Technical Report

Student IDs and Names

|  |  |  |
| --- | --- | --- |
| Date of Change | Contributor | Summary of Change |
| 11/09/2023 | Manish Saily | Created and completed task 1 |
| 11/09/2023 | Edward Winston | Yml file for docker compose task 2 |
| 12/09/2023 | Manish Saily | Fixed docker-compose file and the project is running. |
| 16/09/2023 | Edward Winston | Created files for task 3 |

|  |  |
| --- | --- |
| Name | S number |
| Edward Winston | s5133176 |
| Manish Saily | S5236772 |

Table of Contents

[0.0 Overview 3](#_Toc143789055)

[1.0 Task 1 3](#_Toc143789056)

[1.1 Customize Images 3](#_Toc143789057)

[1.2 Run Individual Containers 3](#_Toc143789058)

[2.0 Task 2 3](#_Toc143789059)

[2.1 Docker Compose 3](#_Toc143789060)

[2.2 Run Containers with Docker Compose 4](#_Toc143789061)

[3.0 Task 3 4](#_Toc143789062)

[3.1 Create Pods 4](#_Toc143789063)

[3.2 Run Cluster with Minikube 4](#_Toc143789064)

[4.0 Benefits of Container Technologies (7623ICT students only) 4](#_Toc143789065)

[5.0 Self-Learning 4](#_Toc143789066)

[References 4](#_Toc143789067)

# Overview

Task 1 is about creating docker images and container that will allow us to build and run the full-stack pizzeria project. We are required to create containers without using docker-compose that:

* Run the front-end web application.
* Run the back end-database.
* Run the GUI for the backend-database.
* Run the nginx proxy to allow https connection.

Our group have successfully completed all requirements for task 1.

Task 2 is about creating a docker-compose file that will automate the creation of the project. The compose file controls all four containers and has internal network, volumes, and exposed ports. Our group successfully completed task 2.

Task 3

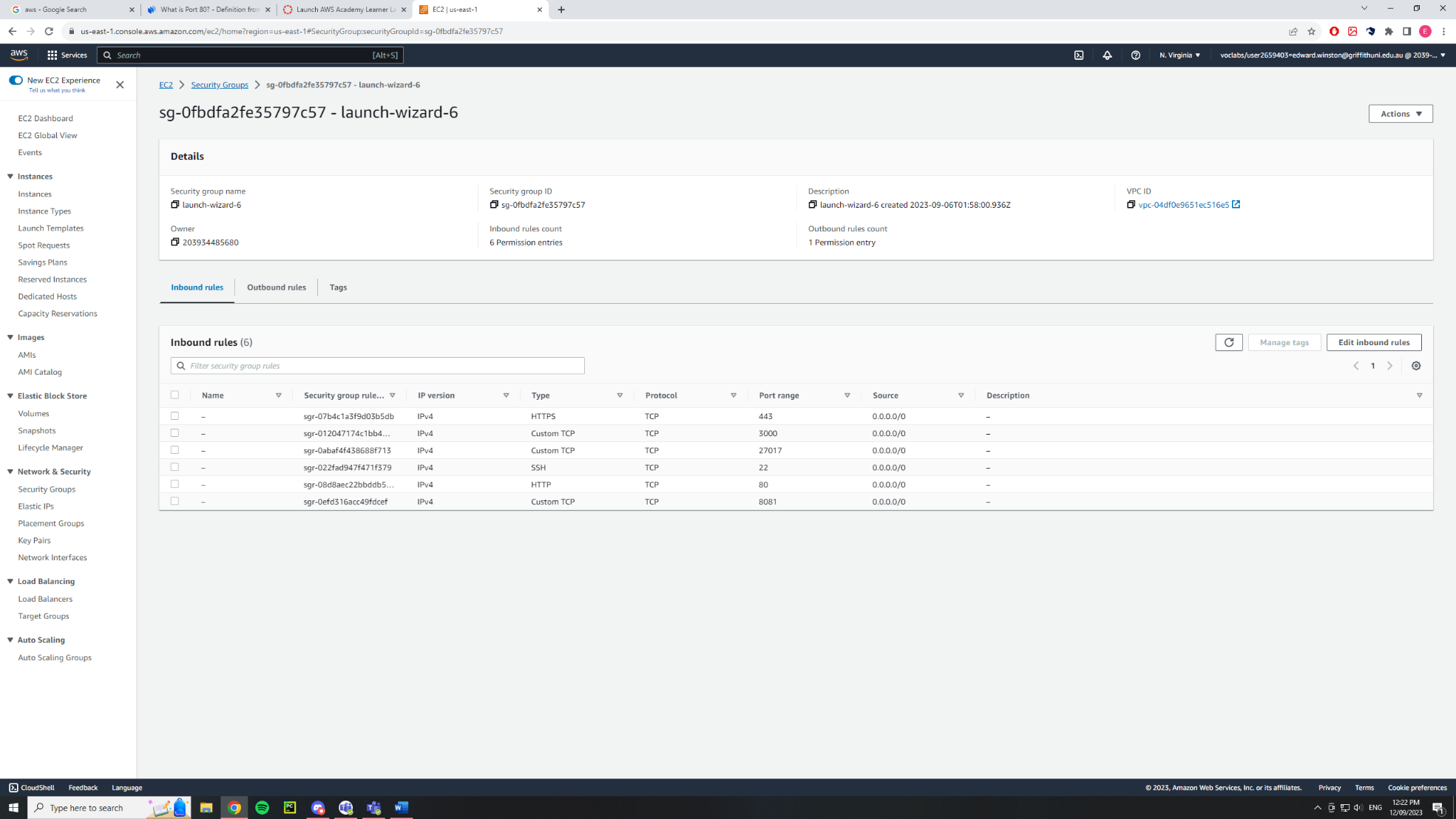
For task 3 we were able to create the ymal files for the four services and we were able to create files that could expose the ports for external access. We were able to create the pods but were unable to successfully complete the task.

