Lesson-End Project

Implementing Microservice Architecture with Docker

Project agenda: To develop and deploy a basic microservices architecture using Docker, which involves creating a web server and a client service. The focus is on isolating each component within its own Docker container to streamline development, testing, and deployment processes.

Description: As a DevOps engineer in a fast-paced startup, your current project focuses on establishing a streamlined microservices architecture for a specific application that serves static content via a web server and a corresponding client, both encapsulated within Docker containers. This architecture supports rapid development, deployment, and scaling and enhances system robustness by isolating each service component. By leveraging Docker and Docker Compose, your task is to ensure that all environments, from development to production, are consistent, and changes can be deployed swiftly and reliably. This setup aims to provide a foundational model for future expansions and service integrations.

Tools required: Docker, Docker Compose, and Python

Prerequisites: None

Steps to be followed:

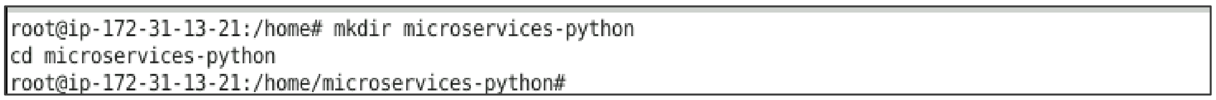
1. Set up project directories and server
2. Configure the client and server Dockerfiles
3. Create and configure Docker Compose
4. Build and deploy with Docker Compose

Step 1: Set up project directories and server

1.1 Run the following command to create and navigate the new directory:

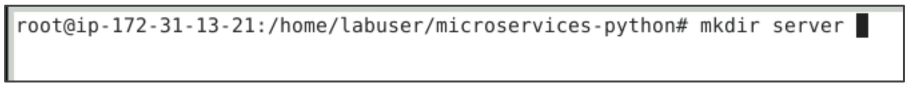
mkdir microservices-python

cd microservices-python



```
root@ip-172-31-13-21:/home# mkdir microservices-python
cd microservices-python
root@ip-172-31-13-21:/home/microservices-python#
```

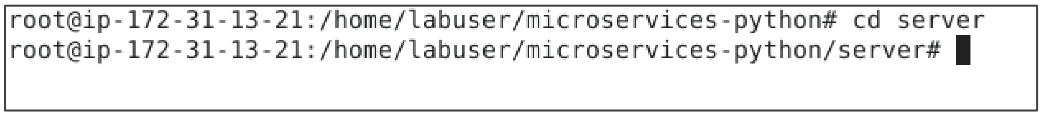
1.2 Run the following command to create a directory for server configurations:
mkdir server



```
root@ip-172-31-13-21:/home/labuser/microservices-python# mkdir server
```

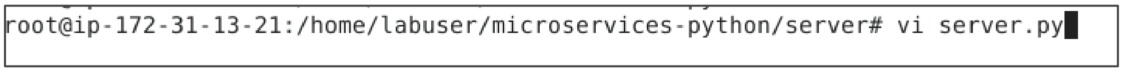
Step 2: Configure the client and server Dockerfiles

2.1 Run the following command to navigate the **server** directory:
cd server



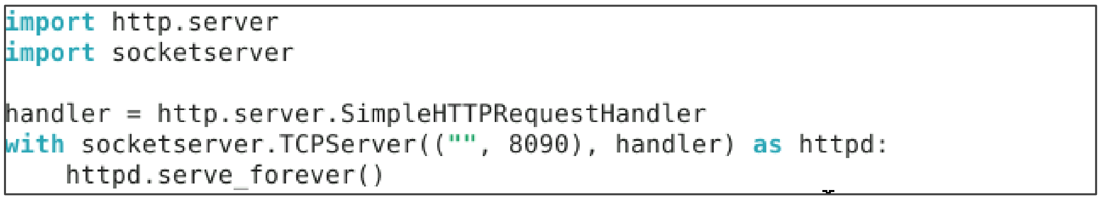
```
root@ip-172-31-13-21:/home/labuser/microservices-python# cd server
root@ip-172-31-13-21:/home/labuser/microservices-python/server#
```

