Giza Systems

Containerized CRUD Application with Automation Scripts

By: Salma Salah

Project’s Description

This project is a containerized CRUD (Create, Read, Update, Delete) application that includes a frontend and backend, which interacts with a MongoDB database. The application is built using Docker, Docker-Compose, and is deployed on an AWS EC2 instance running Ubuntu. Also, a daily backup of the MongoDB database is automated using a cron job. Use Monitoring tools to tack the application’s performance.

Part 01

Deployment

Steps

I. Creating CRUD App

II. Confirming that application is working locally

III. Dockerize the application in three containers; the first for the application front end , the second for API and third for the database

IV. Running the application using Docker-Compose

V. Pushing the Application into git repo

VI. Clon application into Main Server on AWS

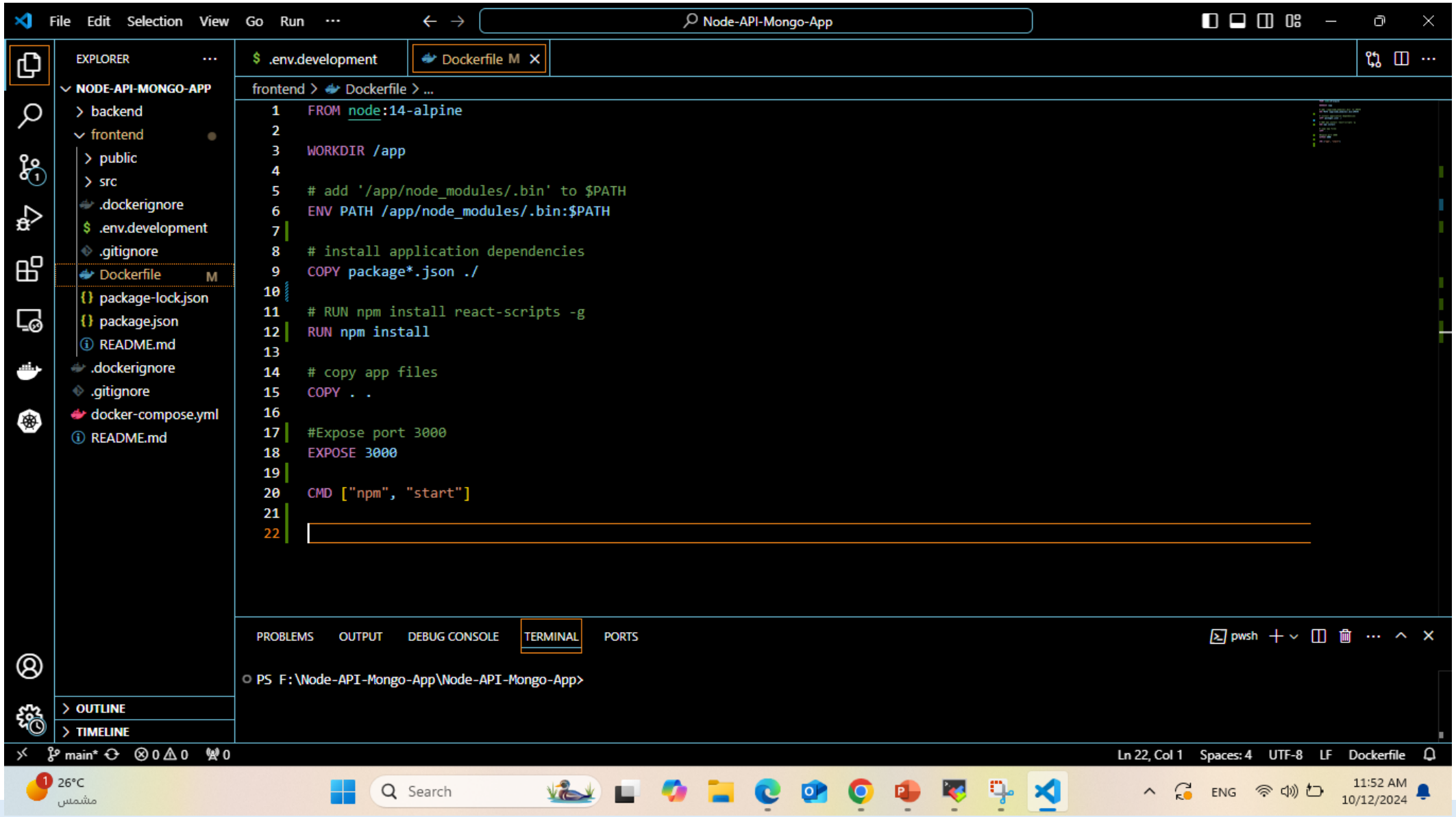
VII. Write scripts to automate building, running, and stopping the Docker containers

VIII. Write scripts to transfer code to target server and build/run the Docker image on the VM

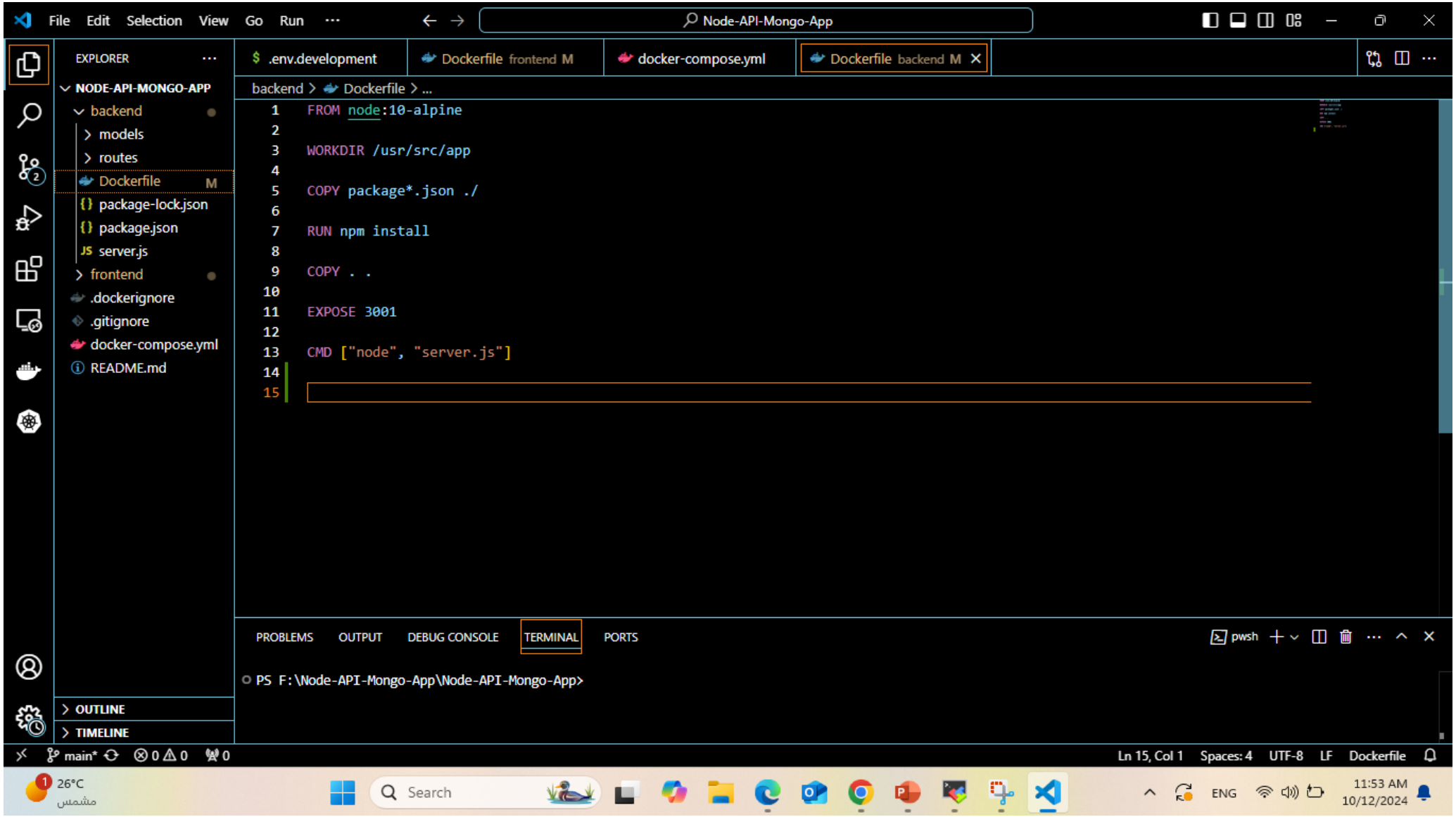
IX. Write scripts to ensure successful deployment.

X. Implement an Automated Daily Task by Automatically back up the application's data once per day.

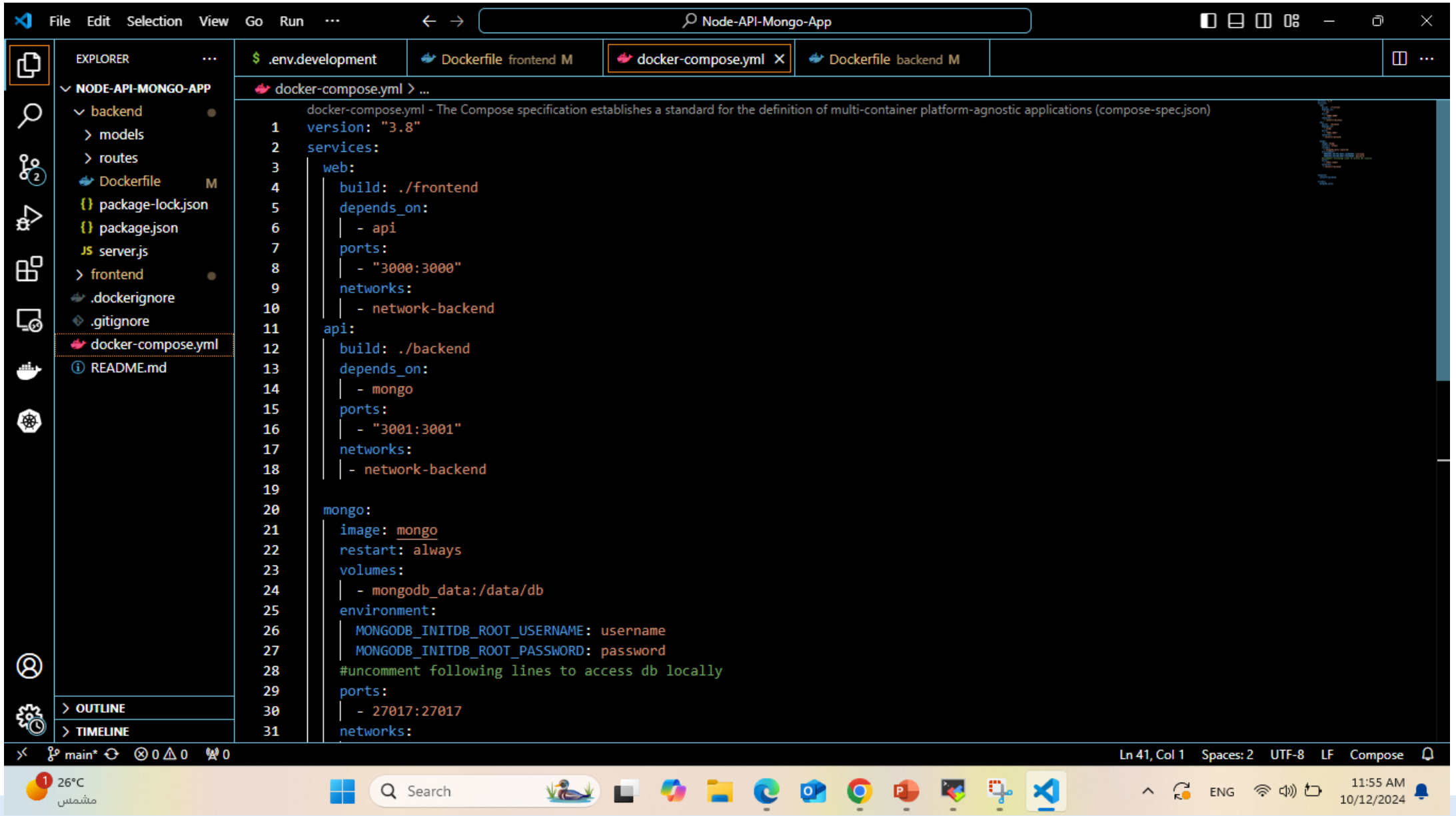
1- Docker file for frontend



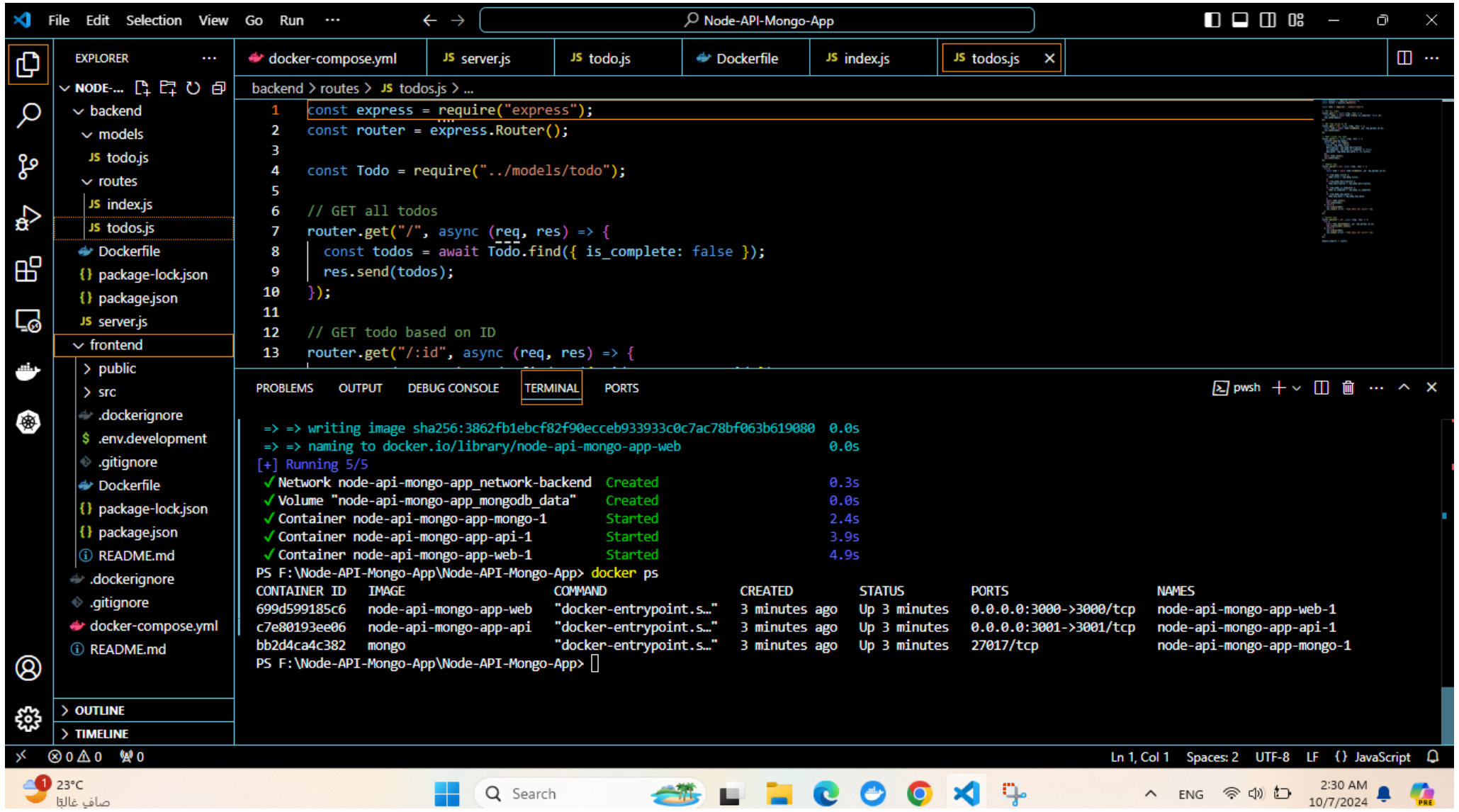
2- Docker file for Backend



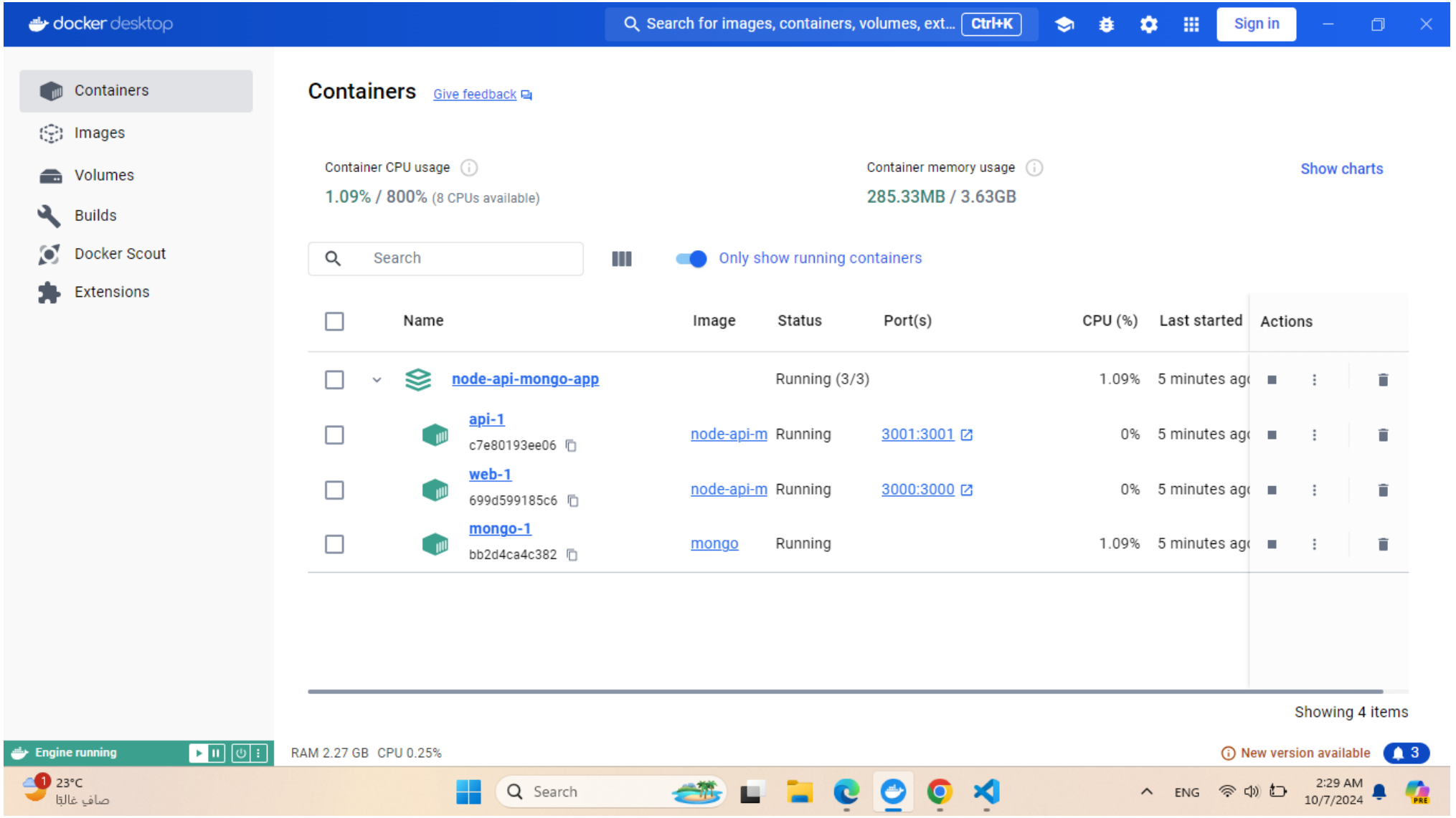
3- Docker-Compose file



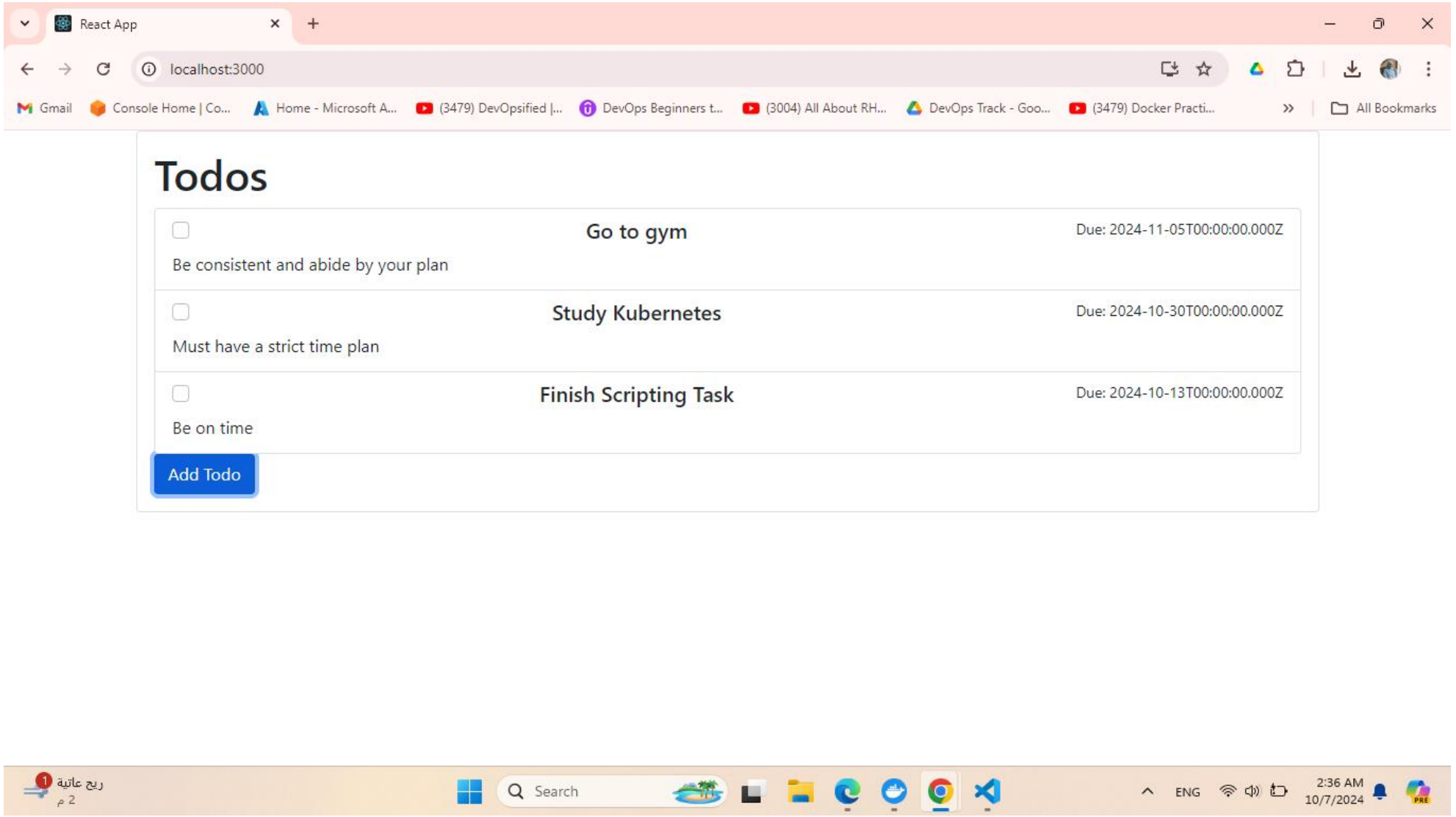
4- Running app



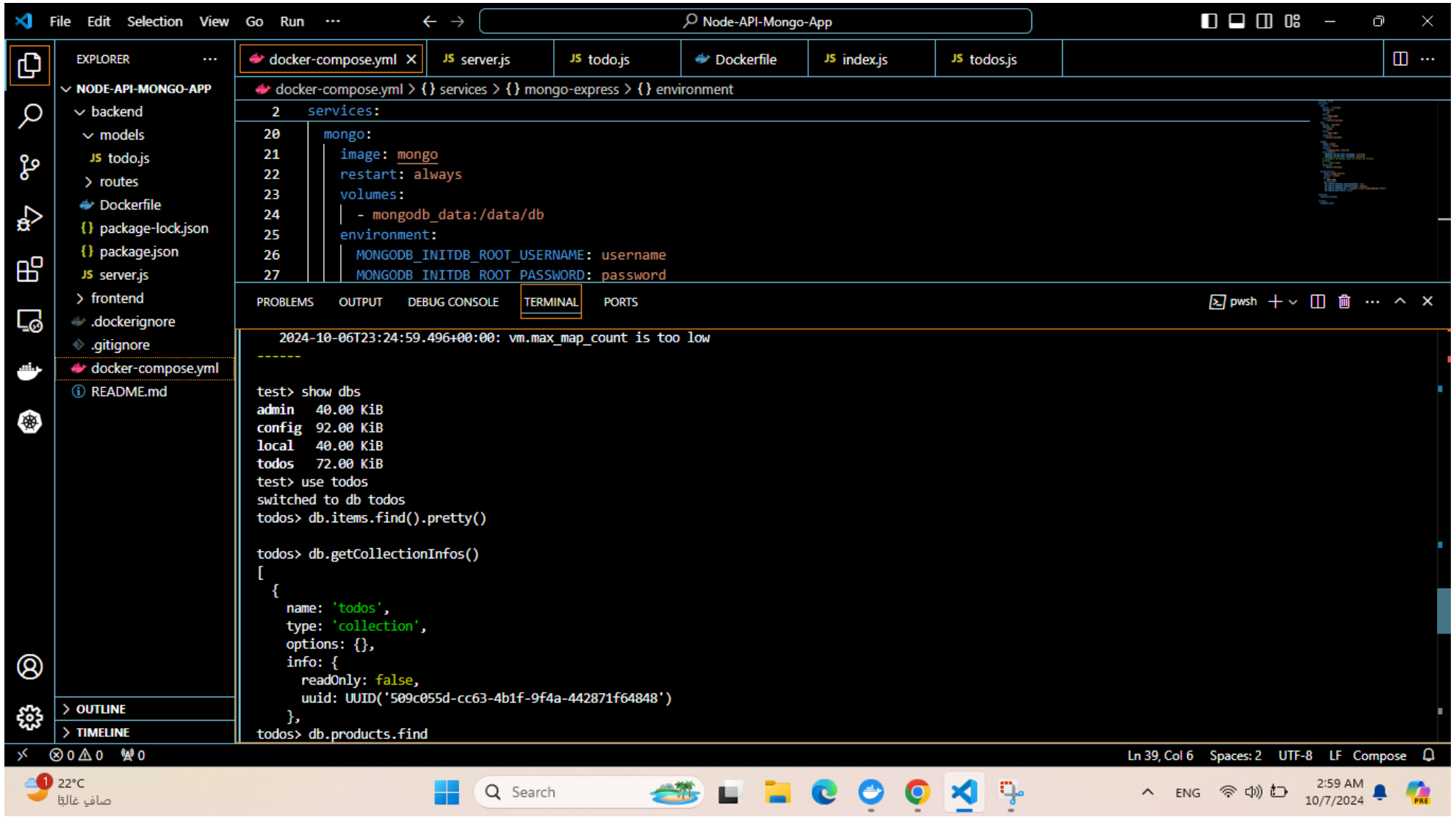
5- Application after running the three containers



