Assignment 3 Report

By

Vishwajeet Ekal

(vekal@iu.edu)

Docker Installation:

1. I utilized Jetstream for Docker installation due to its preinstalled Docker in the root user.

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

exouser@vekal:~$ docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
exouser@vekal:~$
```

Directory Structure:

The directory structure for this assignment is as follows:
ASSIGNMENT3:

- Contains docker-compose.yml, server, and client.

client:

- Includes Dockerfile and client.py.

server:

- Contains Dockerfile and server.py.

```
exouser@vekal:~/ASSIGNMENT3$ ls -lrt
total 16
-rw-rw-r-- 1 exouser exouser 1223 Apr 25 20:09 docker-compose.yml
drwxrwxr-x 2 exouser exouser 4096 Apr 25 21:57 client
-rw-rw-r-- 1 exouser exouser   24 Apr 25 22:00 file.txt
drwxrwxr-x 2 exouser exouser 4096 Apr 25 22:06 server
exouser@vekal:~/ASSIGNMENT3$ cd client/
exouser@vekal:~/ASSIGNMENT3/client$ ls -lrt
total 8
-rw-rw-r-- 1 exouser exouser  456 Apr 25 21:12 Dockerfile
-rw-rw-r-- 1 exouser exouser 1453 Apr 25 21:57 client.py
exouser@vekal:~/ASSIGNMENT3/client$ cd ..
exouser@vekal:~/ASSIGNMENT3$ cd server/
exouser@vekal:~/ASSIGNMENT3/server$ ls -lrt
total 8
-rw-rw-r-- 1 exouser exouser 284 Apr 25 21:11 Dockerfile
-rw-rw-r-- 1 exouser exouser 890 Apr 25 22:06 server.py
exouser@vekal:~/ASSIGNMENT3/server$
```

Files:

1. Docker-compose.yml:
   - Defines two services, "server" and "client," connected to a custom network named "assignment3_app-network".
   - Each service utilizes named volumes ("servervol" and "clientvol").

- Volumes are utilized to persist data for the "server" and "client" services in the /serverdata and /clientdata directories respectively.
- The "client" service depends on the "server" service, ensuring proper service startup order.

```
exouser@vekal:~/ASSIGNMENT3$ cat docker-compose.yml
version: '3.8'

services:
  server:
    build: ./server             # Build the server service using the Dockerfile in the ./server directory
    volumes:
      - servervol:/serverdata   # Mount a volume for server data
    ports:
      - "8080:8080"             # Map port 8080 on the host to port 8080 in the container
    networks:
      - app-network             # Attach the service to the "app-network"

  client:
    build: ./client             # Build the client service using the Dockerfile in the ./client directory
    volumes:
      - clientvol:/clientdata   # Mount a volume for client data
    depends_on:
      - server                  # Ensure the server service is started before the client service
    networks:
      - app-network             # Attach the service to the "app-network"
    stdin_open: true            # Keep stdin open for the client service
    tty: true                   # Allocate a pseudo-TTY for the client service

volumes:
  servervol:                    # Define a volume for server data
  clientvol:                    # Define a volume for client data

networks:
  app-network:                  # Define a bridge network for communication between services
    driver: bridge

exouser@vekal:~/ASSIGNMENT3$
```

2. Dockerfile.server:
   - Sets up a Python 3.8-slim environment, sets the working directory as /app, and specifies a volume at /serverdata.
   - Copies the Python script named server.py into the container's /app directory.
   - Specifies the default command to run the Flask server on port 8080 when the container starts

```
exouser@vekal:~/ASSIGNMENT3/server$ cat Dockerfile
# Dockerfile for server
FROM python:3.8-slim

# Create directory for server data
RUN mkdir /serverdata

# Set working directory
WORKDIR /app

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

# Specify the command to run the server script
CMD ["python3", "server.py", "8080"]
```

3. Server.py:
   - Creates a random file (randomFile.txt).

- Calculates its SHA-256 checksum and accepts the file with the checksum included in the response headers.
- The server listens on port 8080.

```
exouser@vekal:~/ASSIGNMENT3$ cat server/server.py
import socket
import os
from hashlib import sha256

def createFileAndChecksum(path, size=1024):
    randomData = os.urandom(size)
    with open(path, 'wb') as file:
        file.write(randomData)
    return sha256(randomData).hexdigest()

def initiateServer(port):
    serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serverSocket.bind(('0.0.0.0', port))
    serverSocket.listen(1)
    print(f"Server listening on port {port}")
    while True:
        clientSocket, clientAddr = serverSocket.accept()
        print(f"Connection from {clientAddr}")
        checksum = createFileAndChecksum('/serverdata/file.txt')
        with open('/serverdata/file.txt', 'rb') as file:
            clientSocket.sendall(file.read())
            clientSocket.sendall(checksum.encode())
        clientSocket.close()

if __name__ == '__main__':
    import sys
    port = int(sys.argv[1])
    initiateServer(port)

exouser@vekal:~/ASSIGNMENT3$
```

4. Dockerfile.client
   - Sets up a Python 3.8 environment for a client application, installs the requests library, and copies a Python script (client.py) into the container's /app directory.
   - Defines a volume at /clientdata.

