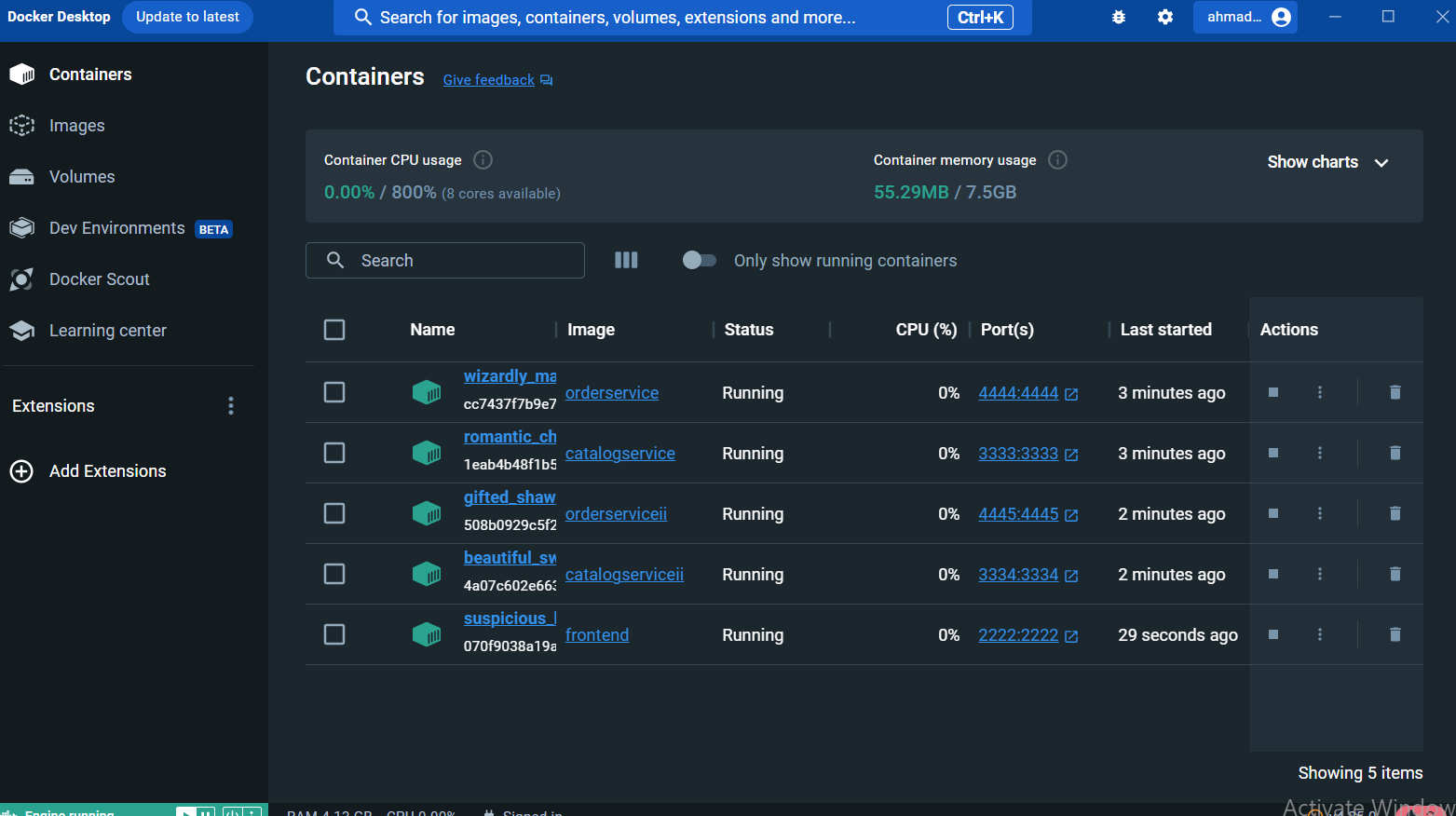