# Task 1

Before we can start working on the pizzeria site we first have to set up an instance. This instance will be using ubuntu through the amazon web services (AWS).

**Security rules**

For the security rules we used the once posted below. We allowed port 22 for the SSH connections. Port 27017 was allowed for the use of mongodb. Port 3000 was allowed so that we could use mongo-express without needing root privileges. Port 80 was allowed as it is the default http port and port 443 was allowed as it is the default port for https. Port 8081 was allowed for mongodb and in doing so we could see if the data base was running.



Then we had to install docker for the instance using the following commands

* sudo apt update
* sudo apt install apt-transport-https ca-certificates curl software-properties-common
* curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
* echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
* sudo apt update
* apt-cache policy docker-ce
* sudo apt install docker-ce
* sudo systemctl status docker

Then to instal the files we were given and to unzip them we used the commands listed below.

* wget -O file.zip http://formal-analysis.com/tmp/pizzeria.zip
* sudo apt install unzip
* unzip file.zip
* cd pizzeria

## Customize Images

First, we created a docker file for the front end. This was done using the nano text editor with nano Dockerfile-frontend.

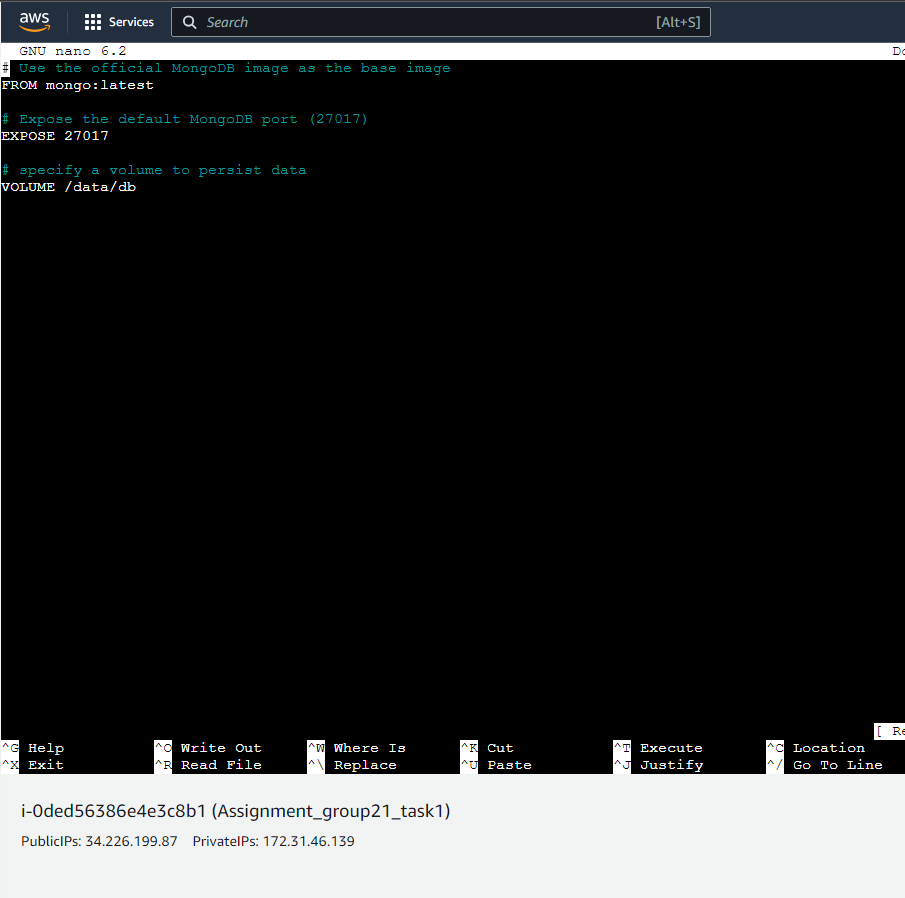
**nano Dockerfile-frontend**



This is the frontend docker file. It uses the base node image to initialise the build. The file then runs the mkdir command to create a directory called app. This is where all the files will be located for the project. The docker file then runs the copy command which copies all the package.json files from the local directory into the image directory app. The command wrkdir specifies that the working directory is app and to execute all following commands in the app directory. Next step of the docker file is the run npm. This command downloads all the dependencies outlined in the package.json files into the app directory. The next command in the docker file is copy. This command copies all remaining files from the local directory into the image directory app. We then expose the port 3000 in which the application will be running on. The next step is to outline all the required environment variables that will be used by server.js. The environment variables inside the docker container are used by server.js to access all the environments the application is running and needs to run. The first environment variables are having a session SECRET. The content of the SECRET environment is required for the frontend application to load the session to connect to the port the application is running on. The second environment variable that is required is the MONGODB\_URI. This environment variable allows for connection between the frontend and backend and saving information into the database. The name backend-database in the MONGODB\_URI is the name of the backend container inside the network that hosts mongodb. The SECRET and MONGODB\_URI are required environment variables for the application to work. The PORT variable simply is a visualization of the port number that will pop up when the application is running.

