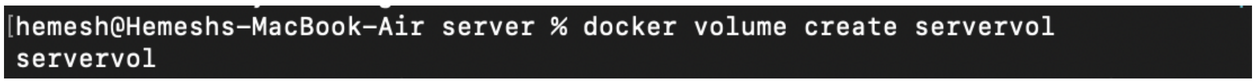


```dockerfile
# Use an official base image
FROM python:3.8-slim

# Set work directory
WORKDIR /app

# Copy server script into the container
COPY server.py /app/

#update and install
RUN apt-get update && apt-get install -y nano

# Expose the port the server will run on
EXPOSE 8000

# Command to run on container start
CMD ["python3", "server.py"]
```

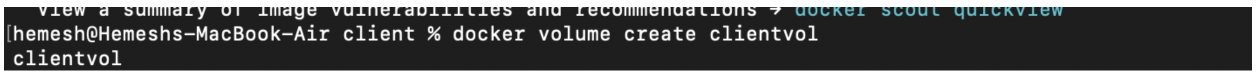
4) Volume Creation:

Created an volume where the server and client data could be stores respectively. The naming convention was according to the question ‘Servervol” and “Clientvol”.

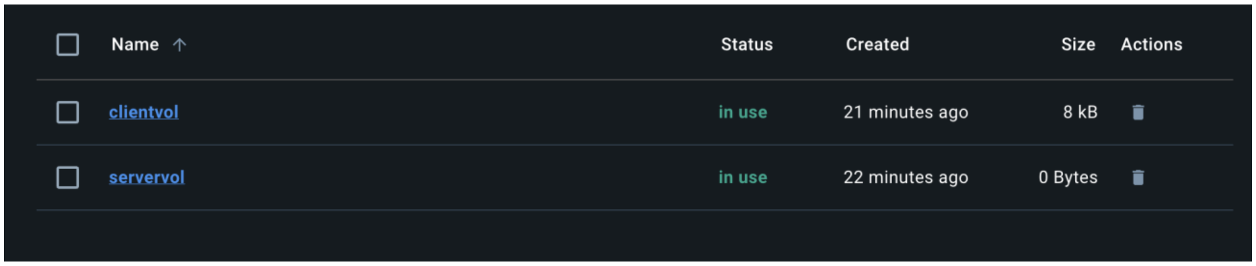
Command: docker volume create <volume name>



```
hemesh@Hemeshs-MacBook-Air server % docker volume create servervol
servervol
```



```
  View a summary of image vulnerabilities and recommendations → docker scout quickview
hemesh@Hemeshs-MacBook-Air client % docker volume create clientvol
clientvol
```

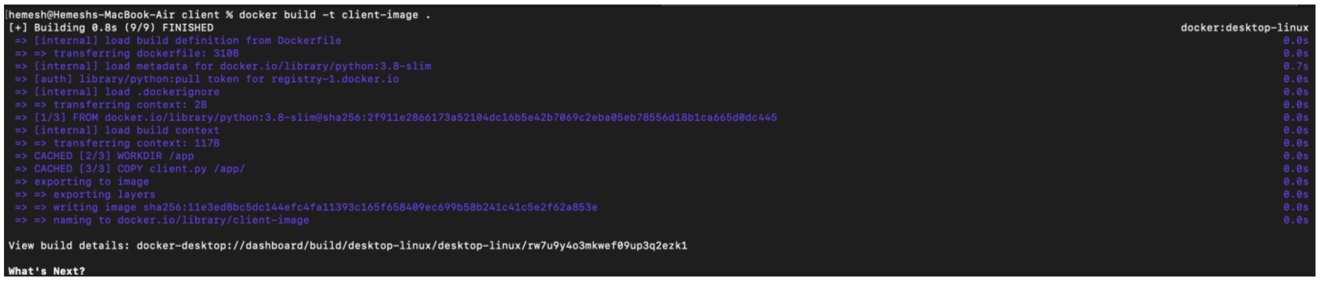


	Name ↑	Status	Created	Size	Actions
☐	clientvol	in use	21 minutes ago	8 kB	🗑
☐	servervol	in use	22 minutes ago	0 Bytes	🗑