2.2 Execute the following command to create the server files:
vi server.py



```
root@ip-172-31-13-21:/home/labuser/microservices-python/server# vi server.py
```

2.3 Add the following Python script in the **server.py** file:
import http.server
import socketserver
handler = http.server.SimpleHTTPRequestHandler
with socketserver.TCPServer(("", 8090), handler) as httpd:
httpd.serve_forever()

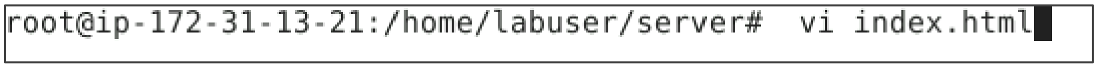


```
import http.server
import socketserver

handler = http.server.SimpleHTTPRequestHandler
with socketserver.TCPServer(("", 8090), handler) as httpd:
    httpd.serve_forever()
```

This code sets up a simple HTTP server on port 8090 using Python's built-in libraries. It handles requests with the default **SimpleHTTPRequestHandler** command.

2.4 Execute the following command to create the **index.html** file for the server:

vi index.html



```
root@ip-172-31-13-21:/home/labuser/server#  vi index.html
```

2.5 Add the following script in the **index.html** file to define the webpage:

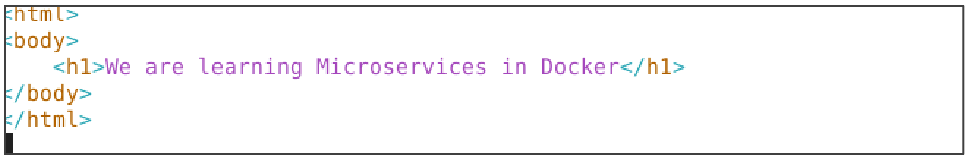
<html>

<body>

<h1>We are learning Microservices in Docker</h1>

</body>

</html>

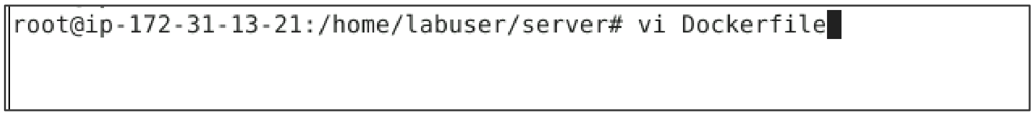


```
<html>
<body>
   <h1>We are learning Microservices in Docker</h1>
</body>
</html>
```

This HTML code will display a webpage with a heading that reads **We are learning Microservices in Docker**.

2.6 Run the following command to create the Dockerfile for the server:

vi Dockerfile



```
root@ip-172-31-13-21:/home/labuser/server# vi Dockerfile
```

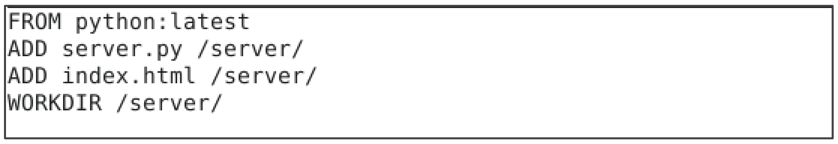
2.7 Add the following script in the **Dockerfile**:

FROM python:latest

ADD server.py /server/

ADD index.html /server/

WORKDIR /server/

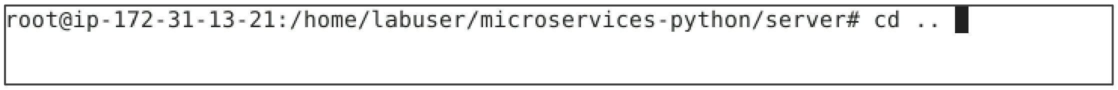


```
FROM python:latest
ADD server.py /server/
ADD index.html /server/
WORKDIR /server/
```

This **Dockerfile** creates a Python-based container and sets up a directory **/server/** with added files **server.py** and **index.html**, making **/server/** the working directory.