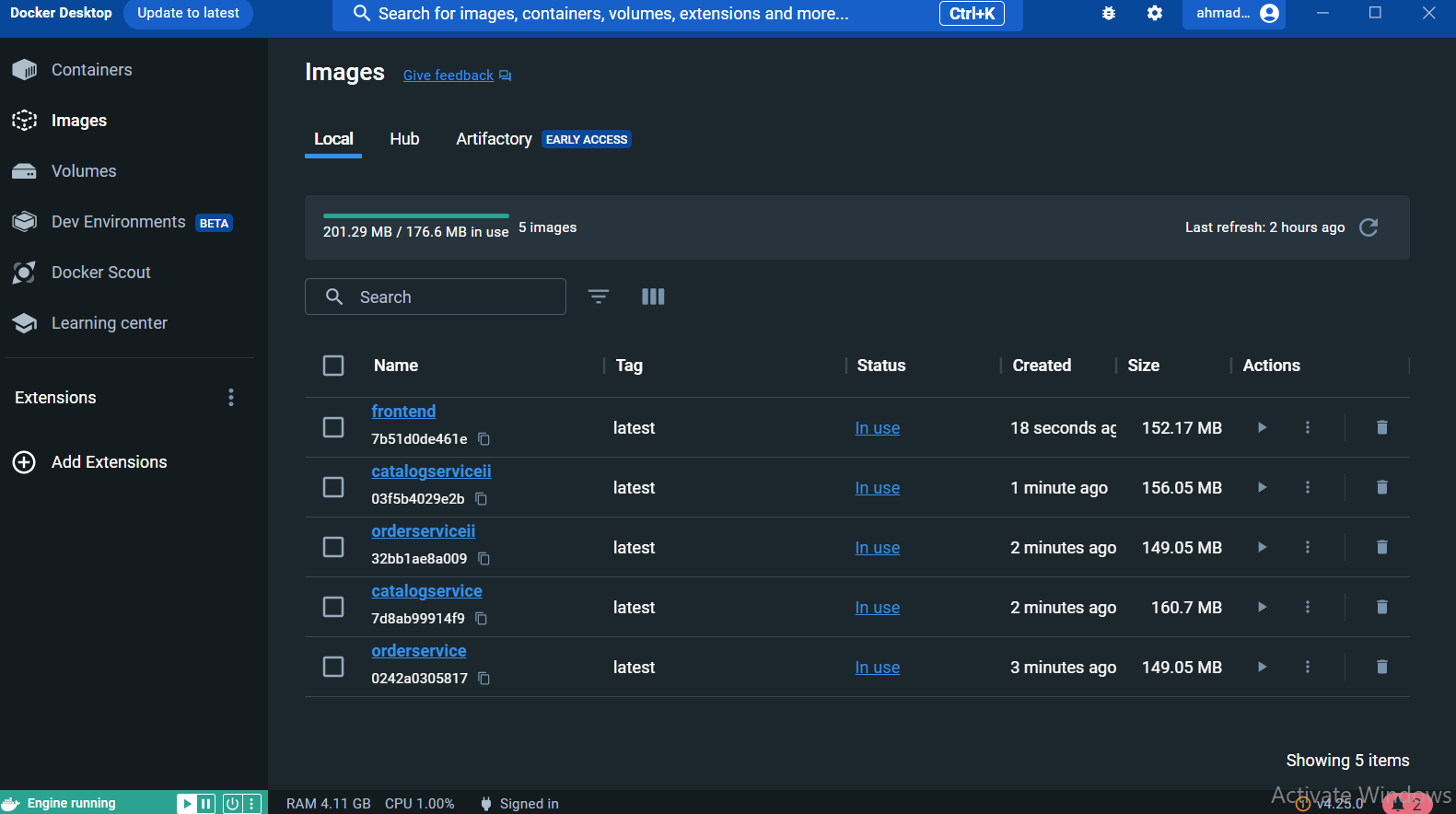
Ahmad Atout