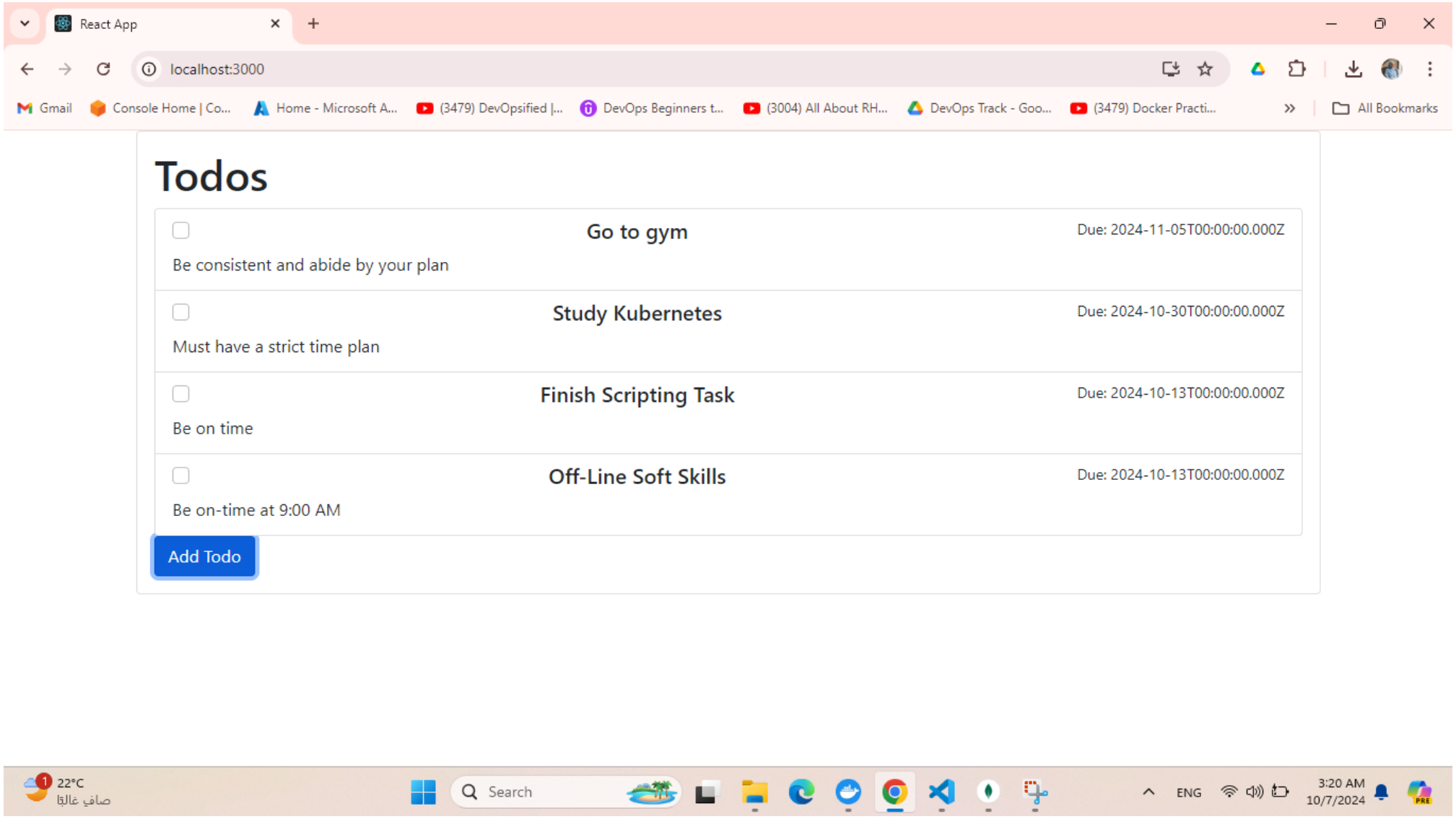
6- Frontend Api & DB working



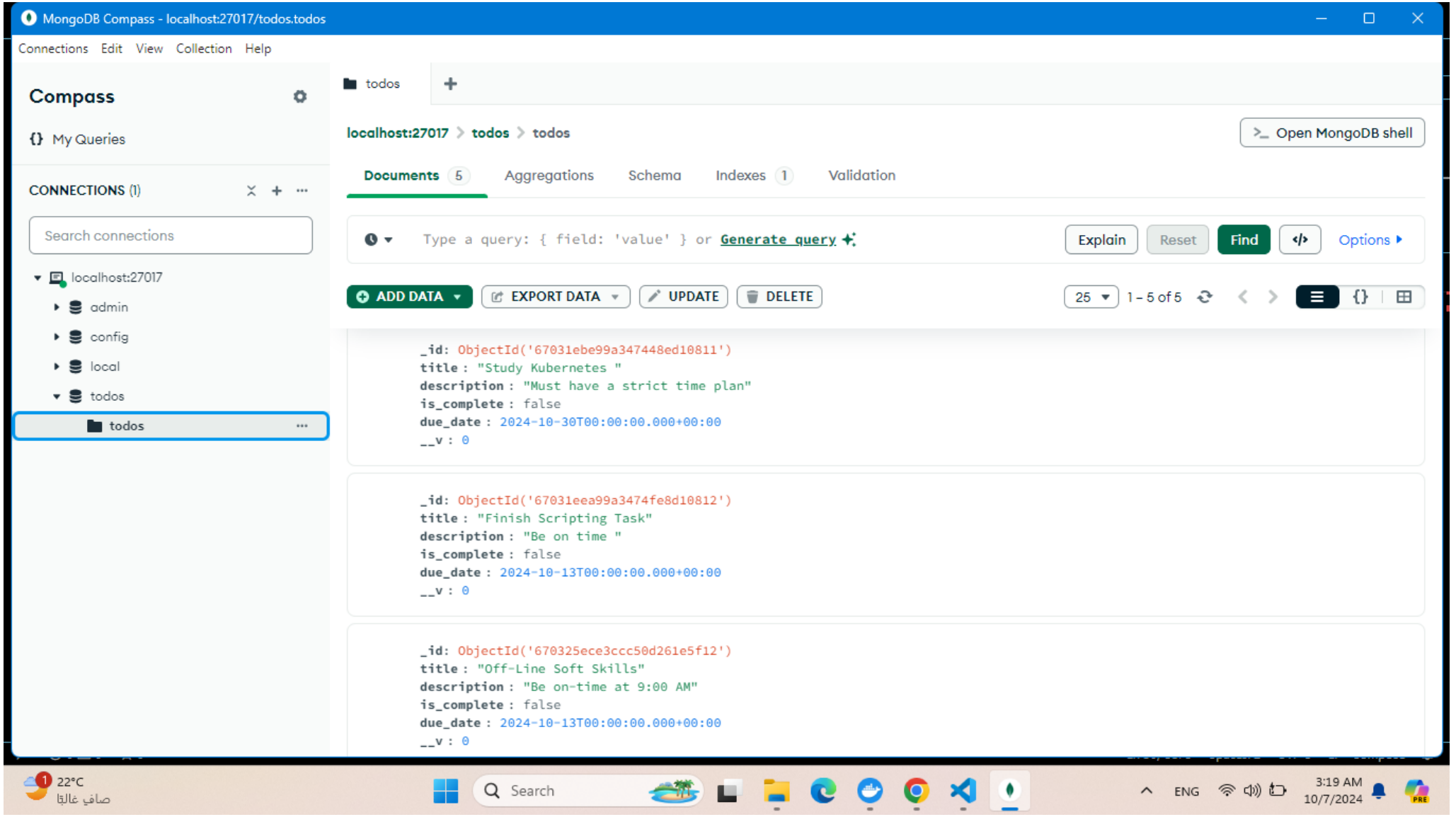
7- Logging into db (todos)



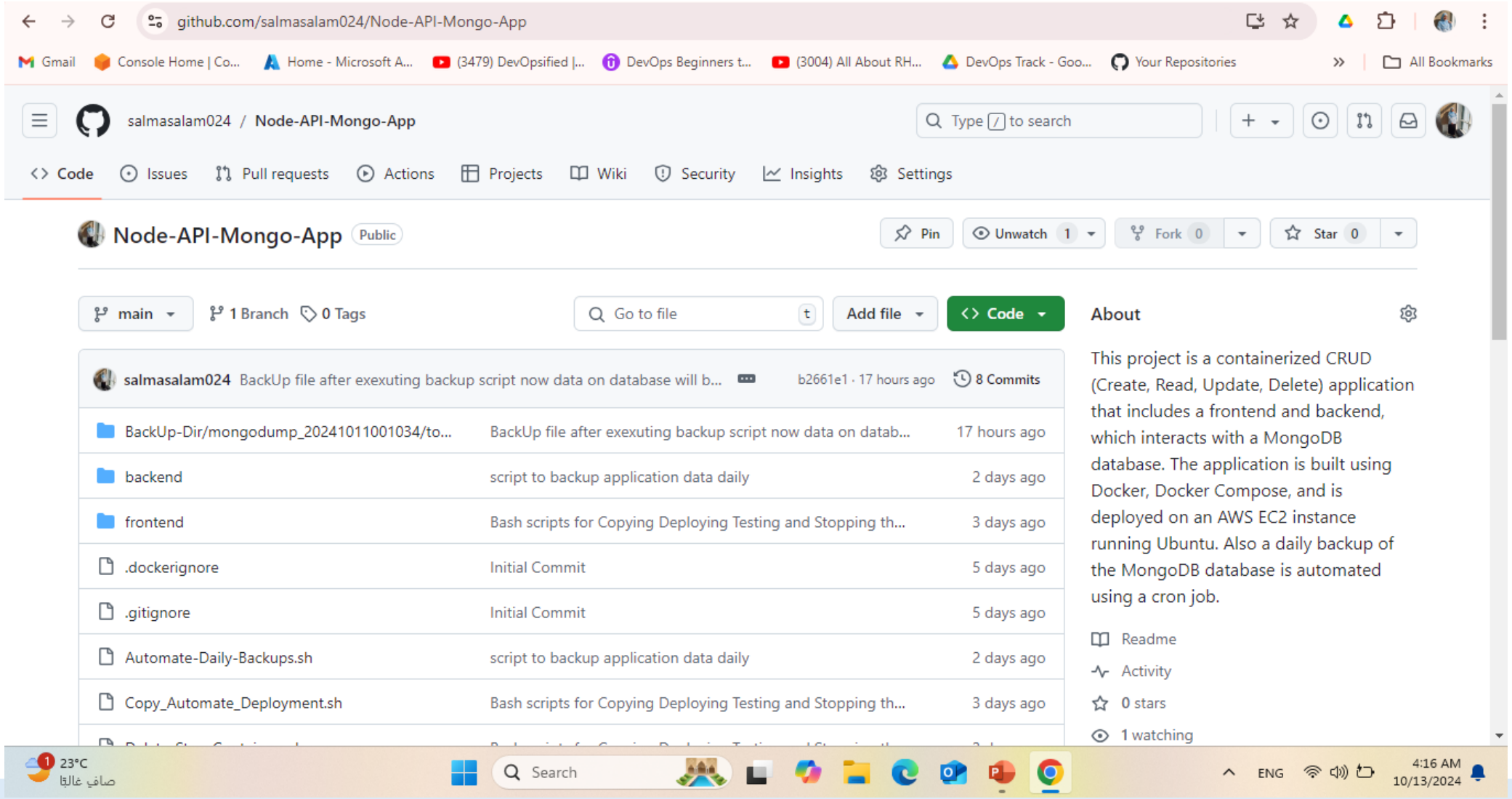
8- Adding more entries to our Todos



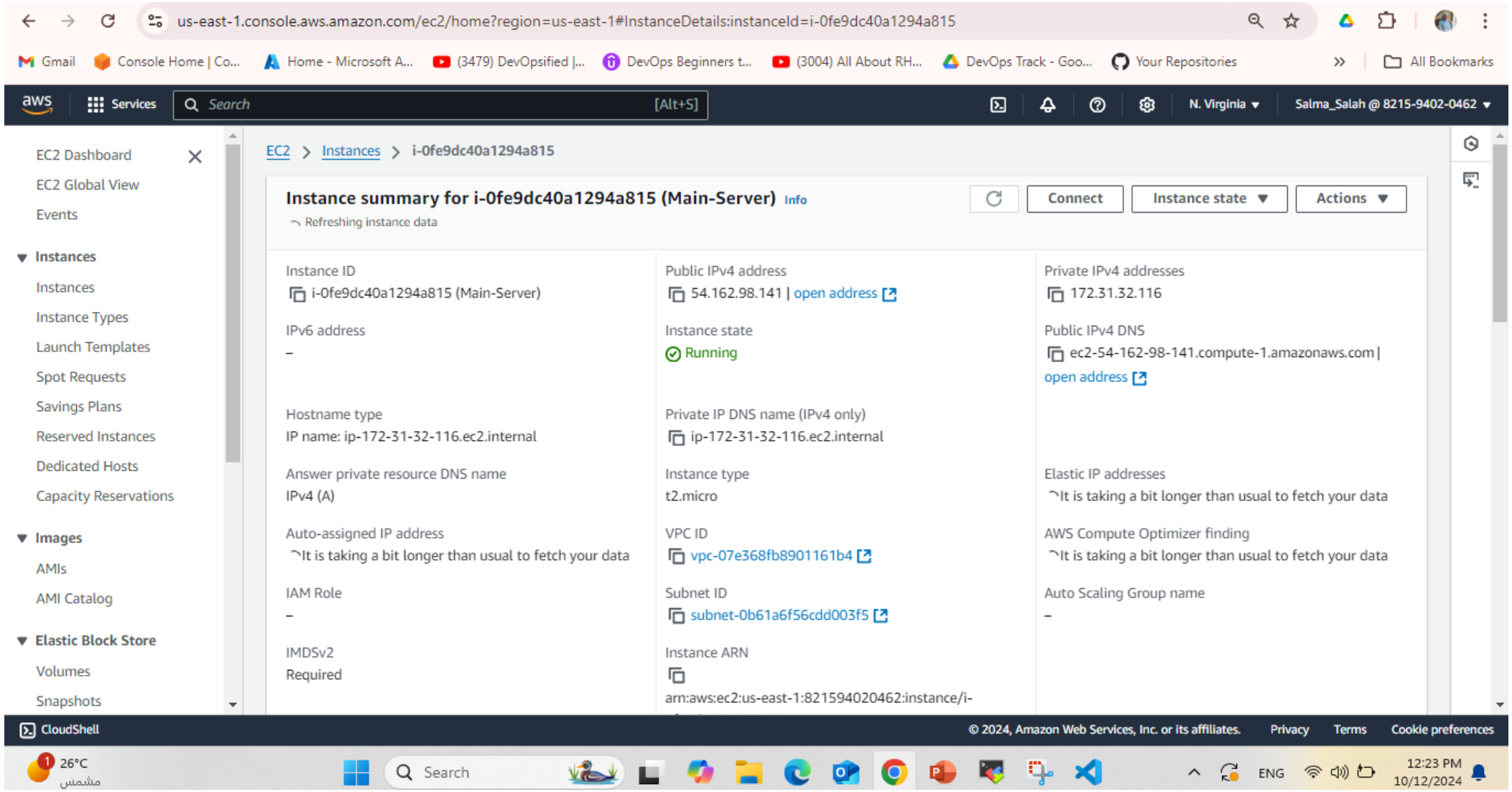
9- Connecting to mongodb using MongoDB Compass (gui)



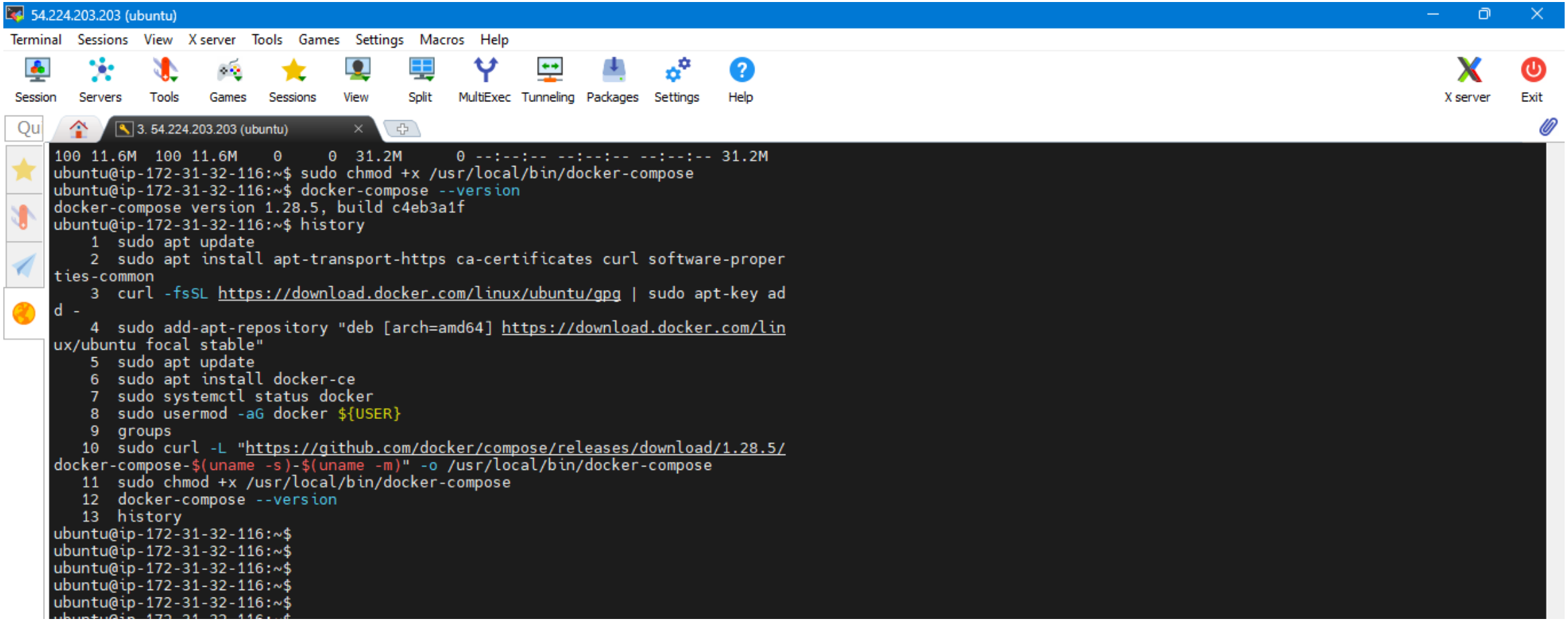
10- Create local and remote git repositories and push all files to remote repo



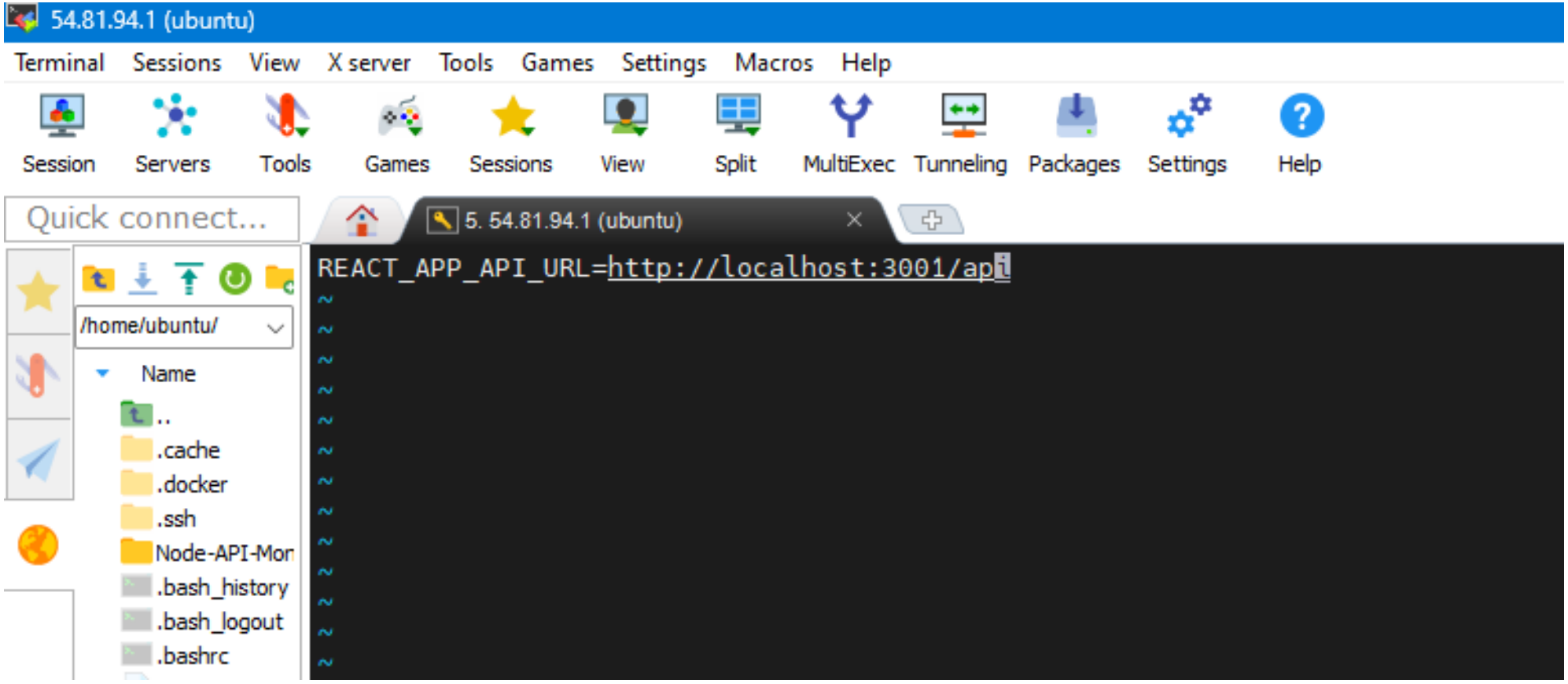
11- Creating Main Server on AWS (Ubuntu) and open required ports



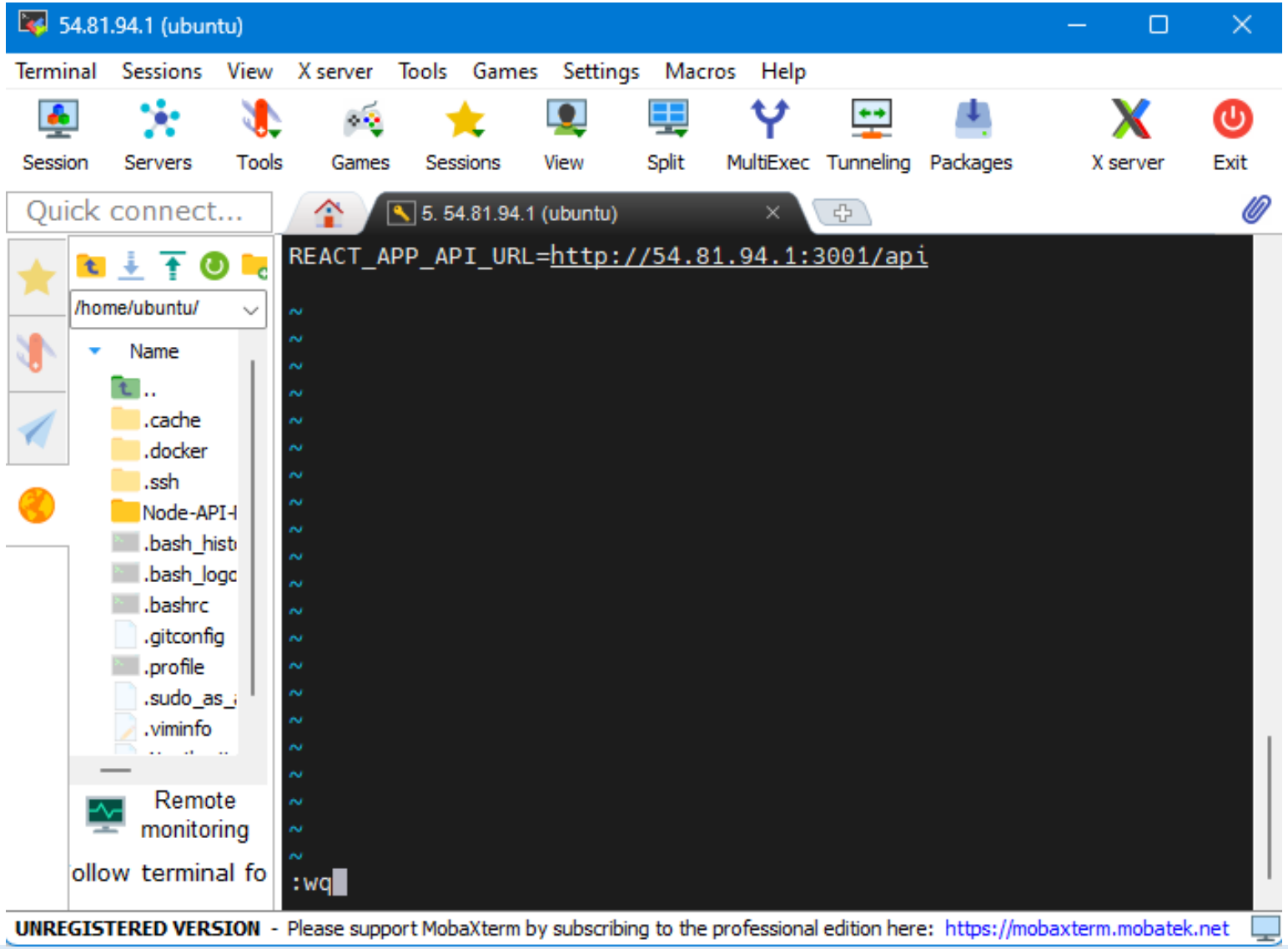
12- Install Docker and Docker-Compose



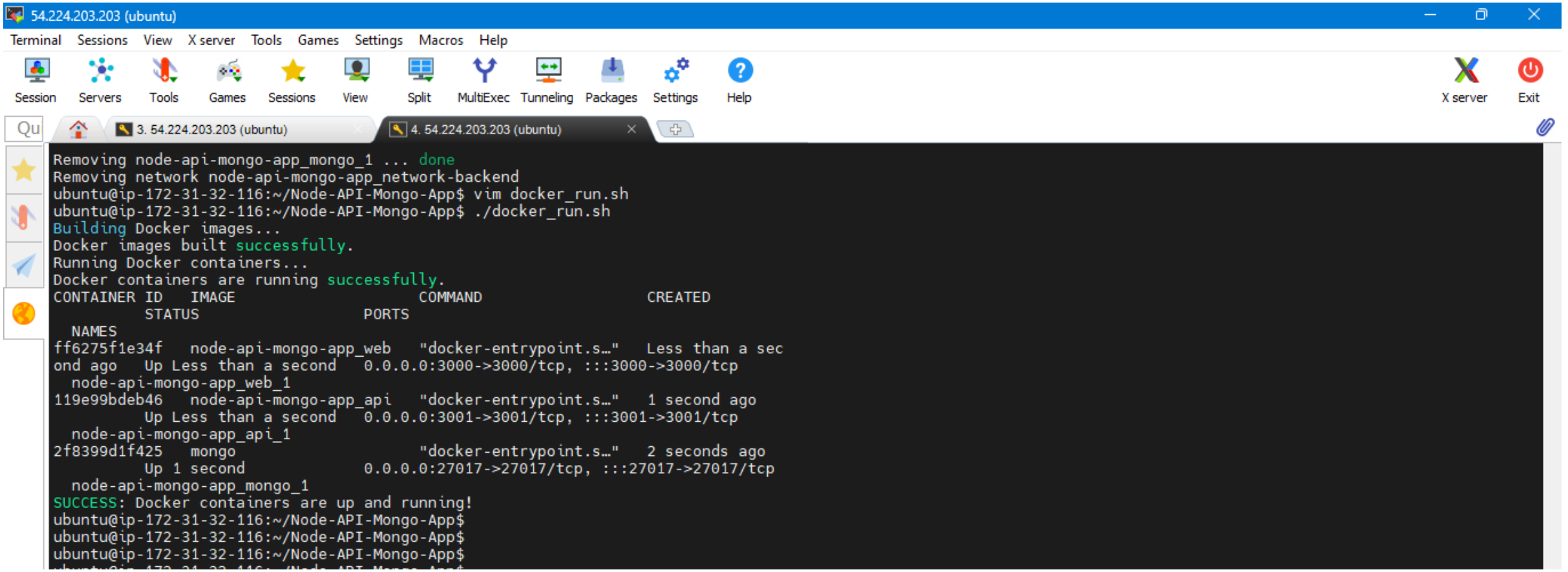
13- Note that at first the application was deployed on host machine so the API URL was http://localhost:3001/api but after deploying on the cloud vm (localhost) must be replaced by the public ip address of ec2 instance



