**Lesson 07 Demo 01**

**Creating a Simple Microservices Architecture Design**

**Objective:** To implement a simple microservices architecture using Docker Compose for showcasing the interaction between a server and client in a containerized environment

**Tools required:** Docker

**Prerequisites:** Knowledge of Python programming language

Steps to be followed:

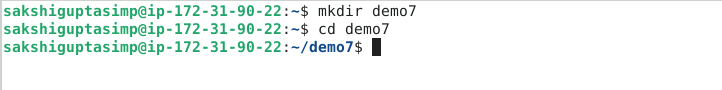
1. Create the server
2. Create the client
3. Create a Docker Compose file
4. Install and run Docker Compose

**Step 1: Create the server**

1. Create a directory for your project and navigate using the following commands:

**mkdir demo7**

**cd demo7**



1. Create a directory for the server component and navigate using the following commands:

**mkdir server**

**cd server**



1. Create a Python script file named **server.py** using the following command:

**vi server.py**



1. Add the following Python script to the **server.py** file:

**import http.server**

**import socketserver**

**handler = http.server.SimpleHTTPRequestHandler**

**with socketserver.TCPServer(("", 8090), handler) as httpd:**

**httpd.serve\_forever()**



The script imports the **http.server** and **socketserver** modules, sets up a simple HTTP request handler, and starts a TCP server on port 8090 to handle incoming requests indefinitely.

1. Create an **index.html** file using the following command:

**vi index.html**



1. Add the following content to the **index.html** file:

**We are learning Microservices in docker**



1. Create a Dockerfile for the server using the following command:

**vi Dockerfile**

****

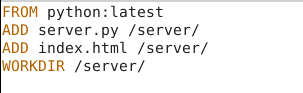
1. Add the following script to the **Dockerfile**:

**FROM python:latest**

**ADD server.py /server/**

**ADD index.html /server/**

**WORKDIR /server/**



1. Navigate back to the project directory using the following command:

**cd ..**

****

**Step 2: Create the client**

1. Create a directory for the client component using the following commands:

**mkdir client**

**cd client**

****

1. Create a Python script file named **client.py** using the following command:

**vi client.py**



1. Add the following Python 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()**

****

1. Create a Dockerfile for the client using the following command:

**vi Dockerfile**

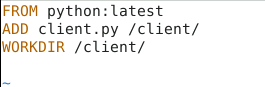


1. Add the following script to the **Dockerfile**:

**FROM python:latest**

**ADD client.py /client/**

**WORKDIR /client/**

****

1. Navigate back to the project directory using the following command:

**cd ..**



**Step 3: Create a Docker Compose file**

1. Create a docker-compose.yml file using the following command:

vi docker-compose.yml



1. Add the following content to docker-compose.yml:

**version: "3"**

**services:**

**server:**

**build: server/**

**command: python ./server.py**

**ports:**

**- 8090:8090**

**client:**

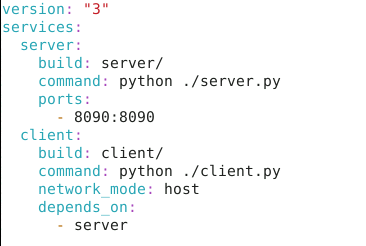
**build: client/**

**command: python ./client.py**

**network\_mode: host**

**depends\_on:**

**- server**



**Step 4: Install and run Docker Compose**

1. Install Docker Compose using the following command:

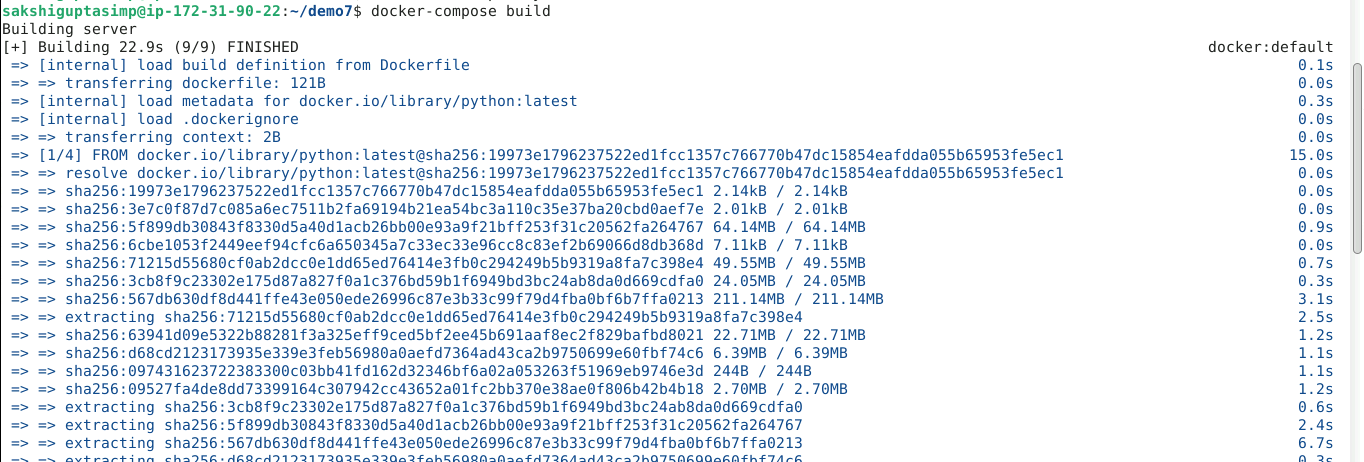
**sudo apt install docker-compose -y**

**A screenshot of a computer program

Description automatically generated**

1. Build the Docker Compose configuration using the following command:

**docker-compose build**



1. Check the built Docker images using the following command:

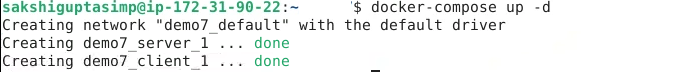
**docker images**

A screenshot of a computer

Description automatically generated

1. Start the Docker containers using the following command:

**docker-compose up -d**



1. Open a web browser and navigate the URL http://localhost:8090



By following these steps, you have successfully implemented a simple microservices architecture using Docker Compose for showcasing the interaction between a server and client in a containerized environment.