**Distributed Operating System**

**Project Part 2**

**Raya Shafai 12010920**

**Lara Amer 12029590**

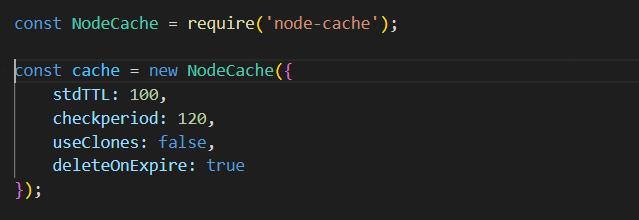
**Introduction:**

Bazar.com, an online bookstore, faced increased demand after a spring break sale, leading to delays and user complaints about latency due to insufficient infrastructure. To address this, the system was rearchitected with two main goals: improving performance through replication, caching, and load balancing to distribute requests and reduce data fetch times, and simplifying deployment via Dockerization. The solution involved replicating catalog and order servers, implementing in-memory caching for faster access, and containerizing components with Docker for portability and easier management.

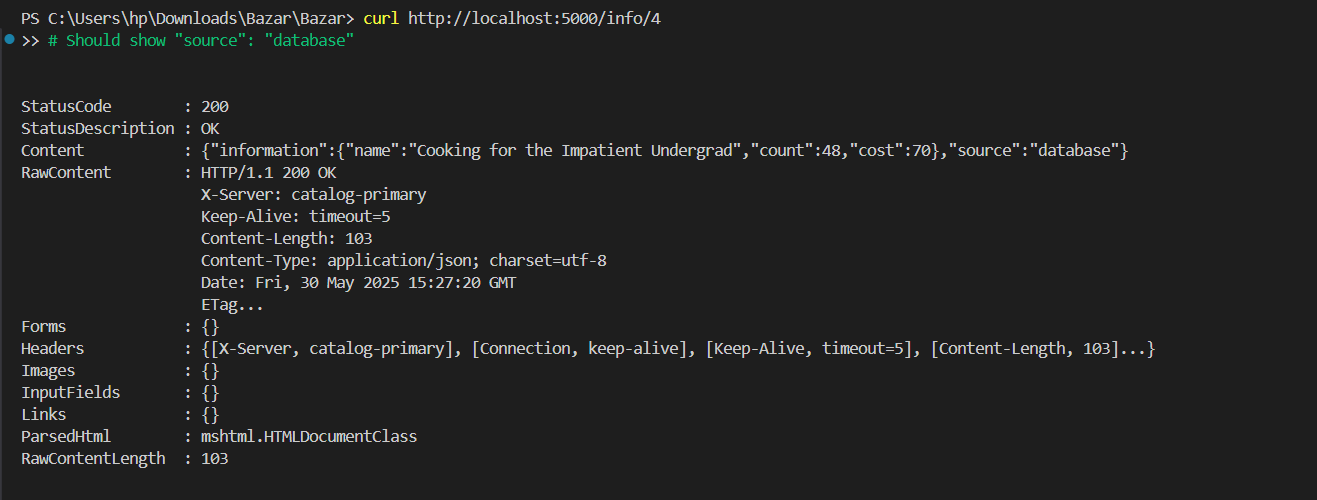
**Replication and Caching:**

In-Memory Cache:

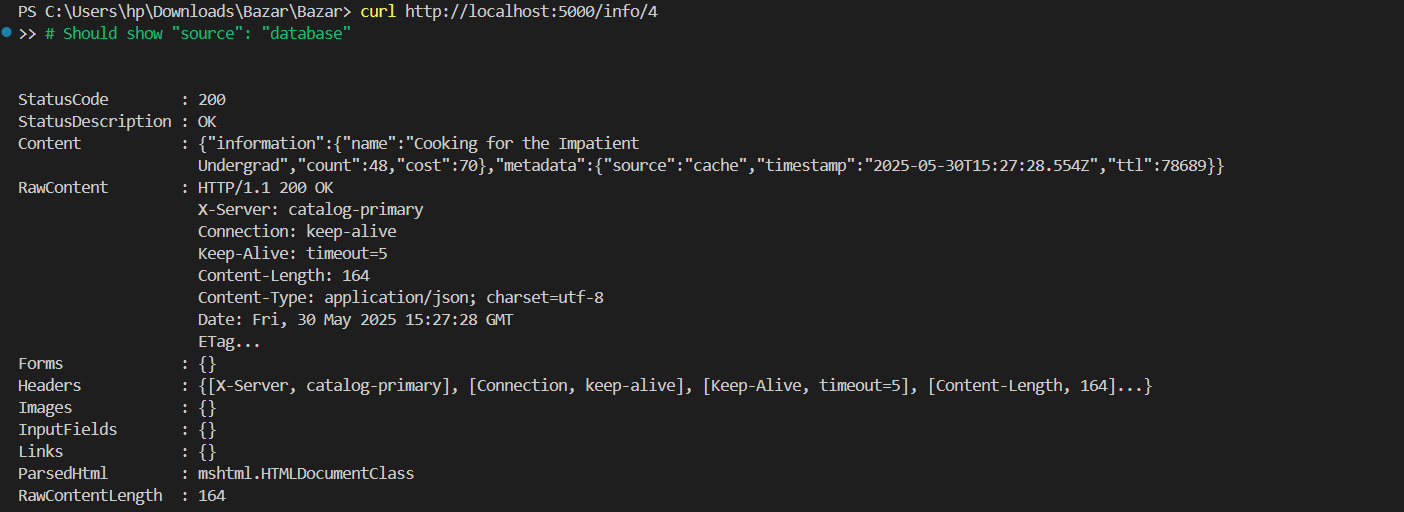
We used node cache for data caching



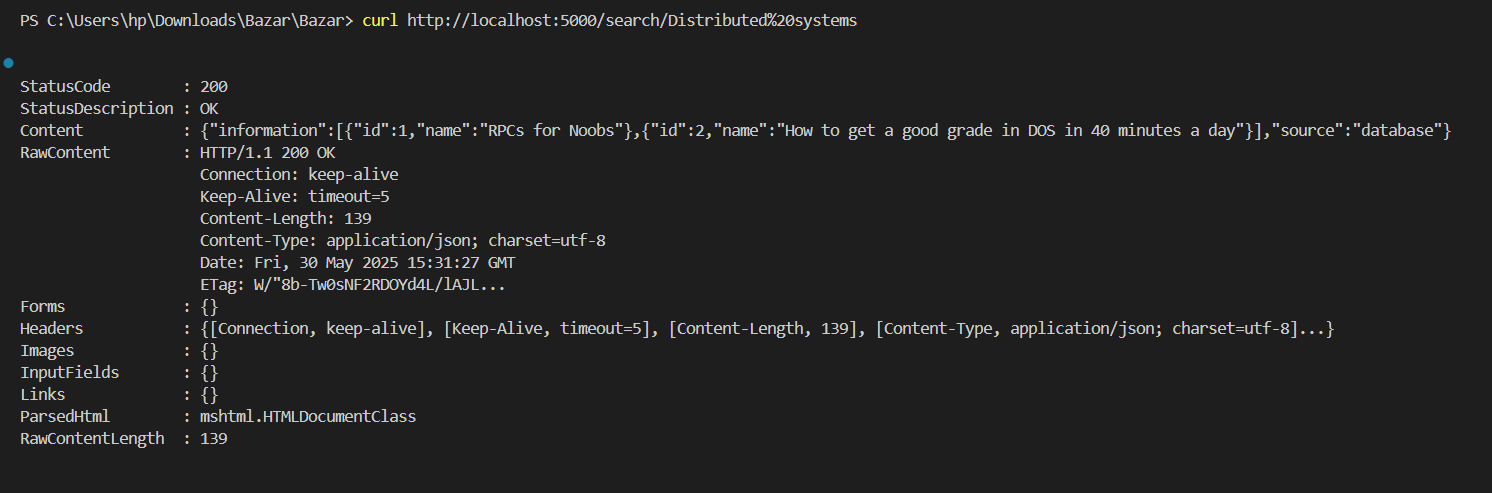
The request time without caching for info (database):