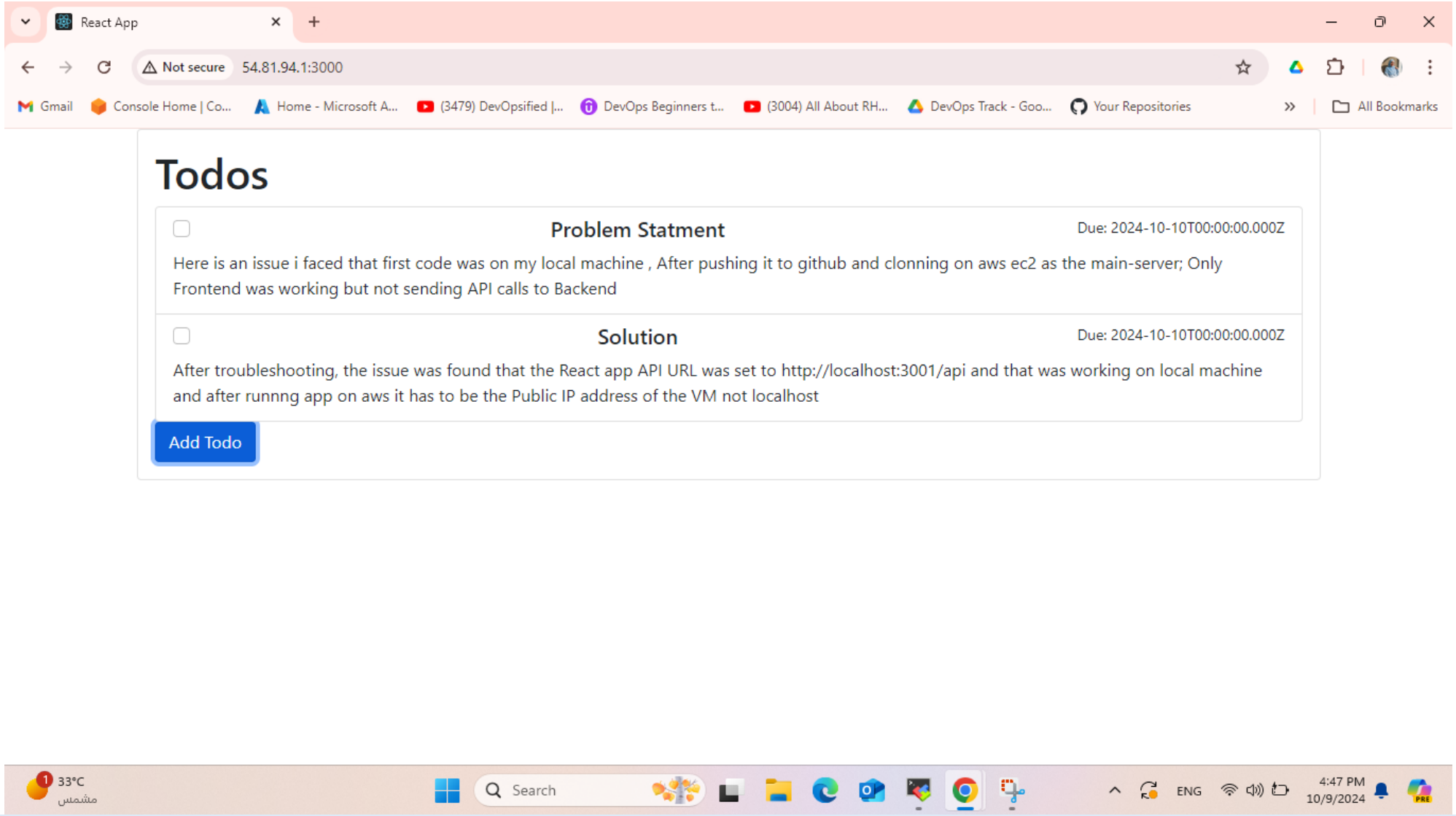
13- API URL after modification with public ip address of ec2 (main server)



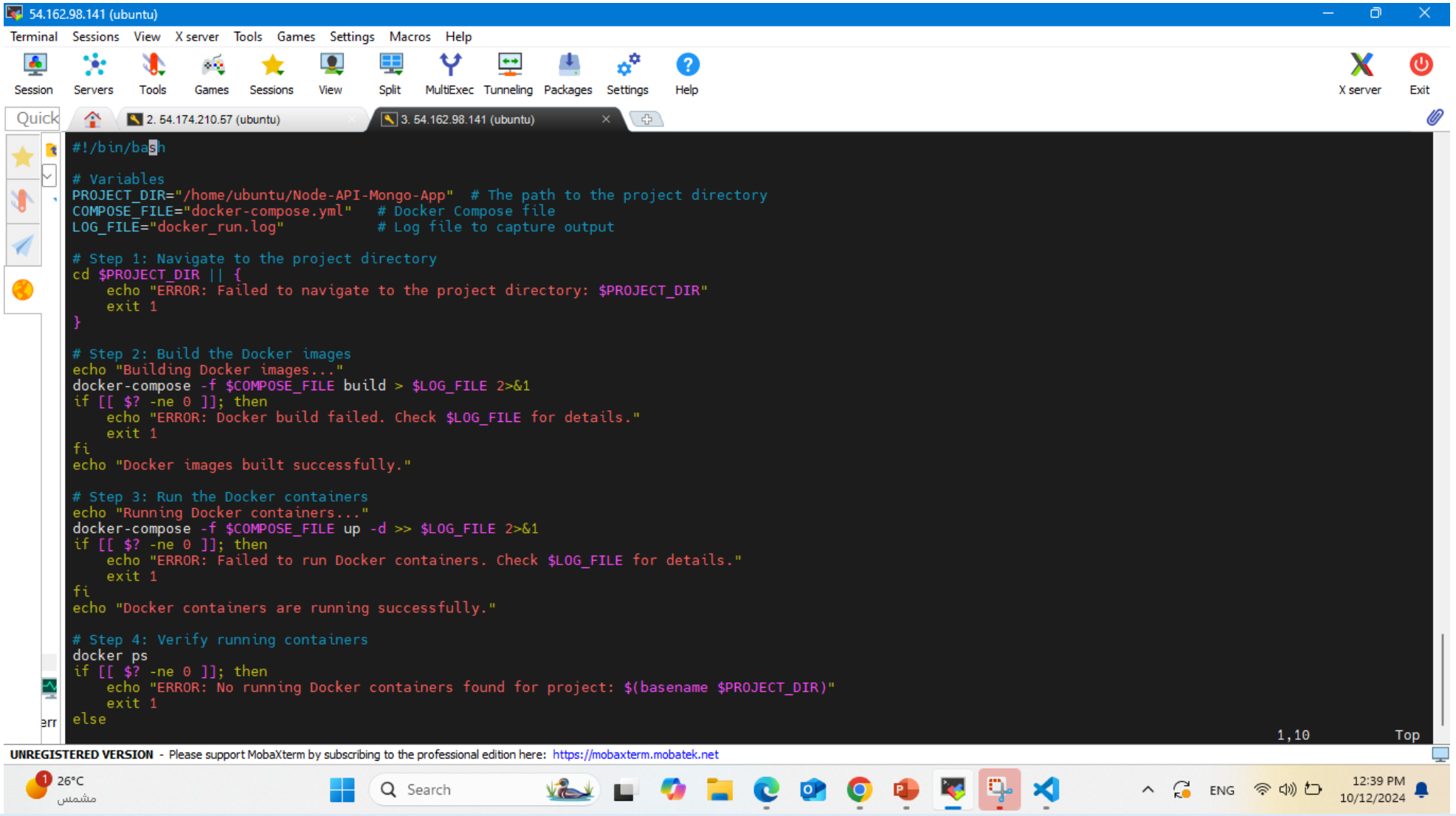
14- Running application from main server on aws



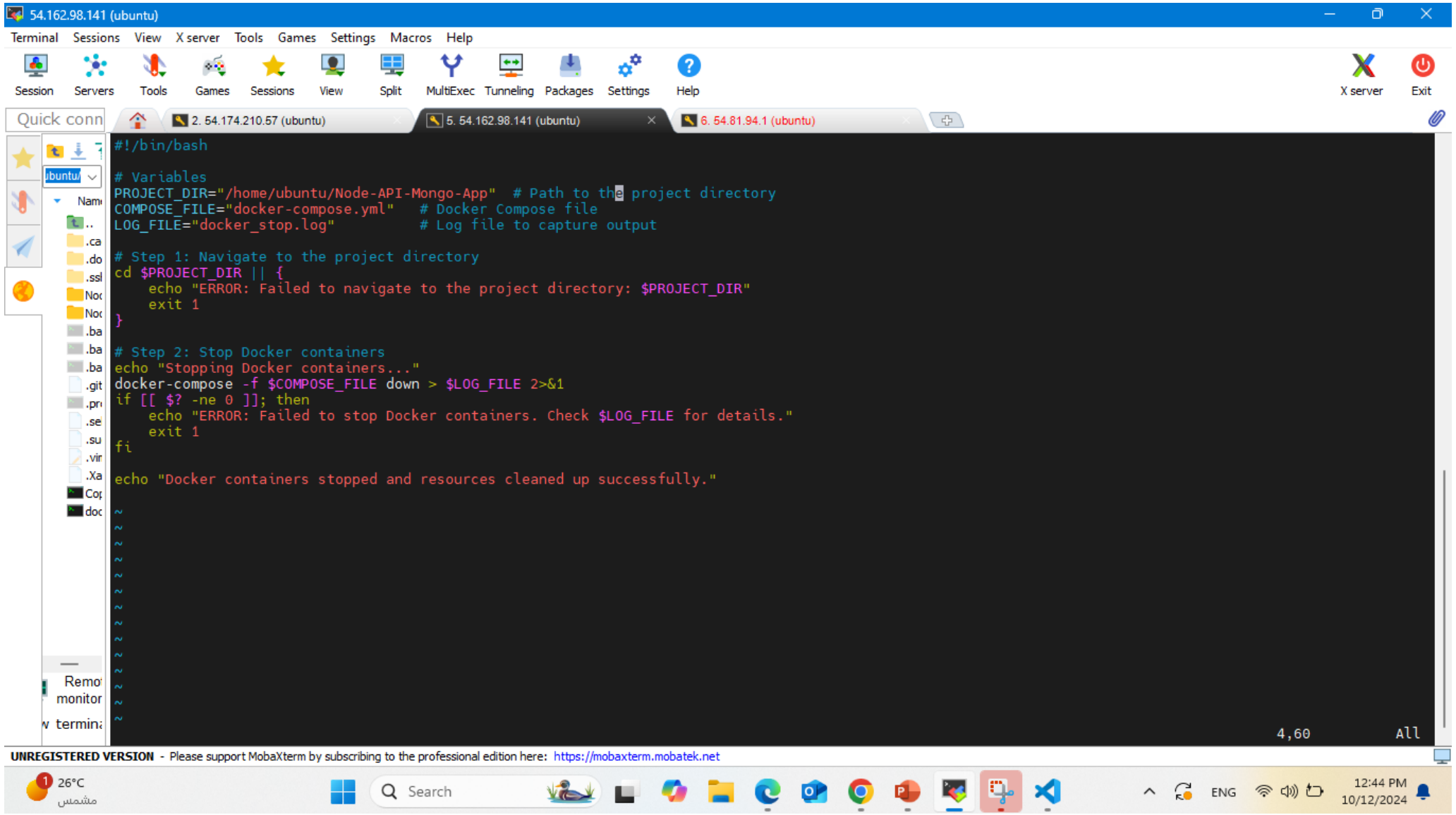
14- Application after deployment in main server on aws



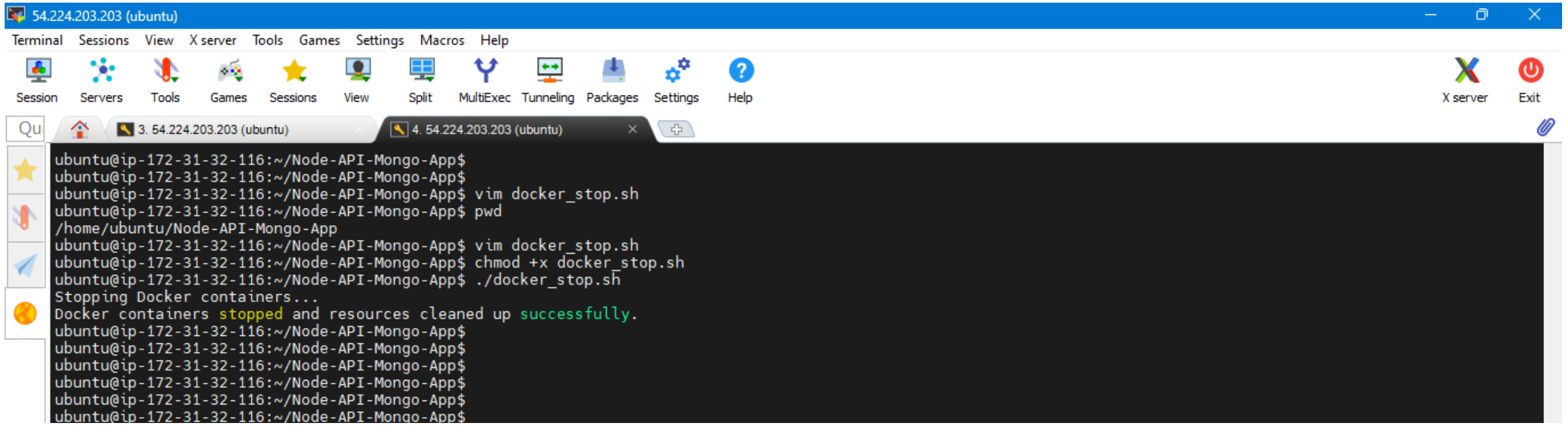
15- Writing Write scripts to automate building and running Docker containers



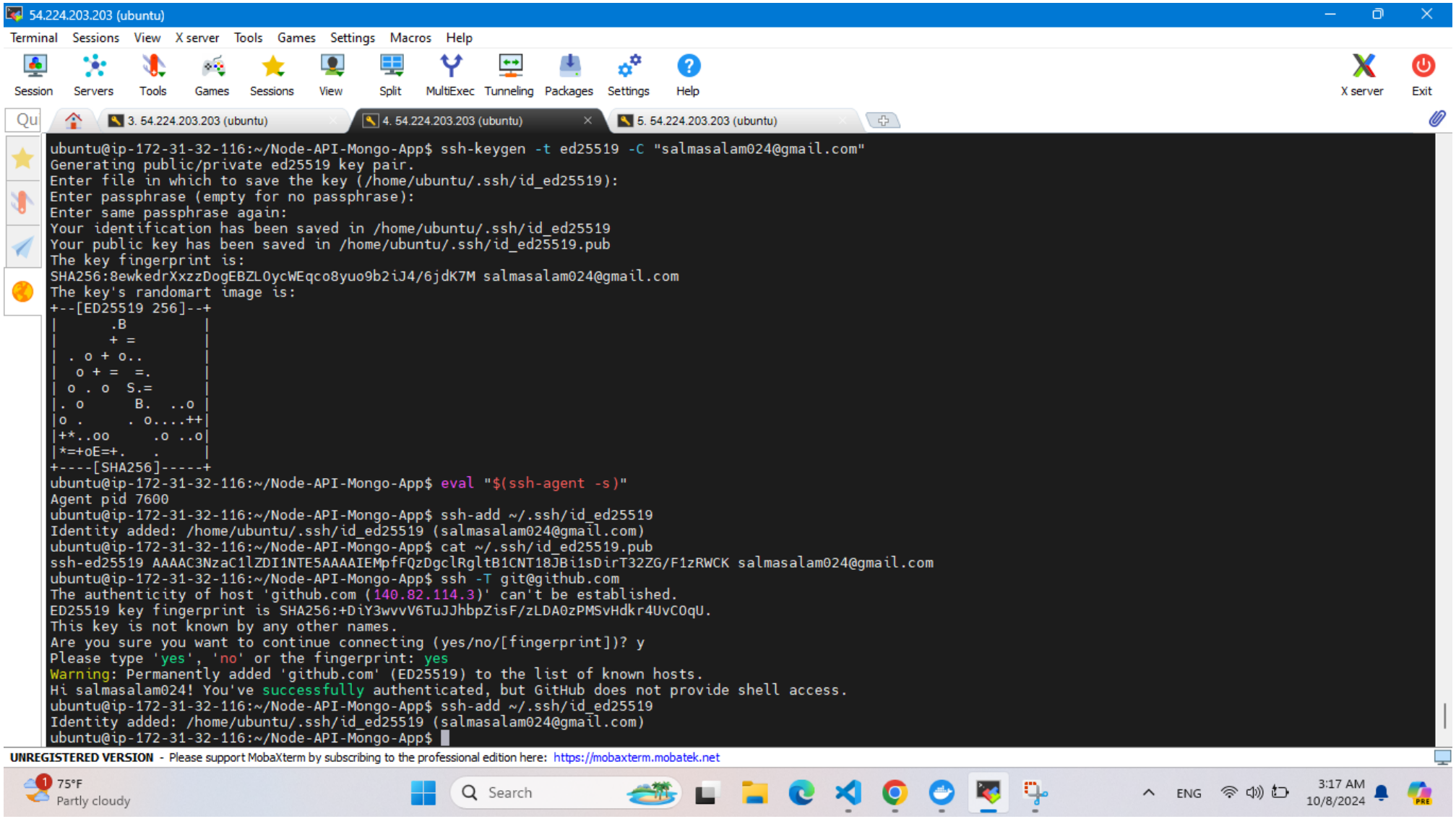
16- Writing a script to automate stopping the Docker containers



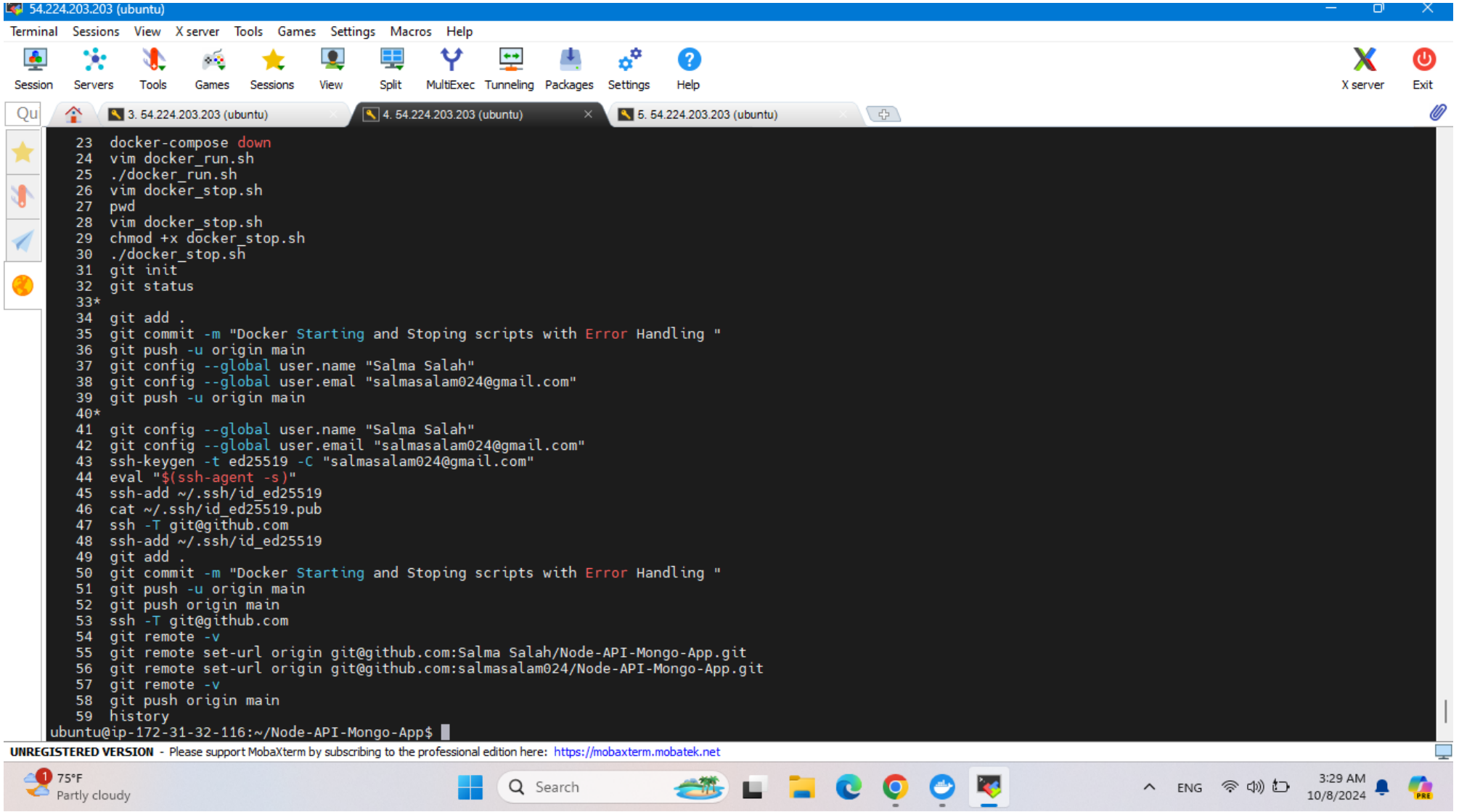
17- Give the running and stopping scrips execute permissions and testing their execution



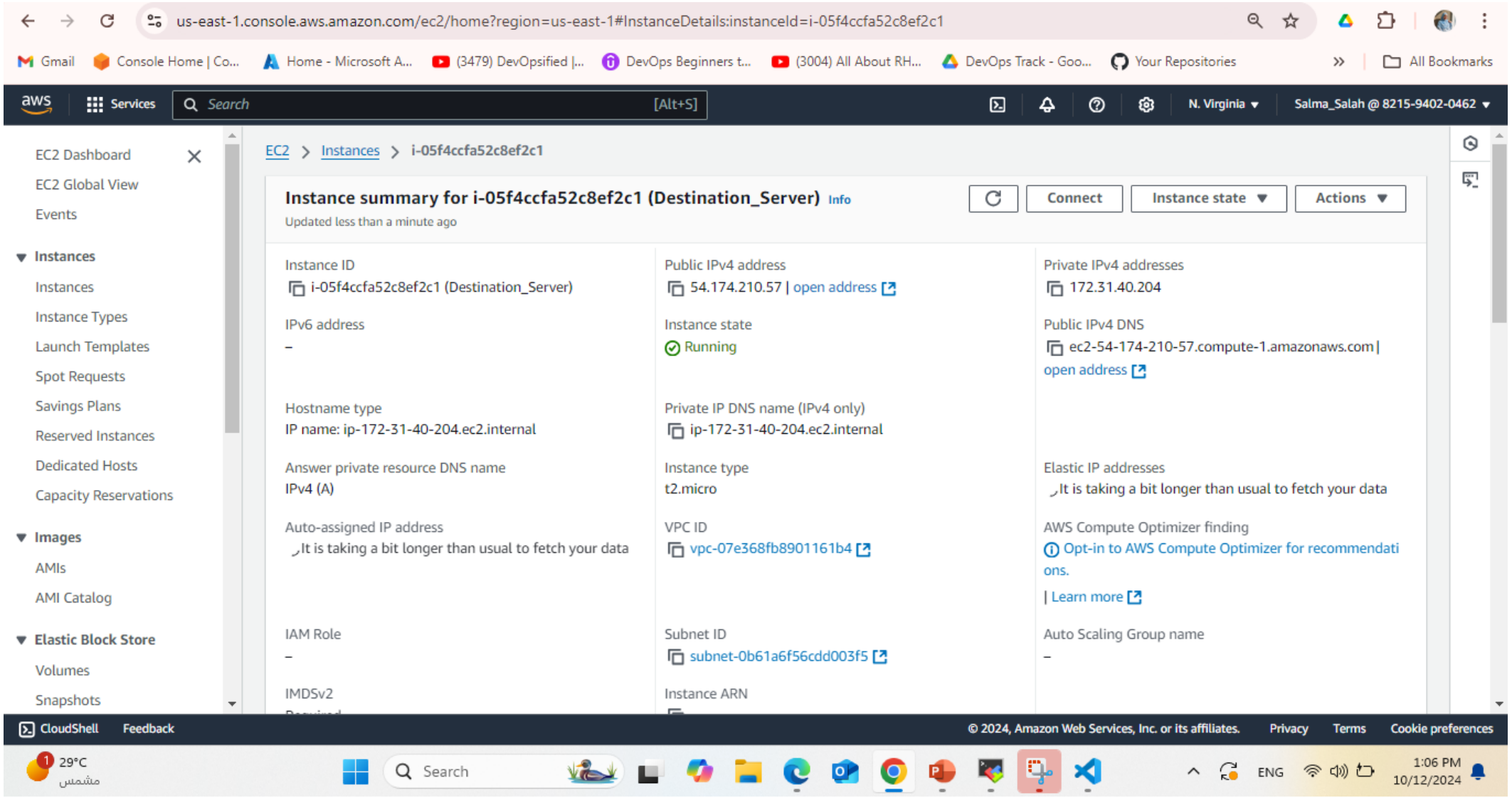
18- Creating ssh keys to authenticate with Github repo to push the new scripts