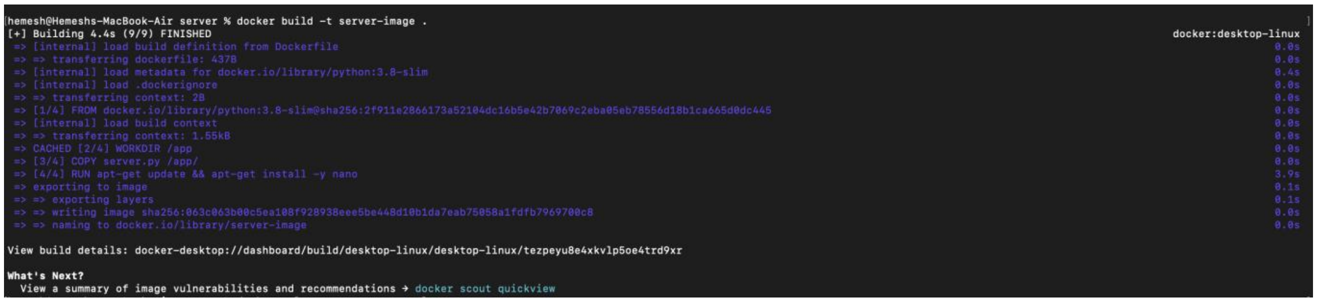
After executing the command the volume could be seen in the application.

5) **Build Docker Images:**
Now we have to create images for the server and client side code, so we will execute the respective docker files.

