ENGR-E 516 – Engineering Cloud Computing

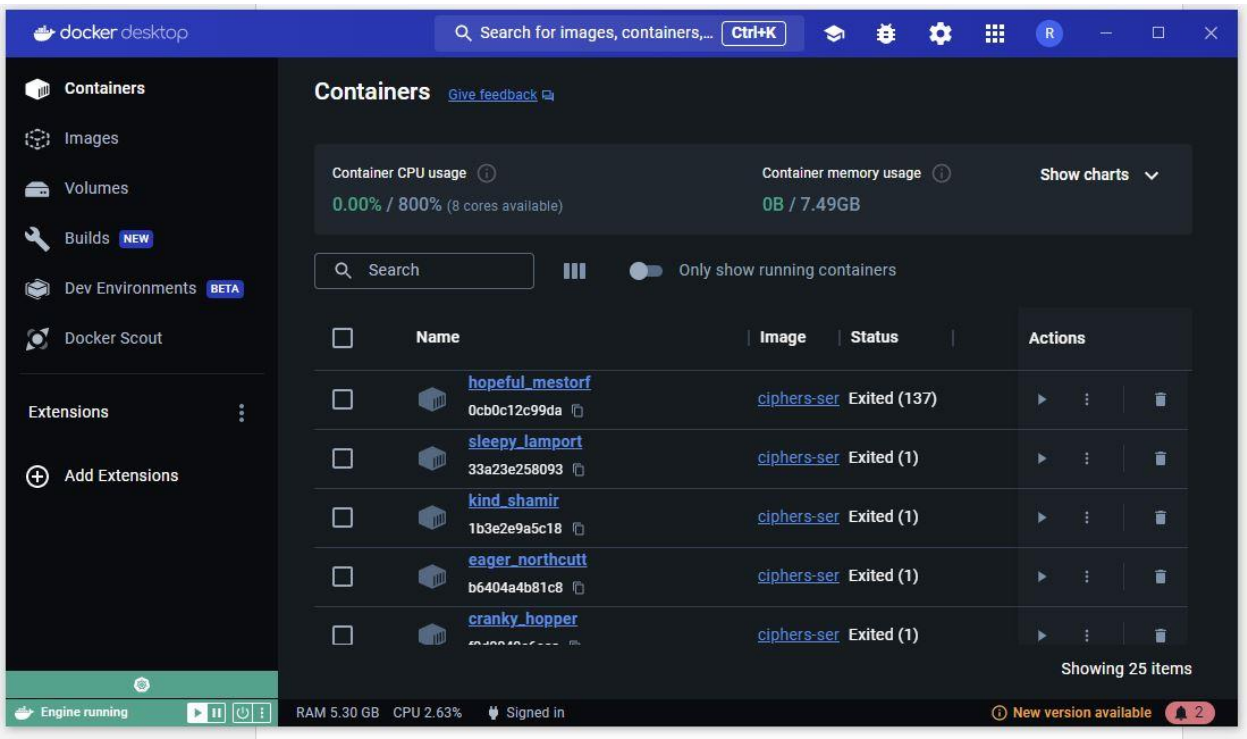
Rohit Goud Kalakuntla

rokala@iu.edu

Assignment-3

Docker Desktop:

For this assignment, I have downloaded and installed docker desktop, which initializes and runs docker daemon in my system. It can be downloaded from the docker website. I will mention URL in references section.



For this assignment, I have chosen **openjdk** docker image as my base image which provides me Java Environment and I developed scripts and code using Java Spring boot

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker pull openjdk
```

Docker Volume Creation

As part of this assignment, I have created 2 docker volumes which are to be used for my programs.

The two volumes I created are

1. **servervol**
2. **clientvol**

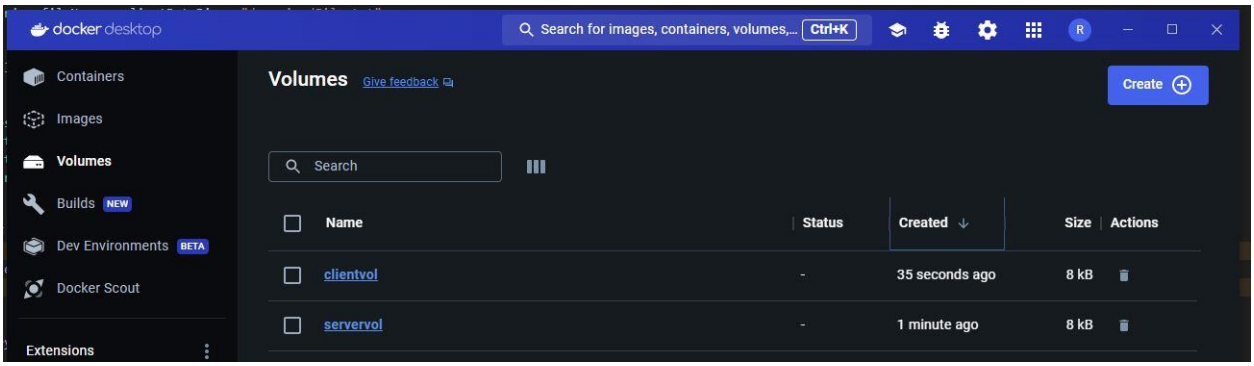
```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker volume create servervol
servervol
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC>
```

"servervol" for server

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker volume create clientvol
clientvol
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC>
```

"clientvol" for client

After creating the volumes, I verified them by looking into the docker desktop



Screenshot of volumes in docker desktop

As we can see that the volumes are created in the docker desktop. I will be using these volumes for further part of the assignment

Docker Network Creation

As part of this assignment, I will have to create a user defined docker network, and the server and client containers should be running under this newly created user defined network.

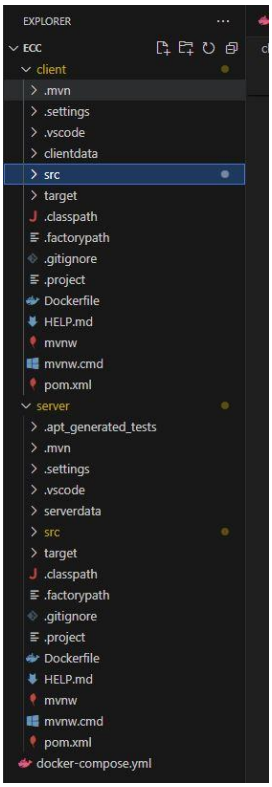
I have created a docker network "**rokala**".

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker network create rokala
316f380d441b5351d1118616ae814da55a521a09992fa14ab247df4f96efdf17
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC>
```

Server And Client Programs & Codes

The next part is to develop server and client scripts, where the server generates a random text file of size 1KB and client receives the files from the server under the network (rokala) I created.

I have used **Java** for the development of server & client scripts.



Project file structure for server & client

Server Script:

server > src > main > java > com > rokala > server > server > ServerApp.java > Language Support for Java(TM) by Red Hat > ServerApp > run(String...)