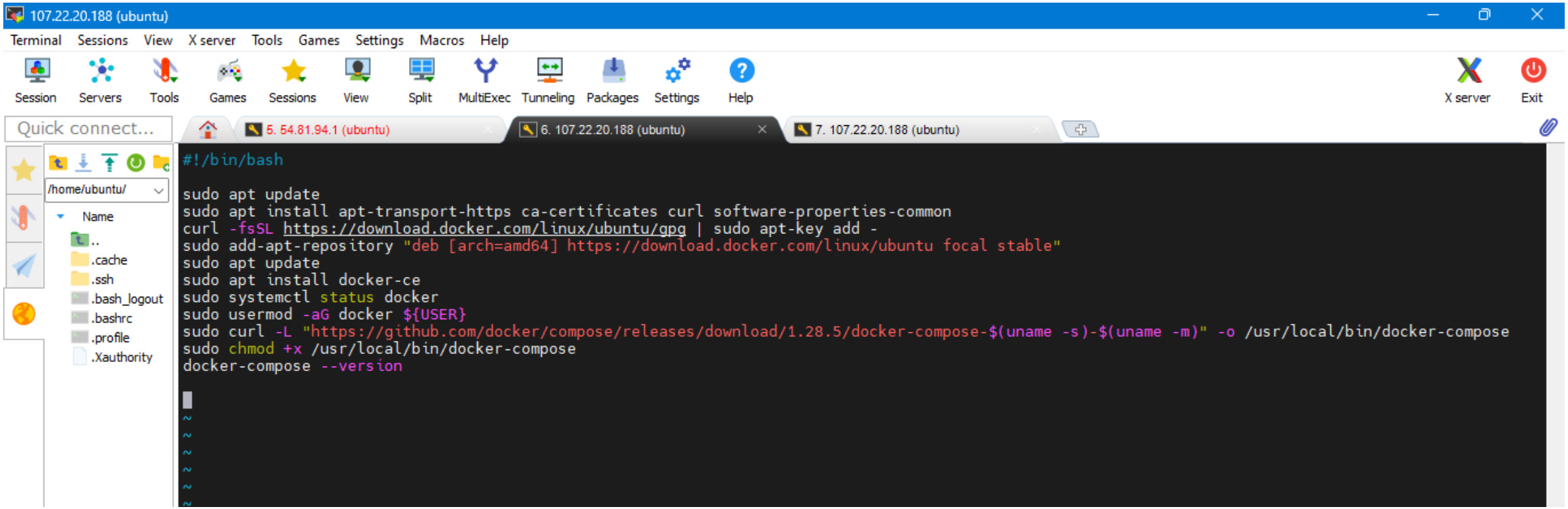
19- Successfully connecting to github repo and push script



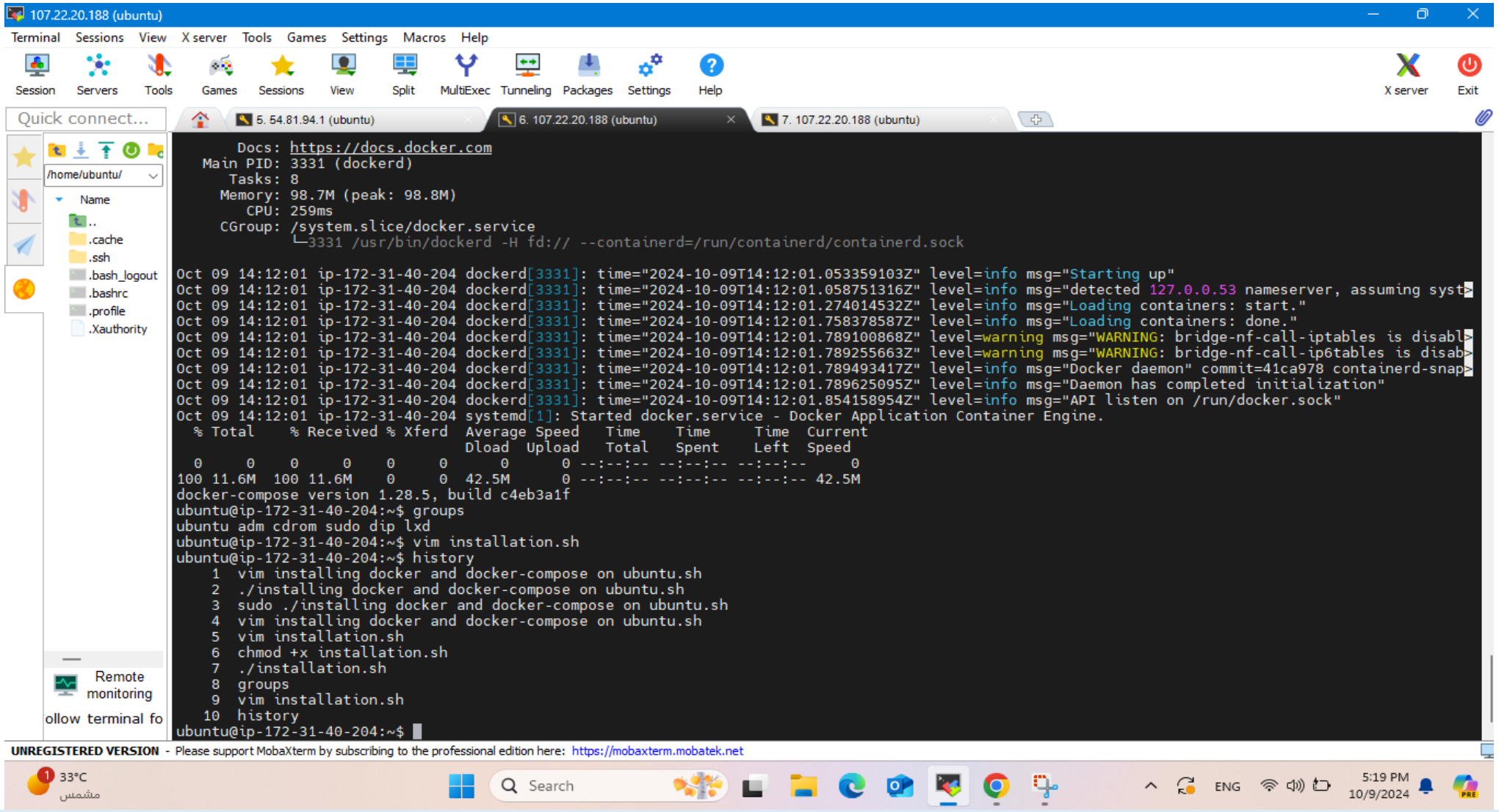
20- Creating Destination Server on AWS (Ubuntu) and open required ports



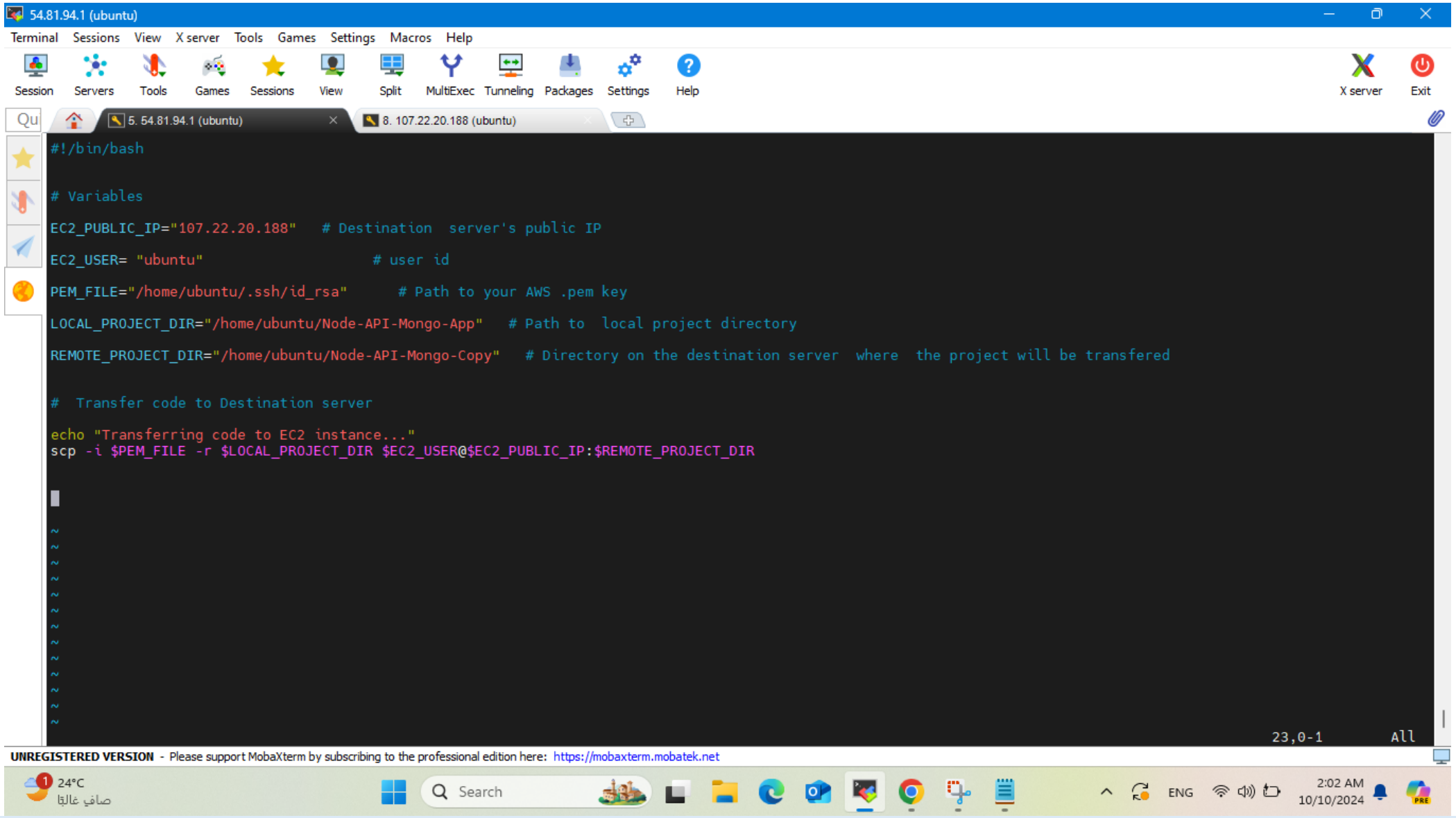
21- Writing bash script to automate installation if docker and docker-compose on destination server



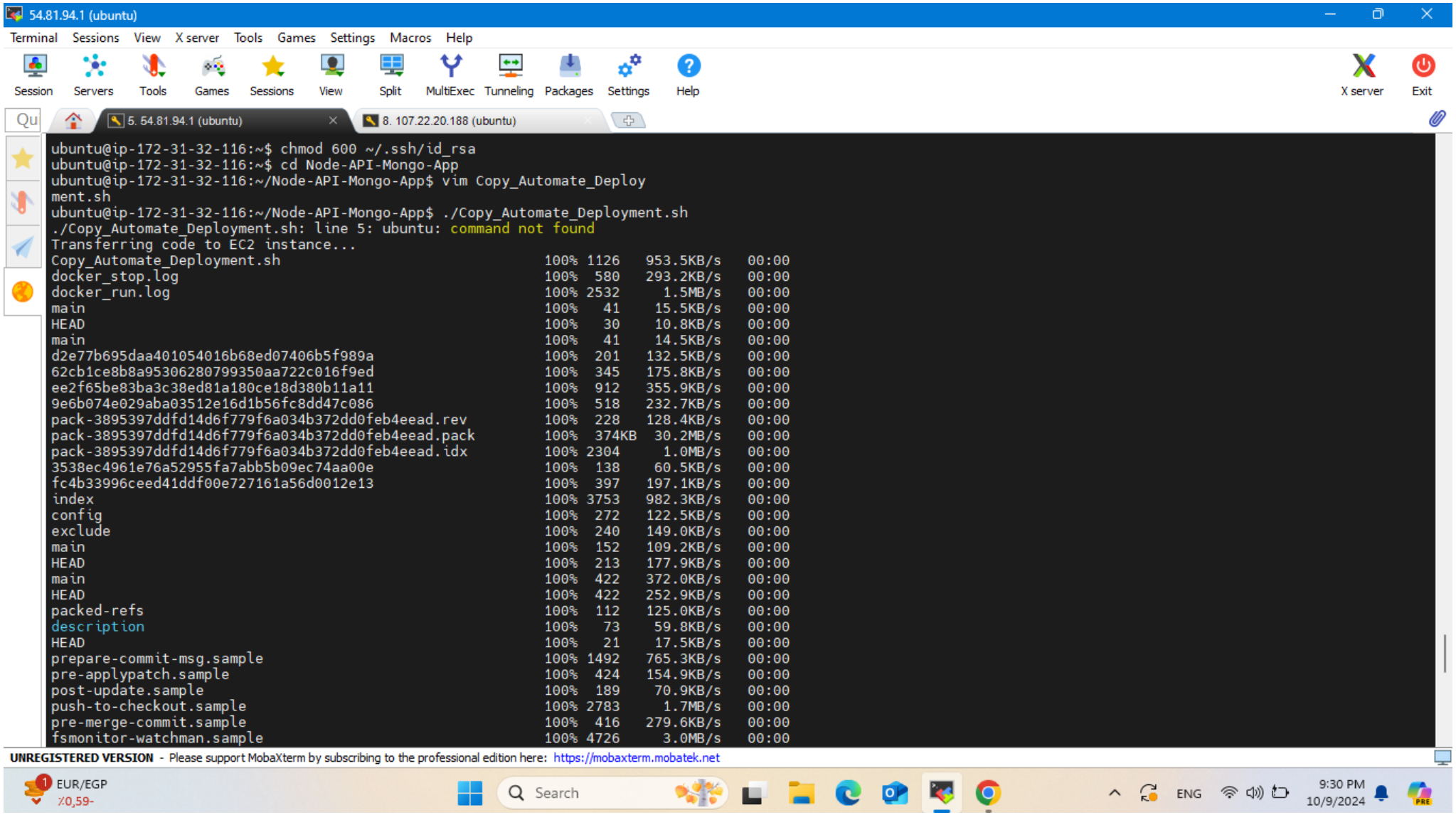
22- Successfully installing docker and docker-compose using script on destination server



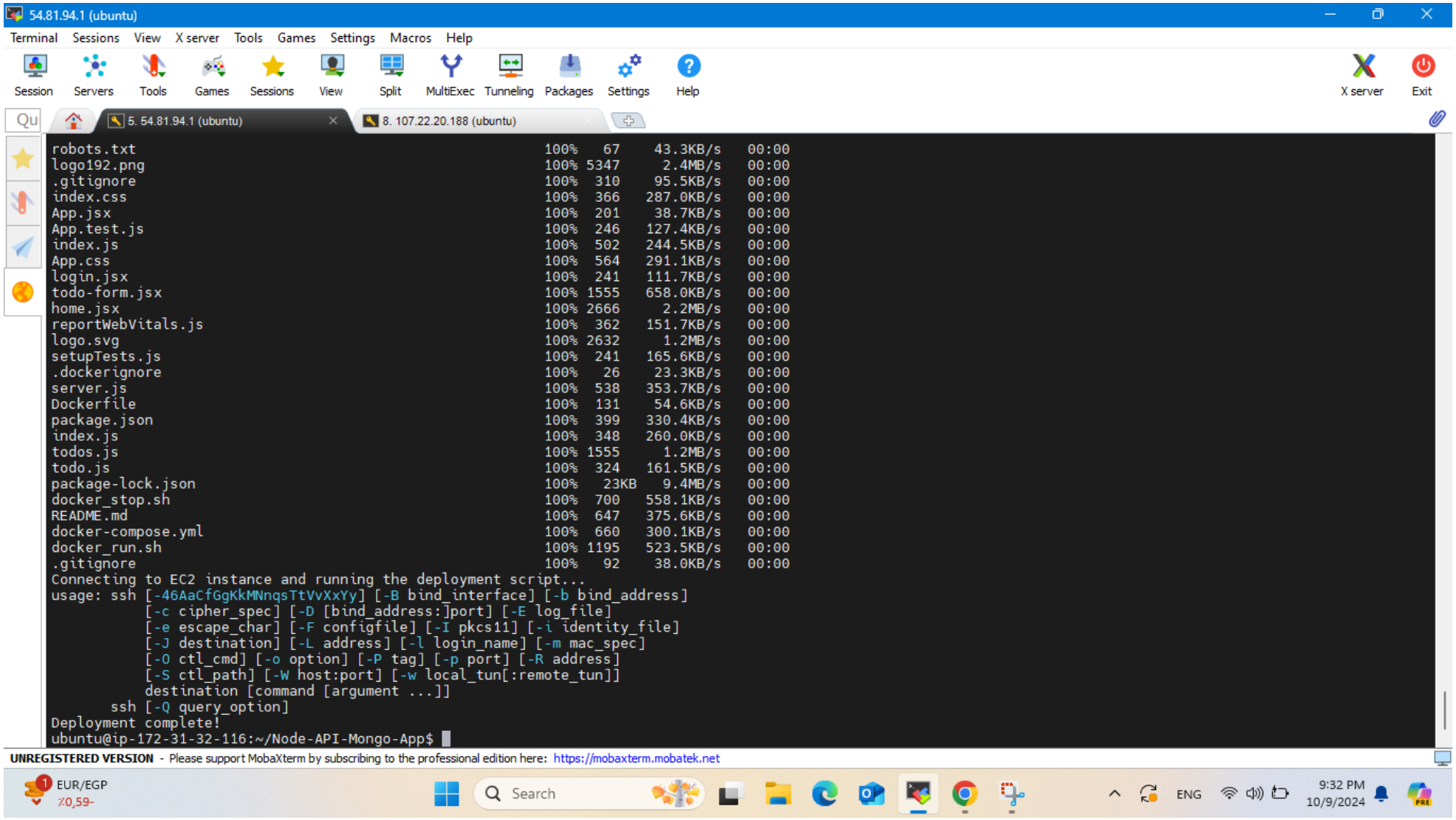
23- Writing bash script to transfer application code to target server



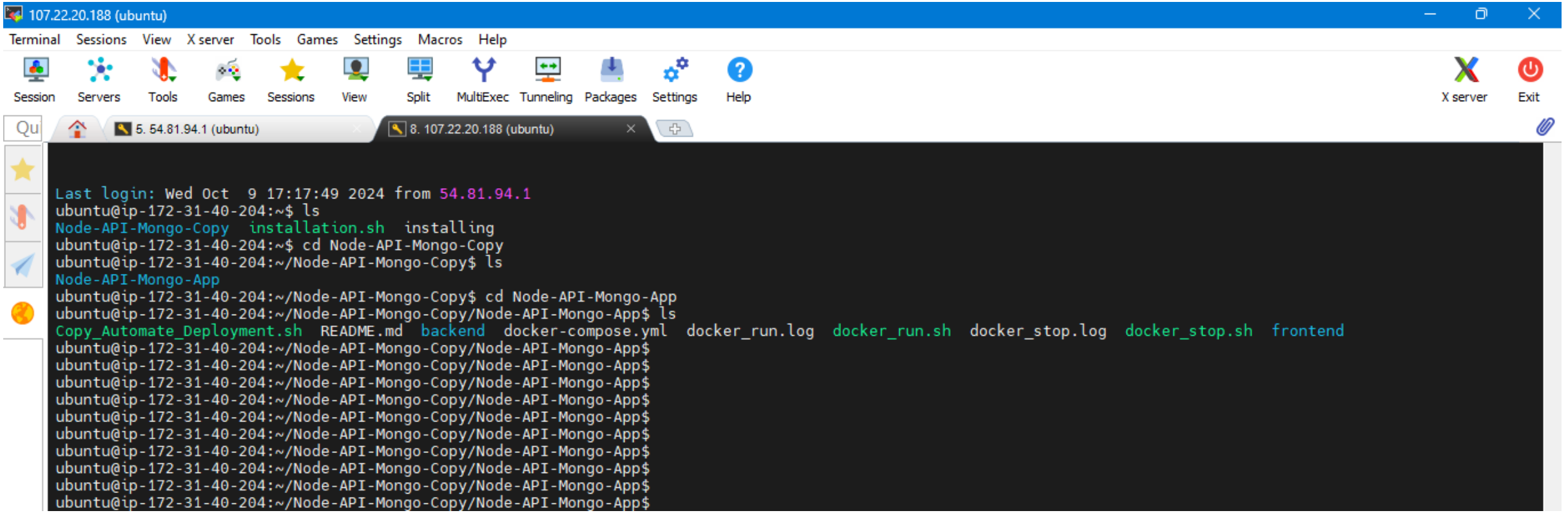
24- Writing bash script to transfer application code to target server



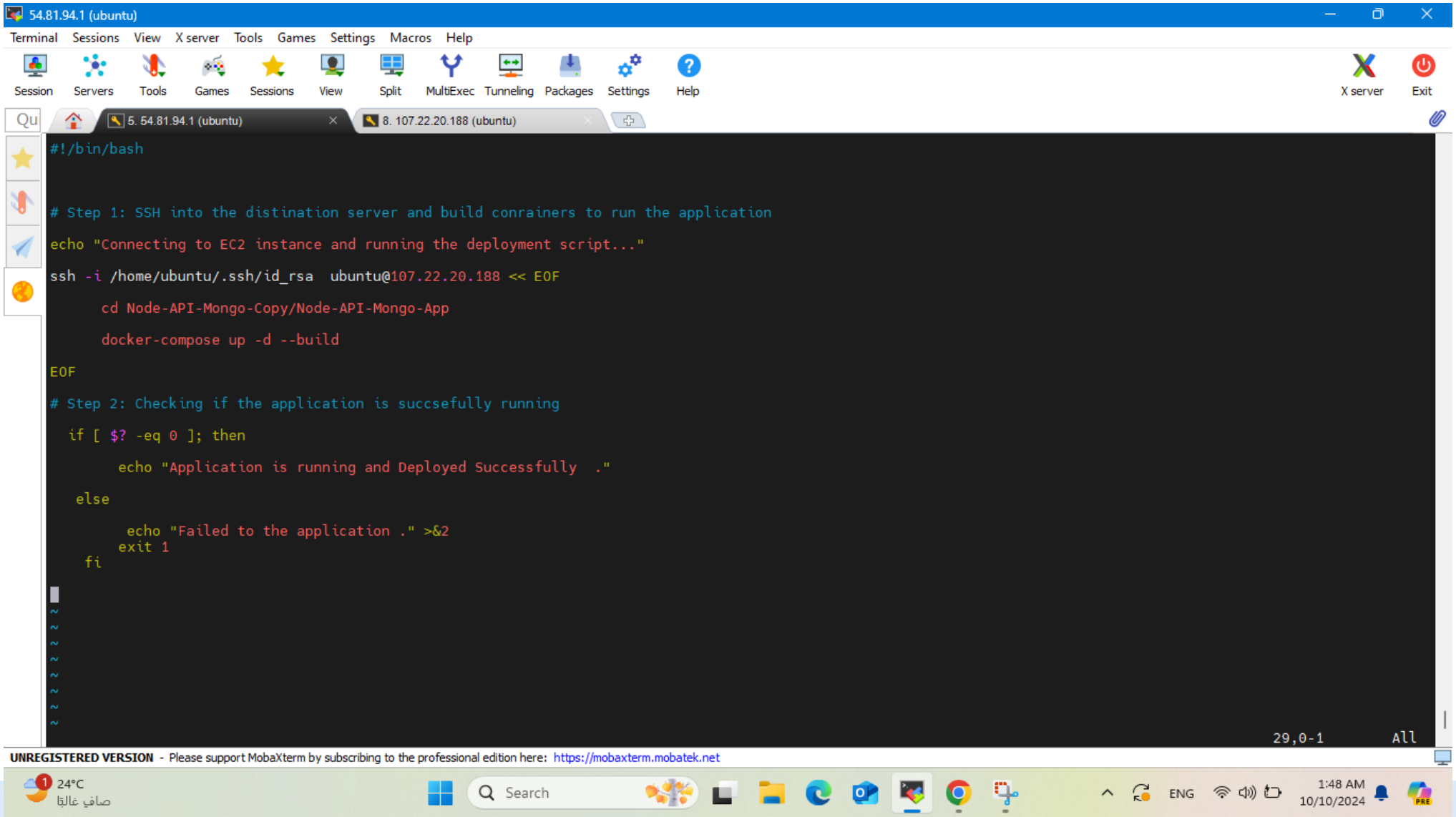
25- Application code is successfully copied to target server



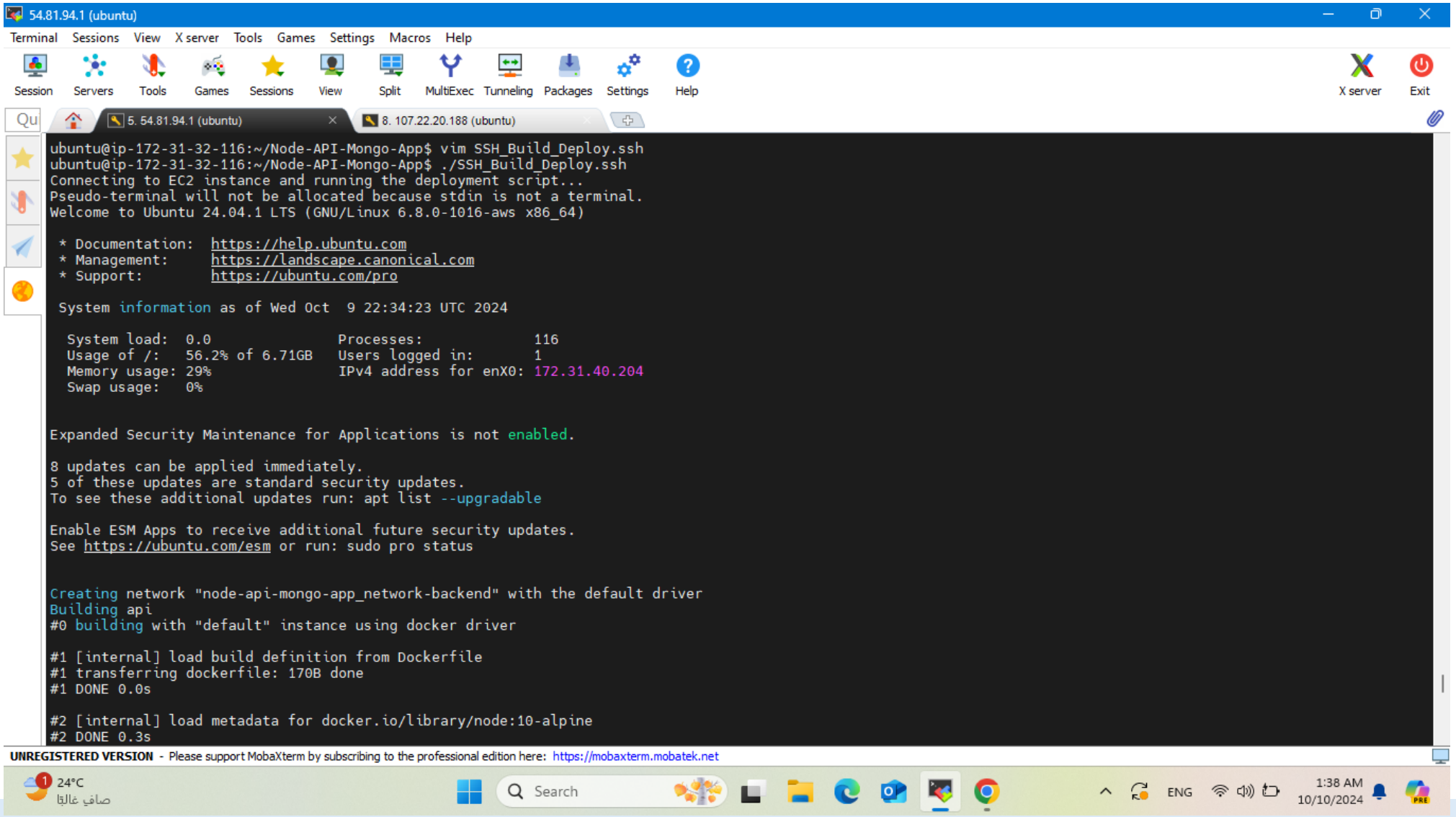
26- Application files in destination server