2.8 Execute the following command to exit from the current directory:

cd ..

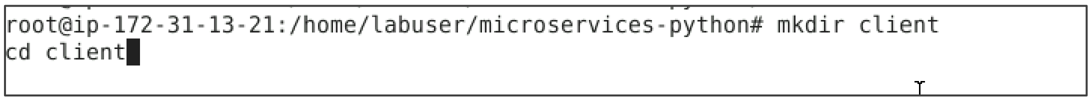


```
root@ip-172-31-13-21:/home/labuser/microservices-python/server# cd ..
```

2.9 Run the following command to create the client directory and navigate it:

mkdir client

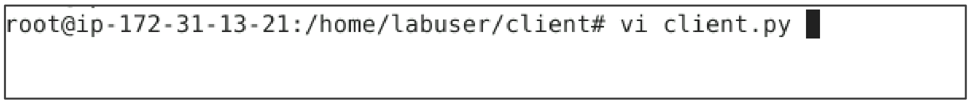
cd client



```
root@ip-172-31-13-21:/home/labuser/microservices-python# mkdir client
cd client
```

2.10 Run the following command to create or edit the file **client.py**:

vi client.py



```
root@ip-172-31-13-21:/home/labuser/client# vi client.py
```

2.11 Add the following script to the **client.py** file:

import urllib.request

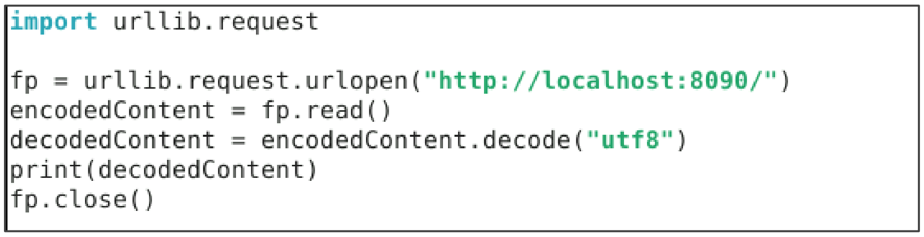
fp = urllib.request.urlopen("http://localhost:8090/")

encodedContent = fp.read()

decodedContent = encodedContent.decode("utf8")

print(decodedContent)

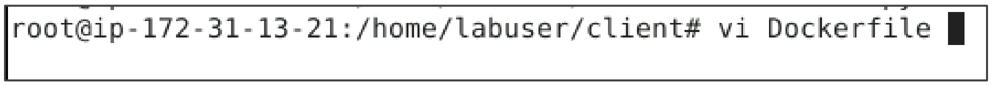
fp.close()



```
import urllib.request

fp = urllib.request.urlopen("http://localhost:8090/")
encodedContent = fp.read()
decodedContent = encodedContent.decode("utf8")
print(decodedContent)
fp.close()
```

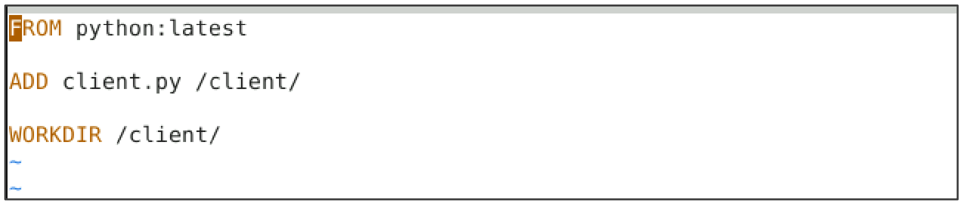
This Python script retrieves the content from the web server running on localhost at port 8090, decodes it from UTF-8 format, and prints the resulting text.

