**Project Report**



AL-NAJAH UNIVERSITY

**DOS**

SUBMITTED TO:

**Dr.Samer Arandi**

PREPARED BY:

**Shahd Khader**

12027906

**Ahmad Nazzal**

12027687

# Architecture:

The Bazar.com system follows a microservice architecture consisting of three main services:

1. **Front-end Service**:
   * Acts as the user interface.
   * Receives requests from the users and forwards them to the appropriate backend services.
   * Provides three main operations:
     + **search(topic)**: Queries the catalog service by topic.
     + **info(item\_number)**: Fetches detailed information about a book.
     + **purchase(item\_number)**: Initiates the purchasing process by interacting with the order service.
2. **Catalog Service**:
   * Manages the inventory of books.
   * Stores details like the number of items in stock, price, and topic.
   * Supports two operations:
     + **query** (by subject or item number): Retrieves available books based on a given topic or item number.
     + **update**: Updates stock levels or modifies the price of a book.
3. **Order Service**:
   * Processes purchase requests by verifying stock availability and updating the catalog accordingly.
   * Supports a single operation:
     + **purchase(item\_number)**: Verifies item availability with the catalog service and updates the stock if a purchase is successful.

# How the project works

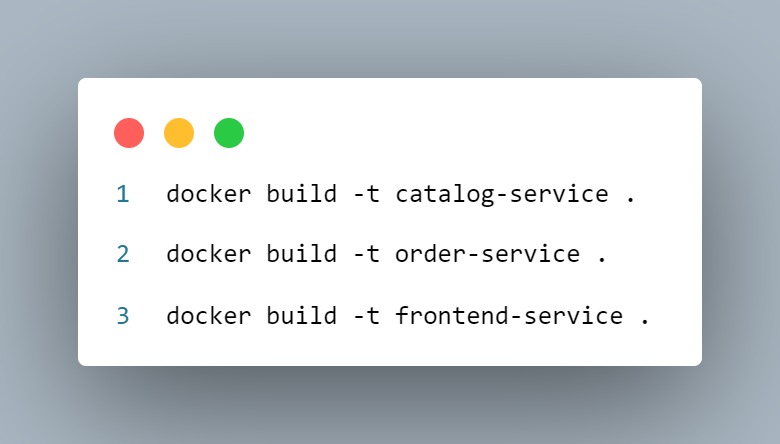
* Git clone <https://github.com/ShahdKhader/Dos.git>
* docker-compose up --build
* Inside the Desktop docker for the frontend service container, open the exec tab and run the command node cli.js.

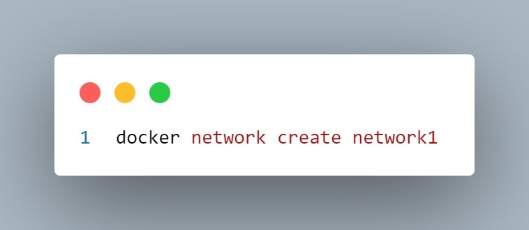
# Work and Results

At first, we wrote the Dockerfile for each service.

#### Frontend service catalog service

#### Order service

Then we need to build the images for each service

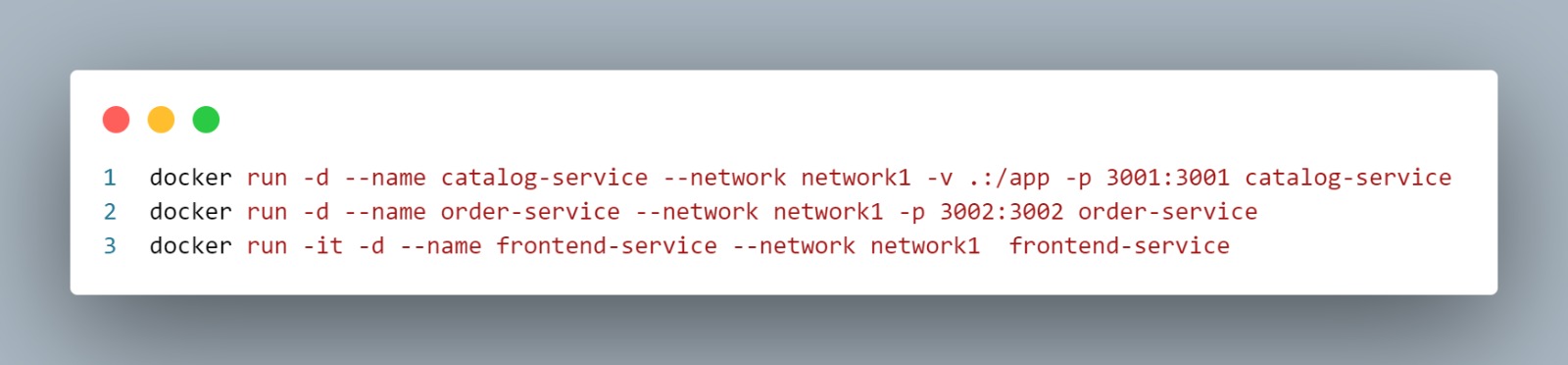
We need to establish a network to enable communication between the containers.