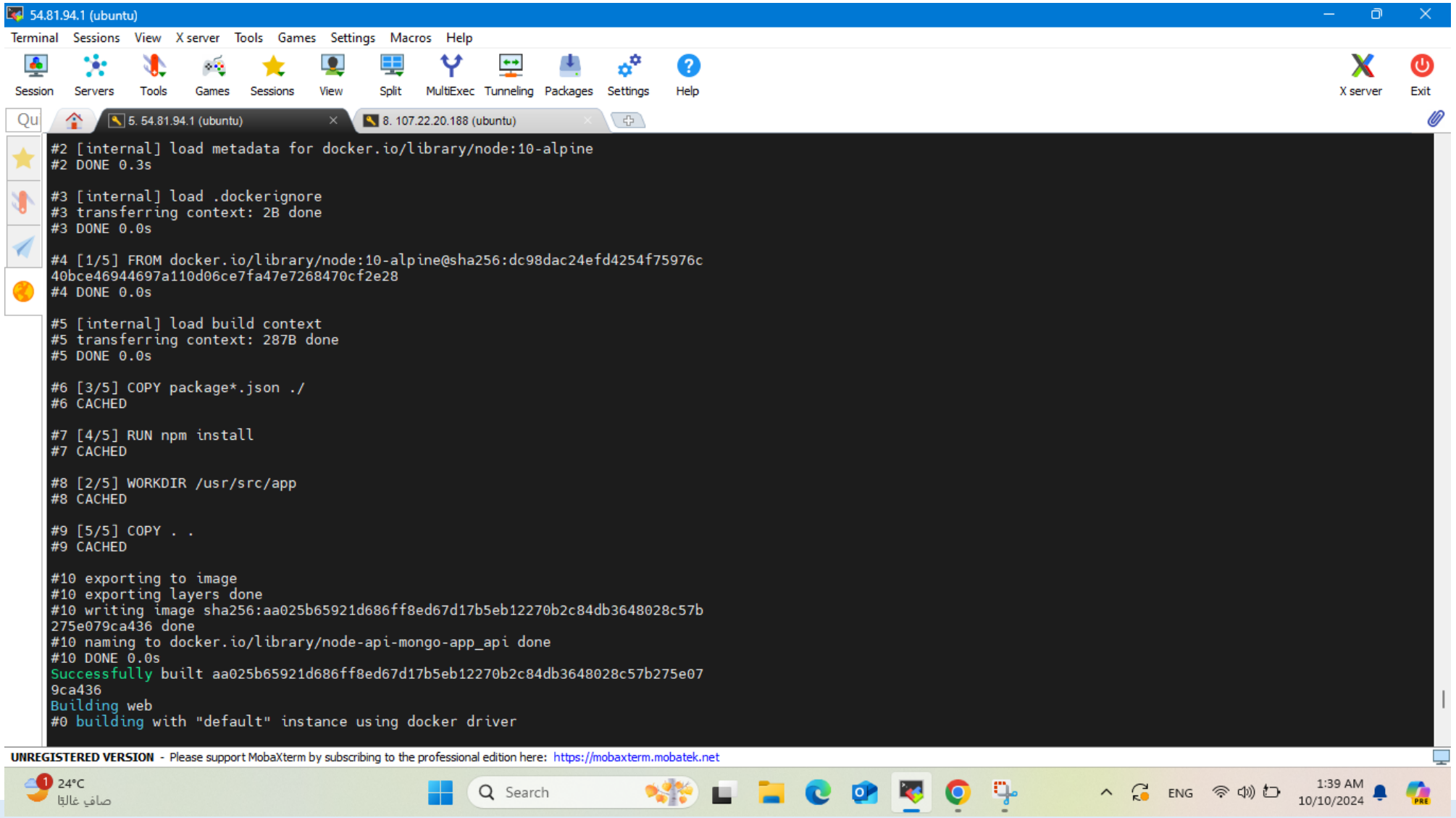
27- Writing bash script to ssh from main server to destination server and run the application using(docker-compose up – build –d)



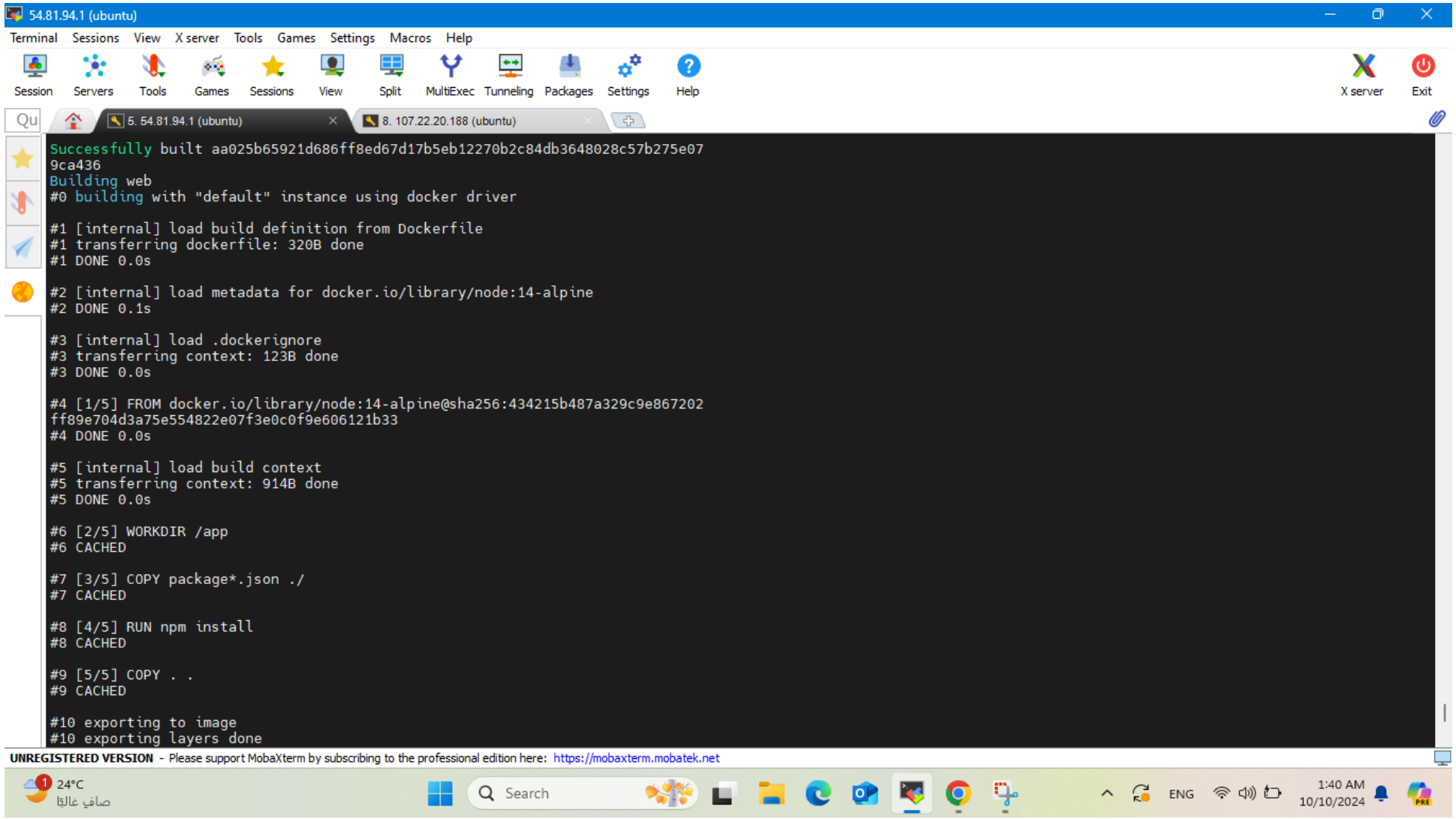
28- Deploying application in destination server by executing the bash script in main server



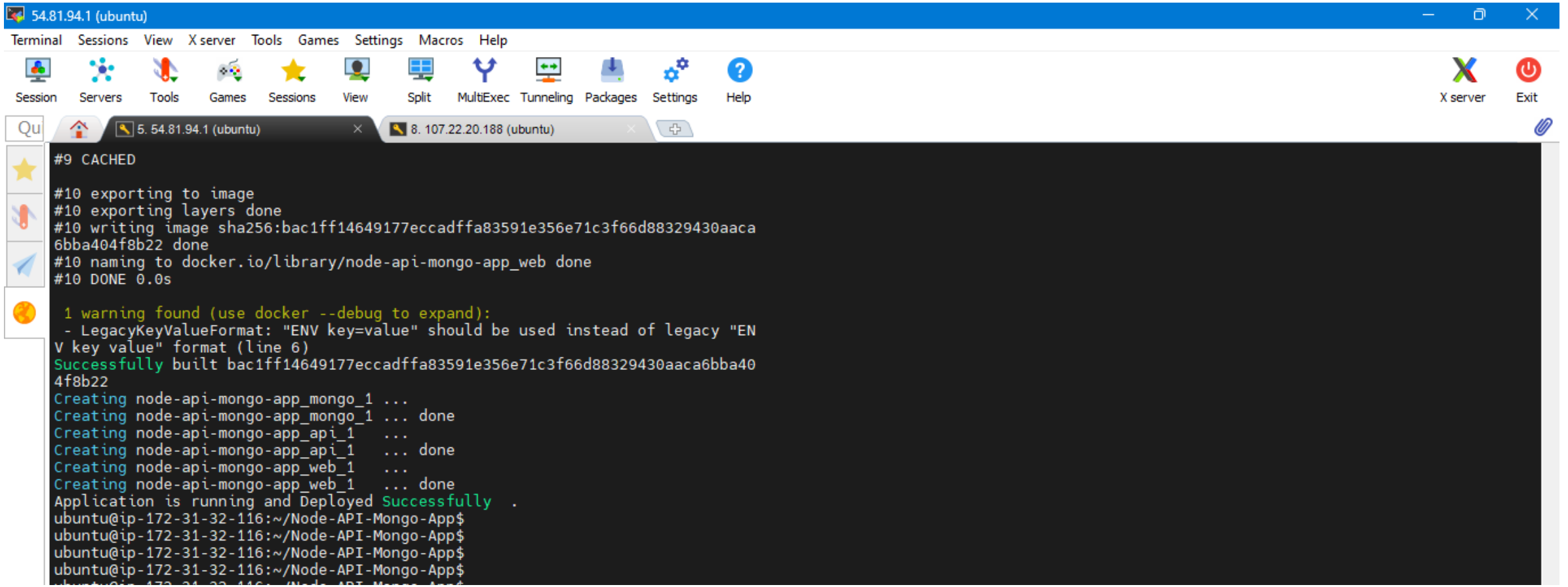
28- Deploying application in destination server by executing the bash script in main server



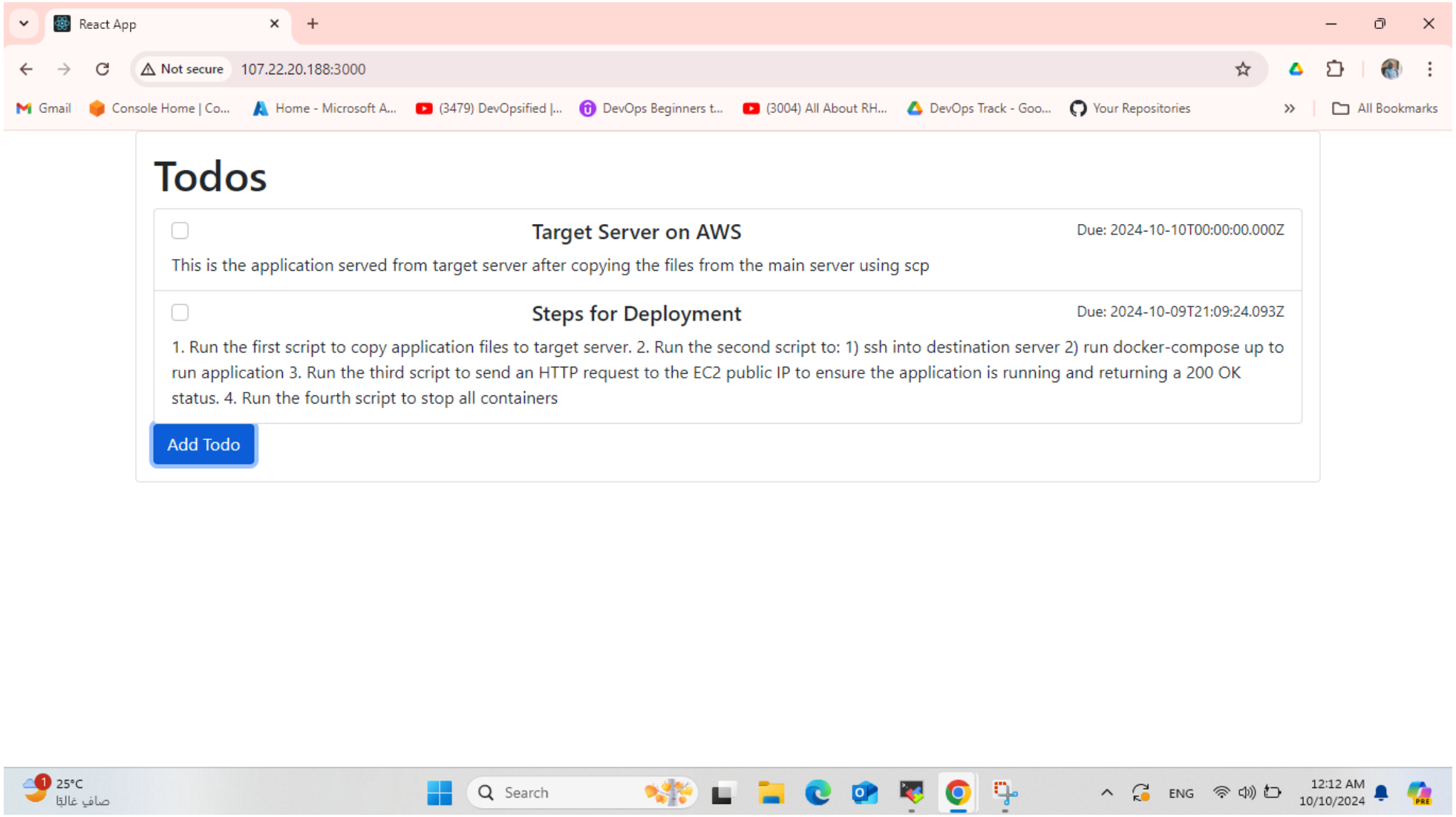
28- Deploying application in destination server by executing the bash script in main server



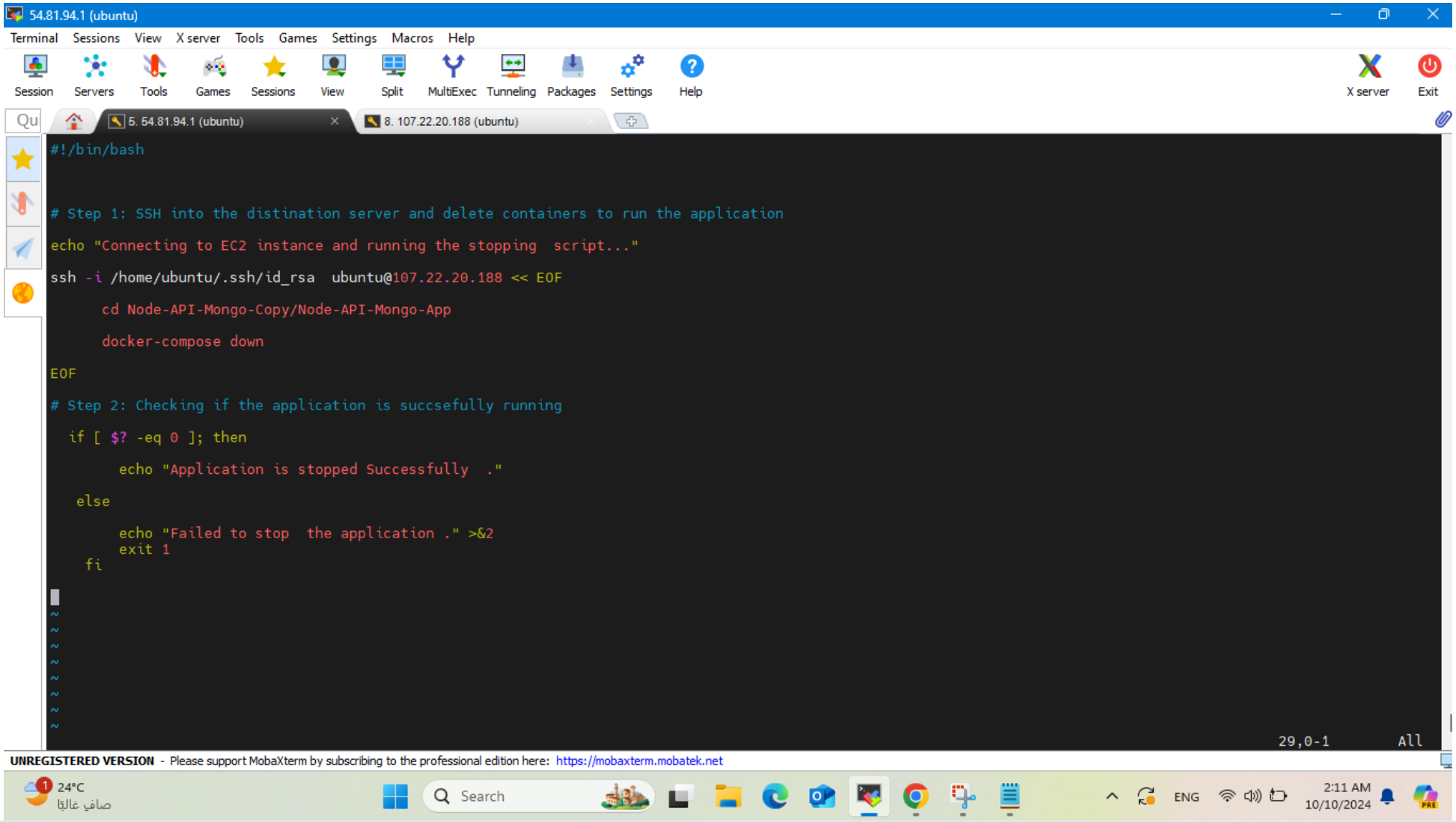
29- Application successfully deployed in main server and the three containers are running by executing run bash script from main server



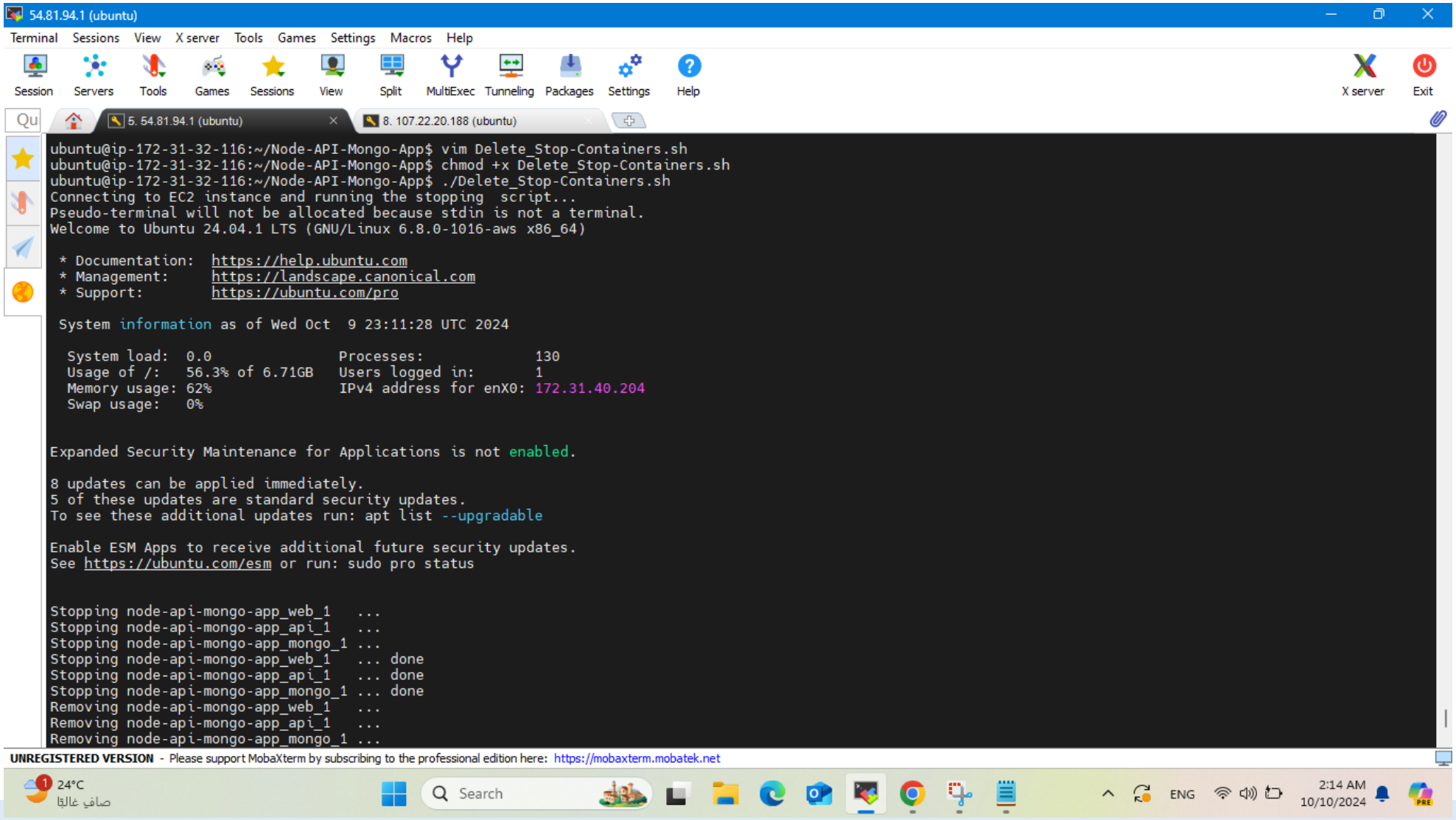
30- Successfully serving the application from destination server



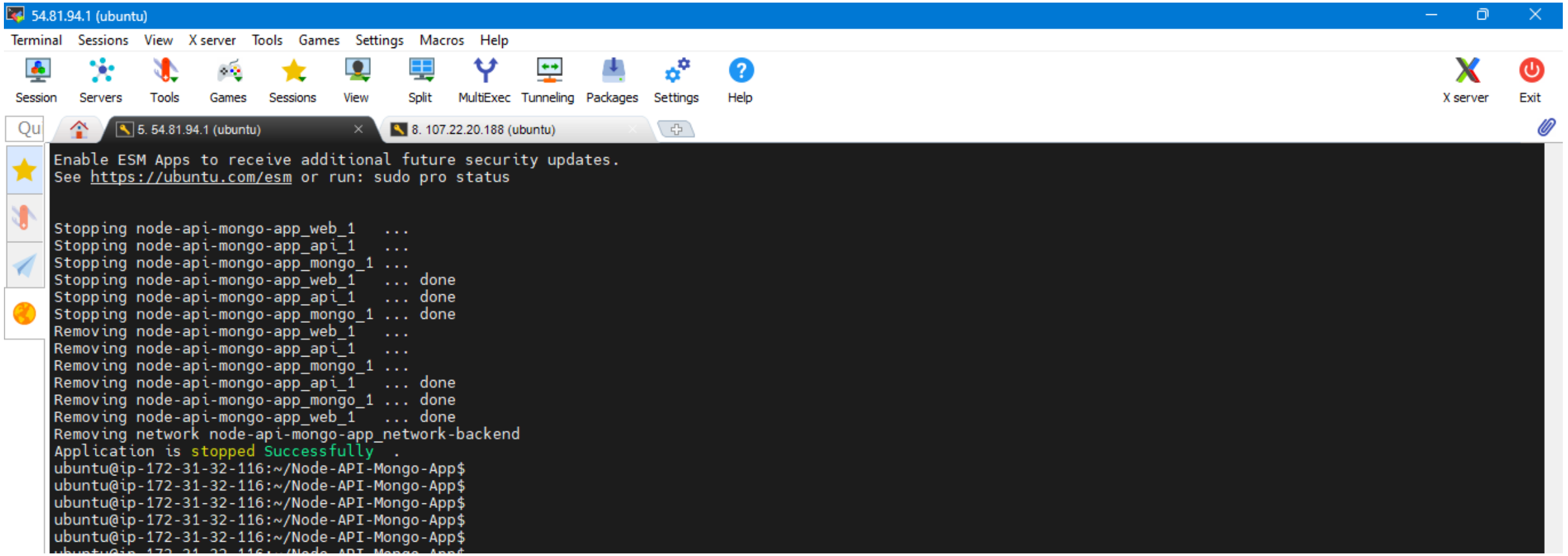
31- Writing bash script to ssh from main server to destination server and stop the application using(docker-compose down)