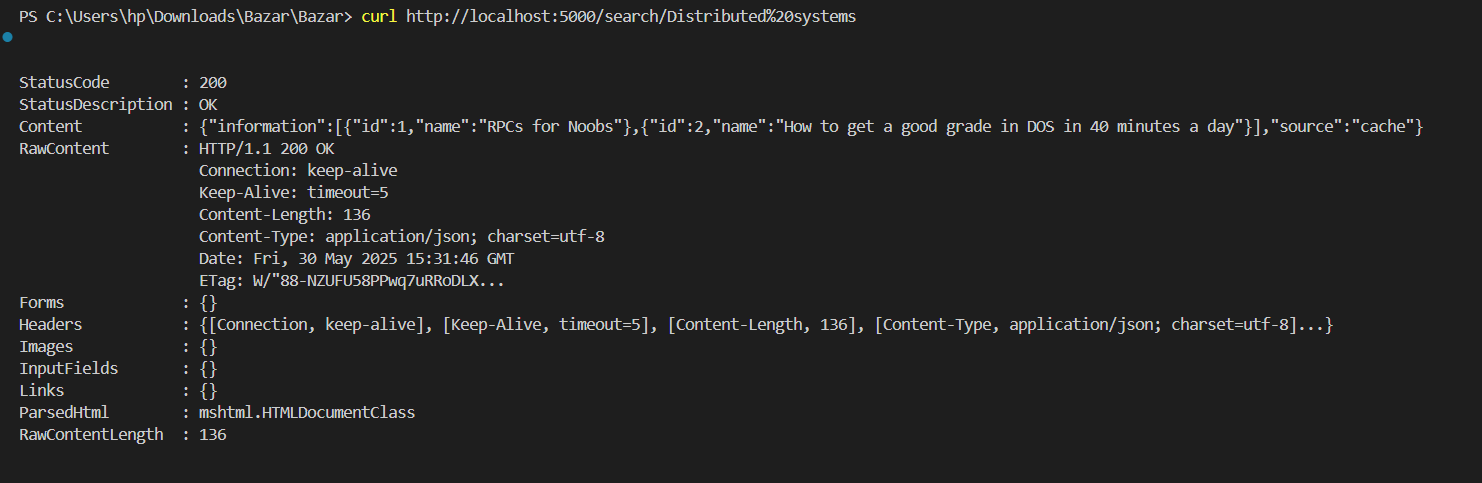
The request time with caching for info:



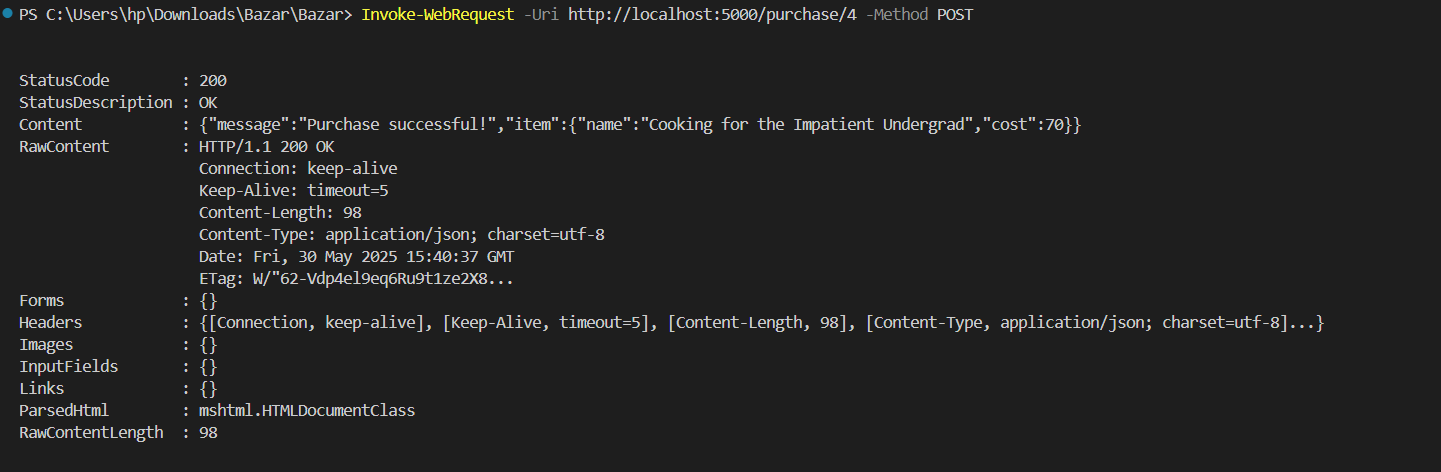
The request time without caching for search (database):