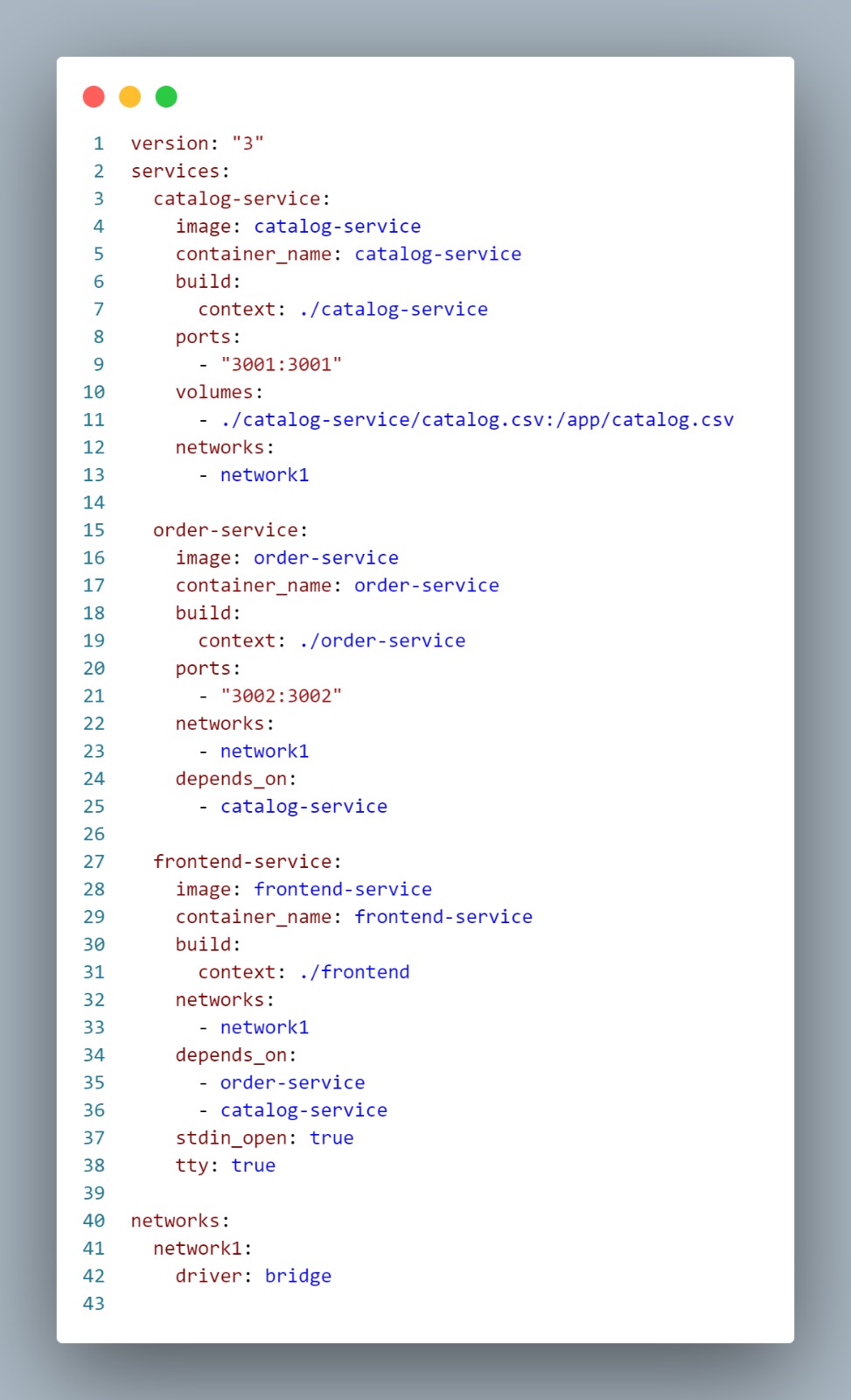
Now we will run the images to make containers, The provided Docker commands launch three services within a specified Docker network named network1.