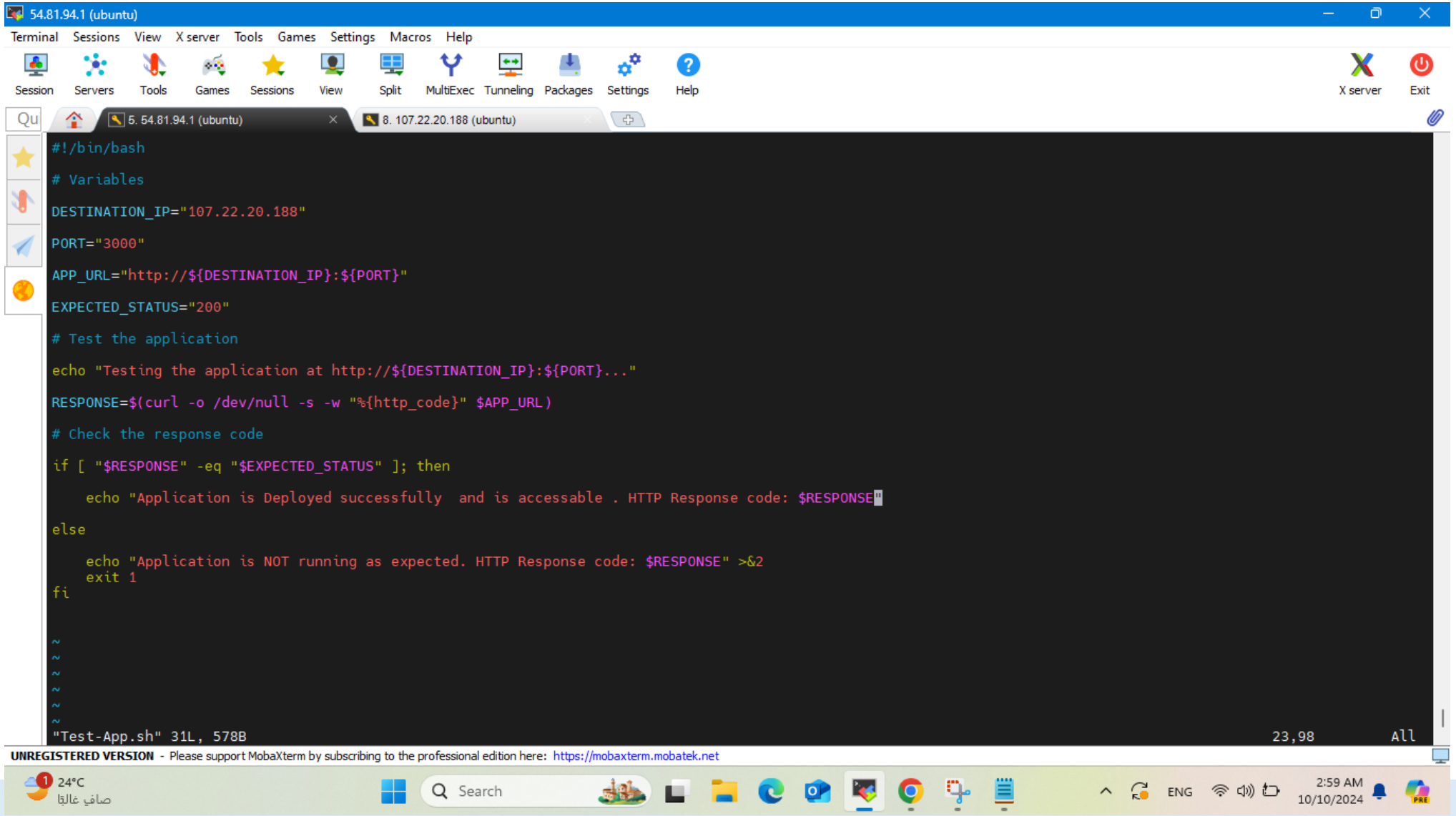
32- Stopping the application in destination server by executing the bash script in main server



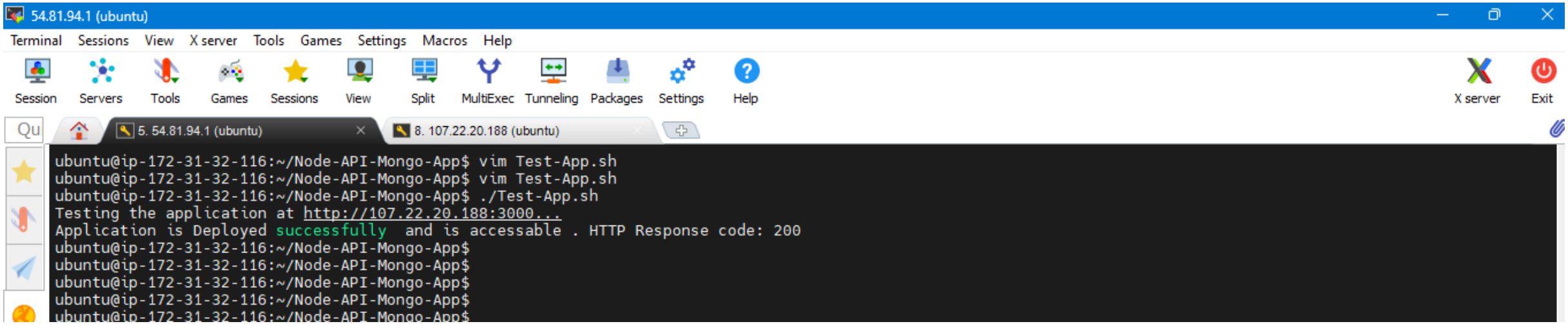
33- Application successfully stopped in destination server and the three containers are removed by executing stop bash script from main server



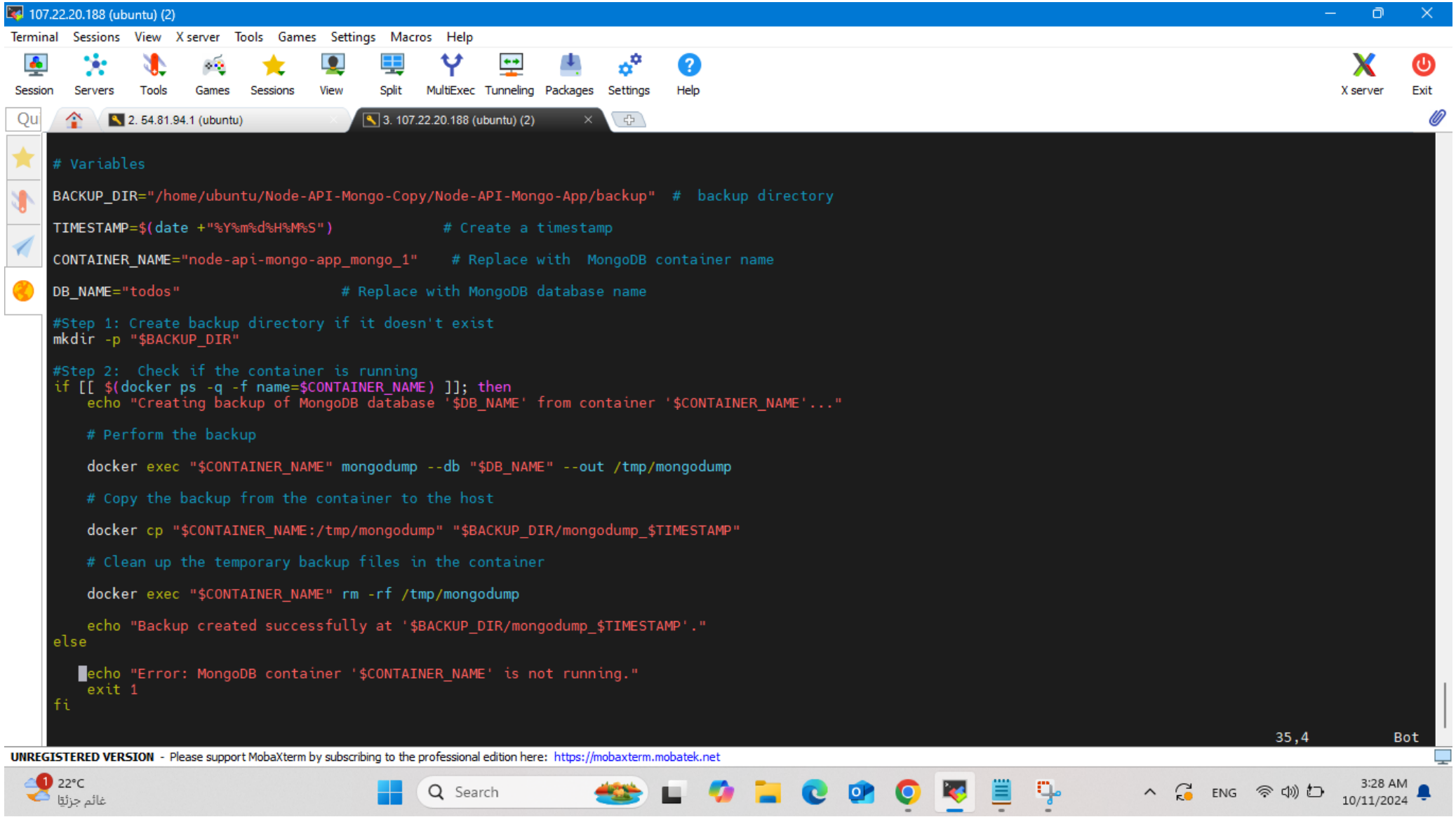
34- Writing bash script to test if the application has successfully deployed using (curl)



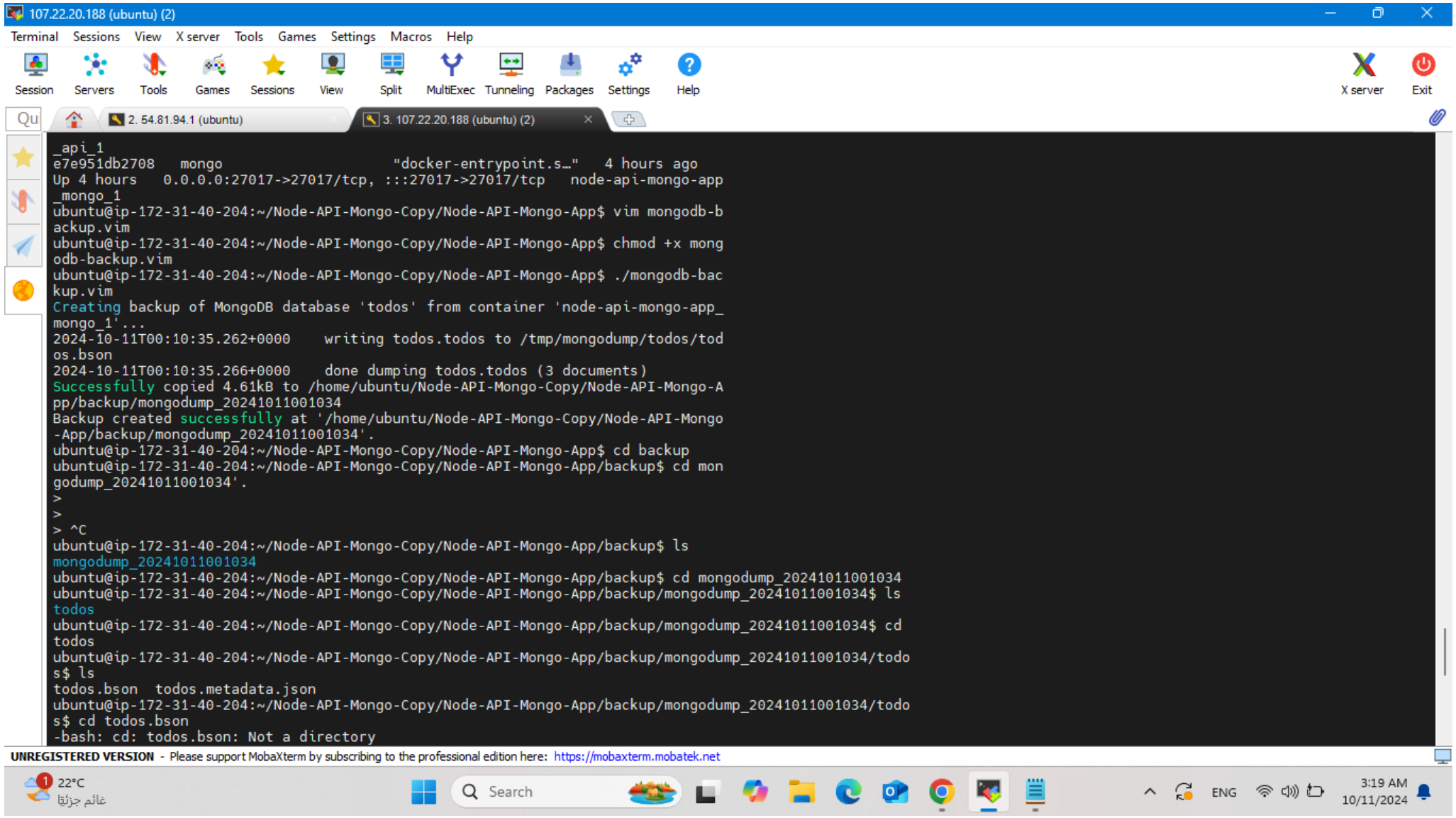
35- Successfully testing the application and get the expected response



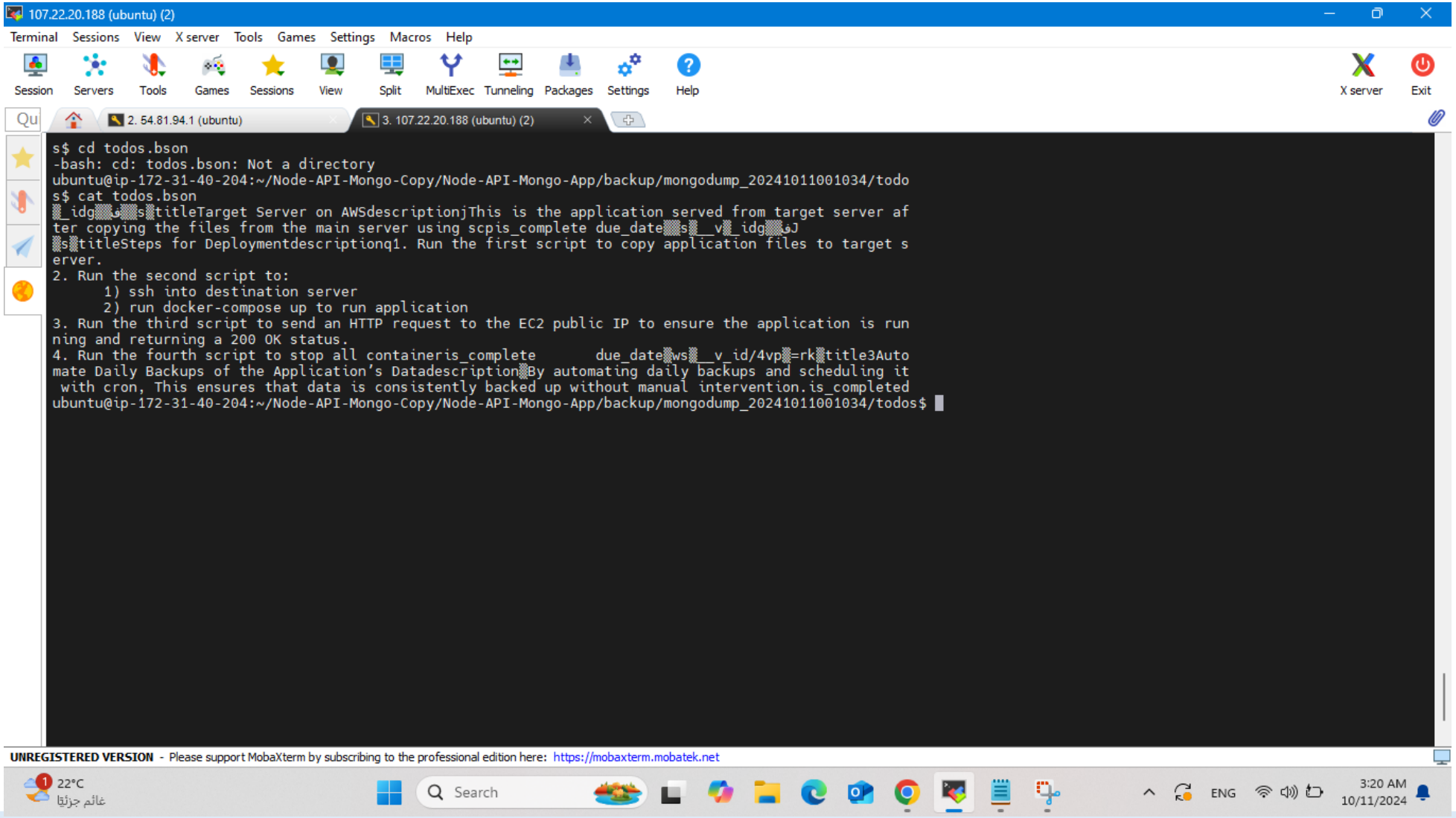
36- Writing bash script to backup all application data



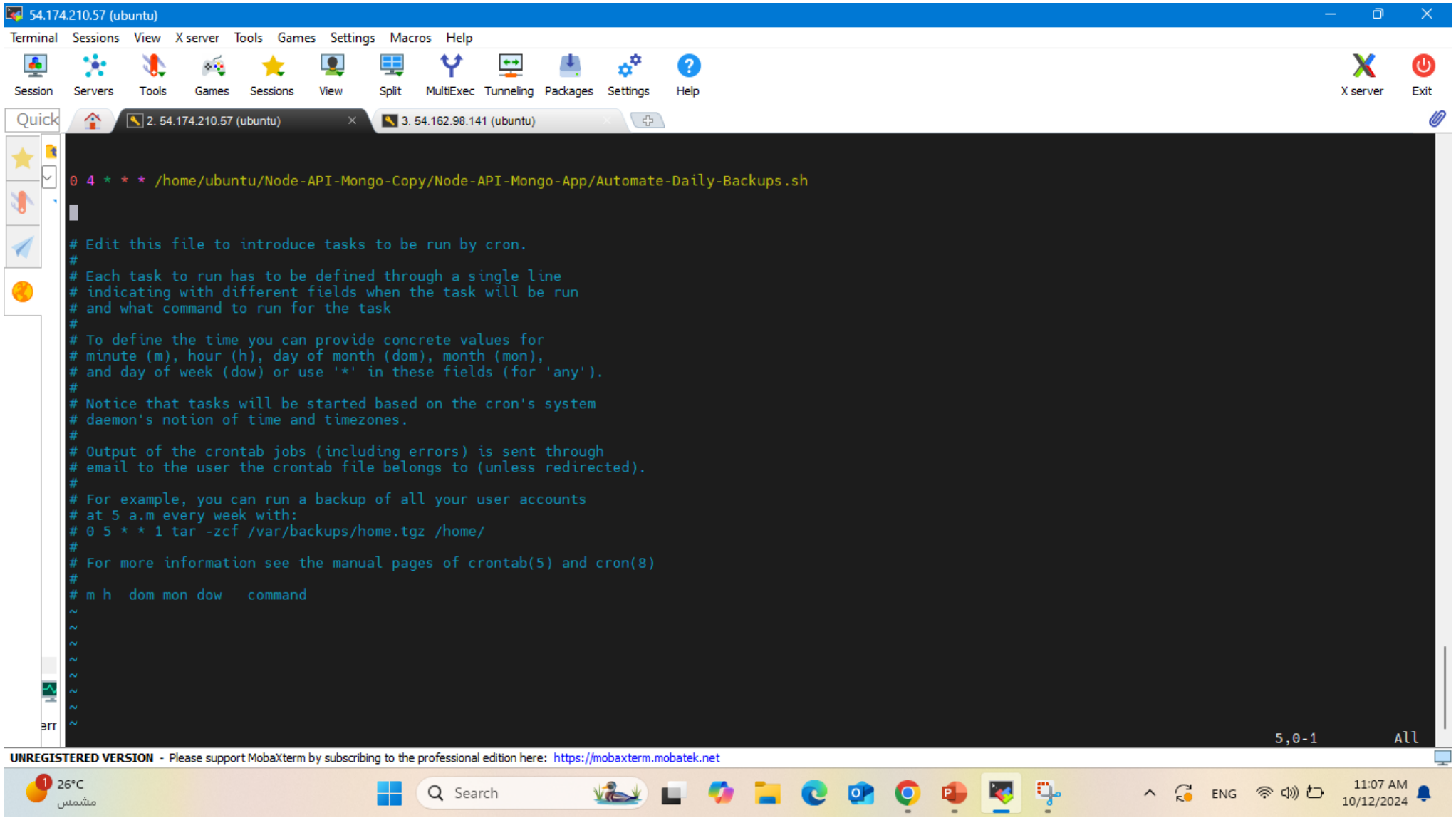
37- Testing and executing the backup script



38- Content of backup file indicating successfully execution of backup script



39- Create a cron job to automate daily backup at 4 am



40- Problems I faced and solved

1- Authentication with git I made ssh

2- API URL was local host and has to be changed to Public IP of aws ec2

3- Run ssh in interactive shell

4- Connection to mongo dB to make backup

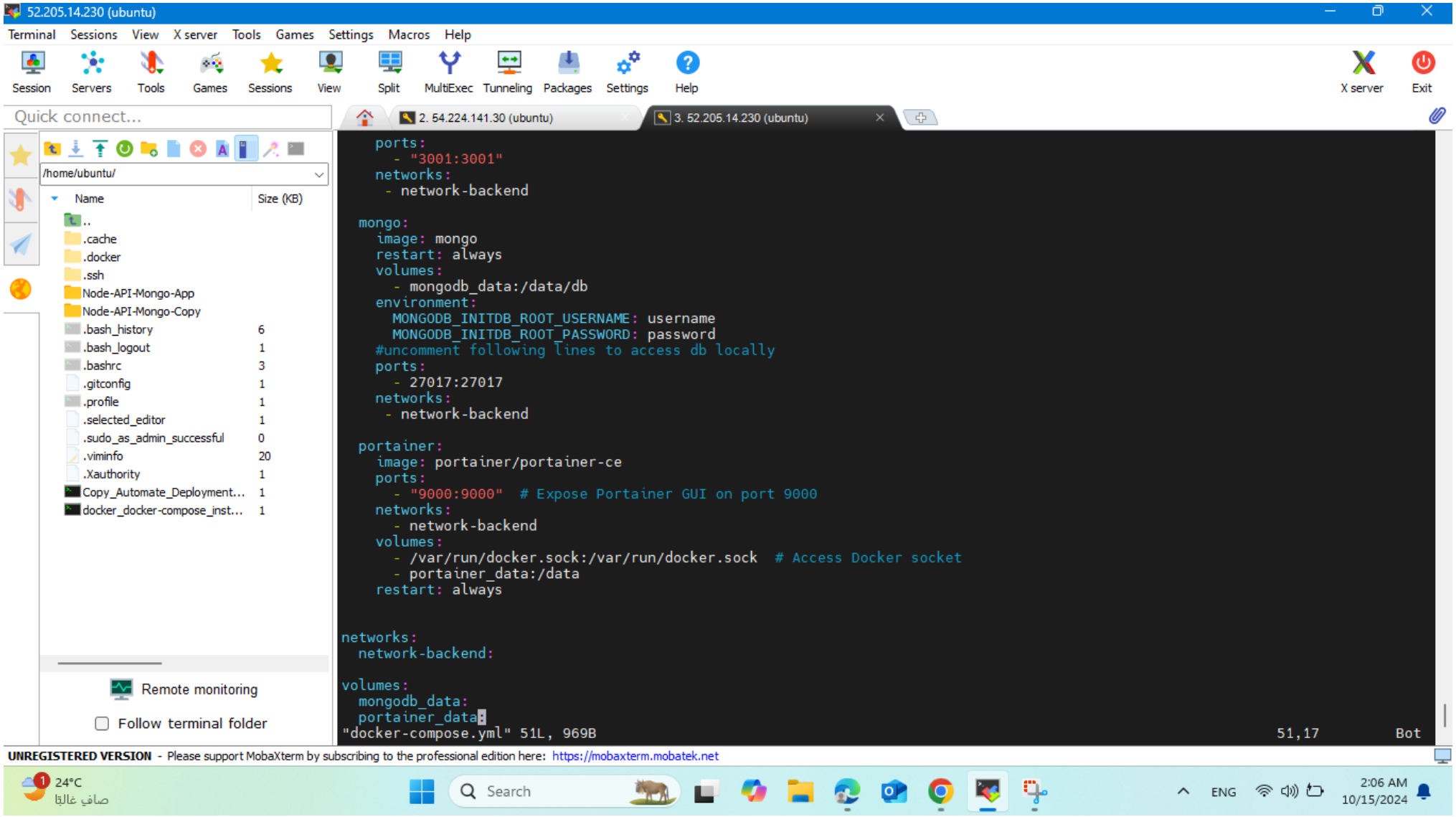
41- Github Repository

https://github.com/salmasalam024/Node-API-Mongo-App

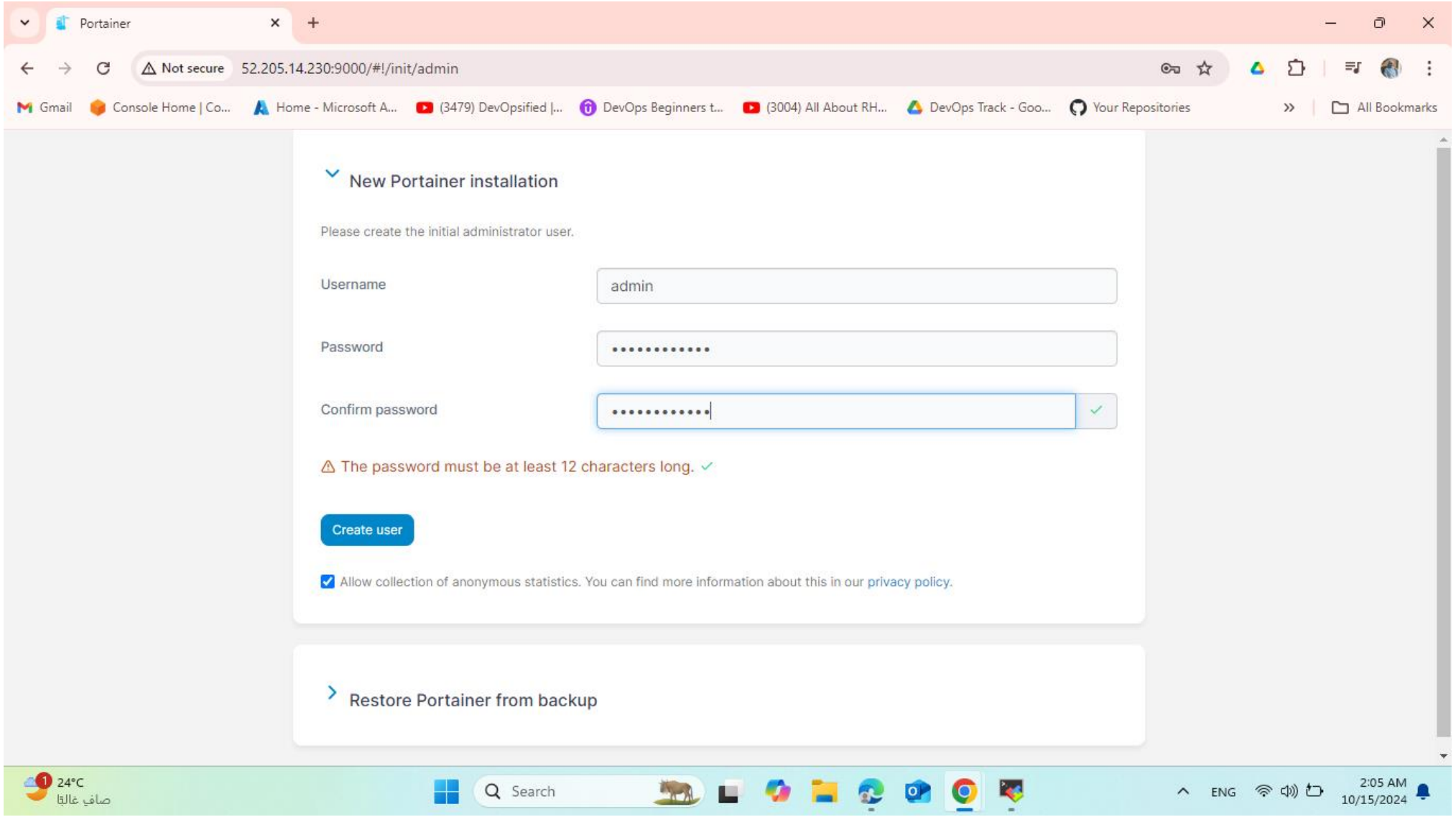
Part 02

Monitoring

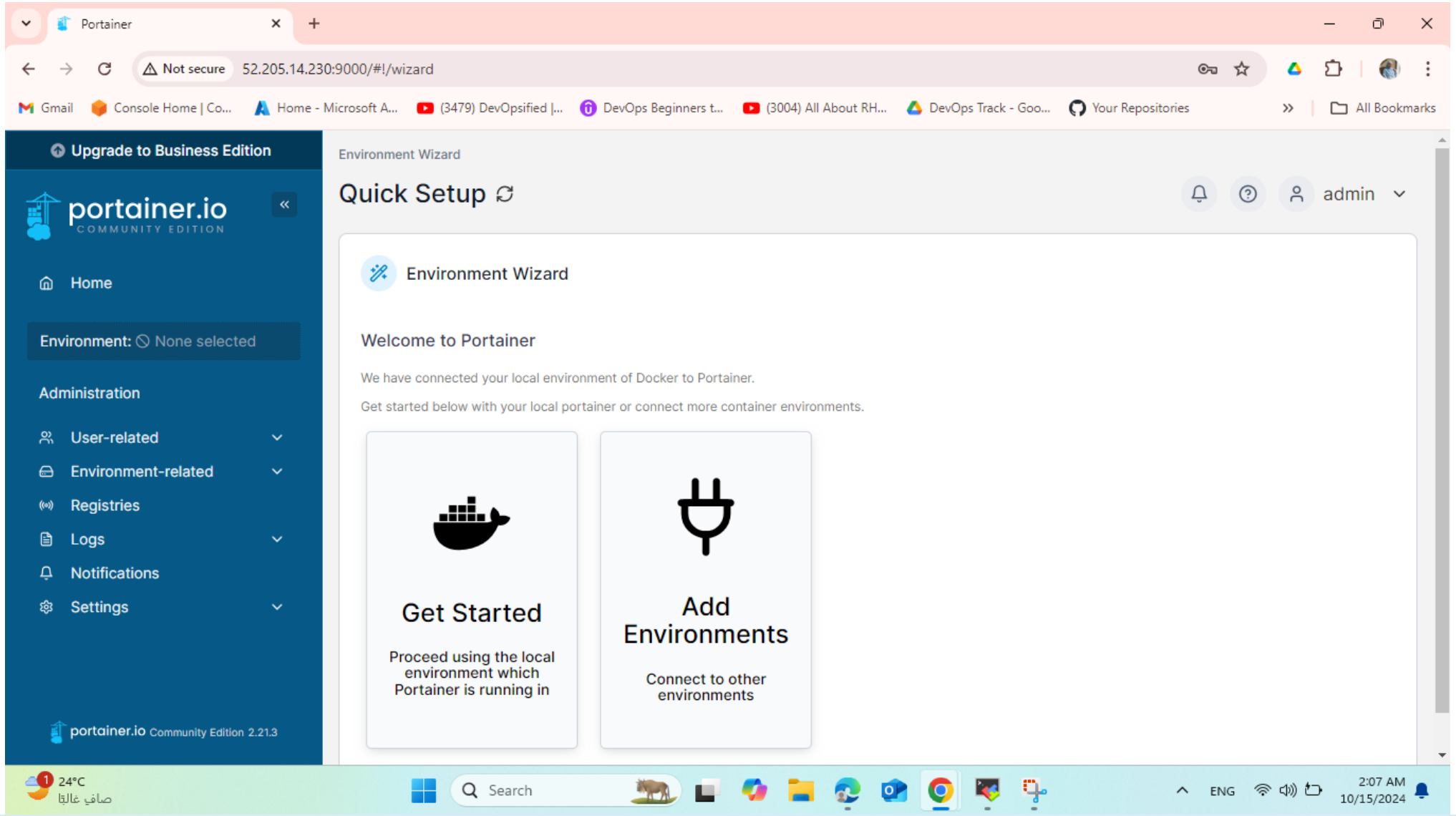
42- Add service in the docker-compose.yml file to run portainer