Docker command: docker build -t <image_name> .



```
hemesh@Hemeshs-MacBook-Air client % docker build -t client-image .
[+] Building 0.8s (9/9) FINISHED                                                                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                      0.0s
 => => transferring dockerfile: 310B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                                                        0.7s
 => [auth] library/python:pull token for registry-1.docker.io                                                             0.0s
 => [internal] load .dockerignore                                                                                         0.0s
 => => transferring context: 2B                                                                                           0.0s
 => [1/3] FROM docker.io/library/python:3.8-slim@sha256:2f911e2866173a52104dc16b5e42b7069c2eba05eb78556d18b1ca665d0dc445  0.0s
 => [internal] load build context                                                                                         0.0s
 => => transferring context: 117B                                                                                         0.0s
 => CACHED [2/3] WORKDIR /app                                                                                             0.0s
 => CACHED [3/3] COPY client.py /app/                                                                                     0.0s
 => exporting to image                                                                                                    0.0s
 => => exporting layers                                                                                                   0.0s
 => => writing image sha256:11e3ed8bc5dc144efc4fa11393c165f658409ec699b58b241c41c5e2f62a853e                              0.0s
 => => naming to docker.io/library/client-image                                                                           0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/rw7u9y4o3mkwef09up3q2ezk1

What's Next?
```



```
hemesh@Hemeshs-MacBook-Air server % docker build -t server-image .
[+] Building 4.4s (9/9) FINISHED                                                                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                      0.0s
 => => transferring dockerfile: 437B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                                                        0.4s
 => [internal] load .dockerignore                                                                                         0.0s
 => => transferring context: 2B                                                                                           0.0s
 => [1/4] FROM docker.io/library/python:3.8-slim@sha256:2f911e2866173a52104dc16b5e42b7069c2eba05eb78556d18b1ca665d0dc445  0.0s
 => [internal] load build context                                                                                         0.0s
 => => transferring context: 1.55kB                                                                                       0.0s
 => CACHED [2/4] WORKDIR /app                                                                                             0.0s
 => [3/4] COPY server.py /app/                                                                                            0.0s
 => [4/4] RUN apt-get update && apt-get install -y nano                                                                   3.9s
 => exporting to image                                                                                                    0.1s
 => => exporting layers                                                                                                   0.1s
 => => writing image sha256:063c063b00c5ea108f928938eee5be448d10b1da7eab75058a1fdfb7969700c8                              0.0s
 => => naming to docker.io/library/server-image                                                                           0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/tezpeyu8e4xkvlp5oe4trd9xr

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

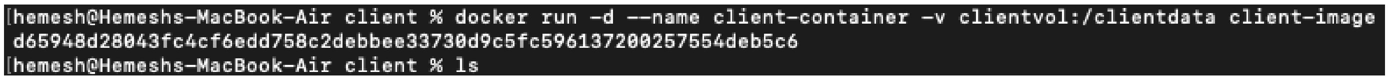
Running the command for both server and client

6) **Run the containers:**

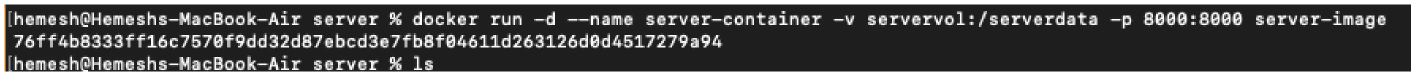
Now we can run the container, once we start the container, it will send random text file to client and will be stored in the volume. Below screenshot will be served as proof that code has given the intended results.

Docker command: docker run -d –name <container name> -v volume1:/volume_location -p port:port <image-name>

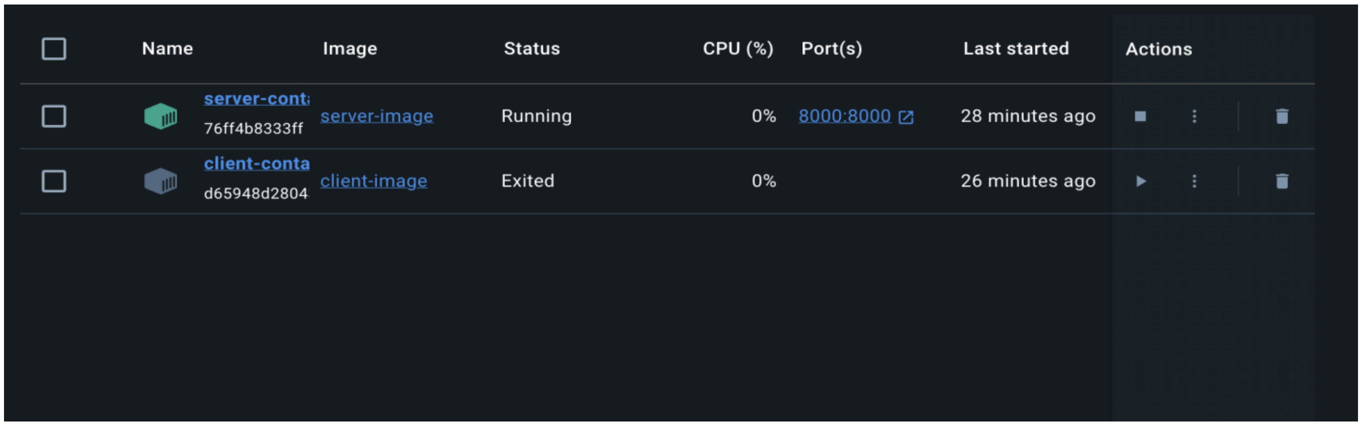
Note: If you have configured special network then we can use that here, before the volume part.