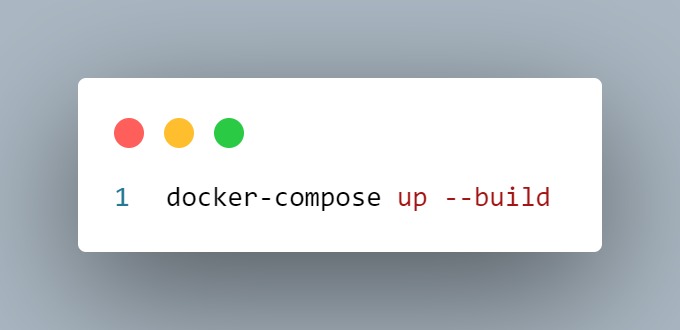
The catalog-service runs in detached mode with its current directory mounted to /app inside the container and maps port 3001 on the host to port 3001 in the container.

The order-service also runs in detached mode, mapping port 3002 on the host to port 3002 in the container.

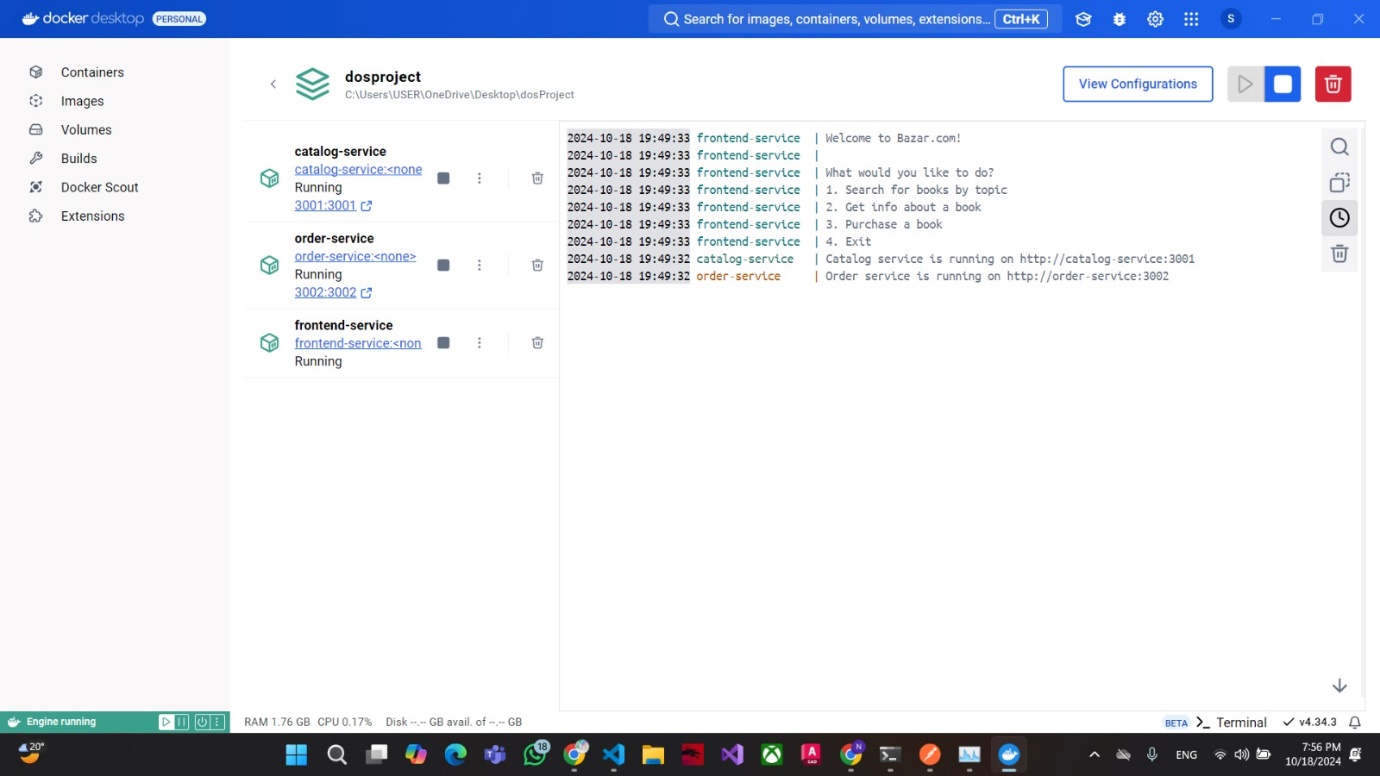
The frontend-service is initiated in detached mode with interactive terminal support, allowing for user interaction while it operates within the same network. Together, these commands set up a microservices architecture where each service can communicate over the defined network.