The request time with caching for search:

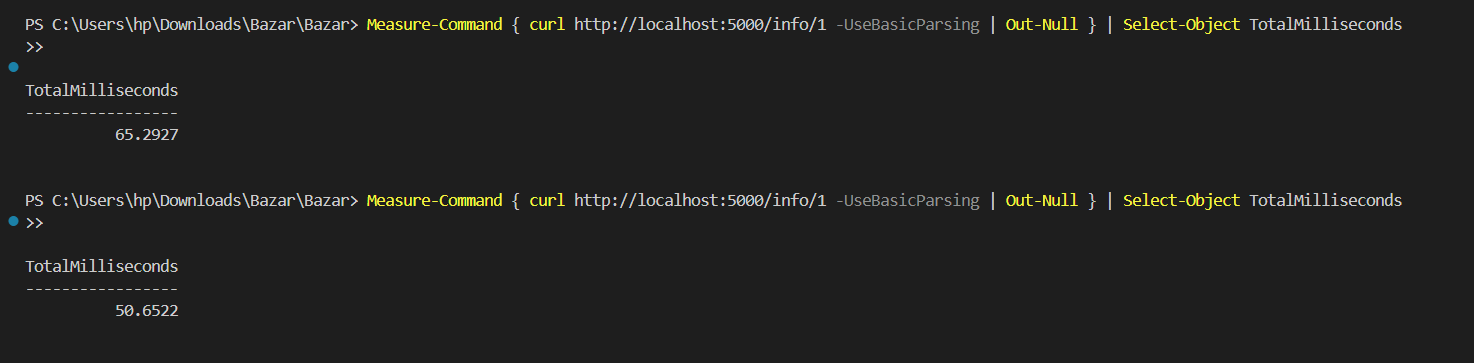


Purchase a book:

