

Distributed and Parallel Computing Lab

CS461 Lab2

Name: Dipean Dasgupta

ID:202151188

TASK: Create and execute multi-server multi-client system in distributed environment.

Experimental Setup

Hardware and Software Requirements

- **Operating System:** Windows (Win10)
- **Docker Version:** 26.1.1, build 4cf5afa
- **Python Version:** 3.11
- **Docker Compose**

The experiment follows the folder structure below:

```
/CS461lab2
├── /client
│   ├── client.py
│   └── Dockfile
├── /server
│   ├── server.py
│   └── Dockfile
├── dock-compose.yml
└── requirements.txt (optional)
```

In docker, a threaded Python script manages numerous client connections while listening on a designated port is executed by the server and answer is echoed to each client that connects to the server and submits a message.

For building and running the docker environment; firstly navigate to root directory that contains the dock-compose.yml file.

```
cd E:\IIITV\SEM7\CS461\CS461lab2
```

Then the docker environment is built and started through:

```
docker-compose up --build
```

Docker Compose File

The [dock-compose.yml](#) file which is used to set up the multi-container environment:

```
version: '26.1.1'

services:
  server:
    build:
      context: ./server
    ports:
      - "12345:12345"

  client1:
    build:
      context: ./client
    environment:
      - SERVER_HOST=server
    command: ["python", "client.py", "Hello World from Client 1"]
  client2:
    build:
      context: ./client
    environment:
      - SERVER_HOST=server
    command: ["python", "client.py", "Hello World from Client 2"]

  client3:
    build:
      context: ./client
    environment:
      - SERVER_HOST=server
    command: ["python", "client.py", "Hello World from Client 3"]
```

Server Dockerfile

The dockerfile located in server directory:

```
FROM python:3.11-slim
WORKDIR /app
COPY server.py /app/

RUN pip install --no-cache-dir cryptography
CMD ["python", "server.py"]
```

Client Dockerfile

Dockerfile in client directory:

```
FROM python:3.11-slim
WORKDIR /app
COPY client.py /app/
RUN pip install --no-cache-dir cryptography
CMD ["python", "client.py"]
```

Code of Execution:

Server side:

```
import socket
import threading

def handle_client(client_socket):
    while True:
        try:
            message = client_socket.recv(1024).decode('utf-8')
            if not message:
                break
            print(f"Received message: {message}")
            response = f"Echo: {message}"
            client_socket.send(response.encode('utf-8'))
        except ConnectionResetError:
            break
        except Exception as e:
            print(f"Error: {e}")
            break

    print("Connection closed")
    client_socket.close()

def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('0.0.0.0', 12345))
    server_socket.listen(5)
    print("Server listening on port 12345")

    while True:
        try:
```

```

        client_socket, addr = server_socket.accept()
        print(f"Accepted connection from {addr}")
        client_handler = threading.Thread(target=handle_client,
args=(client_socket,))
        client_handler.start()
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()

```

Client Side:

```

import socket
import sys

def main():
    if len(sys.argv) != 2:
        print("Usage: python client.py <message>")
        sys.exit(1)

    message = sys.argv[1]
    server_ip = 'server' # Docker Compose service name for the server

    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((server_ip, 12345))

        # Send the message
        client_socket.send(message.encode('utf-8'))

        # Receive the response from the server
        response = client_socket.recv(1024).decode('utf-8')
        print(f"Server response: {response}")

        client_socket.close()
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()

```

OUTPUT:

The clients must be started in order for them to connect and send messages because the server is already operational. To accomplish this, execute the subsequent command for every client:

```
dock-compose up client1 client2 client3
```

In the output no input is taken from client side; sending directly from client.

```
[+] Running 3/0
✓ Container lab2-client1-1 Created
✓ Container lab2-client3-1 Created
✓ Container lab2-client2-1 Created
Attaching to client1-1, client2-1, client3-1
client1-1 | Server response: Echo: Hello world from Client 1
client3-1 | Server response: Echo: Hello world from Client 3
client2-1 | Server response: Echo: Hello world from Client 2
client1-1 exited with code 0
client3-1 exited with code 0
client2-1 exited with code 0
```

END of LAB ASSIGNMENT