```java
public class ServerApp implements CommandLineRunner {
    @Override
    public void run(String... args) throws Exception {
        int port = Integer.parseInt("8085");
        // Get the project directory dynamically
        String projectDir = System.getProperty("user.dir");

        // Append the "data" directory to the project directory
        String serverDataDir = projectDir + "/serverdata";
        String fileName = serverDataDir + "/transferFile.txt";
        int fileSize = 1024; // 1KB
        String randomText = generateRandomText(fileSize);

        // Create directory if it does not exist
        Files.createDirectories(Paths.get(serverDataDir));

        // Create file if it does not exist and write random text to it
        if (!Files.exists(Paths.get(fileName))) {
            Files.write(Paths.get(fileName), randomText.getBytes(StandardCharsets.US_ASCII), StandardOpenOption.CREATE,
                    StandardOpenOption.TRUNCATE_EXISTING);

        }
        System.out.println(Paths.get(fileName));    Replace this use of System.out by a logger.

        // Calculate checksum
        String checksum = calculateChecksum(fileName);

        System.out.println("Server started on port " + port);    Replace this use of System.out by a logger.
        System.out.println("File with random text generated and saved: " + fileName);    Replace this use of System.out by a
        System.out.println("Checksum: " + checksum);    Replace this use of System.out by a logger.
        System.out.println("File size Generated: " + fileSize + " bytes");    Replace this use of System.out by a logger.
    }

    private String generateRandomText(int size) {
        char[] chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789".toCharArray();
        StringBuilder sb = new StringBuilder(size);
        Random random = new Random();    Save and re-use this "Random".
        for (int i = 0; i < size; i++) {
            char c = chars[random.nextInt(chars.length)];
            sb.append(c);
        }
        String randomText = sb.toString();
        if (randomText.length() < size) {
            randomText = String.format("%1$-" + size + "s", randomText); // Pad the string to ensure it is exactly 1024    Fo
                                                                         // characters
        }
        randomText = randomText.toString().replace("\r\n", "\n").replace("\r", "\n");    "randomText" is already a string, th
        return "***ECC-ASSIGNMENT-3-rokala***\n" + randomText;
    }

    private String calculateChecksum(String fileName) throws NoSuchAlgorithmException, IOException {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        FileInputStream fis = new FileInputStream(fileName);    Use try-with-resources or close this "FileInputStream" in a "
        byte[] byteArray = new byte[1024];
        int bytesCount;
        while ((bytesCount = fis.read(byteArray)) != -1) {
            digest.update(byteArray, 0, bytesCount);
        }
        fis.close();
        byte[] bytes = digest.digest();
        StringBuilder sb = new StringBuilder();
        for (byte aByte : bytes) {
            sb.append(Integer.toString((aByte & 0xff) + 0x100, 16).substring(1));
        }
        return sb.toString();
    }
}
```

The above code is for server, this code will create a file names “**transferFile.txt**” of size **1KB** with random text data in “**/serverdata**” location and then transfer the file to the client. It will also create a **checksum** and send it along with the file in HTTP headers so that the client can verify the checksum and then receive the file.

Client Script:

client > src > main > java > com > rokala > client > client > ClientApp.java > Language Support for Java(TM) by Red Hat > ClientApp > run(String...)

```java
public class ClientApp implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
        String serverUrl = ("http://server:8085");
        // Get the project directory dynamically
        String projectDir = System.getProperty("user.dir");

        // Append the "data" directory to the project directory
        String clientDataDir = projectDir + "/clientdata";

        String fileName = clientDataDir + "/receivedFile.txt";

        try {
            // Create directory if it does not exist
            Files.createDirectories(Paths.get(clientDataDir));

            // Create file if it does not exist
            if (!Files.exists(Paths.get(fileName))) {
                Files.createFile(Paths.get(fileName));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Retrieve file and checksum from server
        ResponseEntity<byte[]> responseEntity = restTemplate.getForEntity(serverUrl + "/getFile", responseType:byte[].class);
        byte[] fileBytes = responseEntity.getBody();
        HttpHeaders headers = responseEntity.getHeaders();
        String checksum = headers.getFirst(headerName:"Checksum");

        // Verify checksum
        if (verifyChecksum(fileName, checksum)) {
            System.out.println("Checksum Verified!");    Replace this use of System.out by a logger.
        } else {
            System.out.println("Checksum verification failed.");    Replace this use of System.out by a logger.
        }
        // Save file to client data directory
        try (FileOutputStream fos = new FileOutputStream(fileName)) {
            fos.write(fileBytes);
        }

        System.out.println("File received and saved: " + fileName);    Replace this use of System.out by a logger.


    }

    private boolean verifyChecksum(String fileName, String expectedChecksum) throws NoSuchAlgorithmException, IOException {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] fileBytes = Files.readAllBytes(Paths.get(fileName));
        byte[] hashBytes = digest.digest(fileBytes);
        String actualChecksum = bytesToHex(hashBytes);
        return actualChecksum.equals(expectedChecksum);
    }

    private String bytesToHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (byte aByte : bytes) {
            sb.append(String.format("%02x", aByte));
        }
        return sb.toString();
    }

}
```

The above code is the client script, which receives a file from the server and stores the received file named “**receivedFile.txt**” in “**/clientdata**” location. It also verifies the **Checksum** that is received from the server.

Dockerfile for Server and Client:

After the Programs are developed and ready, Now I have created Dockerfile scripts for both server and client so that I can run them in docker containers.

- First I created a **dockerfile** for server, which pulls a docker base image. Since I am using Java, I am pulling openjdk base image from docker hub.
- Then I have created a home directory "/home"
- Then, I copied the server jar file, which us used to run the server

```
  GNU nano 7.2
FROM openjdk:latest

# Set the working directory
WORKDIR /home

# Copy the generated JAR file
COPY ./target/server-0.0.1-SNAPSHOT.jar server.jar

# Set the entrypoint to run the application
ENTRYPOINT ["java", "-jar", "server.jar", "--server.port=8085"]
```

- Now, I created a dockerfile for the client, which pulls a docker base image. Since I am using Java, I am pulling openjdk base image from docker hub.
- Then I have created a home directory "/home"
- Then, I copied the client jar file, which us used to run the server

```
  GNU nano 7.2
FROM openjdk:latest

# Set the working directory
WORKDIR /home

# Copy the generated JAR file
COPY ./target/client-0.0.1-SNAPSHOT.jar client.jar

# Set the entrypoint to run the application
ENTRYPOINT ["java", "-jar", "client.jar"]
```

Now the application is ready to be built and run in docker.

For building the docker image, I have used the following command. The build command, pulls the image and performs all the steps as mentioned in dockerfile and creates a image with name we mention

Server docker build:

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC\server> docker build -t server -f Dockerfile .
[+] Building 13.5s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 303B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/openjdk:latest
 => [auth] library/openjdk:pull token for registry-1.docker.io
 => [1/3] FROM docker.io/library/openjdk:latest@sha256:9b448de897d211c9e0ec635a485650aed6e28d4eca1efbc34940560a480b3f1f
 => [internal] load build context
 => => transferring context: 19.76MB
 => CACHED [2/3] WORKDIR /home
 => [3/3] COPY ./target/server-0.0.1-SNAPSHOT.jar server.jar
 => exporting to image
 => => exporting layers
 => => writing image sha256:ae780fa9a1bc59d6e84c9dd78414aca67f68af250eda30c70a169bc4413abd7a
 => => naming to docker.io/library/server

View build details: docker-desktop://dashboard/build/default/default/r1tpxa274zjg1dvst2rddp51j

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC\server>
```