2.12 Run the following command to create the Dockerfile for the client:
vi Dockerfile



```
root@ip-172-31-13-21:/home/labuser/client# vi Dockerfile
```

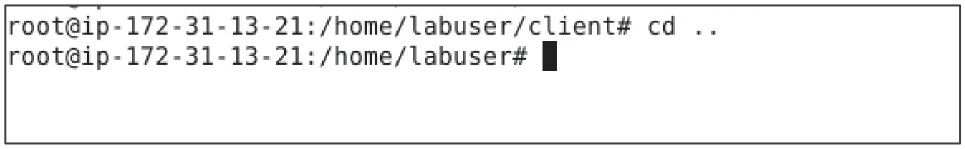
2.13 Add the following script in the **Dockerfile** to define the client Docker environment:
FROM python:latest
ADD client.py /client/
WORKDIR /client/



```
FROM python:latest

ADD client.py /client/

WORKDIR /client/
~
~
```

This Dockerfile snippet sets up a container using the latest Python image, adds a file named **client.py** to the **/client/** directory inside the container, and sets **/client/** as the working directory for executing commands.

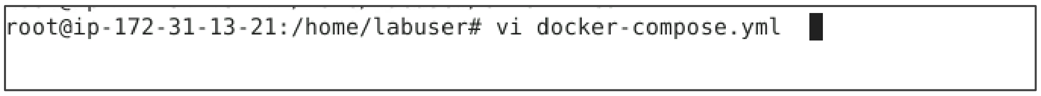
2.14 Execute the following command to exit from the current directory:
cd ..



```
root@ip-172-31-13-21:/home/labuser/client# cd ..
root@ip-172-31-13-21:/home/labuser#
```

Step 3: Create and configure Docker Compose

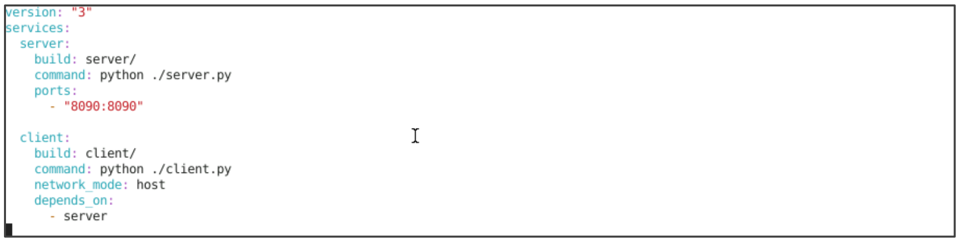
3.1 Execute the following command to create the **docker-compose.yml** file:
vi docker-compose.yml



```
root@ip-172-31-13-21:/home/labuser# vi docker-compose.yml
```

3.2 Add the following script in the **docker-compose.yml** file:

```
version: "3"
services:
  server:
    build: server/
    command: python ./server.py
    ports:
      - 8090:8090
  client:
    build: client/
    command: python ./client.py
    network_mode: host
    depends_on:
      - server
```



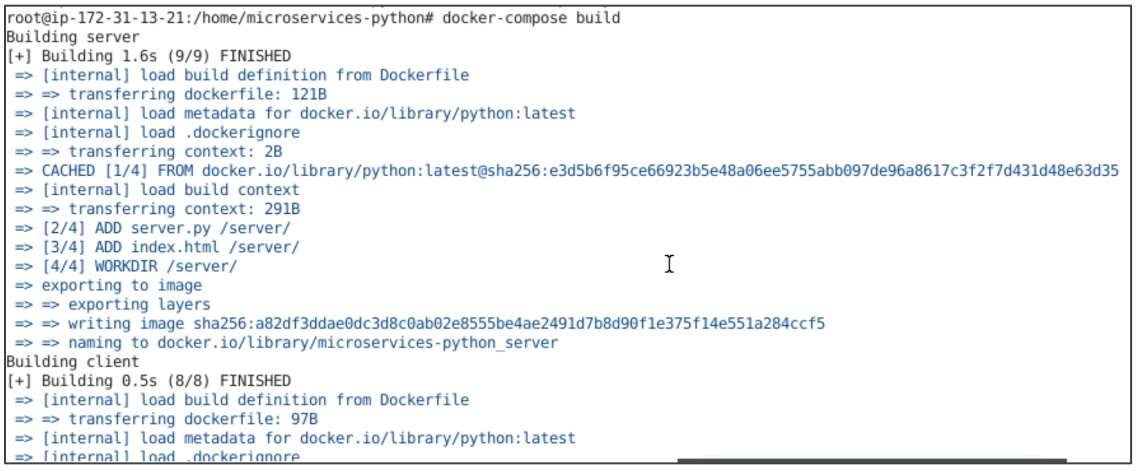
```
version: "3"
services:
  server:
    build: server/
    command: python ./server.py
    ports:
      - "8090:8090"

  client:
    build: client/
    command: python ./client.py
    network_mode: host
    depends_on:
      - server
```

