Part 2 Design Note

Design note for containerizing and deploying microservices using Docker.

Design choices:

1) Dockerizing the Microservices:
    a) Used Alpine which is a lightweight base image to keep the size of the Docker image small and optimize performance.
    b) Set the working directory to the location of the application code.
    c) A requirement.txt file for the created project was exported and copied into the working directory. It contains information about all the libraries, modules, and packages which are specific to the project while developing it.
    d) Installed all the dependencies in the requirements.txt file using pip.
    e) Copied the application code of the microservice into the image.
    f) Used ENTRYPOINT instruction which is used to configure the executables that will run after the container is started.
    g) Tested the Dockerfile by building and the running the Docker image locally to ensure the microservice runs as expected.
2) Shell script to build images:
    a) Created a build.sh script that build all the docker images in one go.
3) Writing a Docker Compose file:
    a) Definition of the services:
        i) Defined all three services using the "services" keyword in the Docker Compose file.
        ii) Adding a build command, to build the image if it is not built already, by passing the location and name of the docker file since three different docker files are in the same root folder.
        iii) Specified the docker image for each service using the “image” keyword.
        iv) Specified custom container name using the “container-name” keyword.
    b) Communication of the services:
        i) Used the "depends_on" keyword to specify the order in which the services should be started, ensuring that dependencies are available before the service starts.
        ii) Used the "networks" keyword to create a user defined bridge network for the services to communicate.

iii) Used the “ports” keyword to create a port forwarding when starting the application container.
iv) Hostnames are added in the server files so it can communicate with the containers in the allocated ports in the user defined network.

c) Mount a directory on the host as a volume:
   i) Used the "volumes" keyword to mount a directory on the host as a volume to the catalog and order services.
   ii) Set the source to the path on the host system as “./backend/files”.
   iii) Set the target to the location inside the container as “/app/data”, where the files will be mounted.

d) Deploying the Microservices:
   i) Used the "docker-compose up" command to start all the containers defined.

4) Latency measurement: The client code measures the latency of each request by calculating the time taken between sending the request and receiving the response. The latency for each run is added, and the total latency for all runs is calculated.

Overall, the design choices helped in containerizing all the micro-services and managing an application consisting of multiple containers using Docker Compose.