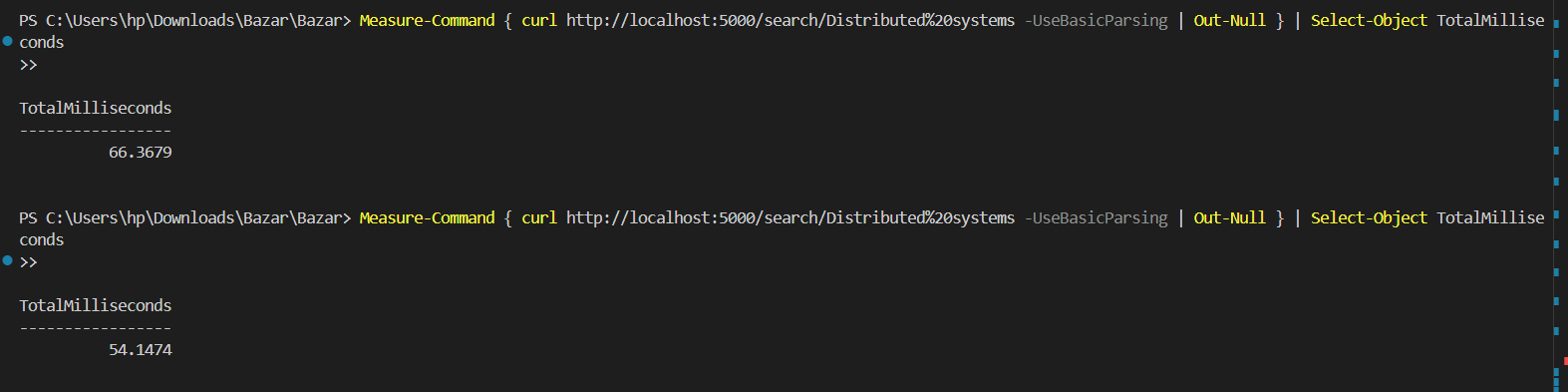


Compute the average response time (query/buy) of your new systems:

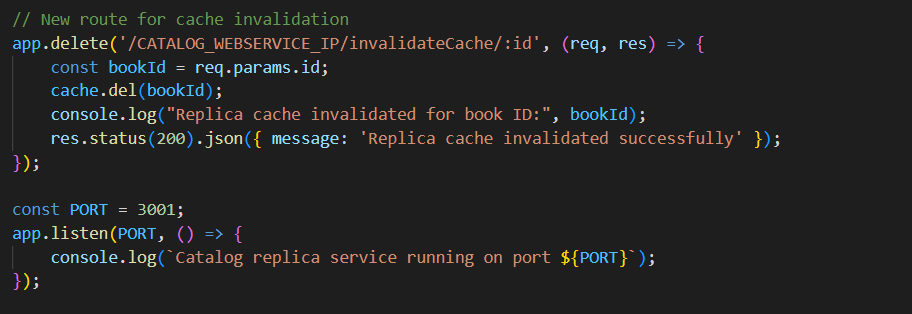
The response time with and without caching for info:

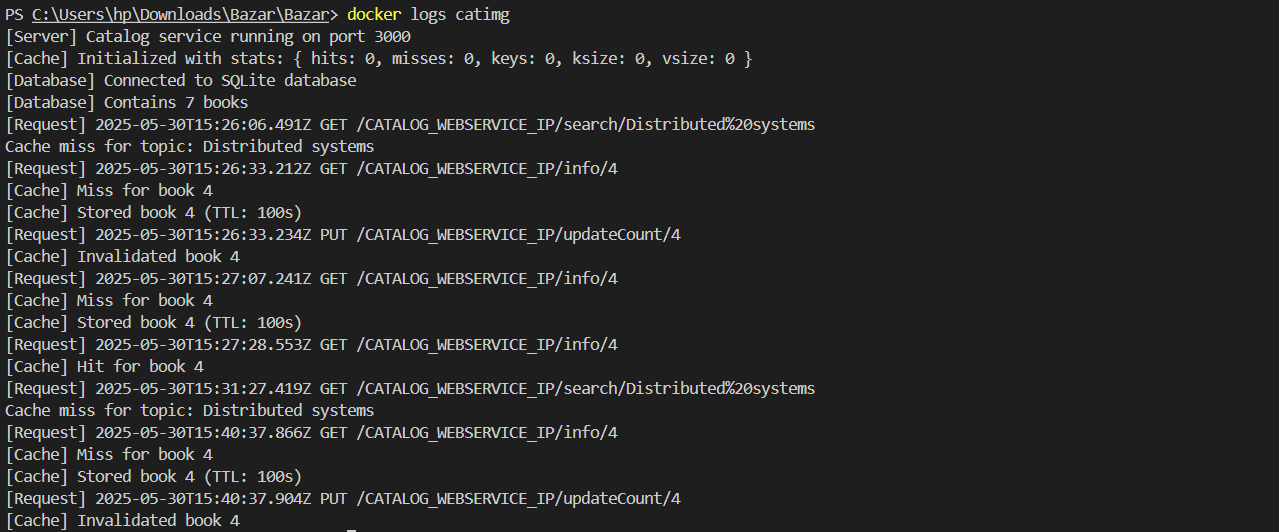


The response time with and without caching for search:



|  |  |  |  |
| --- | --- | --- | --- |
|  | Without cache | With cache | #times cache speed |
| Get/info | 65.2ms | 50.6 | 1.28 faster |
| Get/search | 66.3 | 54.1 | 1.22 faster |