Mohammad Jury

The project was implemented using nodeJS-expressjs as the backend service framework

And then every service was deployed on a nodejs docker container

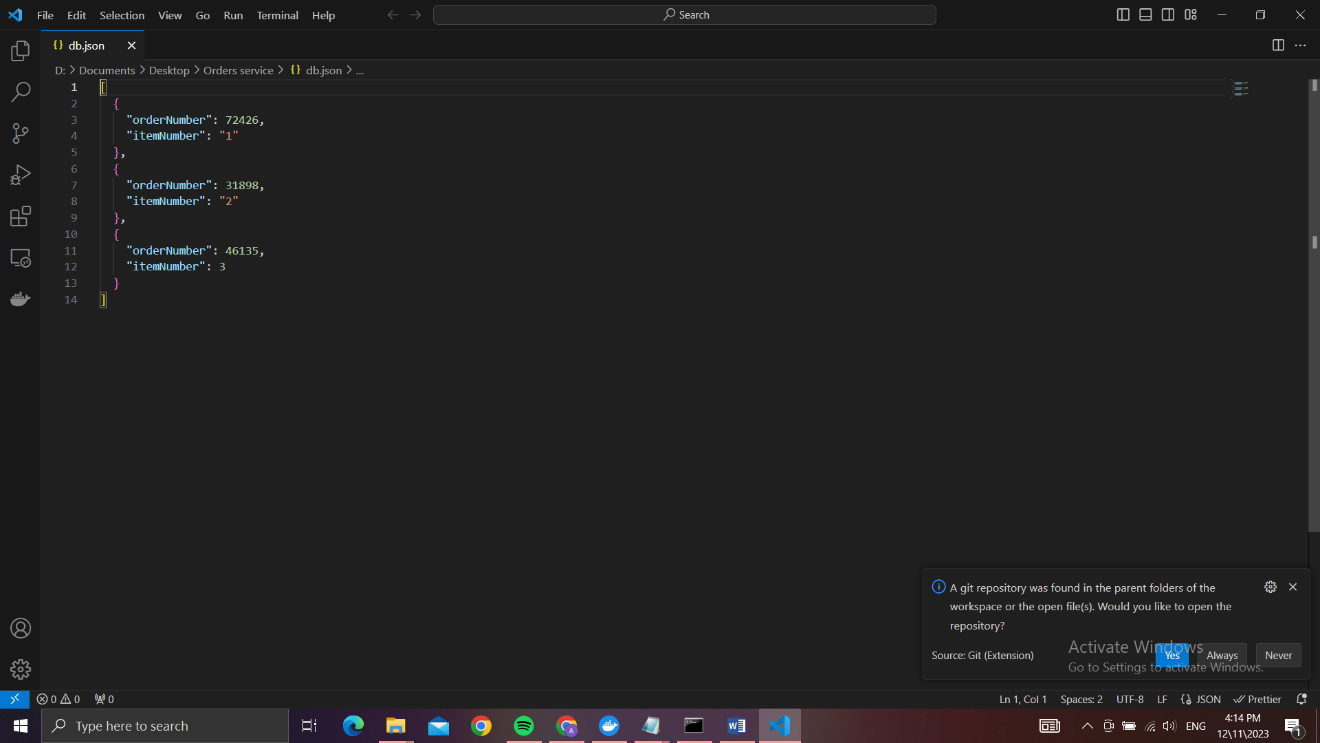
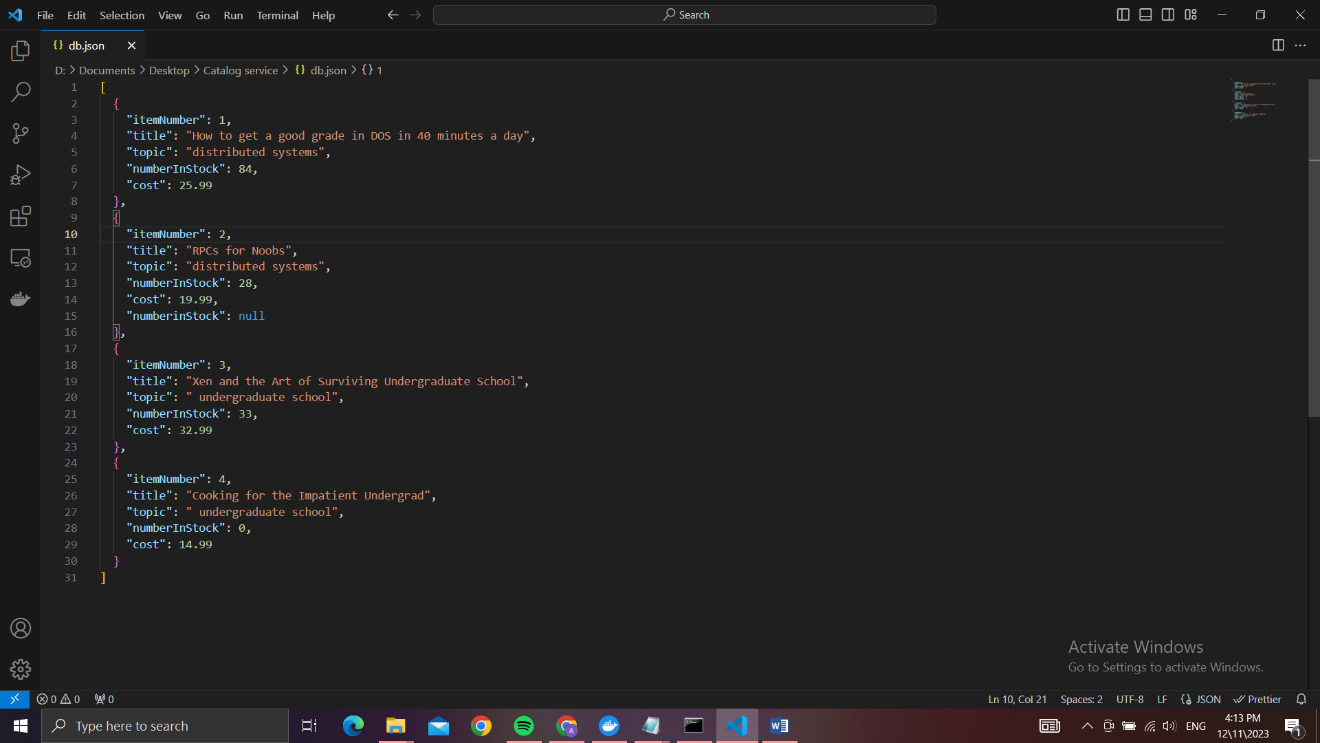