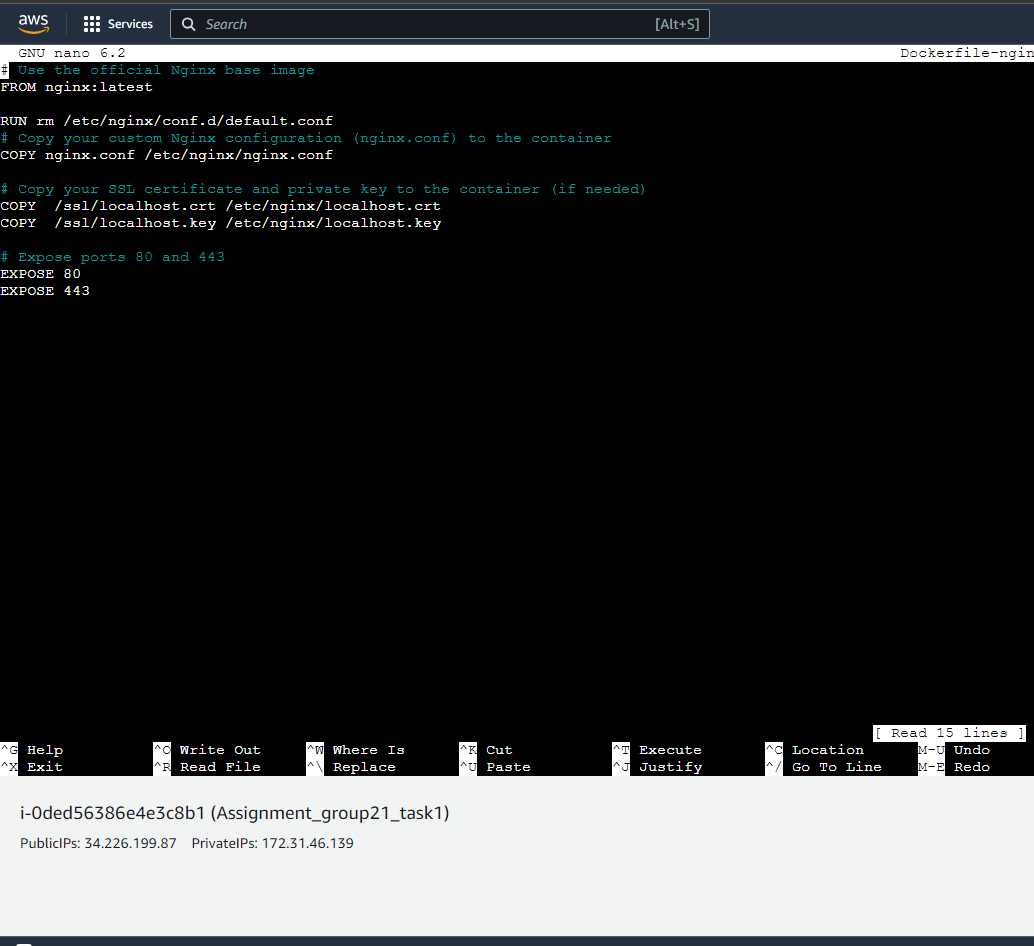
Then we created the backend data base using the command