Cache Invalidation and Consistency:   




Load balancing (round-robin):

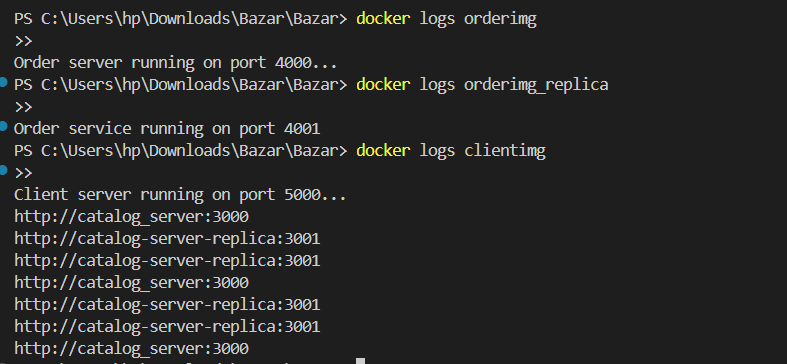
3000: catalog server

3001: catalog-replica server

4000: order server

4000: order-replica server

5000: client server





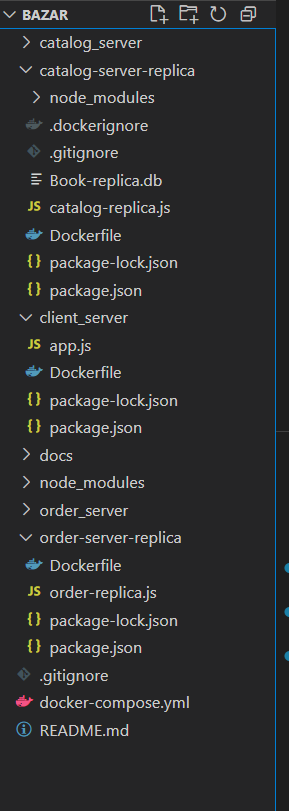
**Dockerization** :

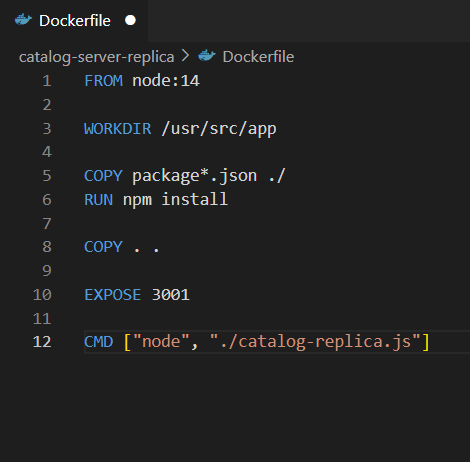
In the second part of the project, the goal was to Dockerize the application to simplify deployment and ensure that the system components could be packaged, shared, and run in isolated containers. The following steps were taken:

Dockerizing Components Each of the major components (front-end server, catalog server and its replica, order server and its replica) was containerized using Docker:

Each service was packaged into its own Docker container , which includes all necessary dependencies and configurations for running the service. The front-end server, catalog server and its replica, order server and its replica were each defined in separate Dockerfiles, specifying the environment, dependencies, and commands to run each component.