This Docker Compose file sets up two services: a server running on port 8090 and a client using the host network, which starts only after the server runs.

Step 4: Build and deploy with Docker Compose

4.1 Run the following command to build the Docker containers:

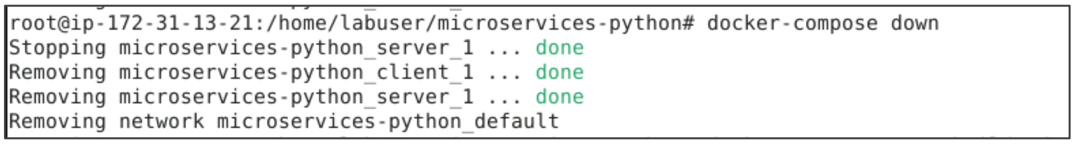
docker-compose build



```
root@ip-172-31-13-21:/home/microservices-python# docker-compose build
Building server
[+] Building 1.6s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 121B
 => [internal] load metadata for docker.io/library/python:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/4] FROM docker.io/library/python:latest@sha256:e3d5b6f95ce66923b5e48a06ee5755abb097de96a8617c3f2f7d431d48e63d35
 => [internal] load build context
 => => transferring context: 291B
 => [2/4] ADD server.py /server/
 => [3/4] ADD index.html /server/
 => [4/4] WORKDIR /server/
 => exporting to image
 => => exporting layers
 => => writing image sha256:a82df3ddae0dc3d8c0ab02e8555be4ae2491d7b8d90f1e375f14e551a284ccf5
 => => naming to docker.io/library/microservices-python_server
Building client
[+] Building 0.5s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 97B
 => [internal] load metadata for docker.io/library/python:latest
 => [internal] load .dockerignore
```

