

COMP 6231 Assignment 3

Task 1: File Server Commands Explanation

Qiang Xue (40300671) Yicheng Cai (26396283) Yikai Chen (40302669)
Yifan Wu (40153584) Alexander Sutherland (40321783)

November 4, 2025

1 Overview

The file server architecture utilizes Docker Swarm for container orchestration, enabling seamless communication between a central server and multiple clients across different physical machines. Each client maintains its own isolated file structure on the server, demonstrating a multi-tenant file storage system.

2 Server Configuration

2.1 Docker Image Build

Dockerfile Content:

```
FROM ozxx33/fileserver-base
WORKDIR /usr/src/app
COPY ./server.py .
CMD ["python", "./server.py"]
```

Build Command:

```
docker build -t task1_server .
```

Explanation: This command creates a Docker image named `task1_server`. The `-t` flag tags the image with the specified name, while the `.` indicates the build context is the current directory containing the Dockerfile and `server.py` file.

2.2 Docker Swarm Initialization

Initialize Swarm:

```
docker swarm init
```

Explanation: This command initializes a Docker Swarm on the manager node. Upon initialization, the system returns a join token that worker nodes can use to join the swarm.

Retrieve Worker Token:

```
docker swarm join-token worker
```

Explanation: Displays the join token for worker nodes. This token is required for worker nodes to authenticate and join the Docker Swarm cluster.

2.3 Create Overlay Network

Create Network:

```
docker network create -d overlay 6231-net
```

Explanation: Creates an overlay network named 6231-net. The -d overlay flag specifies the network driver type.

2.4 Verify Configuration

List Cluster Nodes:

```
docker node ls
```

Explanation: Lists all nodes in the Docker Swarm, displaying their status, availability, and role (manager or worker). This verification step ensures all nodes have successfully joined the cluster.

List Networks:

```
docker network ls
```

Explanation: Lists all networks in the Docker Swarm, confirming the creation of the overlay network 6231-net.

Inspect Specific Network:

```
docker network inspect 6231-net
```

Explanation: Provides detailed information about the overlay network, including connected containers and their IP addresses.

2.5 Run Server Container

Launch Server Container:

```
docker run -it --rm --name server --hostname server  
--network 6231-net task1_server bash
```

Explanation - Command Parameters:

- **-it:** Runs container in interactive mode with terminal access
- **--rm:** Automatically removes the container upon termination
- **--name server:** Assigns the container name "server"
- **--hostname server:** Sets hostname for name resolution within the network
- **--network 6231-net:** Connects container to the overlay network

- **bash:** Opens bash shell for manual command execution

Verify Container IP:

```
hostname -i
```

Output: 10.0.1.28

Explanation: Displays the IP address assigned to the container within the overlay network, enabling communication with other containers.

Execute Server Application:

```
python server.py
```

Explanation: Launches the file server application, which listens on 0.0.0.0:65432 for incoming client connections and organizes received files in client-specific directories.

3 Client Configuration

3.1 Docker Image Build

Dockerfile Content:

```
FROM ozxx33/fileserver-base
WORKDIR /usr/src/app
COPY . .
ENV SERVER_IP=server
ENV SERVER_PORT=65432
CMD ["python", "./client.py"]
```

Build Command:

```
docker build -t task1_client .
```

Explanation: Creates a Docker image named `task1_client`. The Dockerfile copies all files from the current directory (including `client.py` and test files) and sets environment variables (`SERVER_IP` and `SERVER_PORT`) for server connection configuration.

3.2 Distribute Client Image

There are two methods to distribute the client image to worker nodes:

Method 1 - Image Export/Import:

```
# On manager node
docker save -o task1_client.tar task1_client

# On worker node
docker load -i task1_client.tar
```

Explanation: The `docker save` command exports the image to a tar archive file, which can be transferred to worker nodes and loaded using `docker load`. This method is ideal when nodes lack access to a shared registry.

Method 2 - Build on Each Node:

Copy the Dockerfile and necessary files to each worker node and execute the build command locally on each node.