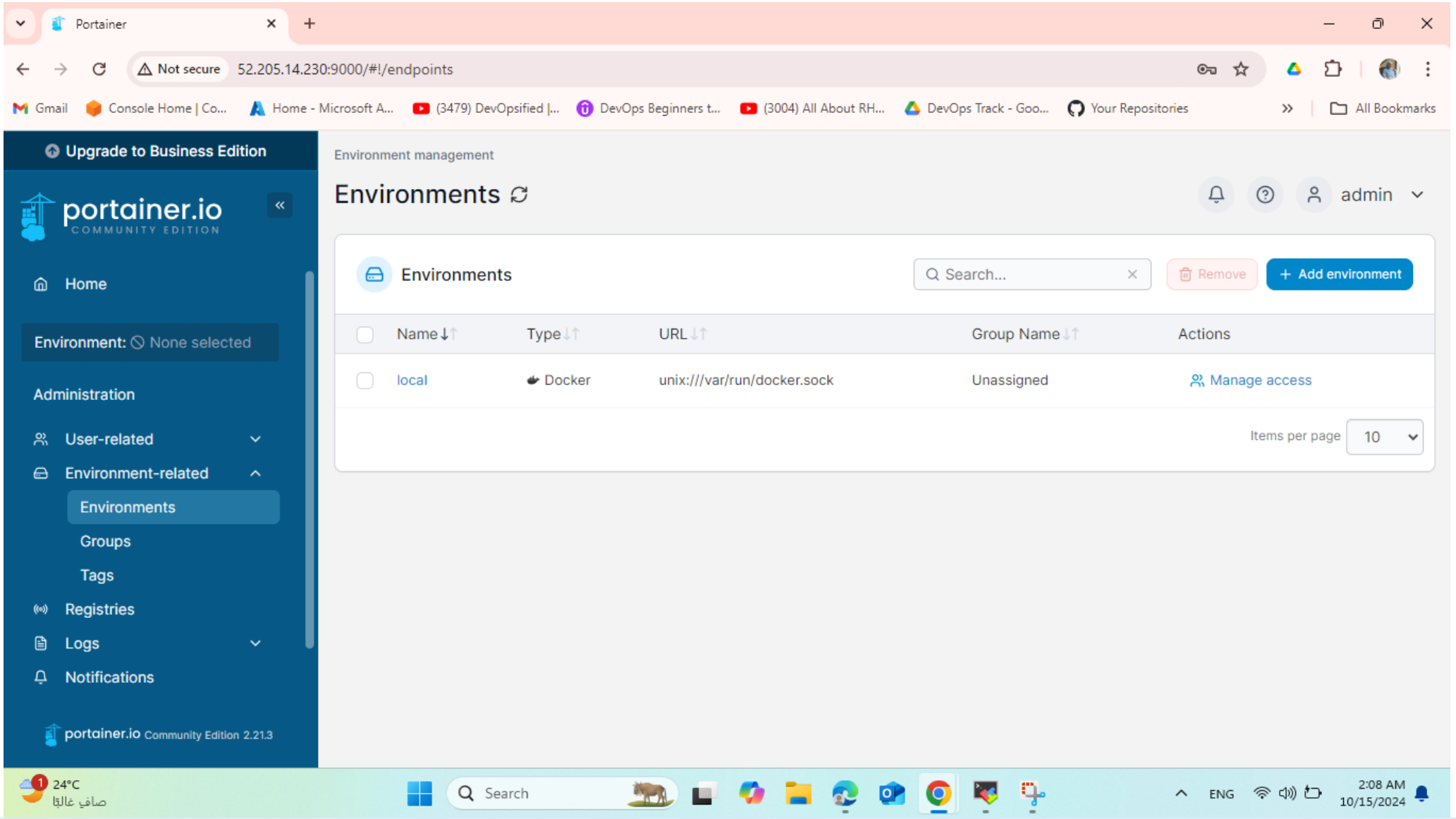
43- Portainer Successfully running on port 9000



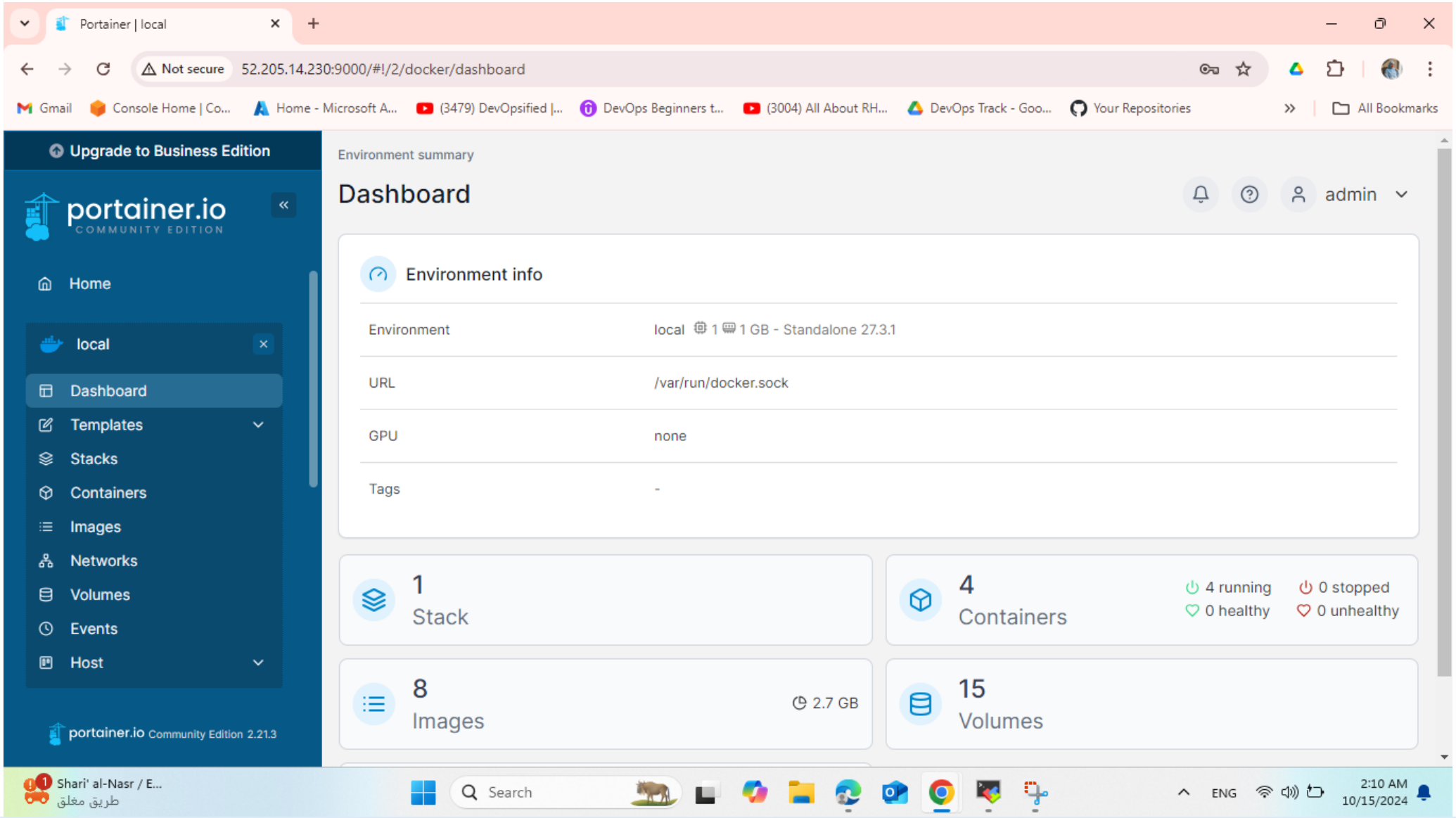
43- Portainer Successfully running on port 9000



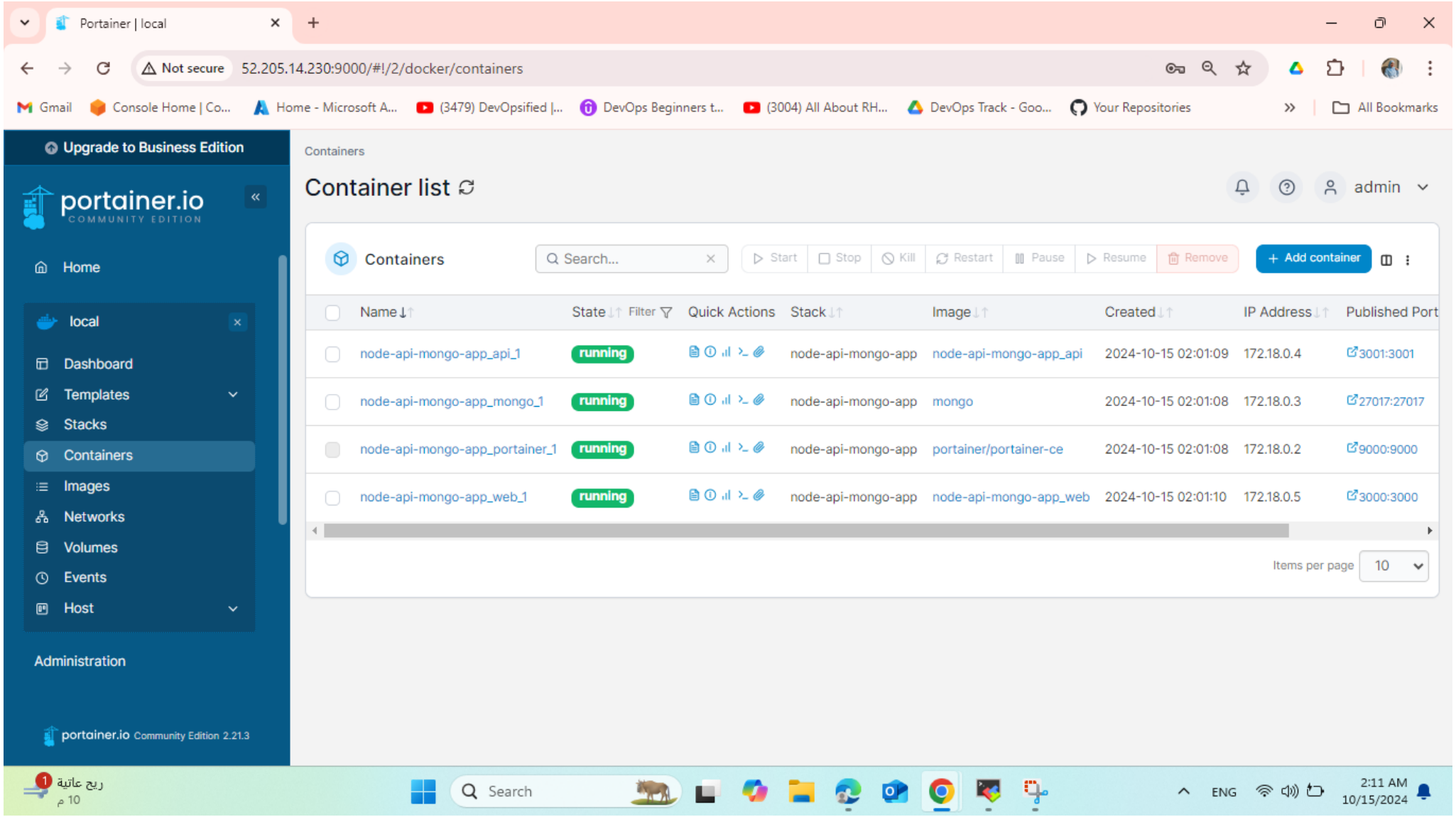
44- Docker in portainer



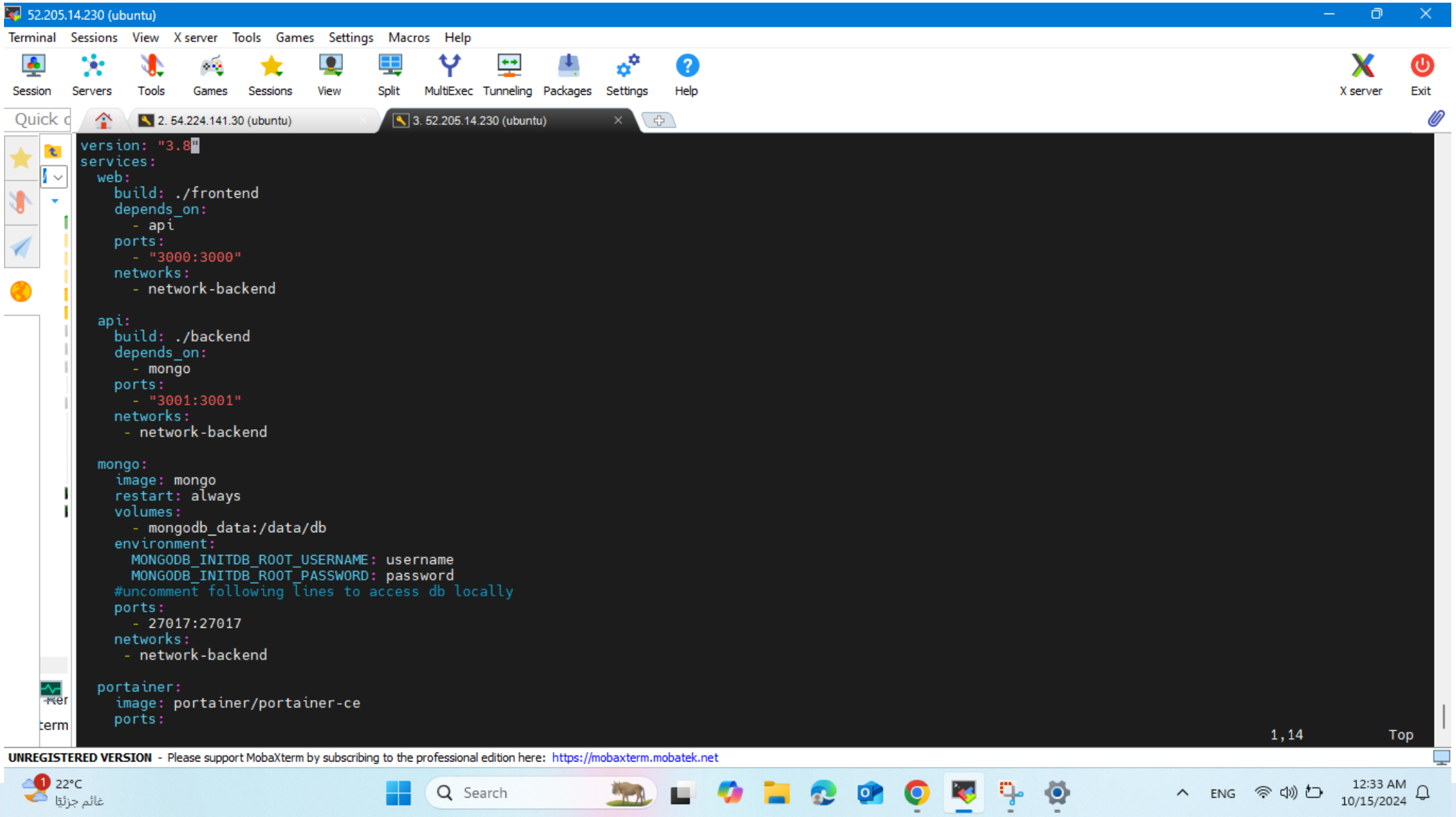
44- Docker in portainer



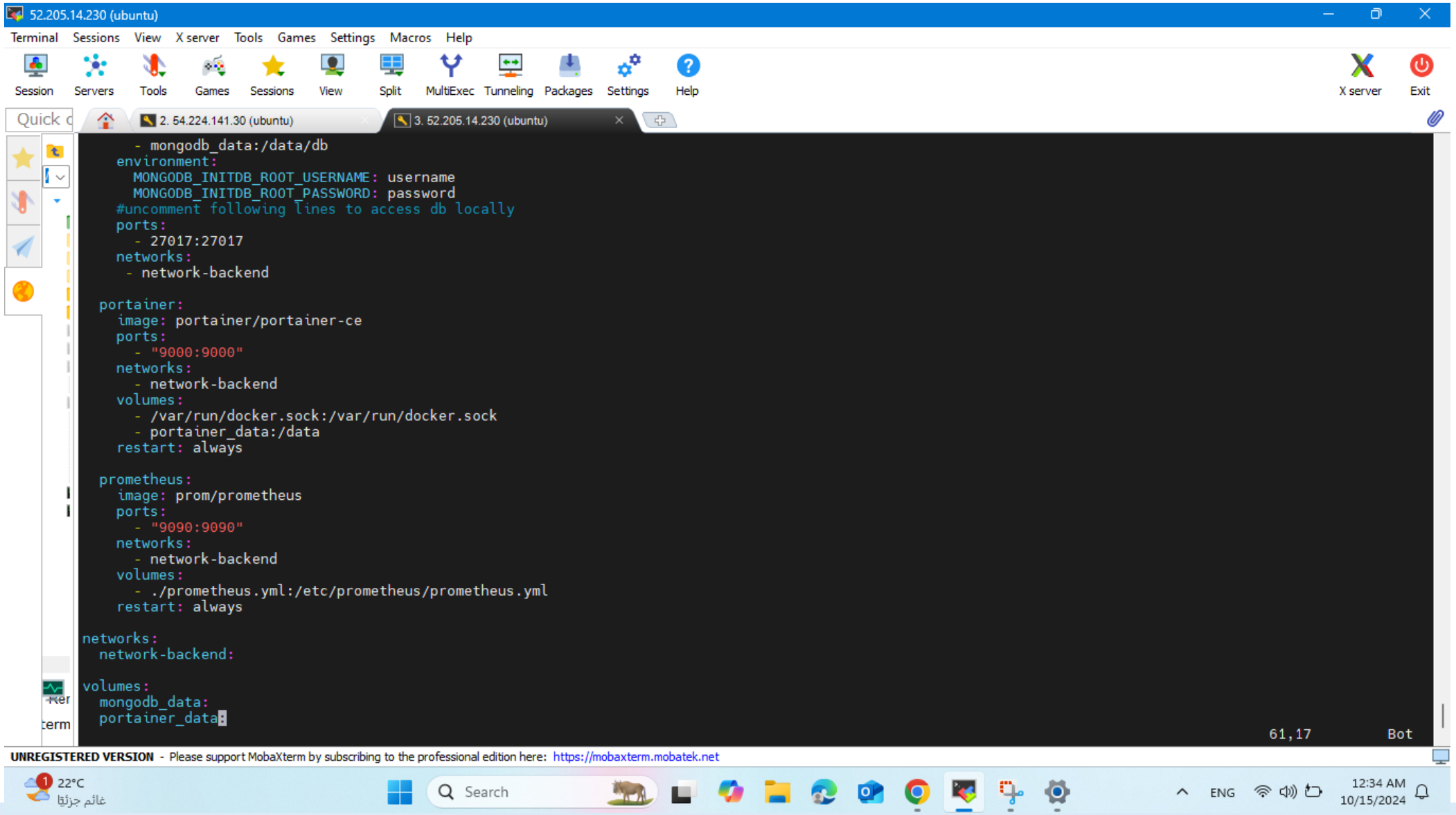
44- Docker containers in portainer



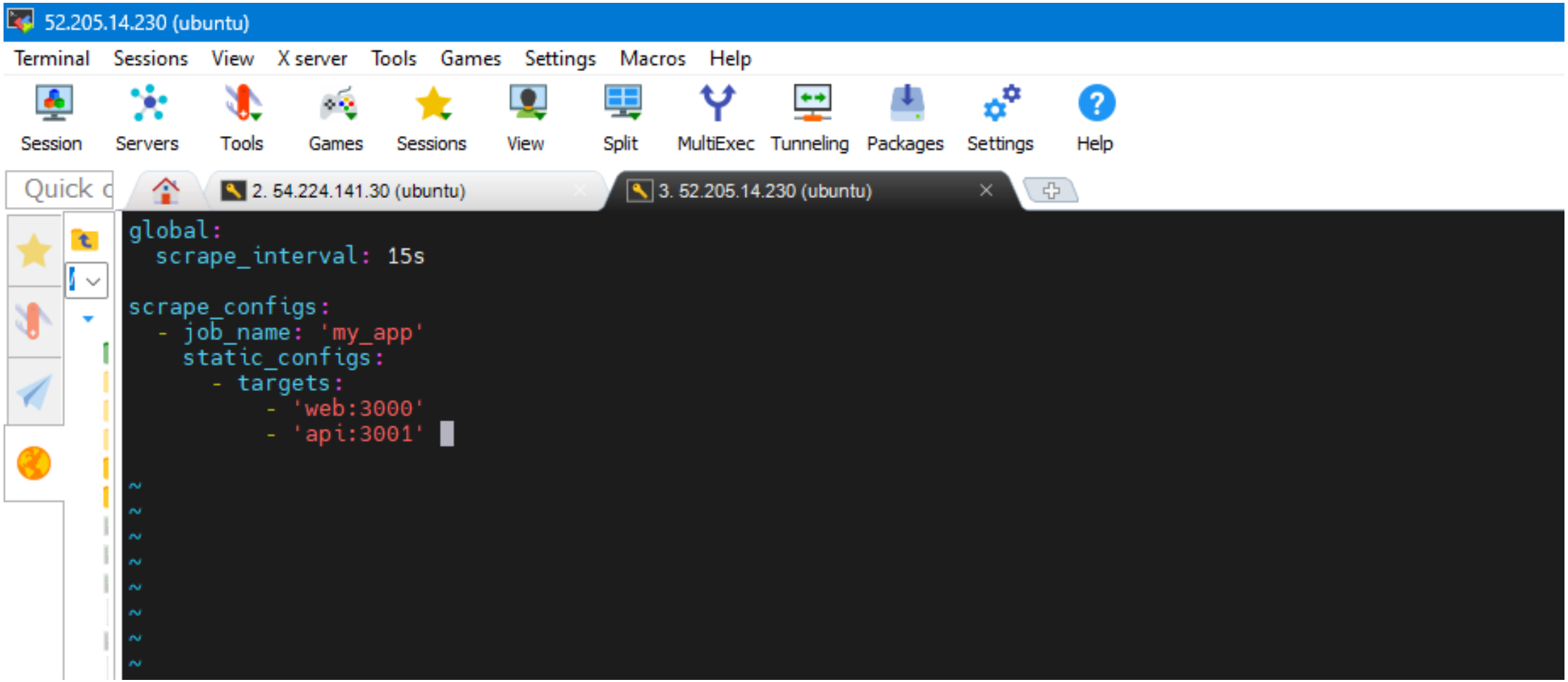
45- Adding service in the docker-compose.yml file to run portainer