Client docker build:

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC\server> docker build -t server -f Dockerfile .
[+] Building 13.5s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 303B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/openjdk:latest
 => [auth] library/openjdk:pull token for registry-1.docker.io
 => [1/3] FROM docker.io/library/openjdk:latest@sha256:9b448de897d211c9e0ec635a485650aed6e28d4eca1efbc34940560a480b3f1f
 => [internal] load build context
 => => transferring context: 19.76MB
 => CACHED [2/3] WORKDIR /home
 => [3/3] COPY ./target/server-0.0.1-SNAPSHOT.jar server.jar
 => exporting to image
 => => exporting layers
 => => writing image sha256:ae780fa9a1bc59d6e84c9dd78414aca67f68af250eda30c70a169bc4413abd7a
 => => naming to docker.io/library/server

View build details: docker-desktop://dashboard/build/default/default/r1tpxa274zjg1dvst2rddp51j

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC\server>
```

Once the builds are successful, the images are created, to verify it I looked on docker desktop to check if the images are created and ready to use.

client 0bc0792db5f2	latest	In use	8 minutes ago	489.68 MB		
server ae780fa9a1bc	latest	In use	22 minutes ag	489.68 MB		

As we can see, the server and client images are created

Running the docker images

As the images are now created and ready to use, I have now used the docker run command to create a container using the images.

After running the both the images, there where 2 containers crated which can be seen below

server c2701729aff1	server	Running	N/A	8085:8085	3	
client 8af4a5c481f5	client	Running	N/A	8086:8086	2	

As seen in above screenshot, the containers are created and running.

Now, In this report, I will show the individual execution of server and client with all the steps

Server docker run:

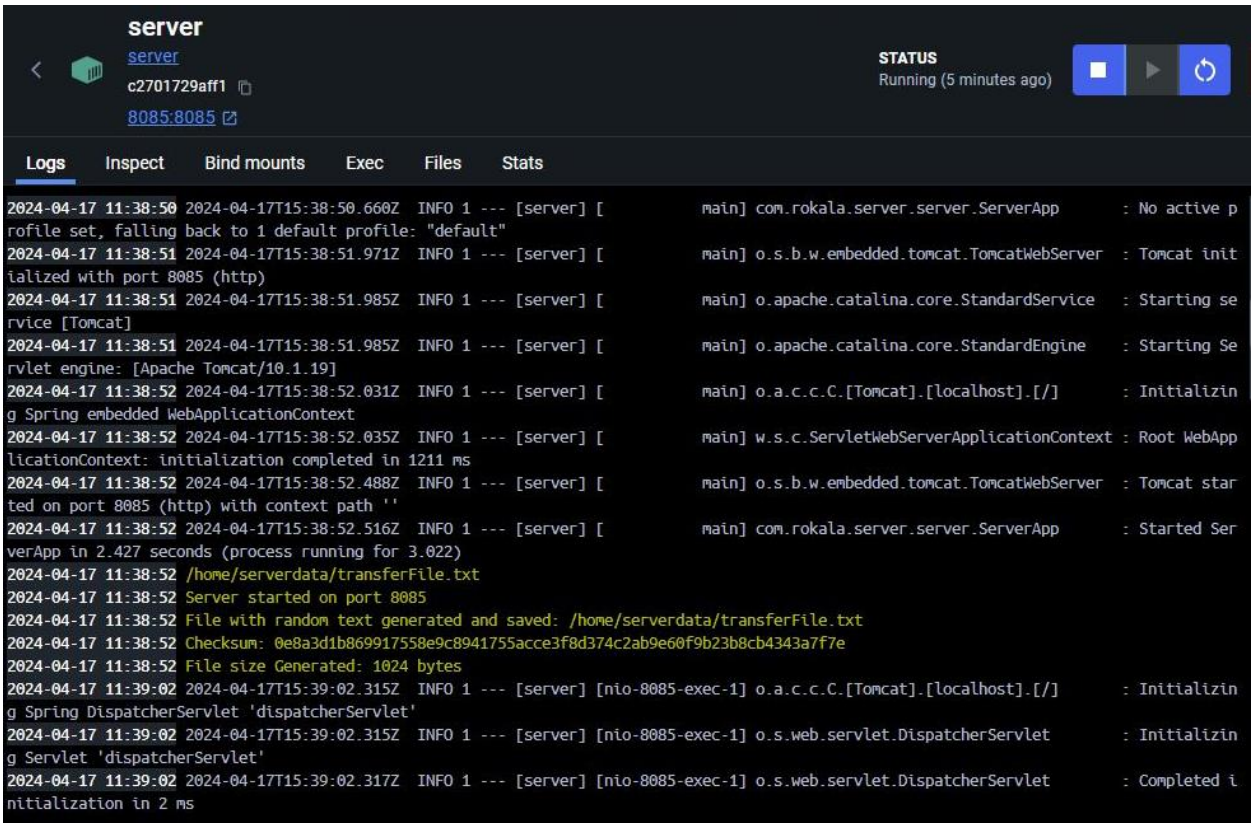
```
docker run -v servervol:/home/serverdata -p 8085:8085 --network rokala --name server server
```

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker run -v servervol:/home/serverdata -p 8085:8085 --network rokala --name server server

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.4)

2024-04-17T15:29:33.916Z  INFO 1 --- [server] [           main] com.rokala.server.server.ServerApp       : Starting ServerApp v0.0.1-SNAP
2024-04-17T15:29:33.920Z  INFO 1 --- [server] [           main] com.rokala.server.server.ServerApp       : No active profile set, falling
2024-04-17T15:29:35.251Z  INFO 1 --- [server] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8
2024-04-17T15:29:35.265Z  INFO 1 --- [server] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-17T15:29:35.266Z  INFO 1 --- [server] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apac
2024-04-17T15:29:35.318Z  INFO 1 --- [server] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded W
2024-04-17T15:29:35.321Z  INFO 1 --- [server] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: in
2024-04-17T15:29:35.730Z  INFO 1 --- [server] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8085 (h
2024-04-17T15:29:35.755Z  INFO 1 --- [server] [           main] com.rokala.server.server.ServerApp       : Started ServerApp in 2.476 sec
/home/serverdata/transferFile.txt
Server started on port 8085
File with random text generated and saved: /home/serverdata/transferFile.txt
Checksum: 0e8a3d1b869917558e9c8941755acce3f8d374c2ab9e60f9b23b8cb4343a7f7e
File size Generated: 1024 bytes
```