The structure of catalo and order replica:



catalog-replica dockerfile(port 3001):  


Docker Compose:

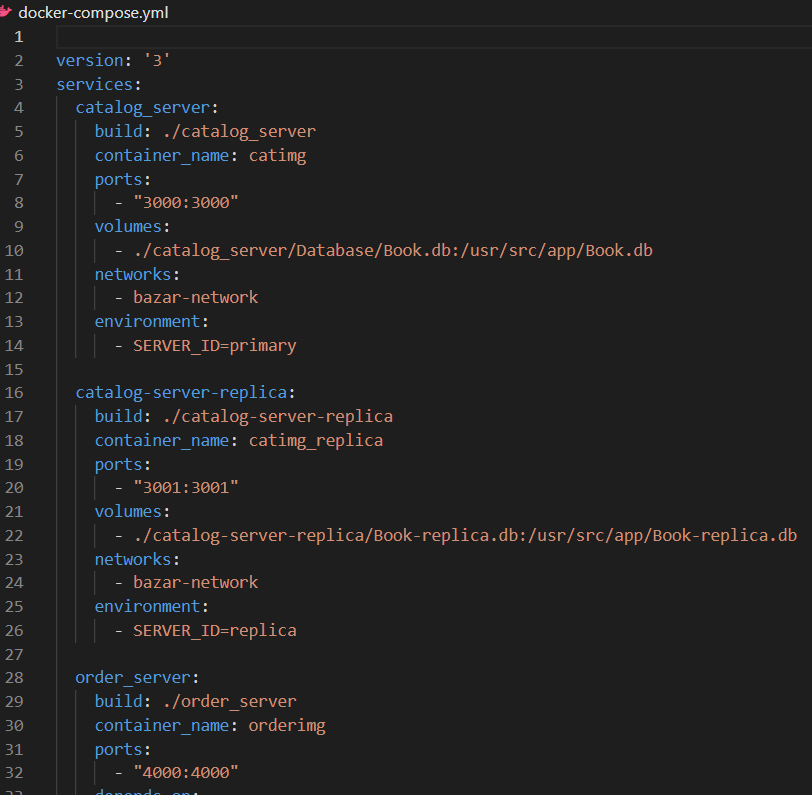
• To manage the multi-container setup, Docker Compose was used to define the

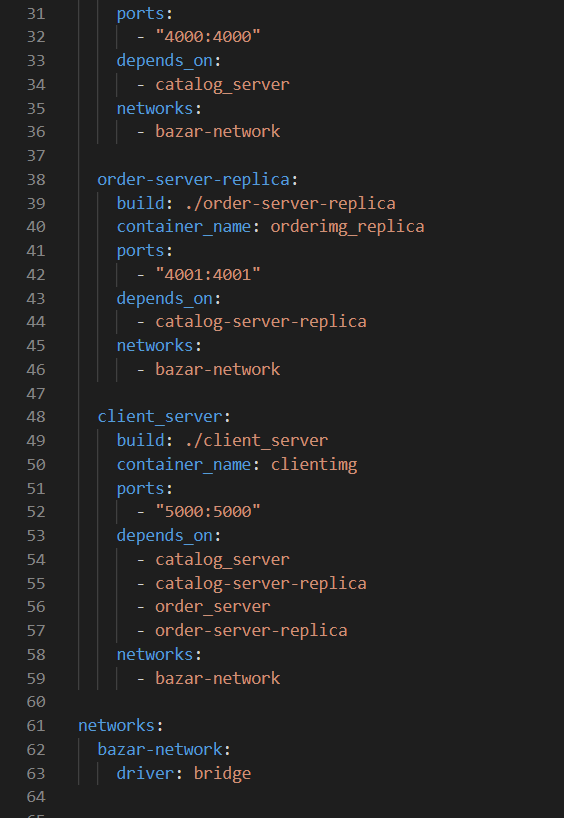
entire stack in a single configuration file (docker-compose.yml).