3.3 Join Swarm from Worker Nodes

Join Worker to Swarm:

```
docker swarm join --token <token> <serverip>:2377
```

Explanation: Joins a worker node to the Docker Swarm. The token authenticates the node, while the IP address (<serverip>) and port 2377 specify the manager node's address. Port 2377 is the default Docker Swarm management port.

3.4 Deploy Client Containers

3.4.1 Client 1 (First Worker Node)

Launch Client Container:

```
docker run -it --rm --name client1 --hostname client1
--network 6231-net task1_client bash
```

Container IP: 10.0.1.30

Execute Client Application:

```
python client.py
```

Explanation: Initiates the client application, which connects to the server using hostname resolution via Docker's DNS and transfers files.

3.4.2 Client 2 (Second Worker Node)

Launch Client Container:

```
docker run -it --rm --name client2 --hostname client2
--network 6231-net task1_client bash
```

Container IP: 10.0.1.32

Execute Client Application:

```
python client.py
```

Explanation: Operates on a separate worker node with unique hostname.

3.4.3 Client 3 (Third Worker Node)

Launch Client Container:

```
docker run -it --rm --name client3 --hostname client3
--network 6231-net task1_client bash
```

Container IP: 10.0.1.34

Execute Client Application:

```
python client.py
```

Explanation: Completes the three-client architecture.

4 Network and Storage Configuration

4.1 Network Configuration

Container	Hostname	IP Address	Port
Server	server	10.0.1.28	65432
Client 1	client1	10.0.1.30	Dynamic
Client 2	client2	10.0.1.32	Dynamic
Client 3	client3	10.0.1.34	Dynamic

Table 1: Container network configuration within the overlay network

4.2 Storage Configuration

- **Server Storage:** Files are stored within the server container at `/usr/src/app/`
- **Client-Specific Directories:** Each client can use `mkdir xxx` to create its own directory on the server for file storage.)
- **No Shared Volumes:** Each container maintains isolated storage; file transfer occurs exclusively via network communication

5 Implementation Results

5.1 Server-Side View

```

>/ Q ± ..231-net task1_server bash
base ~/Codes/6231/Assignment_3/Task1/server git:(main) 2 files changed, 45 insertions(+), 3 deletions(-)
docker run -it --rm --name server --hostname server --network 6231-net task1_server bash
Connection from: ('10.0.1.32', 54390)
Accepted connection from ('10.0.1.34', 46028)
Connection from: ('10.0.1.34', 46028)
Received command: mkdir client1 from ('10.0.1.30', 40222)
Received command: cd client1 from ('10.0.1.30', 40222)
Received command: ul orca.jpg from ('10.0.1.30', 40222)
[UL] Start: name=orca.jpg
[UL] Size header ok: expected_len=58111, remainder_after_header=1383
[UL] Reading exact bytes: to_read=56728
[UL] Done for orca.jpg: wrote=58111 bytes at /usr/src/app/client1/orca.jpg
Received command: cd client2 from ('10.0.1.32', 54390)
Received command: mkdir client2 from ('10.0.1.32', 54390)
Received command: cd client2 from ('10.0.1.32', 54390)
Received command: ul jellyfish.jpg from ('10.0.1.32', 54390)
[UL] Start: name=jellyfish.jpg
[UL] Size header ok: expected_len=77539, remainder_after_header=1383
[UL] Reading exact bytes: to_read=76156
[UL] Done for jellyfish.jpg: wrote=77539 bytes at /usr/src/app/client2/jellyfish.jpg
Received command: mkdir client3 from ('10.0.1.34', 46028)
Received command: cd client3 from ('10.0.1.34', 46028)
Received command: ul about.txt from ('10.0.1.34', 46028)
[UL] Start: name=about.txt
[UL] Size header ok: expected_len=1625, remainder_after_header=1384
[UL] Reading exact bytes: to_read=241
[UL] Done for about.txt: wrote=1625 bytes at /usr/src/app/client3/about.txt
Received command: wordcount about.txt from ('10.0.1.34', 46028)