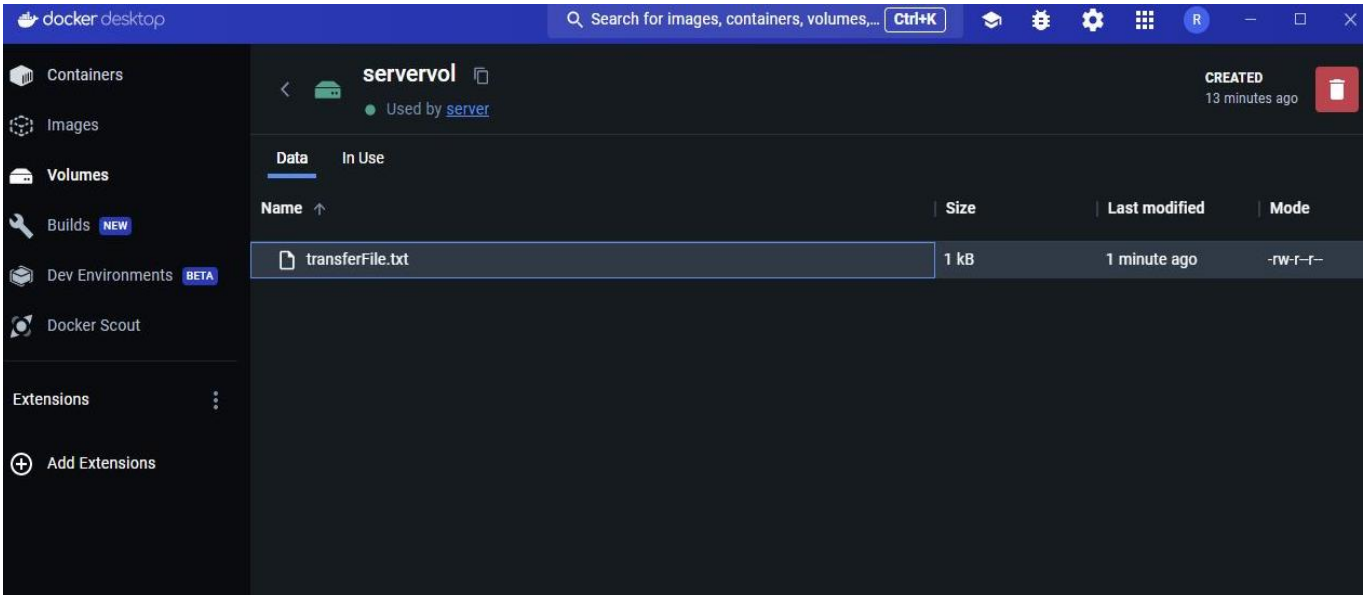
After running the above docker run command, we could see that the server is up and running. I have explicilty mentioned the port number to be 8085 so it is running on 8085.

And also it can be seen that the transferFile.txt has been generated and stored in /home/serverdata/. Checksum is also generated and can be seen the file size 1024 bytes which is 1KB.

I have also verified using the docker desktop to see if the container is up and running and also volumes.

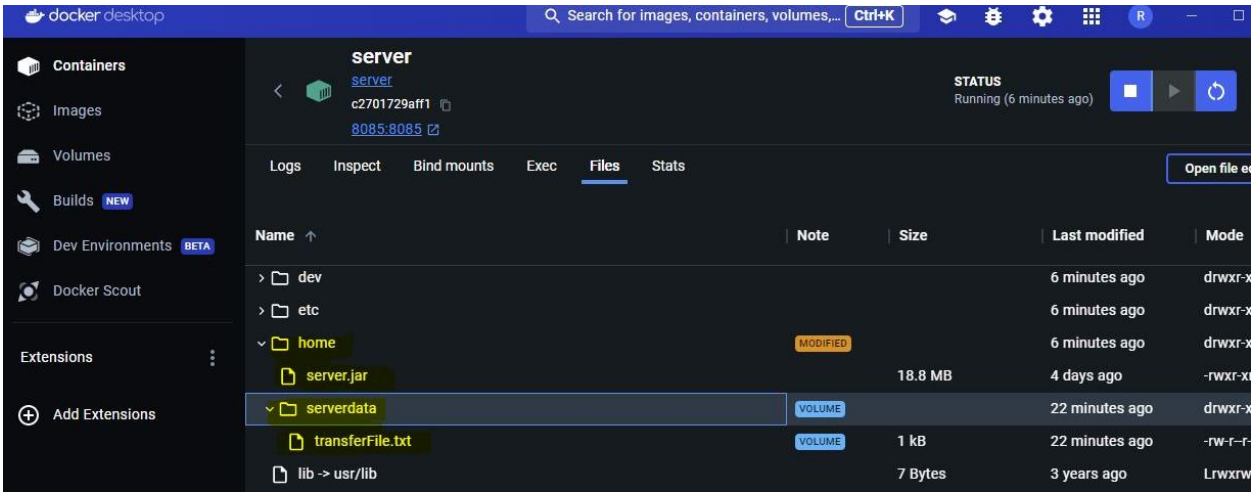


Volume: (servervol)



Verifying in server shell if the file is generated.