**nano Dockerfile-backend**



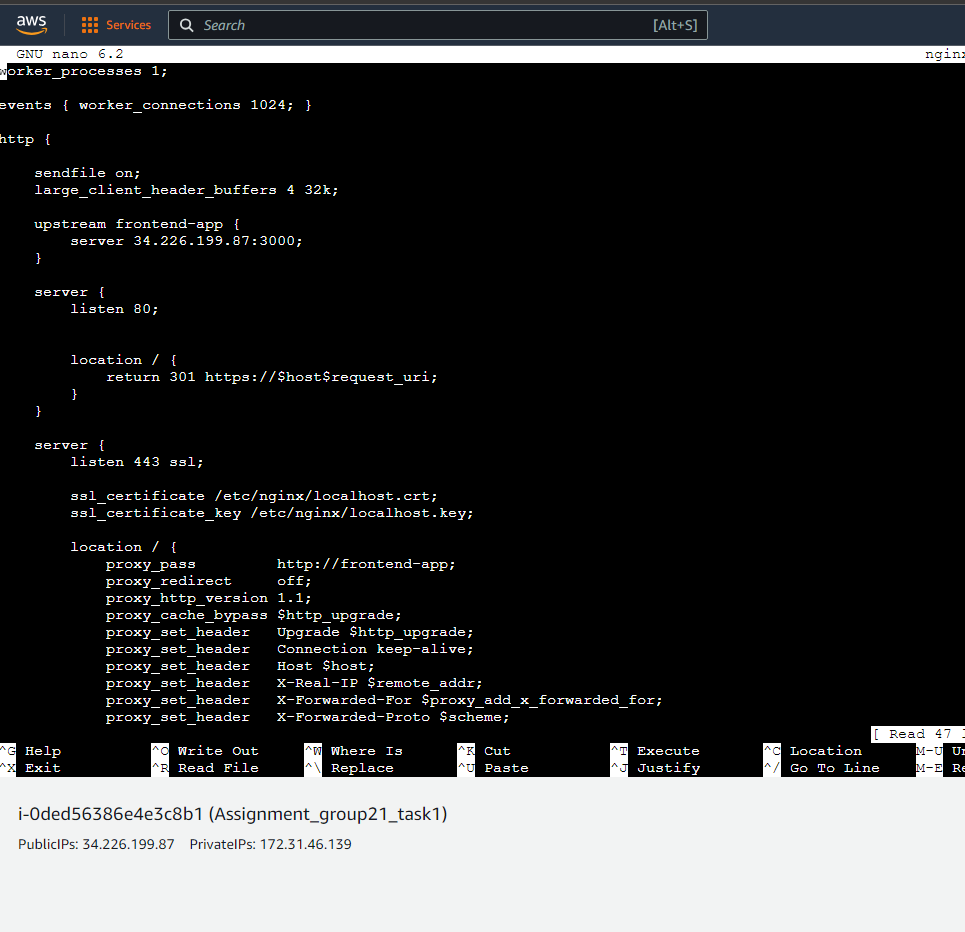
This dockerfile is a simple image deployment file that uses the base mongodb image and exposes the database on port 27017.

**Nano Dockerfile-nginx**



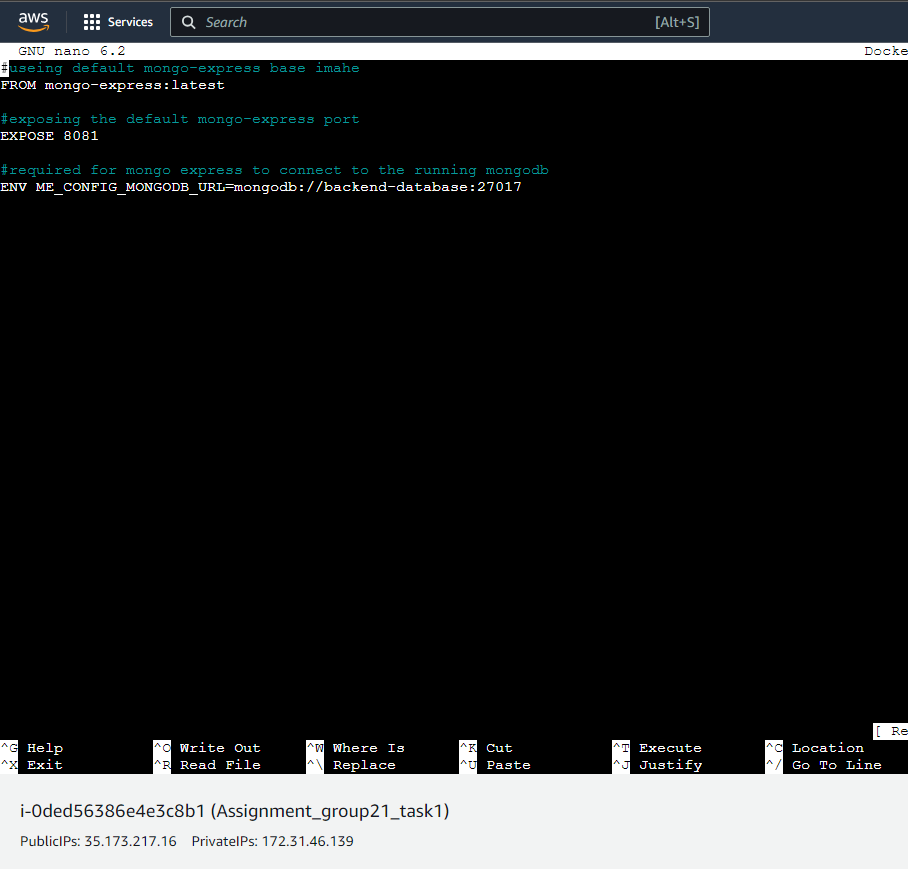
This is the docker file to build the nginx image. This file starts by using the nginx base image. We then execute the RUN command which will delete the default.conf file that has the commands for nginx proxy. We then replace the deleted .conf file with our custom nginx.conf file from our local directory using the copy command and placing it into the nginx image directory. Now we copy the ssl certificate and key from our local directory into the nginx image directory. We expose the ports that will be listening to the traffic which are port 80 and 443. We specify these two ports because port 80 will be accepting the traffic and forwarding it to https 443 for a secure connection.