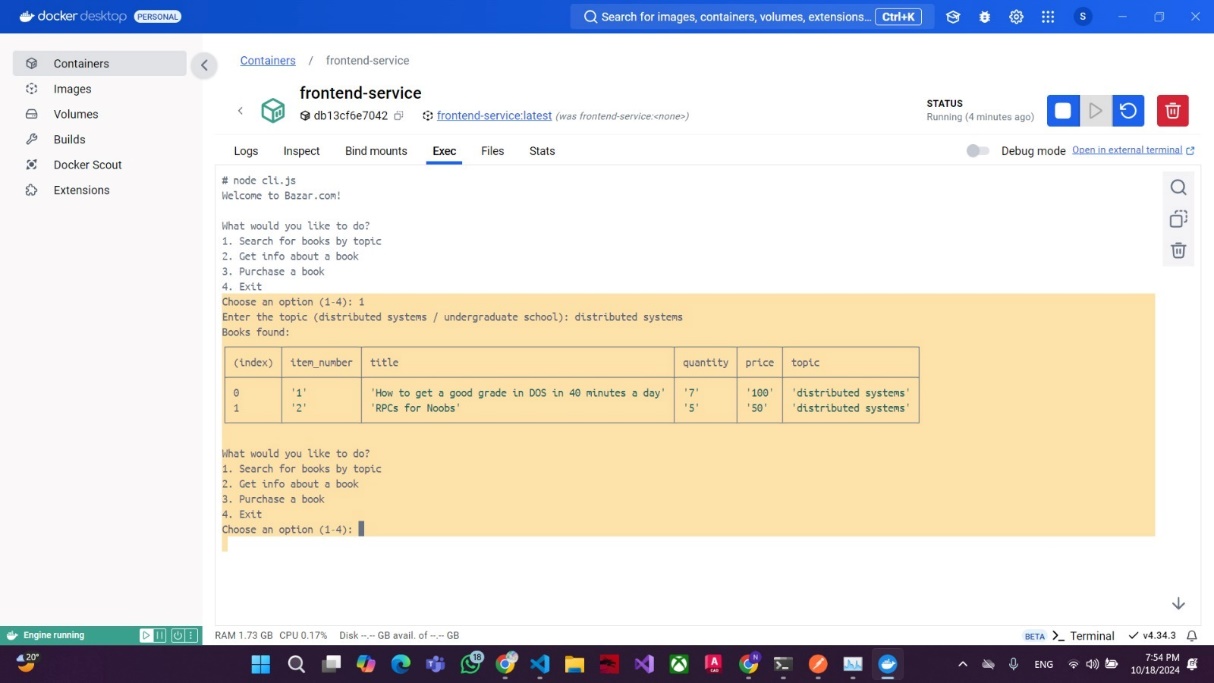
However, Instead of running each container individually, we can utilize Docker Compose to automate the deployment and management of our containerized applications.