```

Figure 1: Server container showing received files organized by client directories

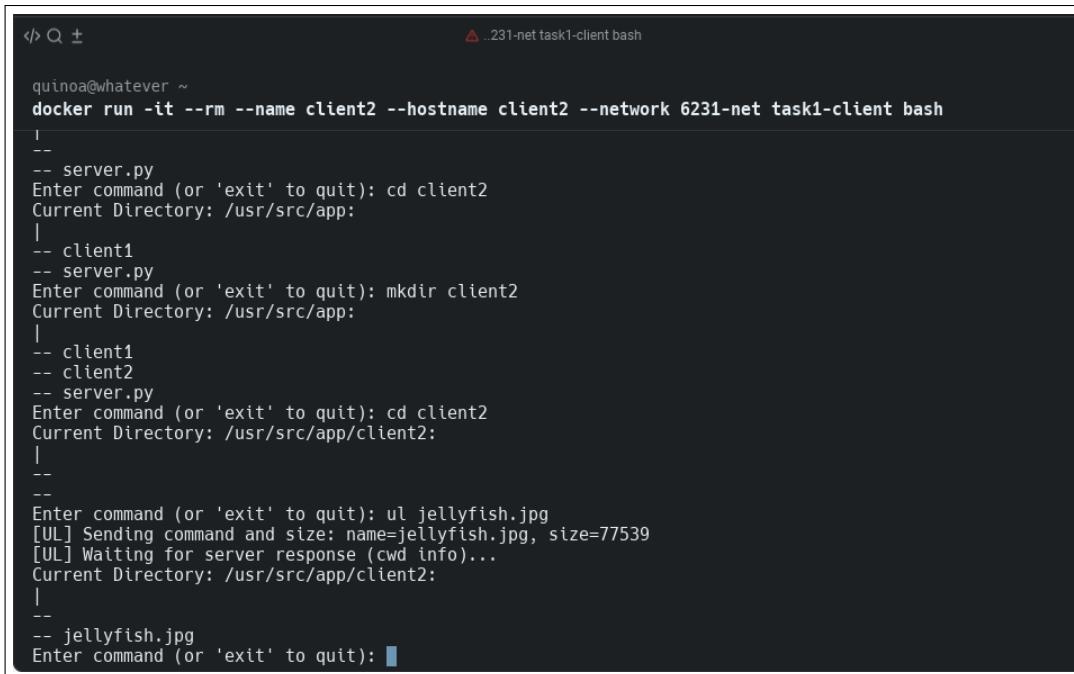
5.2 Client-Side Views

```

>/ Q ± ..231-net task1_client bash
binliu@binliu-RedmiBook ~/6231
docker run -it --rm --name client1 --hostname client1 --network 6231-net task1_client bash
--server.py manuel_ur_uuck.jpg tavo_vett.jpg
client.py orca.jpg user-guide-nokia-1-4-user-guide.pdf
root@client1:/usr/src/app# python client.py
Connected to server at IP: server and Port: 65432
Handshake Done. EOF is: bytearray(b'<qqTqhMQCw>')
Current Working Directory: Current Directory: /usr/src/app:
|
-- server.py
Enter command (or 'exit' to quit): mkdir client1
Current Directory: /usr/src/app:
|
-- client1
-- server.py
Enter command (or 'exit' to quit): cd client1
Current Directory: /usr/src/app/client1:
|
-- 
-- Enter command (or 'exit' to quit): ul orca.jpg
[UL] Sending command and size: name=orca.jpg, size=58111
[UL] Waiting for server response (cwd info)...
Current Directory: /usr/src/app/client1:
|
-- 
-- orca.jpg
Enter command (or 'exit' to quit): █