**Nano ngin.conf**



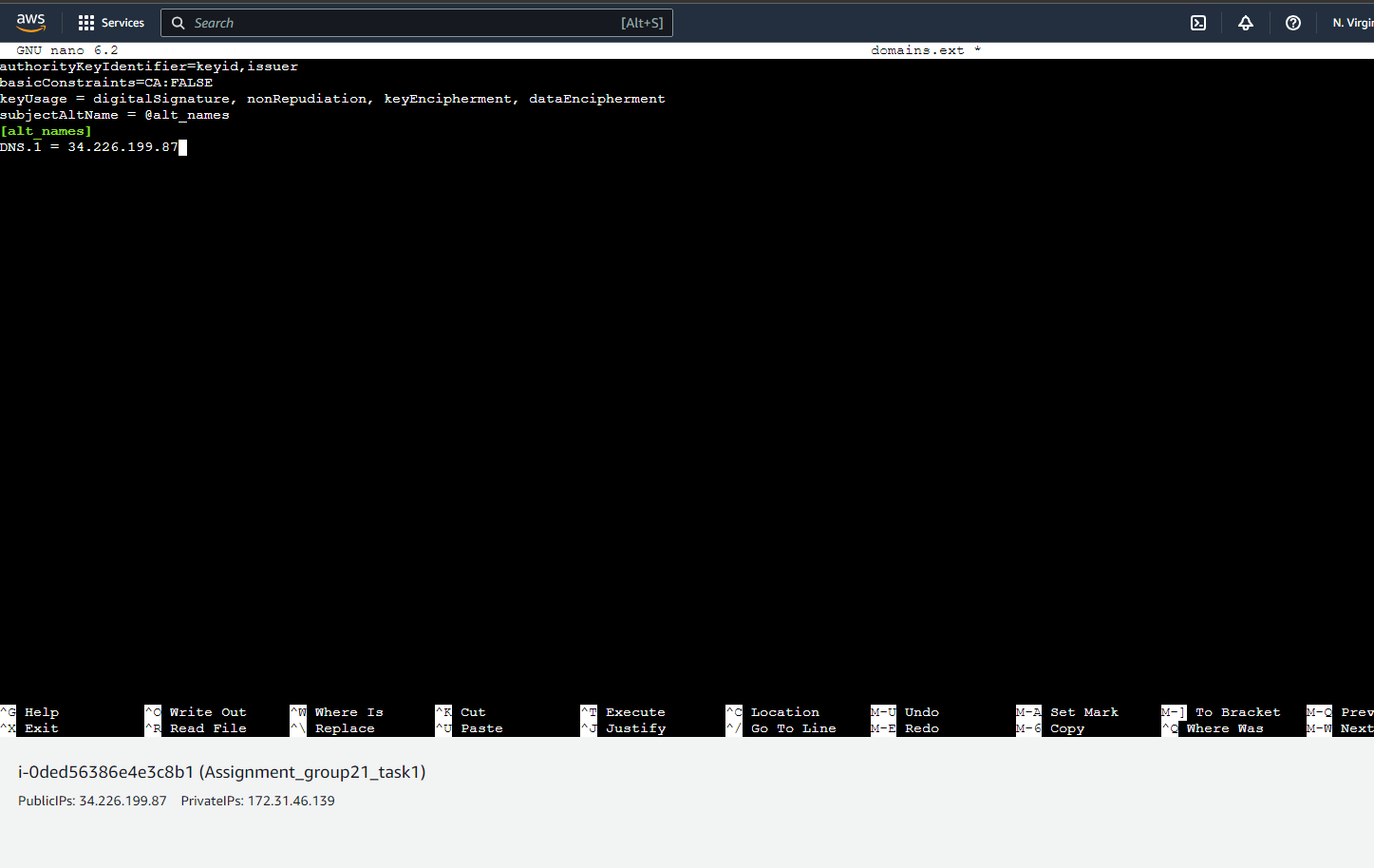
This the custom nginx.conf file that we replaced in our nginx image. To begin with, we setup the port and URL the frontend application will be on inside the upstream function. We name the upstream frontend-app this puts all the listed servers inside one group called frontend-app. The file will then specify the listing port 80. This is done so that all http traffic is first picked up by port 80 which is the default http port. The listening traffic URL is then renamed https. In the second server function we have listen 443. This port is the https connection port which will be used to create a secure https connection. The ssl certificate and key are then specified in which folder they are in inside the nginx image. The proxy\_pass command will then forward all the incoming traffic from our frontend-app and pass it through our ports 80 and 443 to create a reverse proxy and secure ssl connection.

**Nano Mongo-express dockerfile**



The environment variable is the same one used in the frontend URI. This allows mongo-express to connect to the database and provide the default user interface for the backend database.