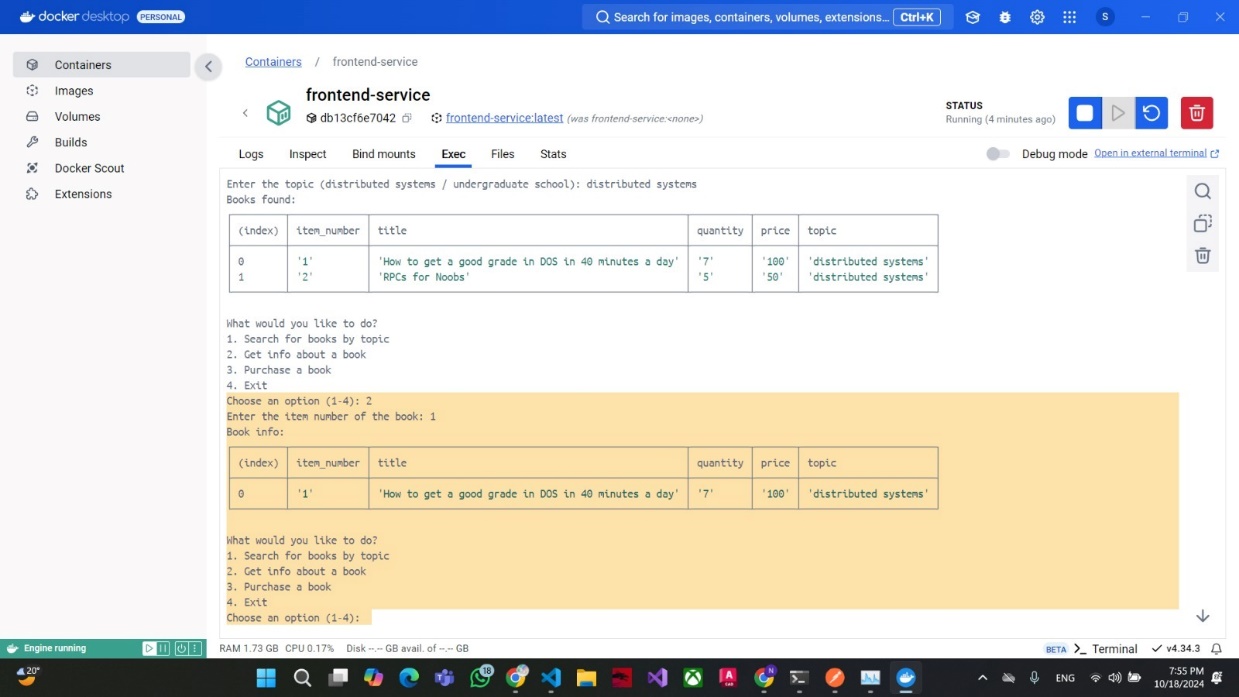
Now we will run the docker compose

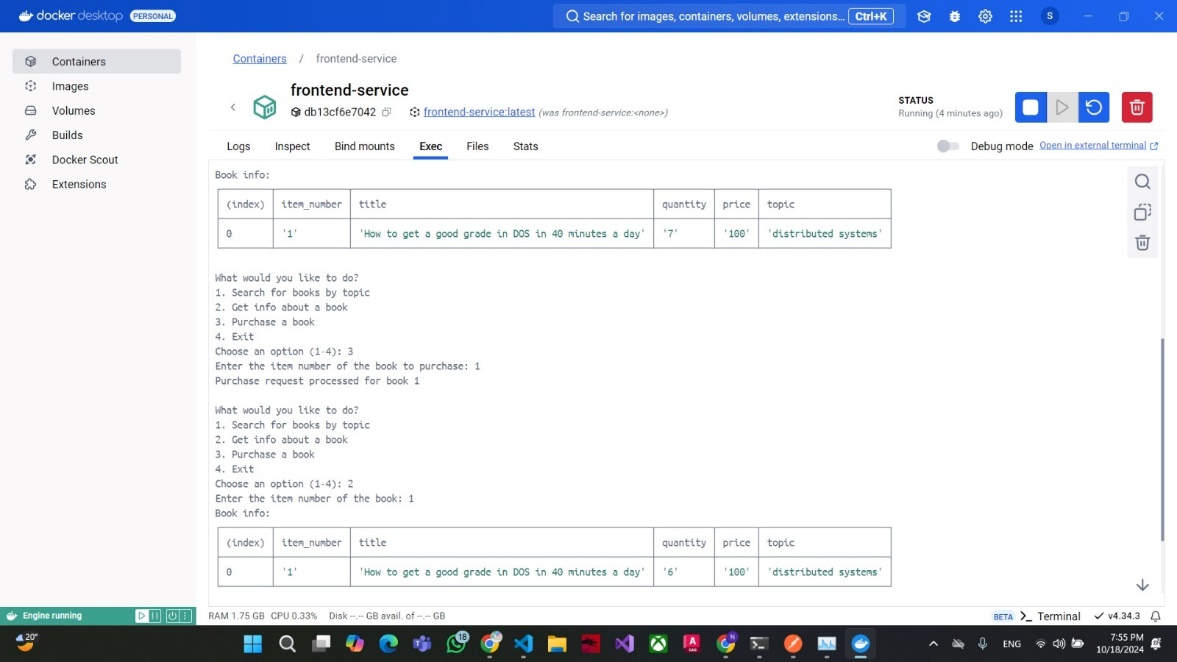
The result in the docker desktop is as shown below.