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker exec -it server /bin/bash
bash-4.4# ls
server.jar  serverdata
bash-4.4# cd serverdata
bash-4.4# ls
transferFile.txt
bash-4.4# cat transferFile.txt
***ECC-ASSIGNMENT-3-rokala***
DjrVrvHs7noXuMSNiN5pOHh0Zb5bbL9aDiGQYcCXrBLsncXI2UvsRR7hU6C0xctyXjCeq8HObImXSE6OzFgJ
M28Mg2vKavzokGjaBp8EtSArqr26jlh8tRmjmmA020r66imes0bB7v6TUTe8CWSEH7WsOZfW5PnIdDXyRdFb
Zw4bDx9XsrgLzif9SFhhBK3iYYMsBAosMFqmQZFlLix3ImKxa693i8NatPHwj1UbFGezzKmIAtInVqMHsXO0
L7mR6pBJV0xKe0qdSUU6Y5AwWOF3qhhpR4Zwy0vlCNYFYMnr4HFQv1HnQS6IGkwHW7AvqDIE3lU52tCAgSDu
```



I have verified all the ways, and It is been verified properly that a random file “transferFile.txt” has been generated and is stored in the serverdata directory an also in the servervol volume.

Client docker run:

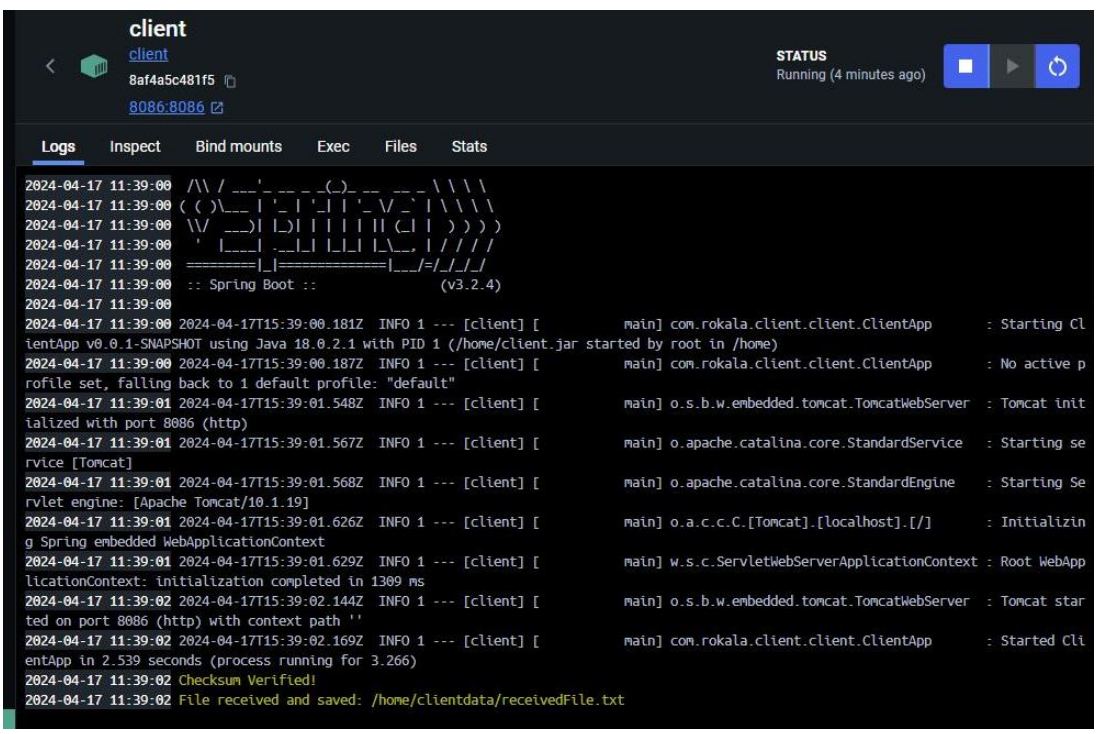
```
docker run -v clientvol:/home/clientdata -p 8086:8086 --network rokala --name client client
```

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC\client> docker run -v clientvol:/home/clientdata -p 8086:8086 --network rokala --name client client

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.4)

2024-04-17T15:35:48.081Z  INFO 1 --- [client] [           main] com.rokala.client.client.ClientApp       : Starting ClientApp v0.0.1-SNAPSHOT usi
2024-04-17T15:35:48.089Z  INFO 1 --- [client] [           main] com.rokala.client.client.ClientApp       : No active profile set, falling back to
2024-04-17T15:35:49.359Z  INFO 1 --- [client] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8086 (htt
2024-04-17T15:35:49.376Z  INFO 1 --- [client] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-17T15:35:49.376Z  INFO 1 --- [client] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomca
2024-04-17T15:35:49.433Z  INFO 1 --- [client] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplic
2024-04-17T15:35:49.436Z  INFO 1 --- [client] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializa
2024-04-17T15:35:49.845Z  INFO 1 --- [client] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8086 (http) wit
2024-04-17T15:35:49.871Z  INFO 1 --- [client] [           main] com.rokala.client.client.ClientApp       : Started ClientApp in 2.447 seconds (pr
Checksum Verified!
File received and saved: /home/clientdata/receivedFile.txt
```

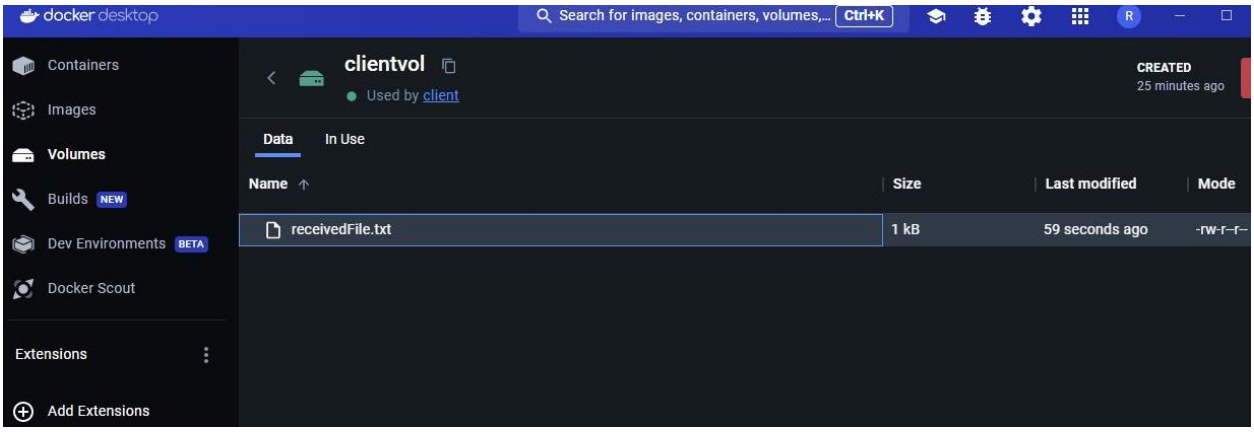
After running the above docker run command, we could see that the server is up and running. I have explicitly mentioned the port number to be 8086 so it is running on 8086.

And also it can be seen that the receivedfile.txt has been received from the server and stored in /home/clientdata/. Checksum is also verified .

I have also verified using the docker desktop to see if the container is up and running and also volumes.



Volume: (clientvol)



Verifying in client shell if the file is generated.

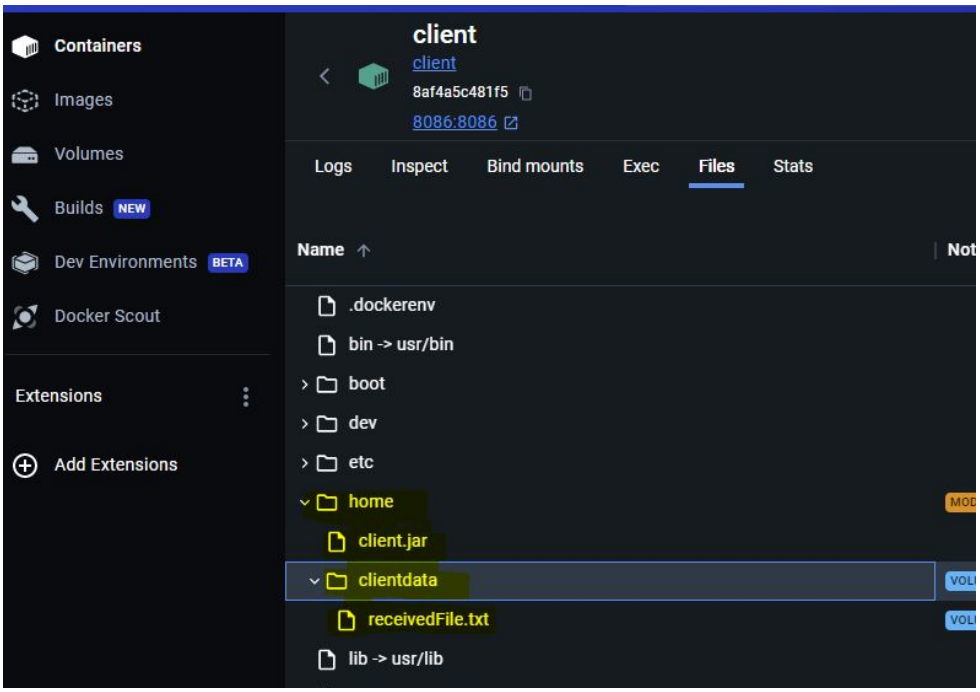
Now, I have entered into the client shell, to check and verify is the file has been received.