**Nano domain.ext**



This is the domain.ext that was used to create the ssl certificate and key. Below are the commands that were used to create the ssl key and certificate within the pizzeria directory:

openssl req -x509 -nodes -new -sha256 -days 1024 -newkey rsa:2048 -keyout RootCA.key -out RootCA.pem -subj "/C=US/CN=My-Root-CA"

openssl x509 -outform pem -in RootCA.pem -out RootCA.crt

openssl req -new -nodes -newkey rsa:2048 -keyout localhost.key -out localhost.csr -subj "/C=US/ST=YourState/L=YourCity/O=Example-Certificates/CN=localhost.local"

openssl x509 -req -sha256 -days 1024 -in localhost.csr -CA RootCA.pem -CAkey RootCA.key -CAcreateserial -extfile domains.ext -out localhost.crt

mkdir ssl

mv localhost.\* ssl/

mv RootCA.\* ssl/

**docker network create pizzeria-project**

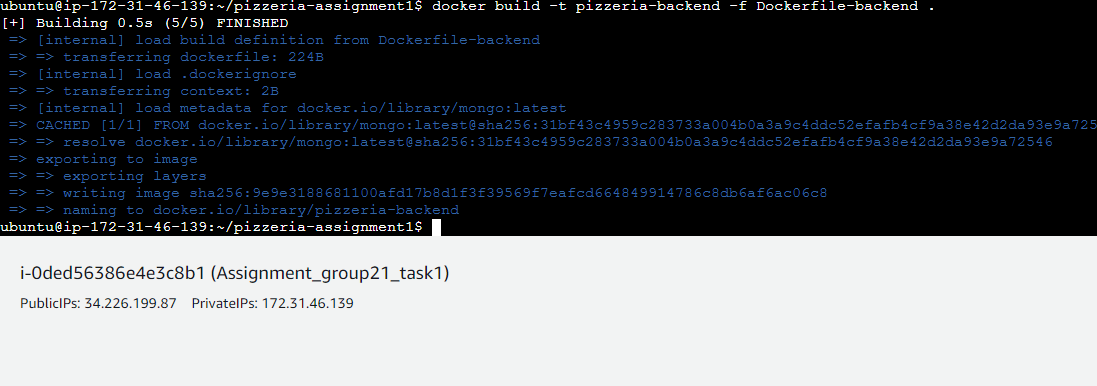
A screenshot of a computer

Description automatically generated

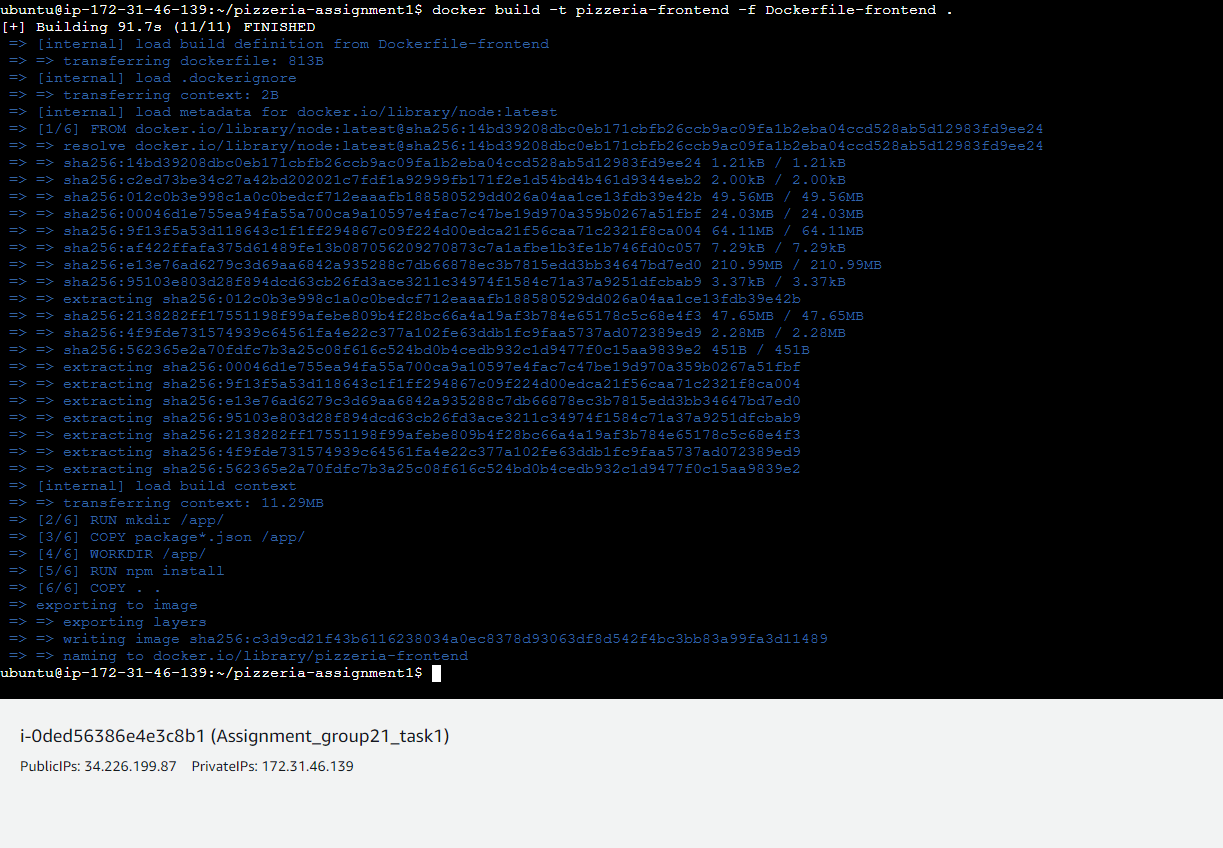
This creates a network called pizzeria-project. The network will host all the containers for the project that will allow for communication between the containers. As container run in isolation running all of the containers inside the network allows them to communicate with each other hence why in the mongodb URI the database name is the name of the container.