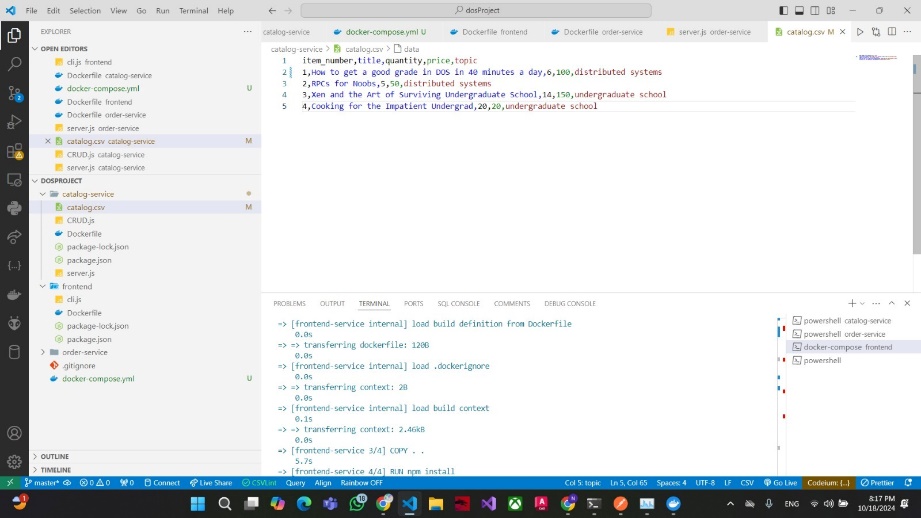


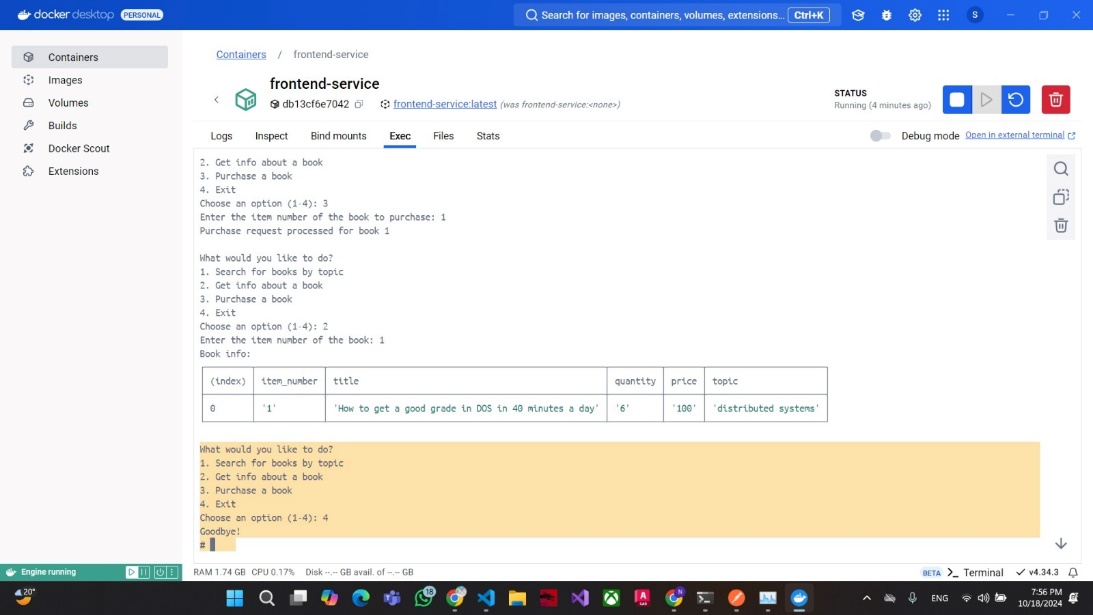
Now we can use the CLI as the input for our project and sends requests to other service containers as shown below.