We Used .json files as databases, we had 2 databases each one inside a service

Database for catalog

Database for orders



The catalog service runs on port 3333/ replica runs on 3334

The orders service runs on port 4444/ replica runs on 4445

The front-end service runs on port 2222

We created Dockerfile in each service to build an image and start a container running that service

FROM node:alpine

WORKDIR /usr/src/app

COPY package\*.json index.js db.json ./

RUN npm install

EXPOSE 4444

CMD ["node", "index.js"]

We used node:alpine because its much lighter version than other node docker images

When an instance is created it runs npm install to install all the required dependencies for the service to run

All the dependencies are mentioned in the package.json file

The requests are directed to the frontend service which then redirects all the requests to the desired service to handle

When the orders service receives a request to purchase it sends a request to the catalog service to query info about the number of the desired book available in stock

If the book is available it sends another request to the catalog server to decrement the number available in stock by 1

Cache and replicas added:

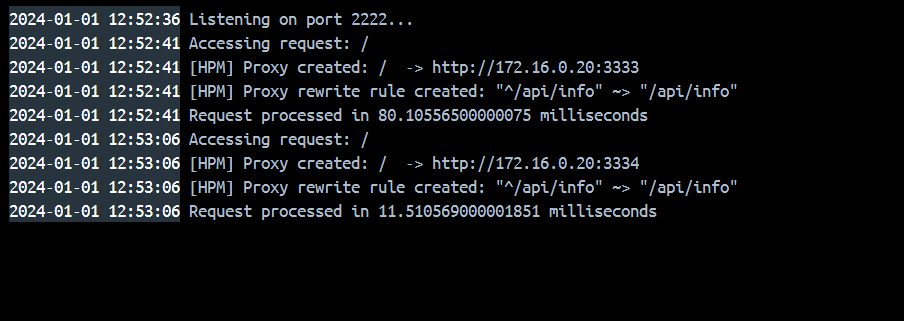
We added cache to the front end service so that it saves a record of each read query on the catalog service and the replica of the catalog service

The cache is implemented as a middleware so that it handles the request before proxying it to the catalog services

We used round robin algo for load balancing the requests as u can see in the bellow picture, each request was send to a different service

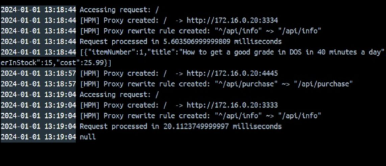
When any write operation is made to the services, each service sends the change happened to the replica so that they are consistent

Then it sends the change happened to the front end service to inform it that the resource has been changed and delete the cached version it has of it



As we can see here the first request didn’t find the result cached so it had to proxy to the catalog service and fetch it from the db and it took almost 80 ms

The second query on the same resource found the result in cache so it took only 11ms



1. We performed api/info/1 request which was already in cache
2. We performed api/purchase/1 so that it cuase the cache in front end service to be deleted
3. We performed api/info/1 again and it took 30 ms bacuese the cache was deleted

How to run our program?

Install each service folder on your computer then run “npm install’ in each folder to install all the required dependencies.

The ip of our local host is hardcoded so change it in each index.js file if needed to your local machine ip address

Run each seavice and test out the project

If you want to run it on different virtual machines

After doing the steps above:-

Bulid an image of each service using the Dockerfile available in each folder of the 3 folders

Then run the images and expose each container the the correct ports

Front-end 2222

Catalog 3333// replica on 3334

Orders 4444// replica on 4445