To enter into shell I have used the command below

```
docker exec -it client /bin/bash
```

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker exec -it client /bin/bash
bash-4.4# ls
client.jar  clientdata
bash-4.4# cd clientdata
bash-4.4# ls
receivedFile.txt
bash-4.4# cat receivedFile.txt
***ECC-ASSIGNMENT-3-rokala***
DjrVrvHs7noXuMSNiN5pOHh0Zb5bbL9aDiGQYcCXrBLsncXI2UvsRR7hU6C0xctyXjCeq8HObImXSE6
M28Mg2vKavzokGjaBp8EtSArqr26jlh8tRmjmmA020r66imes0bB7v6TUTe8CWSEH7WsOZfW5PnIdDX
Zw4bDx9XsrgLzif9SFhhBK3iYYMsBAosMFqmQZFlLix3ImKxa693i8NatPHwj1UbFGezzKmIAtInVqM
L7mR6pBJV0xKe0qdSUU6Y5AwWOF3qhhpR4Zwy0vlCNYFYMnr4HFQv1HnQS6IGkwHW7AvqDIE3lU52tC
```

Upon entering into client shell and navigating to the clientdata folder, I could see that the file from the server has been received. I have manually also verified its it is the same file that server generated. After validating checksum, It is confirmed that it is the same file that server generated.



I have verified all the ways, and It is been verified properly that a random file "transferFile.txt" has been received and is stored in the clientdata directory with a file name receivedFile.txt.

Checksum has also been verified by the system.

Verifying the Network:

I have also verified if the user defined nework is working as expected by using below command

```
docker network inspect rokala
```

Upon running the above command, It displayed which all containers are on the network as show in below screenshot.

It can be seen that, the containers server and client are on rokala network, Which is expected.

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker network inspect rokala
[
    {
        "Name": "rokala",
        "Id": "316f380d441b5351d1118616ae814da55a521a09992fa14ab247df4f96efdf17",
        "Created": "2024-04-17T15:27:08.537678104Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.21.0.0/16",
                    "Gateway": "172.21.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "8af4a5c481f5b562a22ee415d22b85cecd78ce81818a5ebf70d6dcdfc31ab04e": {
                "Name": "client",
                "EndpointID": "d28044c68d2498cc8d21b7ee17755d88c9c743f4540fb0e3e3c2c9293008acd2",
                "MacAddress": "02:42:ac:15:00:03",
                "IPv4Address": "172.21.0.3/16",
                "IPv6Address": ""
            },
            "c2701729aff175e8ed6effb9717c2b098e3ad7a6cf82cec01e2742ec2023dcbe": {
                "Name": "server",
                "EndpointID": "cefe6caa20526f470c05f472d7703af339ae36bf8e7b1c77904c6c238fe19366",
                "MacAddress": "02:42:ac:15:00:02",
                "IPv4Address": "172.21.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC>
```

Additional Exercise

To explore more of docker, I have now tried out on how to use docker-compose. I have added a docker-compose.yml script in my assignment which includes both server and client.

Firstly, I have created a docker-compose.yml as shown in below screenshot.

```yaml
version: '3'
services:
  server:
    build:
      context: ./server
      dockerfile: Dockerfile
    networks:
      - rokala
    volumes:
      - servervol:/home/serverdata
    container_name: server
    ports:
      - "8085:8085"

  client:
    build:
      context: ./client
      dockerfile: Dockerfile
    networks:
      - rokala
    volumes:
      - clientvol:/home/clientdata
    container_name: client
    ports:
      - "8086:8086"
    depends_on:
      - server

networks:
  rokala:

volumes:
  servervol:
  clientvol:
```

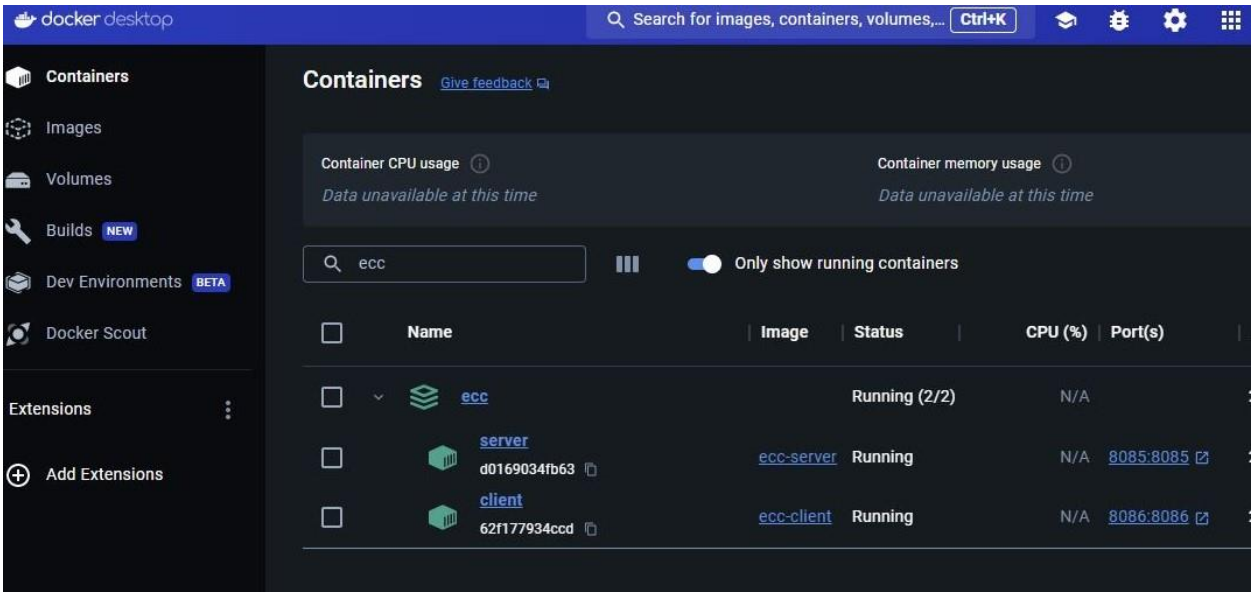
In the above screenshot, we can see that the 2 services server & client are configured, along with the desired port numbers, network and volumes respectively.