Get an information about the Distributed system book

Get information of a book from it’s id

Purchase a book and the stock

And it decrements by one in catalog.csv

Finally, we exit the program

# Design Trade-offs

**Use of Text File (CSV) vs. Database**:

A CSV file was used for simplicity to store the catalog and orders. While this is efficient for small-scale data, it may become a bottleneck as the system scales. A lightweight database such as SQLite would offer better query performance and data integrity in a more complex system.

# Known Limitations

 **Concurrent Purchase Conflicts**:

* Multiple users purchasing the same book simultaneously may lead to overselling due to race conditions in stock verification.

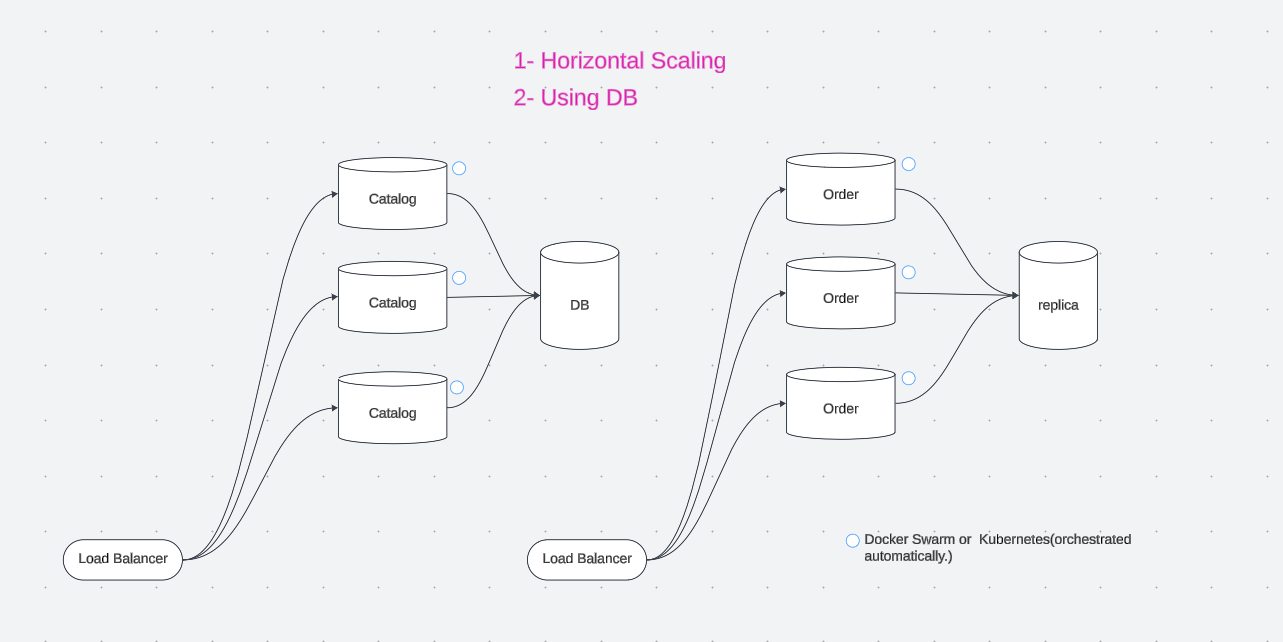
 **Data Persistence with CSV**:

* Concurrent writes to the CSV file can cause data corruption, leading to inconsistent inventory data.

**Single Point of Failure**:

* If one of the microservices (e.g., the catalog service or order service) goes down or is unreachable, the system will not function properly.

# Improvements and Extensions

* **Database Integration**:
  + Switching from a CSV file to a lightweight database such as SQLite would provide better scalability, faster queries, and improved data consistency.
* **Implement Docker Swarm or Kubernetes**:
  + By adopting Docker Swarm or Kubernetes for orchestration, the services can be scaled horizontally (i.e., adding more instances of each service) automatically. This is especially useful if you expect the catalog or order service to have high traffic at times.
* **Use a Database**:
  + Migrate from using a CSV file to a relational database.