```
[hemesh@Hemeshs-MacBook-Air client % docker run -d --name client-container -v clientvol:/clientdata client-image
d65948d28043fc4cf6edd758c2debbee33730d9c5fc596137200257554deb5c6
[hemesh@Hemeshs-MacBook-Air client % ls
```

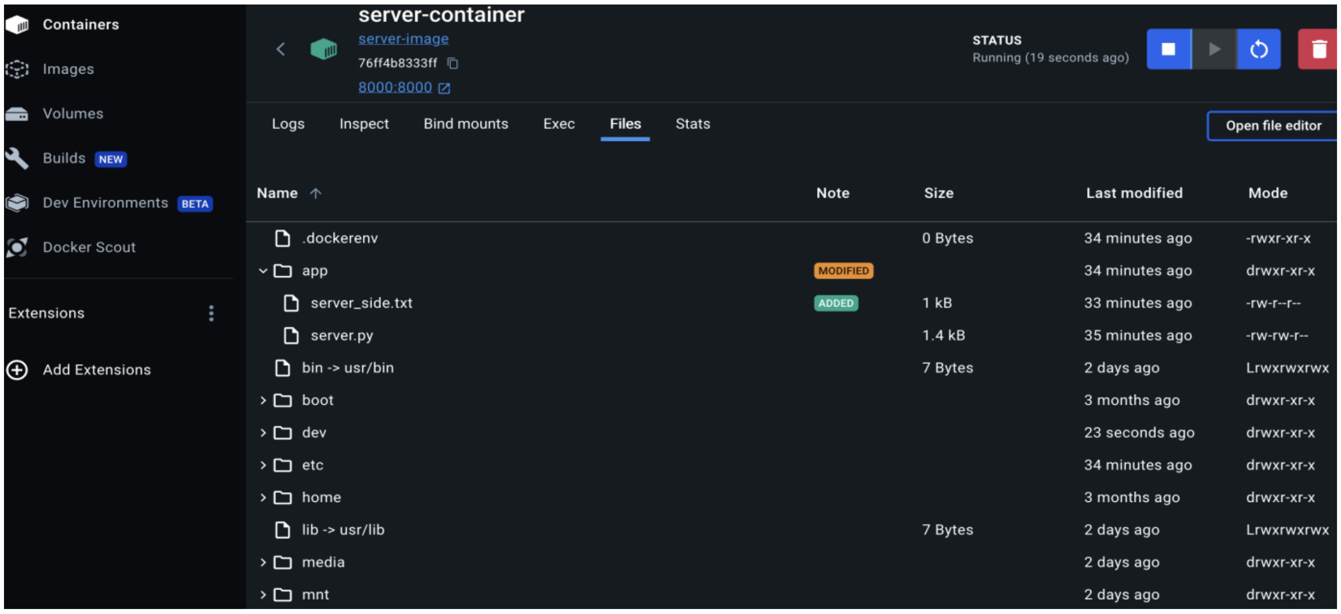