```
exouser@vekal:~/ASSIGNMENT3/client$ cat Dockerfile
# Use the official Python 3.8 slim image as base
FROM python:3.8-slim

# Create a directory for storing client data inside the container
RUN mkdir /clientdata

# Set the working directory to /app inside the container
WORKDIR /app

# Copy the client.py script from the host into the container's /app directory
COPY client.py /app/

# Define the command to run the client script when the container starts
CMD ["python3", "-u", "client.py", "server", "8080"]
```

5. Client.py:
    - Defines a client application that retrieves a file with checksum from a specified server URL (http://server:8080/).
    - Saves the file at /clientdata/receivedFile.txt, and verifies its checksum using SHA-256.
    - Prints an error message if the checksum verification fails.

```
import socket
import sys
from hashlib import sha256

def checkFileIntegrity(filePath, expectedHash):
    with open(filePath, 'rb') as fileStream:
        data = fileStream.read()
    calculatedHash = sha256(data).hexdigest()
    return calculatedHash == expectedHash, calculatedHash

def main(serverAddr, serverPort):
    try:
        clientSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        clientSock.connect((serverAddr, serverPort))

        with open('/clientdata/received_file.txt', 'wb') as fileStream:
            data = clientSock.recv(1024)
            while data:
                fileStream.write(data)
                data = clientSock.recv(1024)
        receivedHash = clientSock.recv(64).decode()
        valid, calculatedHash = checkFileIntegrity('/clientdata/received_file.txt', receivedHash)

        if valid:
            print("File received and verified successfully.")
        else:
            print(f"File verification failed. Expected {receivedHash}, got {calculatedHash}")

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        clientSock.close()
        input("Press Enter to exit...")

if __name__ == '__main__':
    if len(sys.argv) < 3:
        print("Usage: python client.py <server_ip> <server_port>")
        sys.exit(1)

    serverAddr = sys.argv[1]
    serverPort = int(sys.argv[2])
    main(serverAddr, serverPort)

exouser@vekal:~/ASSIGNMENT3$
```

Questions:

Build and Run your server and client container.

- I utilized docker-compose.yml to define and configure multi-containers and built and ran the containers using the following steps:
- Executed docker-compose build to build the containers (assignment3_client_1 and assignment3_server_1).
- Ran the services mentioned in the Docker Compose file using docker-compose up -d.
- Used the -d flag for detached mode to run services in the background.

```
exouser@vekal:~/ASSIGNMENT3$ docker-compose up --build -d
Building server
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  3.584kB
Step 1/5 : FROM python:3.8-slim
 ---> 5b29acce18dd
Step 2/5 : RUN mkdir /serverdata
 ---> Using cache
 ---> f7013bd03b5f
Step 3/5 : WORKDIR /app
 ---> Using cache
 ---> dcc41bb54e4b
Step 4/5 : COPY server.py /app/
 ---> Using cache
 ---> efc31e8021b5
Step 5/5 : CMD ["python3", "server.py", "8080"]
 ---> Using cache
 ---> 71d4794c72fa
Successfully built 71d4794c72fa
Successfully tagged assignment3_server:latest
Building client
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  4.096kB
Step 1/5 : FROM python:3.8-slim
 ---> 5b29acce18dd
Step 2/5 : RUN mkdir /clientdata
 ---> Using cache
 ---> 6606d2218bad
Step 3/5 : WORKDIR /app
 ---> Using cache
 ---> e7009af3ec91
Step 4/5 : COPY client.py /app/
 ---> Using cache
 ---> 518b5bf6fd7f
Step 5/5 : CMD ["python3", "-u", "client.py", "server", "8080"]
 ---> Using cache
 ---> e4ab43a85309
Successfully built e4ab43a85309
Successfully tagged assignment3_client:latest
Starting assignment3_server_1 ... done
Starting assignment3_client_1 ... done
```

- assignment3_app-network has been established, featuring two volumes named assignment3_servervol and assignment3_clientvol, and hosting two services, namely assignment3_server_1 and assignment3_client_1.
- Below, you can see the result of the docker-compose ps command, confirming that both containers are up and functioning.

Communication between the two:

- Communication between containers is facilitated through client.py and server.py, enabling file transfer within the network connecting the client and server containers.
- A network titled assignment3_app_network has been established, as depicted in the accompanying screenshot.
- Both assignment3_server_1 and assignment3_client_1 containers are operating within the same network, assignment3_app-network, as evidenced in the provided screenshot.
- These configurations have been set up in docker-compose.yml, as detailed in the Files section above.
- Additionally, two volumes, assignment3_clientvol and assignment3_servervol, have been created to store container data.

```
exouser@vekal:~/ASSIGNMENT3$ docker-compose ps
        Name                       Command               State                    Ports
-------------------------------------------------------------------------------------------------------------
assignment3_client_1   python3 -u client.py serve ...   Up
assignment3_server_1   python3 server.py 8080           Up      0.0.0.0:8080->8080/tcp,:::8080->8080/tcp
exouser@vekal:~/ASSIGNMENT3$
exouser@vekal:~/ASSIGNMENT3$ docker network ls
NETWORK ID     NAME                        DRIVER    SCOPE
79c0f30f2594   assignment3_app-network     bridge    local
d8b94f3f1a83   bridge                      bridge    local
0f362a52020b   guacamole_exo-guac-net      bridge    local
16b638b4f904   host                        host      local
9d90c106fd49   none                        null      local
exouser@vekal:~/ASSIGNMENT3$
exouser@vekal:~/ASSIGNMENT3$ docker volume ls
DRIVER    VOLUME NAME
local     assignment3_clientvol
local     assignment3_servervol
```

Client Container Shell:

- Here is the snapshot of the client container shell.
- I accessed the client container shell using the command docker-compose exec client sh.
- I proceeded to navigate to the /clientdata directory to confirm the reception of the file.
- As illustrated below, the file has been successfully received, and the checksum matches as expected.