Next, we needed to build all containers and run the containers.

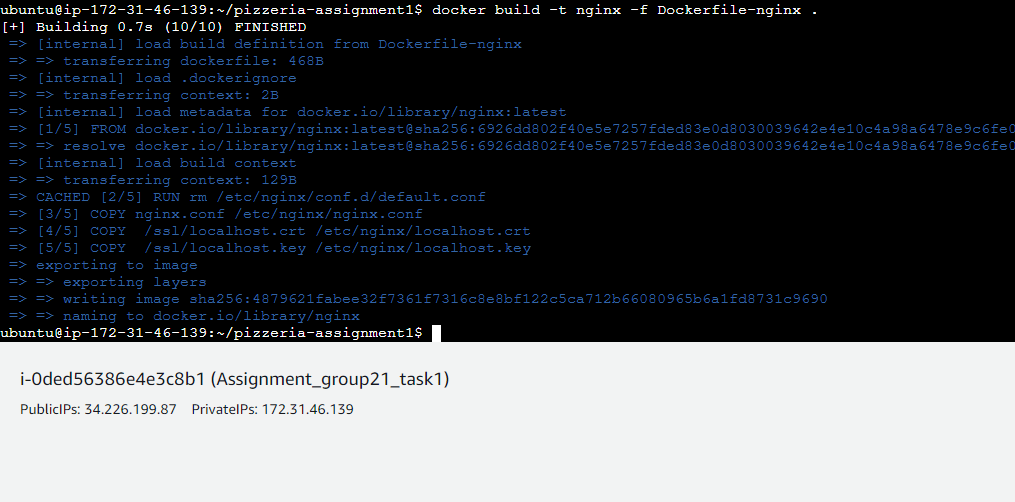
docker build -t pizzeria-backend -f Dockerfile-backend .



docker build -t pizzeria-frontend -f Dockerfile-frontend .



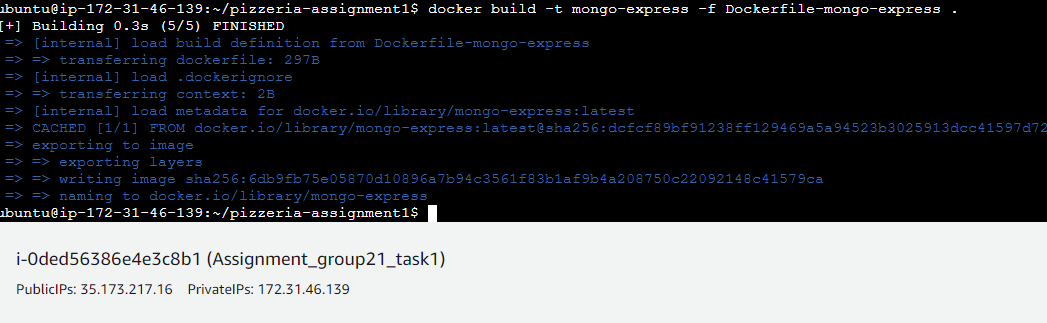
Docker build –t nginx –f Dockerfile-nginx .



This command built a docker container called pizzeria-frontend and included the docker file called Docker-frontend.