4.2 Run the following command to stop and remove Docker containers, networks, and volumes:

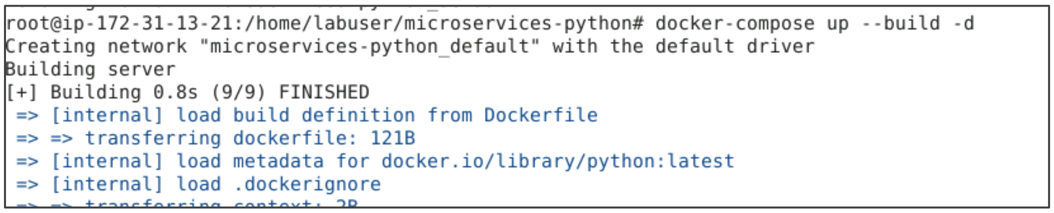
docker-compose down



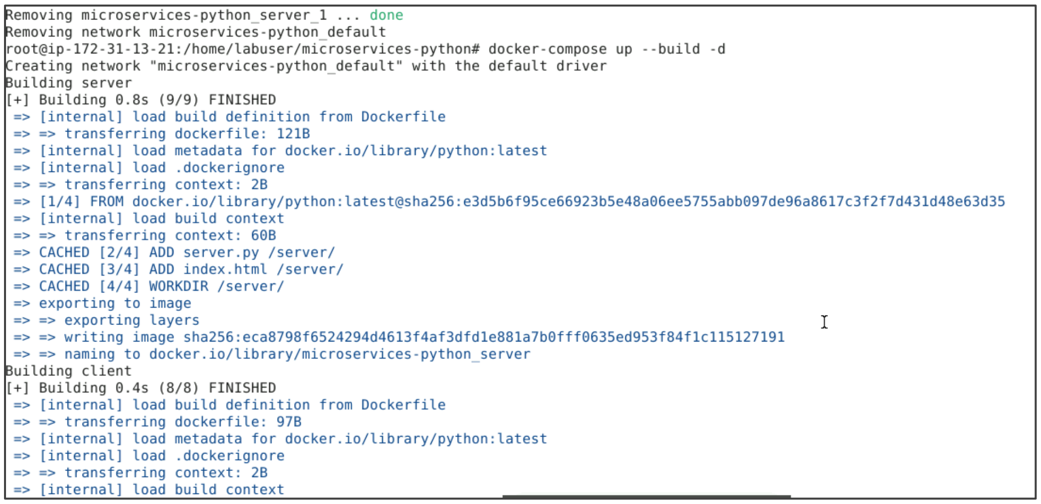
```
root@ip-172-31-13-21:/home/labuser/microservices-python# docker-compose down
Stopping microservices-python_server_1 ... done
Removing microservices-python_client_1 ... done
Removing microservices-python_server_1 ... done
Removing network microservices-python_default
```

4.3 Run the following command to build images and start containers in detached mode:

docker-compose up --build -d

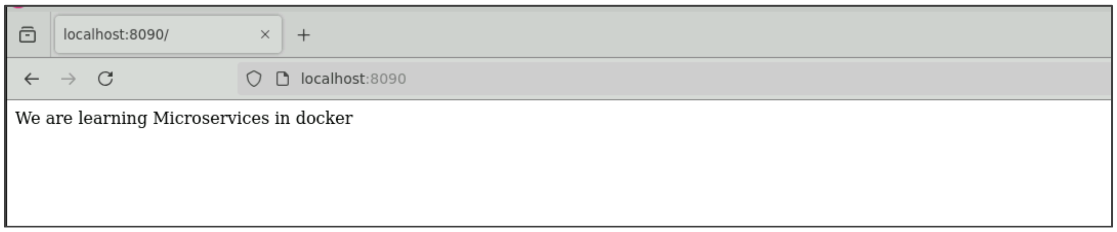


```
root@ip-172-31-13-21:/home/labuser/microservices-python# docker-compose up --build -d
Creating network "microservices-python_default" with the default driver
Building server
[+] Building 0.8s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 121B
 => [internal] load metadata for docker.io/library/python:latest
 => [internal] load .dockerignore
```



```
Removing microservices-python_server_1 ... done
Removing network microservices-python_default
root@ip-172-31-13-21:/home/labuser/microservices-python# docker-compose up --build -d
Creating network "microservices-python_default" with the default driver
Building server
[+] Building 0.8s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 121B
 => [internal] load metadata for docker.io/library/python:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/4] FROM docker.io/library/python:latest@sha256:e3d5b6f95ce66923b5e48a06ee5755abb097de96a8617c3f2f7d431d48e63d35
 => [internal] load build context
 => => transferring context: 60B
 => CACHED [2/4] ADD server.py /server/
 => CACHED [3/4] ADD index.html /server/
 => CACHED [4/4] WORKDIR /server/
 => exporting to image
 => => exporting layers
 => => writing image sha256:eca8798f6524294d4613f4af3dfd1e881a7b0fff0635ed953f84f1c115127191
 => => naming to docker.io/library/microservices-python_server
Building client
[+] Building 0.4s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 97B
 => [internal] load metadata for docker.io/library/python:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
```

4.4 Verify the deployment by accessing the server at **http://localhost:8090**



We are learning Microservices in docker

Note: In an ideal situation, there will be multiple machines on which client and server containers will be deployed, and the same can be configured in your compose file.

By following these steps, you have successfully created a scalable microservices architecture using Docker. This setup involves separate containers for the server and the client components, allowing for efficient management and isolation of services.