• This file specifies how each service (front-end, catalog, order, cache) is built,

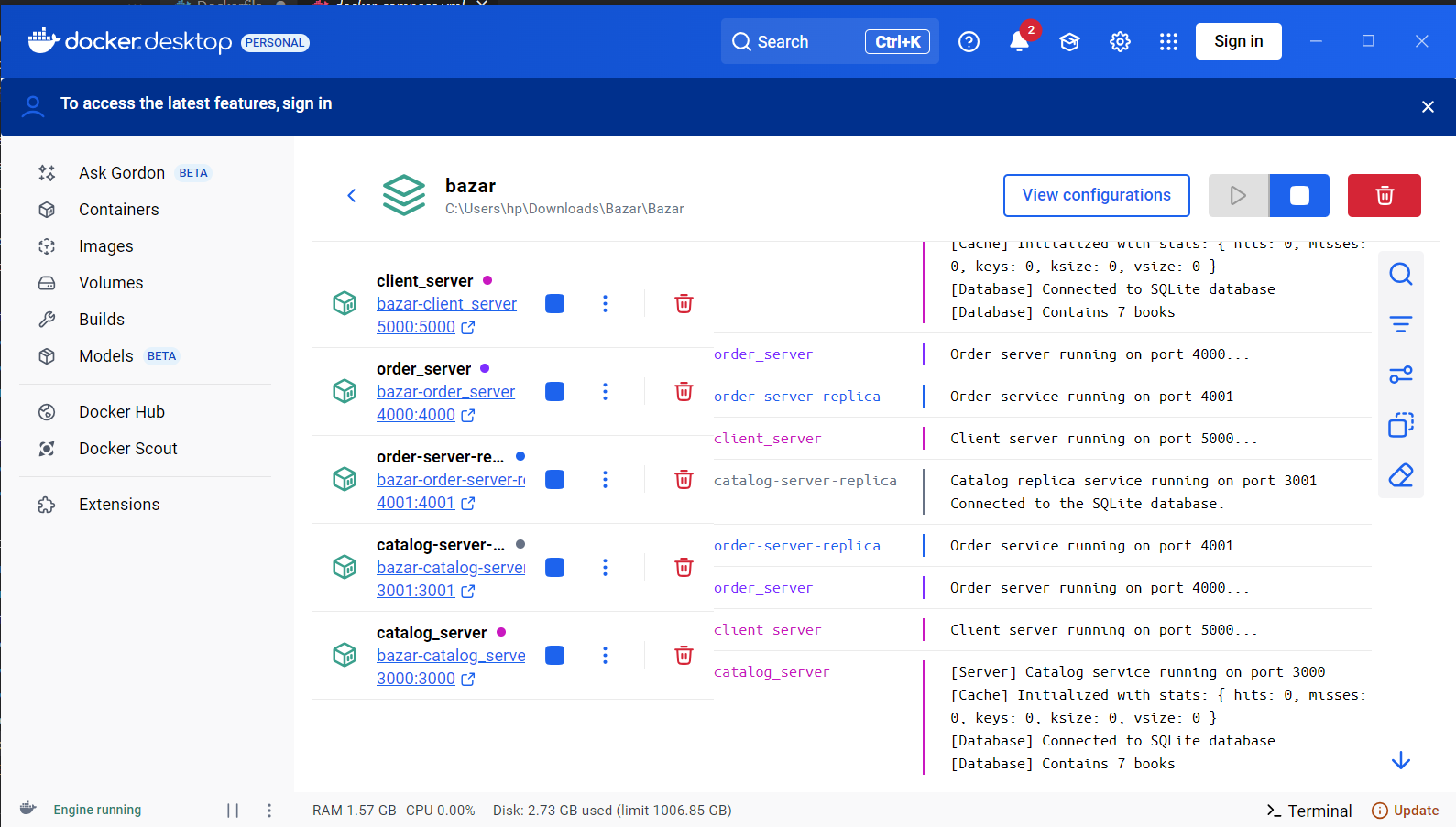
how they communicate with each other, and how ports are exposed for external

access.





And here our containers in the Docker Desktop:



**Conclusion:**  
This project successfully implemented an in-memory caching mechanism using NodeCache to improve the performance of the Bazar.com distributed bookstore system. The caching strategy was applied to both the primary catalog server and its replica, ensuring faster response times for frequently accessed book data while maintaining consistency through proper cache invalidation.