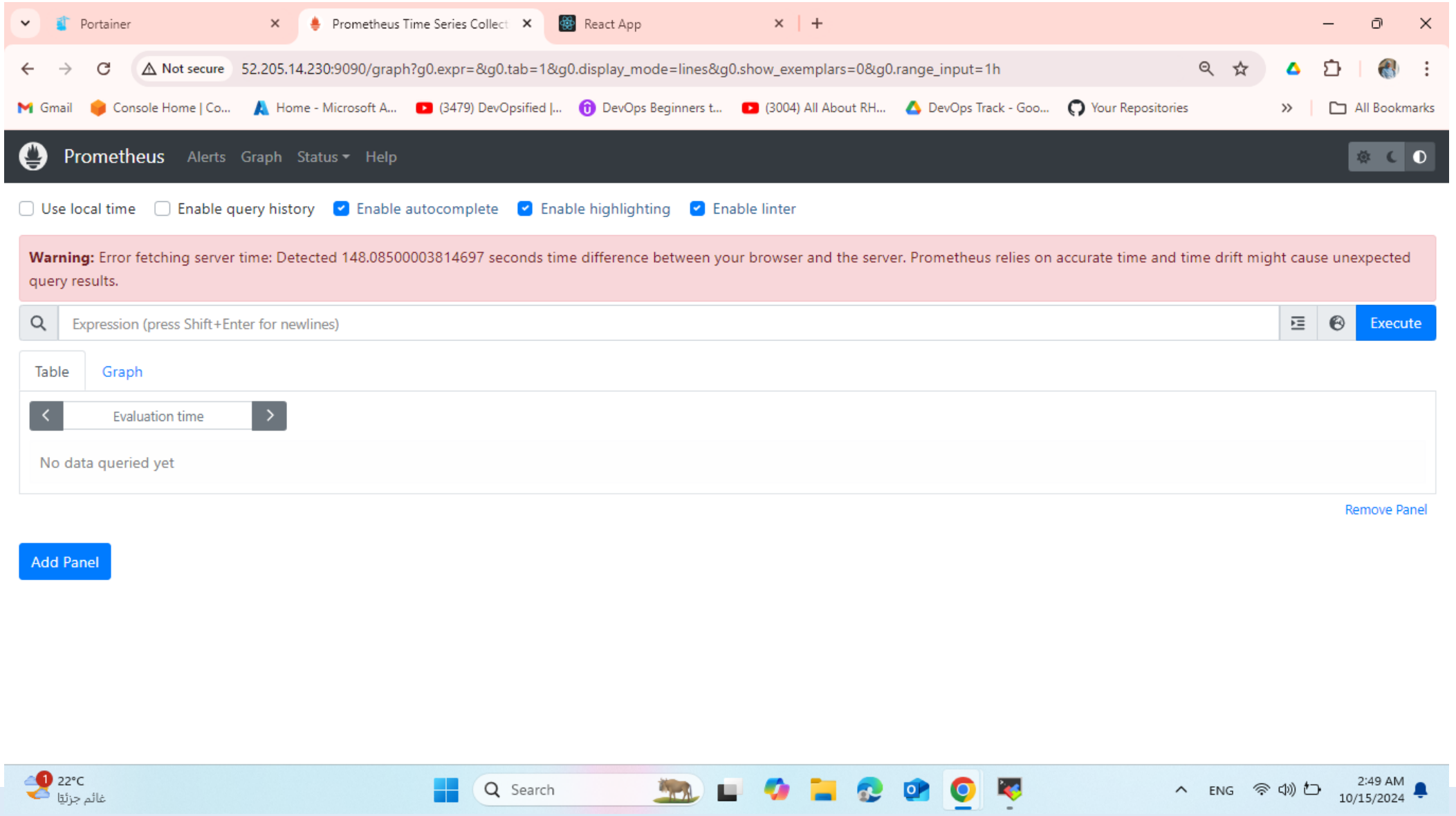
45- Adding service in the docker-compose.yml file to run portainer



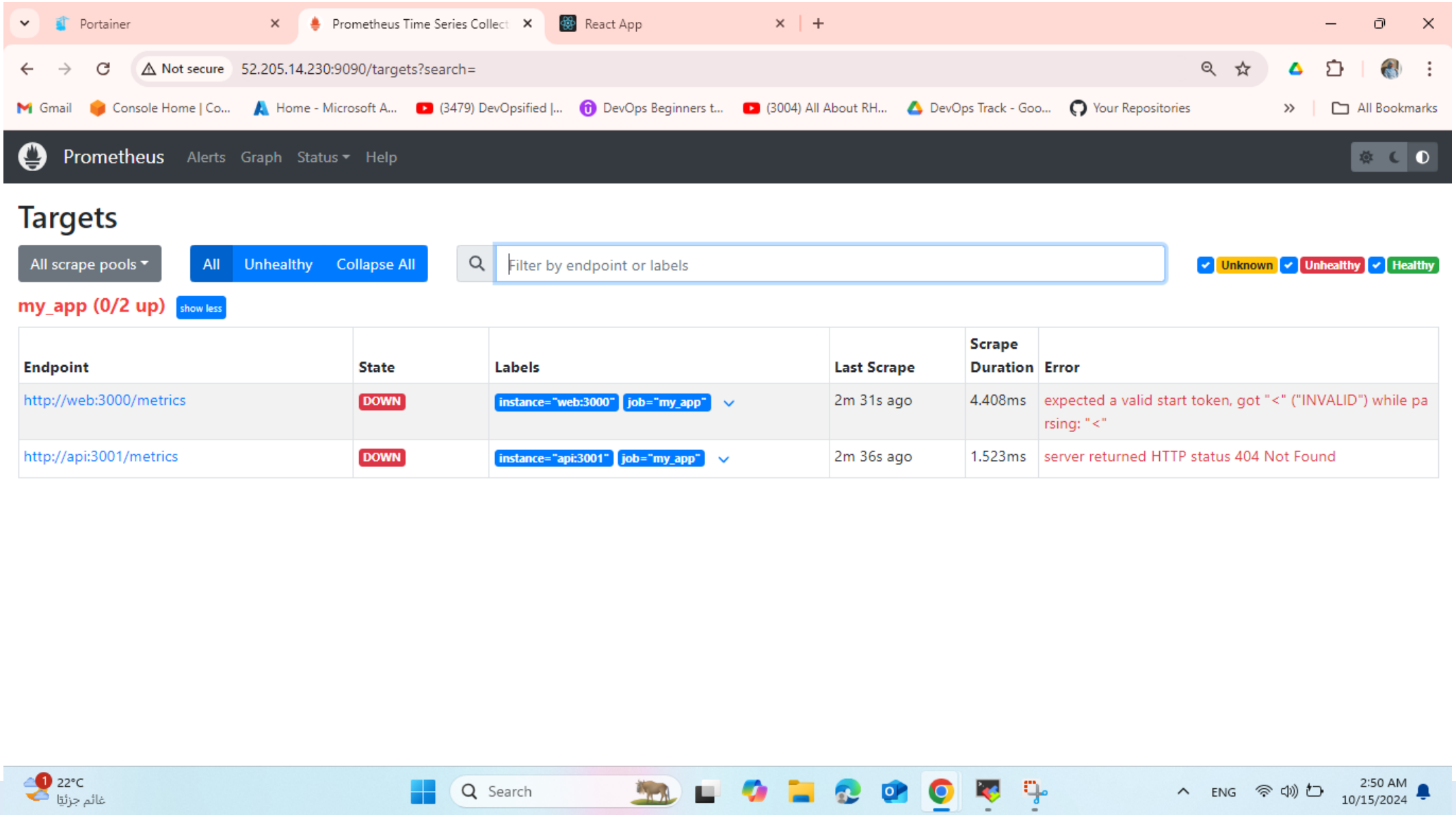
46- Prometheus.yml file with targets declaration



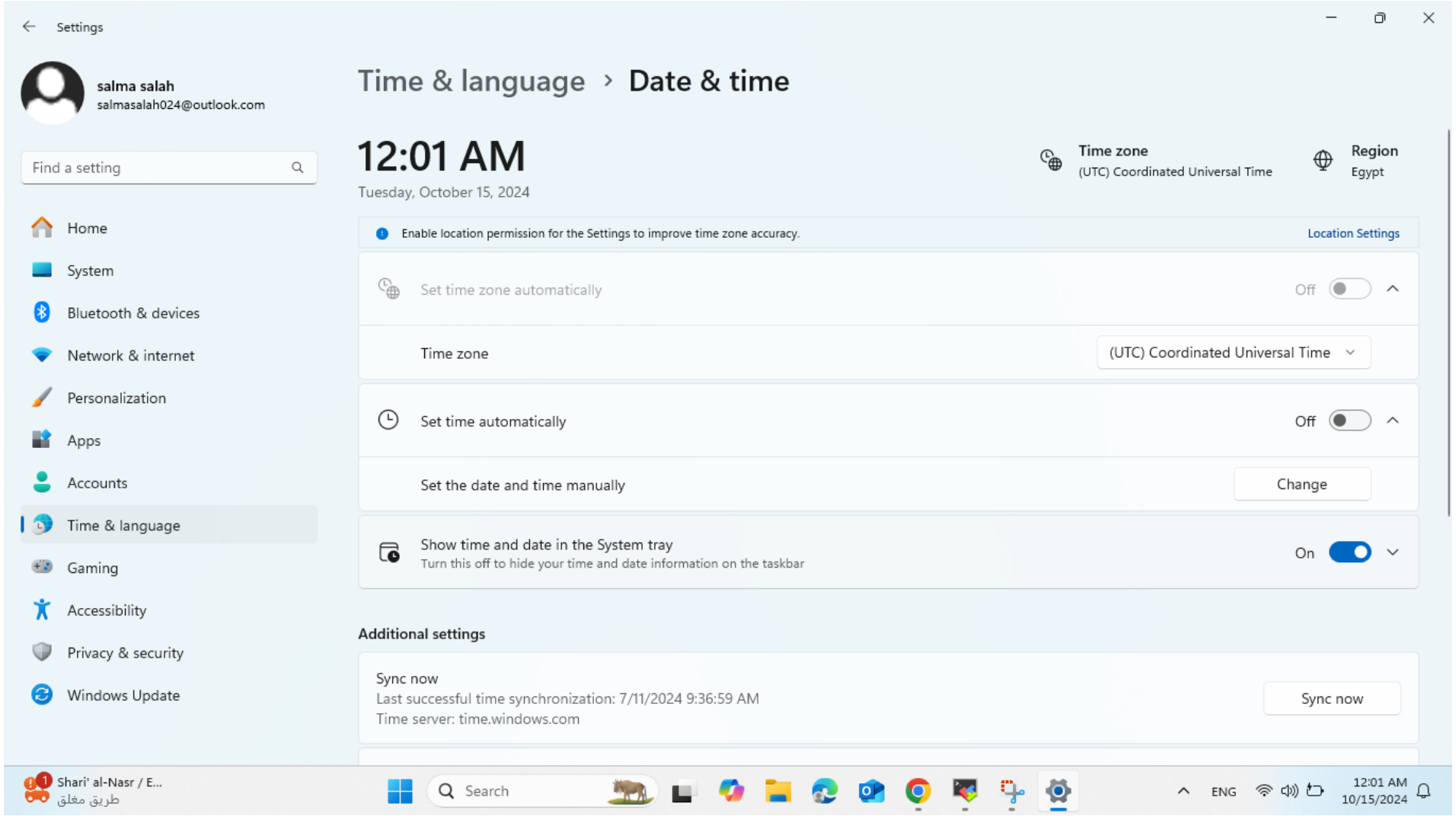
47- An error in time synchronization between the server and browser



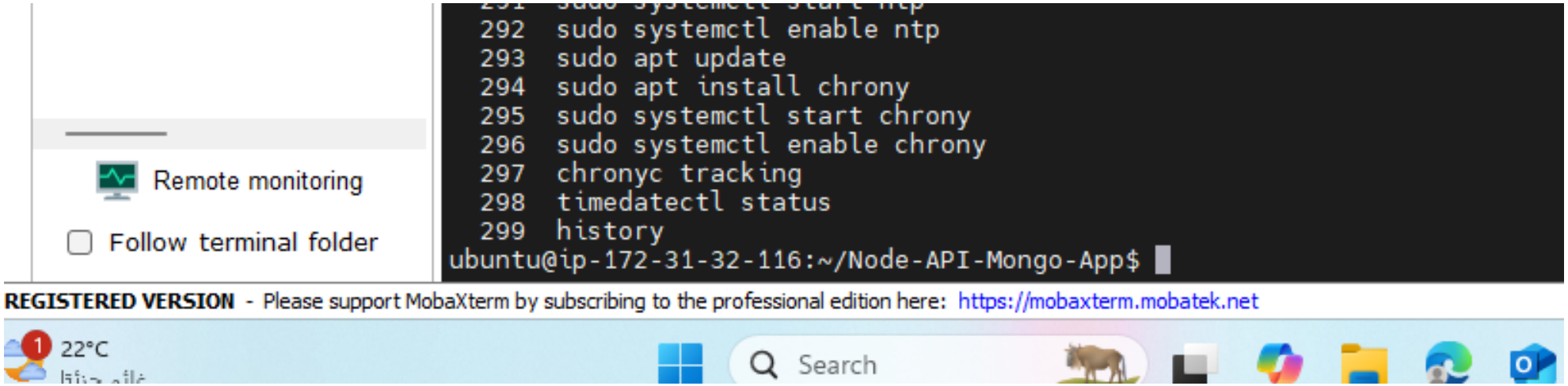
47- An error in time synchronization between the server and browser



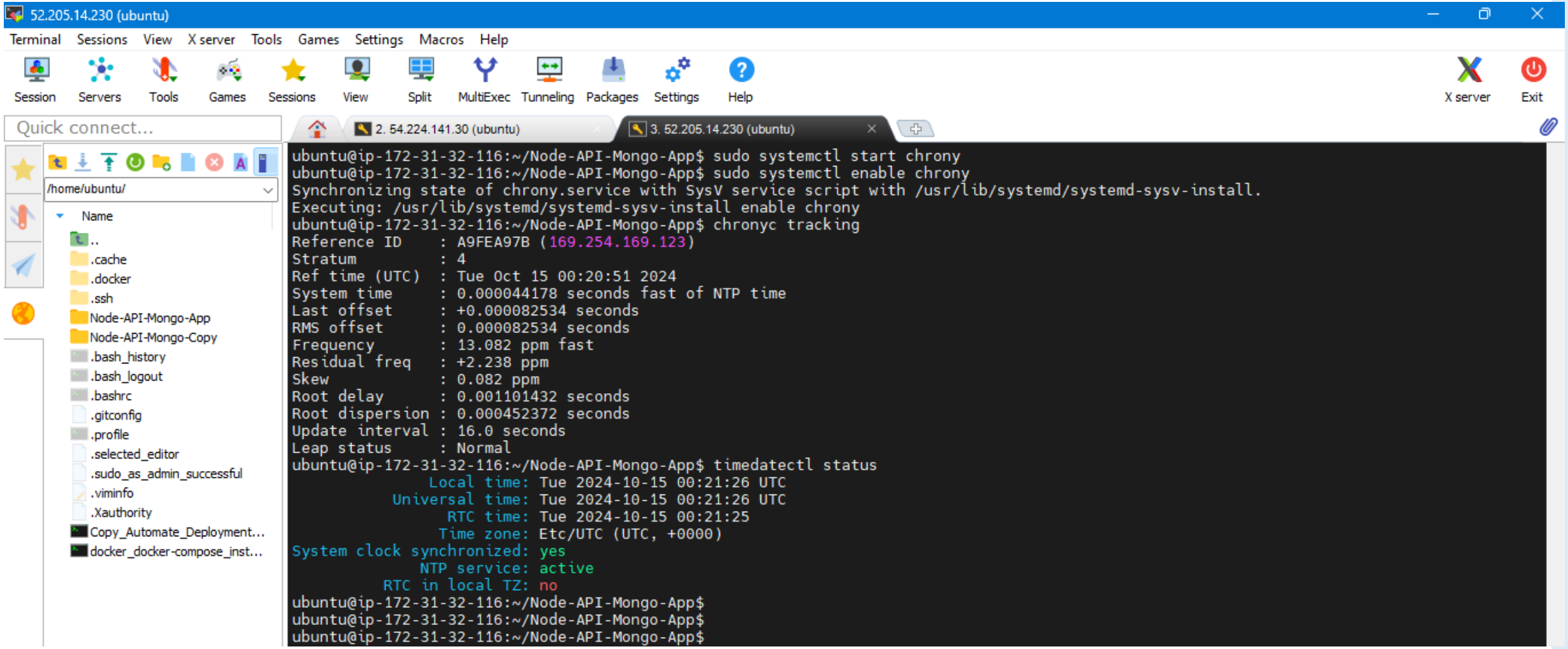
48- Changing system’s time to be UTC as the server



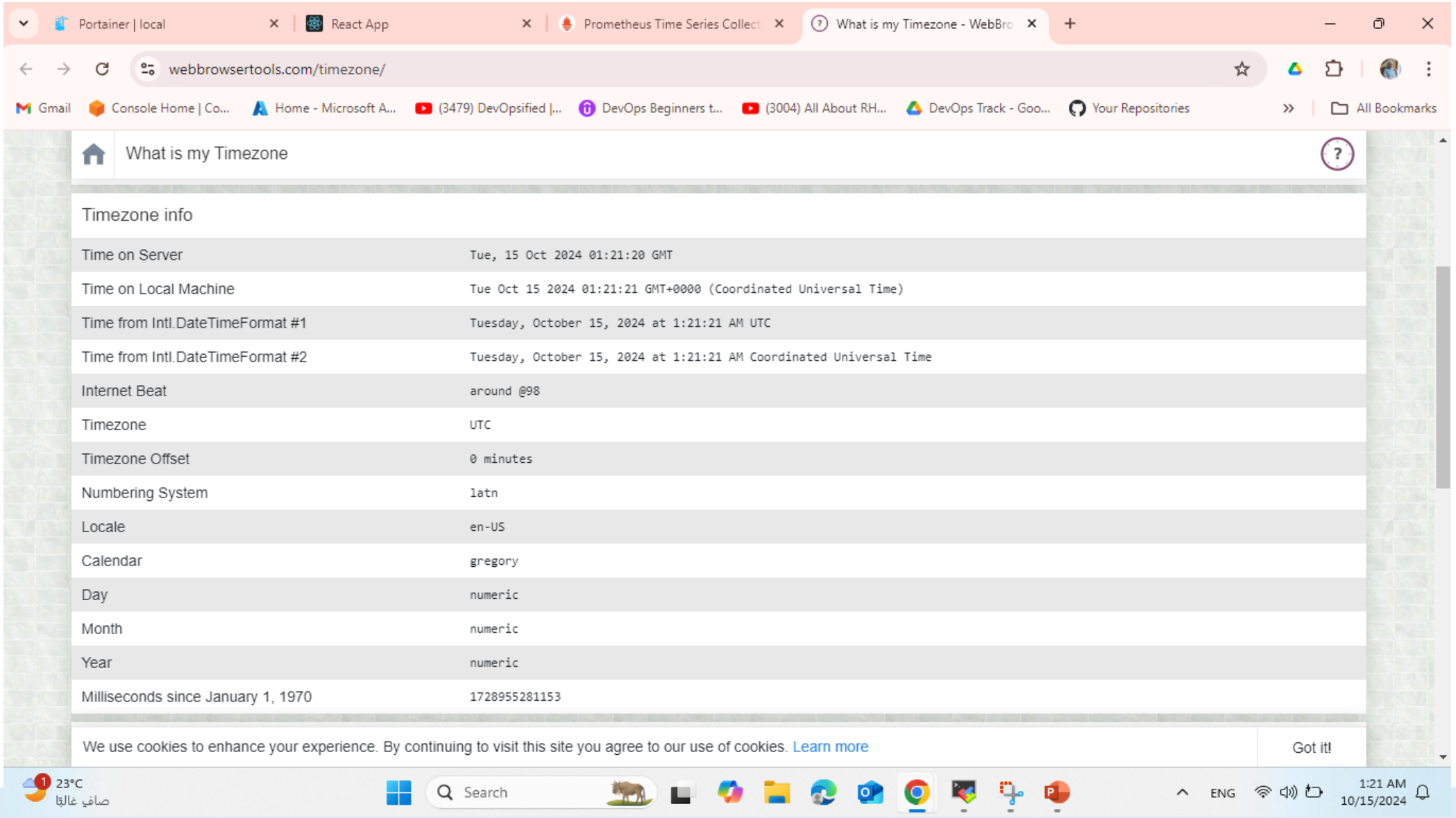
49- Installing chrony



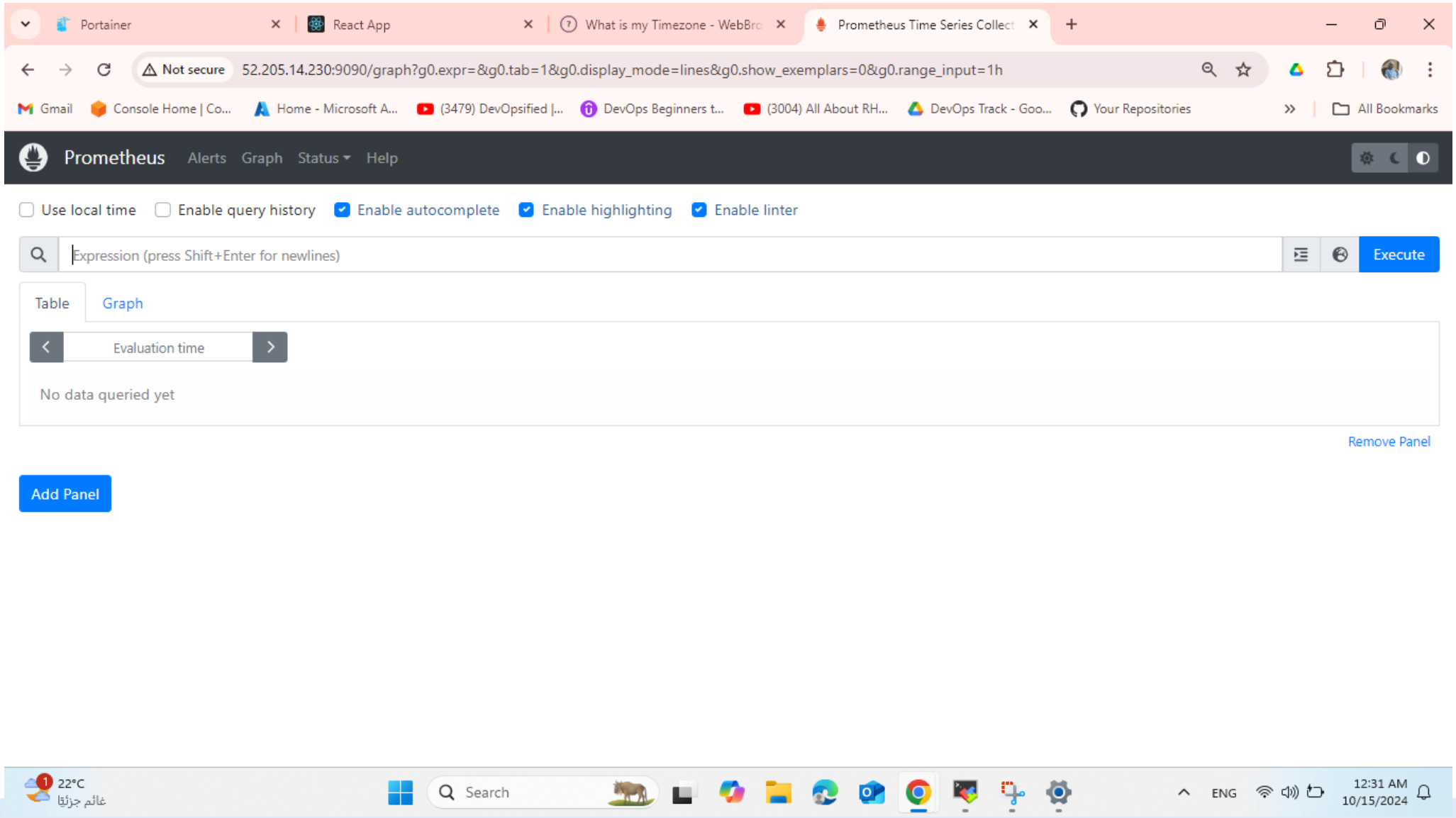
50- Checking synchronization



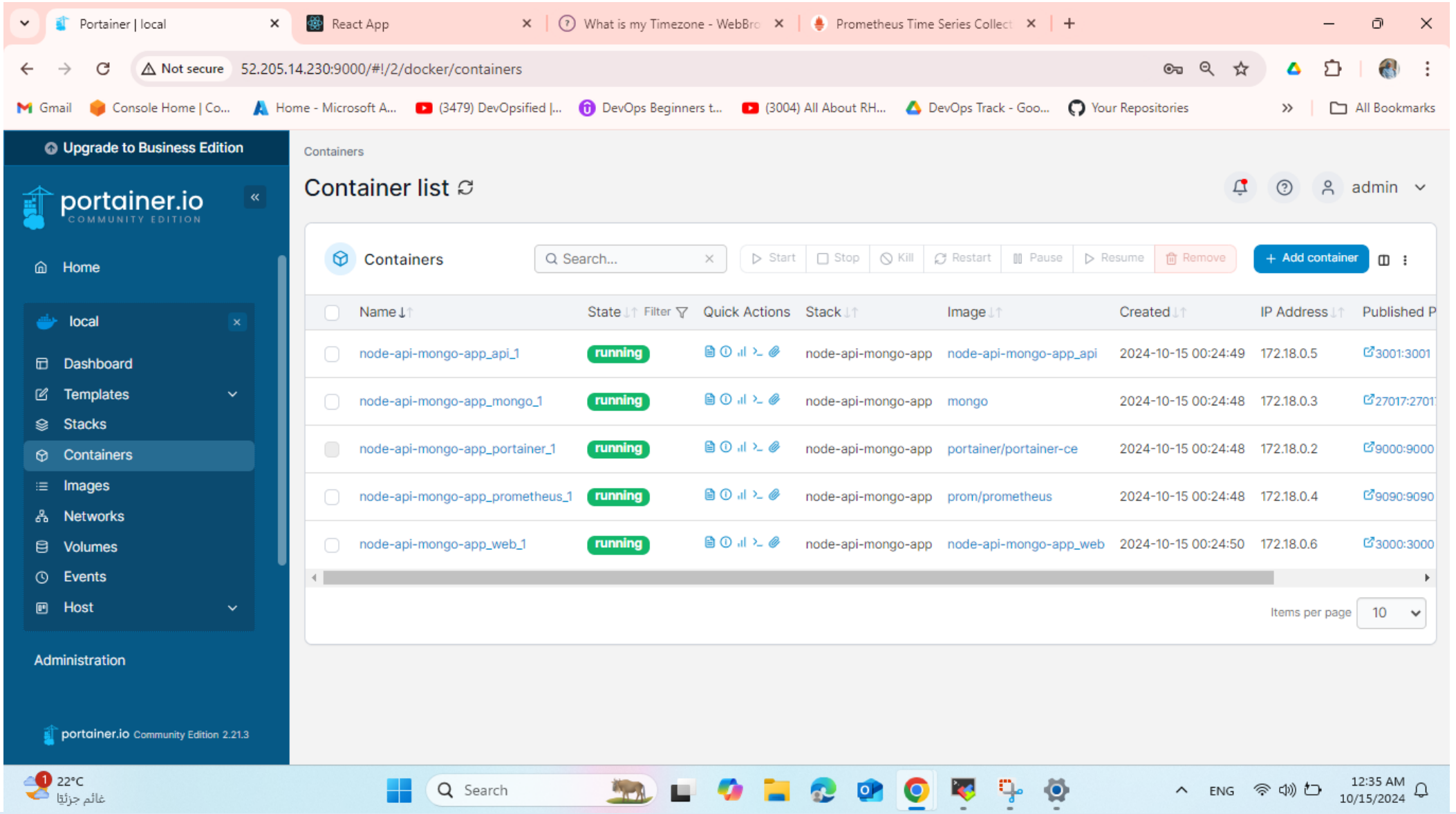
50- Checking synchronization



51- Prometheus Successfully running



51- Prometheus Successfully running



52- Next Steps

- Install (prom-client) to expose metrics in a format that Prometheus can scrape
- Set Up the Metrics Endpoint in the applications file
- Add service for Grafana to Visualize metrics