Now, I used the following command to run the docker-compose. It creates the containers, volumes and network all together in one go.

```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker-compose up -d
[+] Running 5/5
 ✓ Network ecc_rokala      Created
 ✓ Volume "ecc_servervol"  Created
 ✓ Volume "ecc_clientvol"  Created
 ✓ Container server        Started
 ✓ Container client        Started
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC>
```

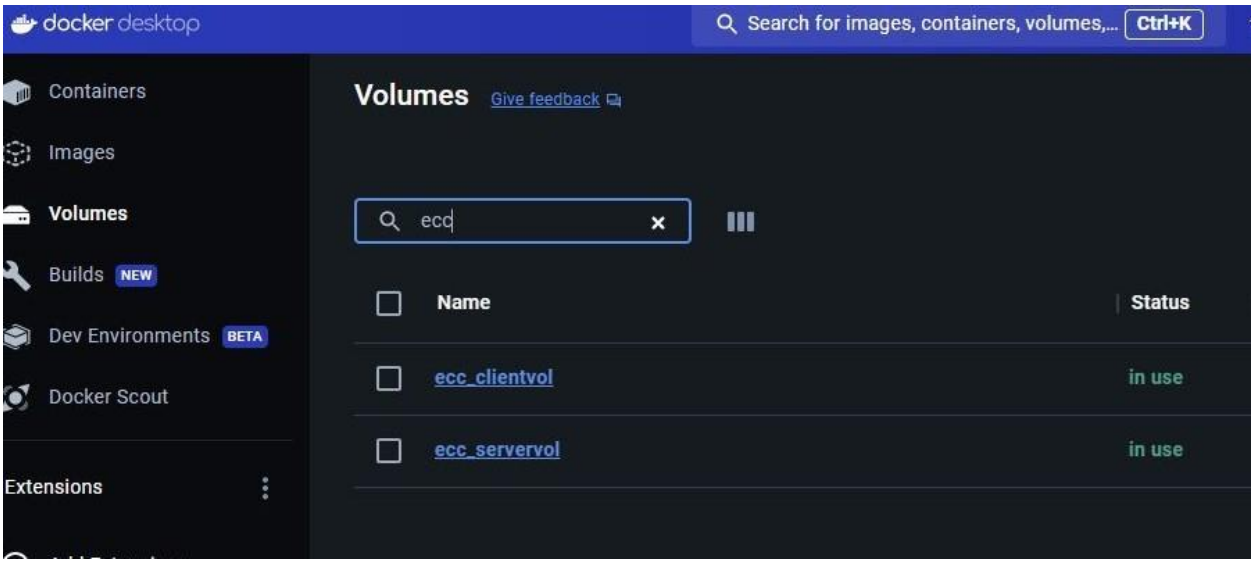
As seen in the screenshot, network “ecc_rokala”, server and client containers has been created and also volumes ecc_servervol, ecc_clientvol has been created

To verify, I looked upon docker desktop to check if containers are created.

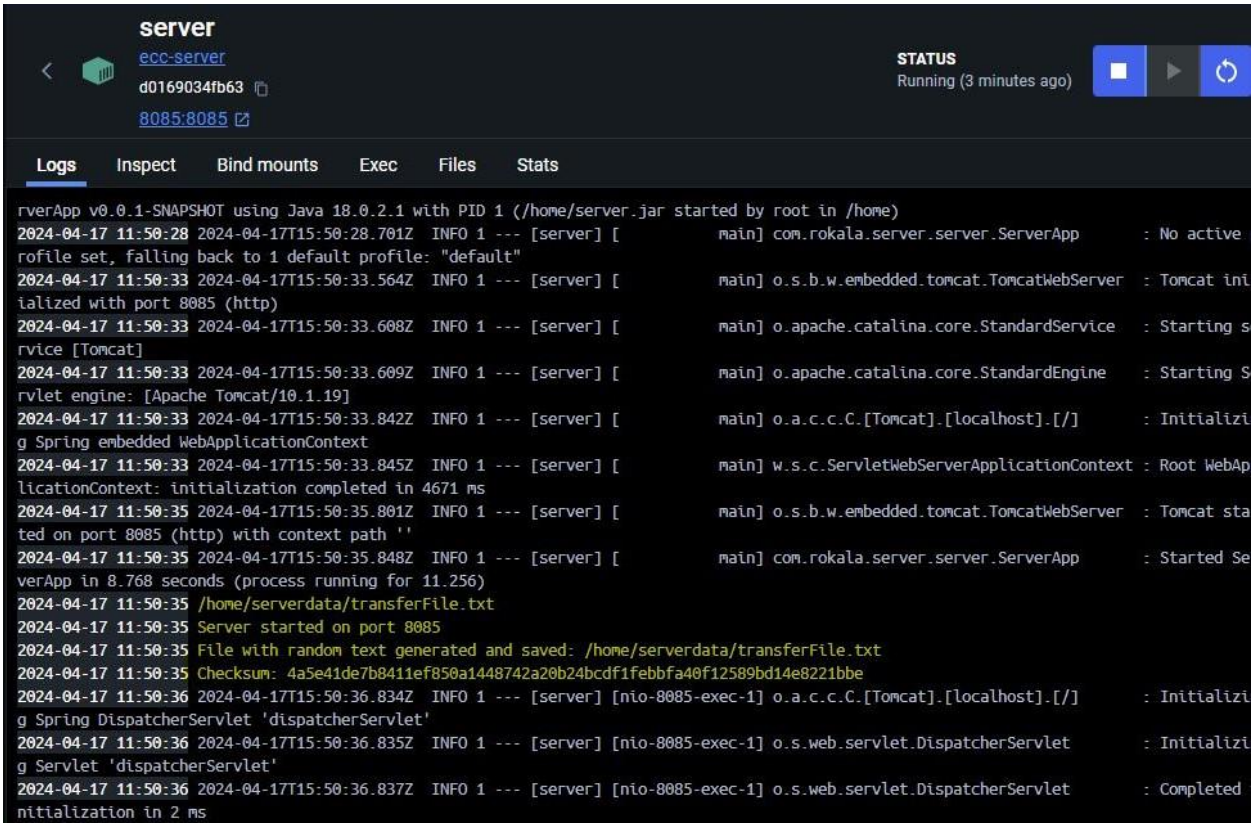


Server and client containers are created.

Volumes are also created, it is shown in below screenshot.