```
[hemesh@Hemeshs-MacBook-Air server % docker run -d --name server-container -v servervol:/serverdata -p 8000:8000 server-image
76ff4b8333ff16c7570f9dd32d87ebcd3e7fb8f04611d263126d0d4517279a94
[hemesh@Hemeshs-MacBook-Air server % ls
```



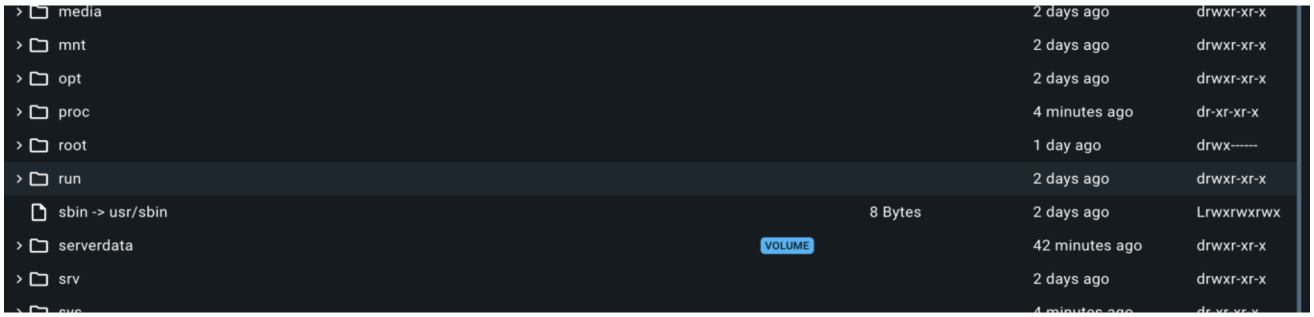
	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
☐	server-cont 76ff4b8333ff	server-image	Running	0%	8000:8000	28 minutes ago	■ ⋮ 🗑
☐	client-conta d65948d2804	client-image	Exited	0%		26 minutes ago	▶ ⋮ 🗑

Screenshots of images containers which are running in docker

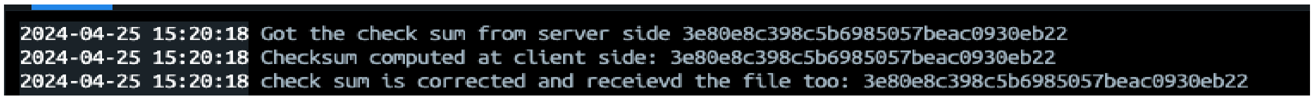


Name	Note	Size	Last modified	Mode
.dockerenv		0 Bytes	34 minutes ago	-rwxr-xr-x
app	MODIFIED		34 minutes ago	drwxr-xr-x
server_side.txt	ADDED	1 kB	33 minutes ago	-rw-r--r--
server.py		1.4 kB	35 minutes ago	-rw-rw-r--
bin -> usr/bin		7 Bytes	2 days ago	Lrwxrwxrwx
boot			3 months ago	drwxr-xr-x
dev			23 seconds ago	drwxr-xr-x
etc			34 minutes ago	drwxr-xr-x
home			3 months ago	drwxr-xr-x
lib -> usr/lib		7 Bytes	2 days ago	Lrwxrwxrwx
media			2 days ago	drwxr-xr-x
mnt			2 days ago	drwxr-xr-x

Screenshot where we can see the server side text being present



media			2 days ago	drwxr-xr-x
mnt			2 days ago	drwxr-xr-x
opt			2 days ago	drwxr-xr-x
proc			4 minutes ago	dr-xr-xr-x
root			1 day ago	drwx------
run			2 days ago	drwxr-xr-x
sbin -> usr/sbin		8 Bytes	2 days ago	Lrwxrwxrwx
serverdata	VOLUME		42 minutes ago	drwxr-xr-x
srv			2 days ago	drwxr-xr-x

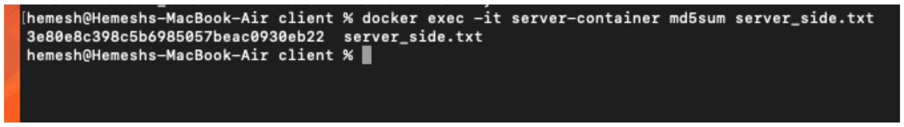


```
2024-04-25 15:20:18 Got the check sum from server side 3e80e8c398c5b6985057beac0930eb22
2024-04-25 15:20:18 Checksum computed at client side: 3e80e8c398c5b6985057beac0930eb22
2024-04-25 15:20:18 check sum is corrected and receievd the file too: 3e80e8c398c5b6985057beac0930eb22
```

For the proof of file being received, at client docker, we can see the logs being printed out in the client side

Lets double check whether the checksum value matches or not, let us generate an checksum for the file which we have received at the client, using md5sum. And we can see that both checksum value below screenshot and client log are same, so our code has executed successfully.

Docker command: docker exec -it server-container md5sum server_side.txt



```
[hemesh@Hemeshs-MacBook-Air client % docker exec -it server-container md5sum server_side.txt
3e80e8c398c5b6985057beac0930eb22  server_side.txt
hemesh@Hemeshs-MacBook-Air client %
```

This above screenshot please previous screenshot where we can see in the logs, the checksum values being equal.

The above screenshot and along with the idea and logic which are explained above have provided proof, that an 1kb file have been sent to the client from server, and saves in '/clientdata'.

Reference:

- https://www.youtube.com/watch?v=eGz9DS-aIeY&ab_channel=NetworkChuck
- https://www.youtube.com/watch?v=pg19Z8LL06w&ab_channel=TechWorldwithNana
- https://www.youtube.com/watch?v=3QiPPX-KeSc&ab_channel=TechWithTim
- https://www.youtube.com/watch?v=FGdiSJakIS4&ab_channel=freeCodeCamp.org