Part 2 Design Document

The Part 2 code is to containerize the Two-Tiered Stock Bazaar Microservices and deploy it as a distributed application using docker.

Design Choices:

1. **Dockerfiles** : Created 3 Dockerfiles for the 3 corresponding microservices as below:

    a. catalog.Dockerfile:
        i. Created a Dockerfile named catalog.Dockerfile for catalog service that lookups the stockname from the catalog database.
        ii. Sets the working directory to /app and used the base image python:3.9-alpine for the Docker container.
        iii. Copies 3 application files into the Docker container, catalog_service.py, stocks_DB.csv, and Read_Write_Lock.py. Read_Write_Lock.py is a python file defined to manage the read write locks while accessing the database.
    b. Order.Dockerfile:
        i. Created a Dockerfile named order.Dockerfile for order service that lookups the stockname from the catalog database and buy/sell the quantity based on the probability and maintain an order log.
        ii. Copies 3 application files into the Docker container, orders_service.py, orders_DB.csv, and Read_Write_Lock.py.
    c. Frontend.Dockerfile:
        i. Frontend.Dockerfile is for the frontend service which supports Lookup and Trade requests from the client using HTTP REST APIs.

2. **Docker-compose file** : Created the compose file for building and tearing up the entire application in a single command.

    a. Inside the docker-compose , Under the services section ,defined the 3 services and named the containers respectively catalog_microservice,frontend_microservice and orders_microservice which are built from the dockerfiles created.
    b. Used the version: "3.9" of the docker compose file format.
    c. Inside the ‘dockerfile’ section, the name of the dockerfile to use for building the service.
    d. **Data Volume Mount** : Specified the volume map between host and container in the ‘volume ‘section.This maps the stocks_DB and orders_DB to the volume in respective containers so that the data will be persisted even after the removal of containers.
    e. In Ports section specified the port that microservice going to expose.

3. **Frontend-server**:

    a. Modified the IP address to the container service name (here: frontend_service) so that , it can locate the catalog and order service in network.

4. **Back-end** :

    a. Catalog_service: Modified the IP address to the container name (catalog_service).
    b. Orders_service: Modified the IP address to the container name (orders_service).

5. **Shell Script** : developed a shell script to buid the dockerfile images for 3 microservices has been implemented using the ‘ docker build -f <dockerfile> -t <container-name> ‘ command.
6. **Latency :** The latency is calculated for 1000 requests both trade and lookup from the client application and recorded.

The Docker compose up starts all the containers and client and can access the exposed port of frontend and send requests.

In conclusion, This docker-compose.yml file provides a simple and effective way to define and manage the different services in a microservices-based application using Docker Compose. By using volumes to map the database files from the host system to the containers, the application can store and retrieve data in a consistent and reliable manner.