Now, I check the server logs to verify if the file has been generated by the server, the logs are shown in below screenshot



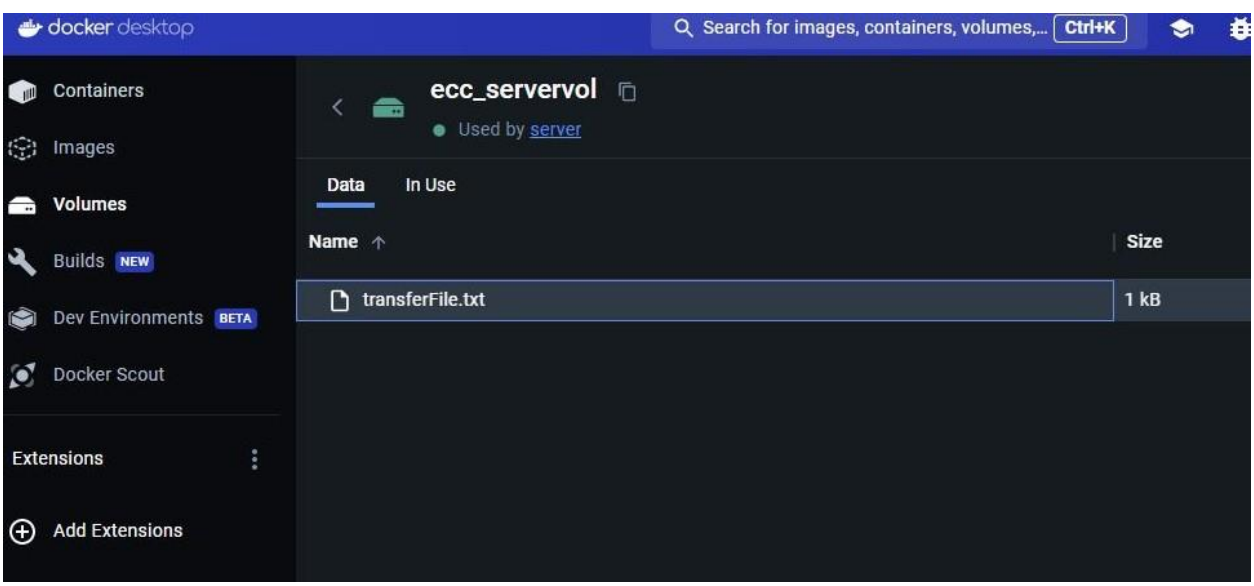
The logs show that the file has been generated and stored, checksum is also generated.

Now, to verify the file in file system of server and client, I opened up the shells of server and client to check if file has been received.

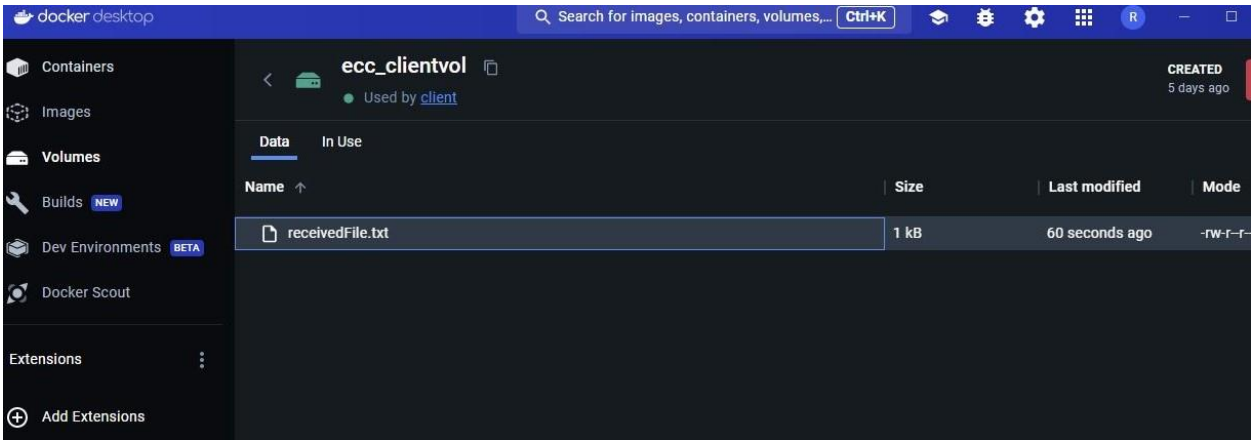
```
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker exec -it server /bin/bash
bash-4.4# ls
server.jar  serverdata
bash-4.4# cd serverdata
bash-4.4# cat transferFile.txt
***ECC-ASSIGNMENT-3-rokala***
GF0qR3Iva1kwSGvgppb61pH98xjgnzuEAugPyjfLI12N7CkmciVhgwMJEMkHBYaHWH8LZcyXubVSUpQVD3bwol
ZwiTk5KmwejvEzTdY3VP0bimZYRIZI8Qp8zdboS88m6yTyPR6M6MBkwvlTDaz47WFGVLvITDTIYZRVhPD0ewo3
uhkUDOcLhsvUYhqJNJHAe3Och3OH5a7C0fvVTz70j2iQlZf54q9hrIKmRW0fPkMXkhmJL0sh2qmScGRolAj019
ijJXCtly0mXIHpgTO12c88OZmASDvUKpTMSON1DZahY5eUfumb1BLcPw9xzA8QmdeMQyfg2iAbash-4.4# ^C
bash-4.4# exit
exit
PS E:\IU2025\SPRING2024\ECC\Assignment-3\ECC> docker exec -it client /bin/bash
bash-4.4# ls
client.jar  clientdata
bash-4.4# cd client
bash: cd: client: No such file or directory
bash-4.4# cd clientdata/
bash-4.4# ls
receivedFile.txt
bash-4.4# cat receivedFile.txt
***ECC-ASSIGNMENT-3-rokala***
GF0qR3Iva1kwSGvgppb61pH98xjgnzuEAugPyjfLI12N7CkmciVhgwMJEMkHBYaHWH8LZcyXubVSUpQVD3bwol
ZwiTk5KmwejvEzTdY3VP0bimZYRIZI8Qp8zdboS88m6yTyPR6M6MBkwvlTDaz47WFGVLvITDTIYZRVhPD0ewo3
uhkUDOcLhsvUYhqJNJHAe3Och3OH5a7C0fvVTz70j2iQlZf54q9hrIKmRW0fPkMXkhmJL0sh2qmScGRolAj019
ijJXCtly0mXIHpgTO12c88OZmASDvUKpTMSON1DZahY5eUfumb1BLcPw9xzA8QmdeMQyfg2iAbash-4.4#
```

I also checked the volumes on docker desktop

ecc_servervol



ecc_clientvol



GitHub Repository: https://github.com/rohit2905/ENGR_E_516_ECC

Instructions to run (via docker build and docker run)

1. Git clone the repository.
2. Install docker in your system and run docker desktop.
3. Navigate to server folder and follow the steps mentioned in this report.
4. Verify the result.

Instructions to run (via docker-compose)

1. Git clone the repository.
2. Install docker in your system and run docker desktop.
3. Navigate to the cloned folder.
4. Run **docker-compose up -d**
5. Verify the result.

References:

1. Docker desktop
2. openjdk image