```

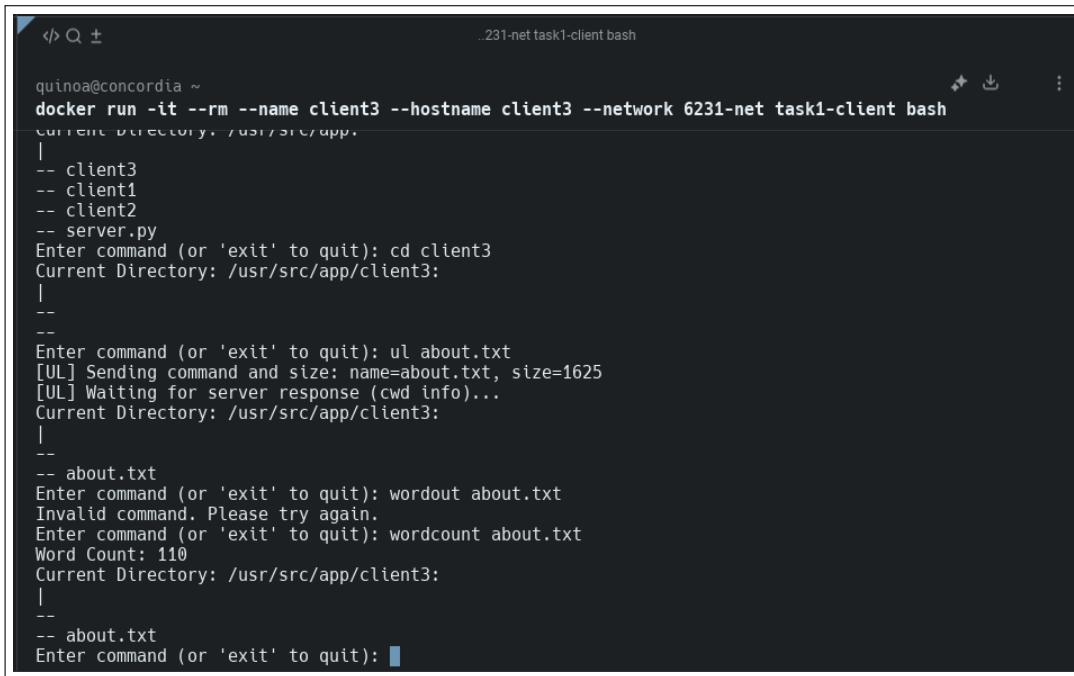
Figure 2: Client 1 successfully creating `client1` directory and sending files to server



```

</> Q ± ..231-net task1-client bash
quino@whatever ~
docker run -it --rm --name client2 --hostname client2 --network 6231-net task1-client bash
|
-- server.py
Enter command (or 'exit' to quit): cd client2
Current Directory: /usr/src/app:
|
-- client1
-- server.py
Enter command (or 'exit' to quit): mkdir client2
Current Directory: /usr/src/app:
|
-- client1
-- client2
-- server.py
Enter command (or 'exit' to quit): cd client2
Current Directory: /usr/src/app/client2:
|
-- jellyfish.jpg
[UL] Sending command and size: name=jellyfish.jpg, size=77539
[UL] Waiting for server response (cwd info)...
Current Directory: /usr/src/app/client2:
|
-- jellyfish.jpg
Enter command (or 'exit' to quit): █

```

Figure 3: Client 2 successfully creating `client2` directory and sending files to server


```

</> Q ± ..231-net task1-client bash
quino@concordia ~
docker run -it --rm --name client3 --hostname client3 --network 6231-net task1-client bash
|
-- client3
-- client1
-- client2
-- server.py
Enter command (or 'exit' to quit): cd client3
Current Directory: /usr/src/app/client3:
|
-- about.txt
[UL] Sending command and size: name=about.txt, size=1625
[UL] Waiting for server response (cwd info)...
Current Directory: /usr/src/app/client3:
|
-- about.txt
Enter command (or 'exit' to quit): wordout about.txt
Invalid command. Please try again.
Enter command (or 'exit' to quit): wordcount about.txt
Word Count: 110
Current Directory: /usr/src/app/client3:
|
-- about.txt
Enter command (or 'exit' to quit): █

```

Figure 4: Client 3 successfully creating `client3` directory and sending `about.txt` files to server and execute `wordcount about.txt` command