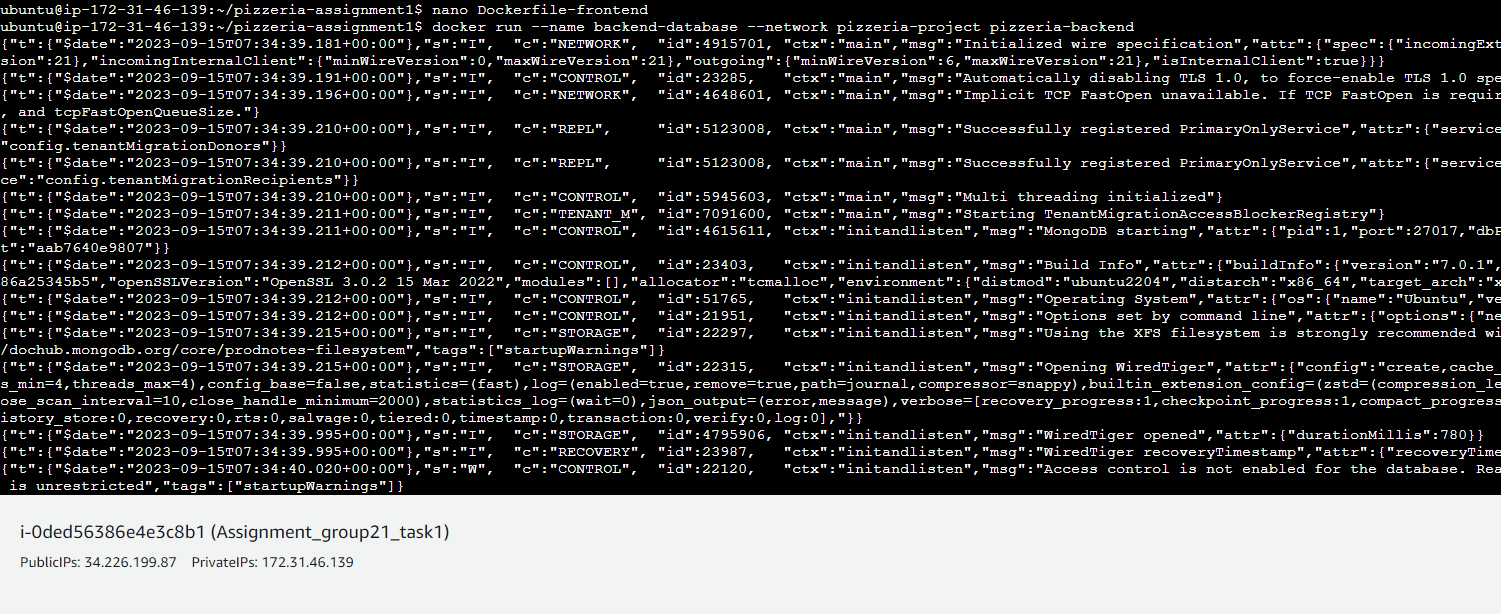
Docker pull mongo-express

Docker build -t mongo-express –f Dockerfile-mongo-express .

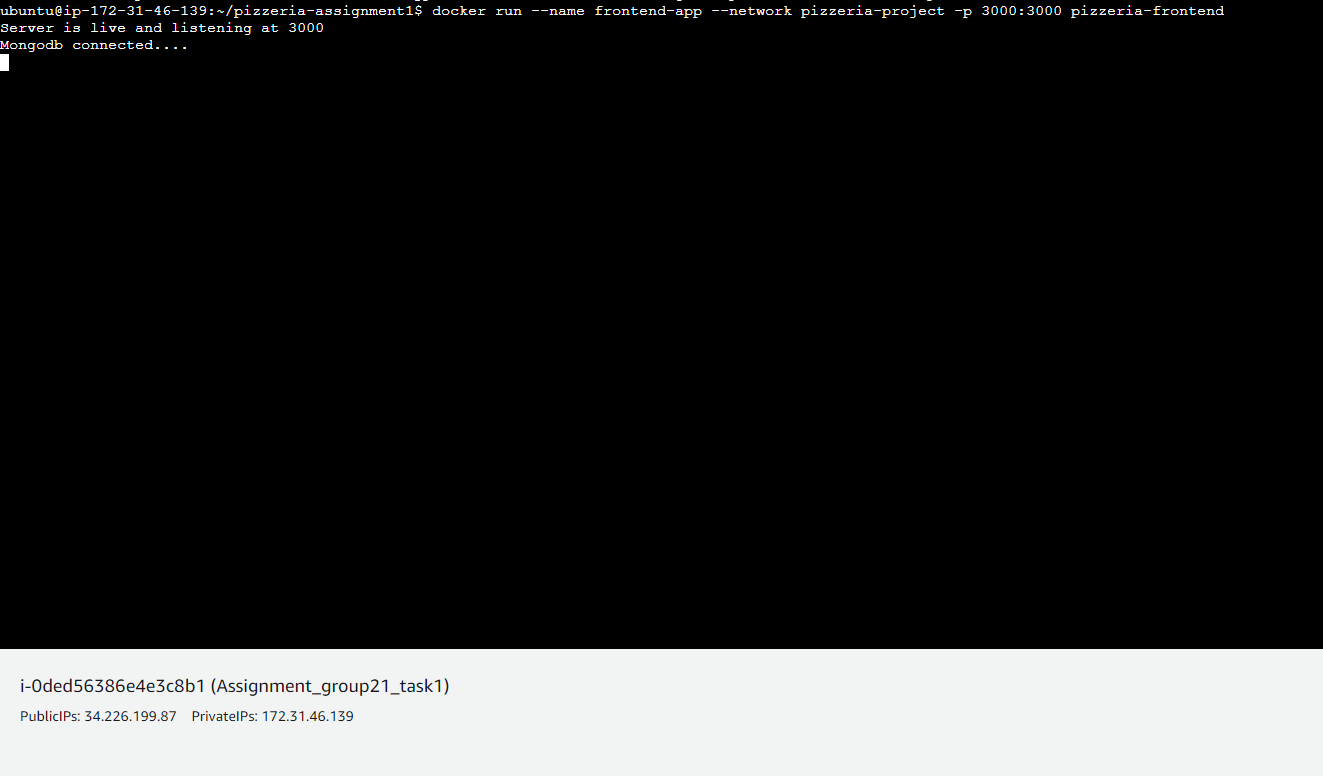


We pull the mongo-express files using the above command and build the mongo-express dockerfile. We use the environment variables in the dockerfile to connect to the mongodb.

docker run --name backend-database --network pizzeria-project pizzeria-backend



docker run --name front-end-app --network pizzeria-project -p 3000:3000 pizzeria-frontend



We run the front-end image inside the pizzeria-project network and use port 3000 to listen to all the traffic. Running the front-end image shows the server port the application is listening to and a successful mongodb connection.

docker run --name backend-db-gui --network pizzeria-project -p 8081:8081 mongo-express

A black and white screen with many small colored lines

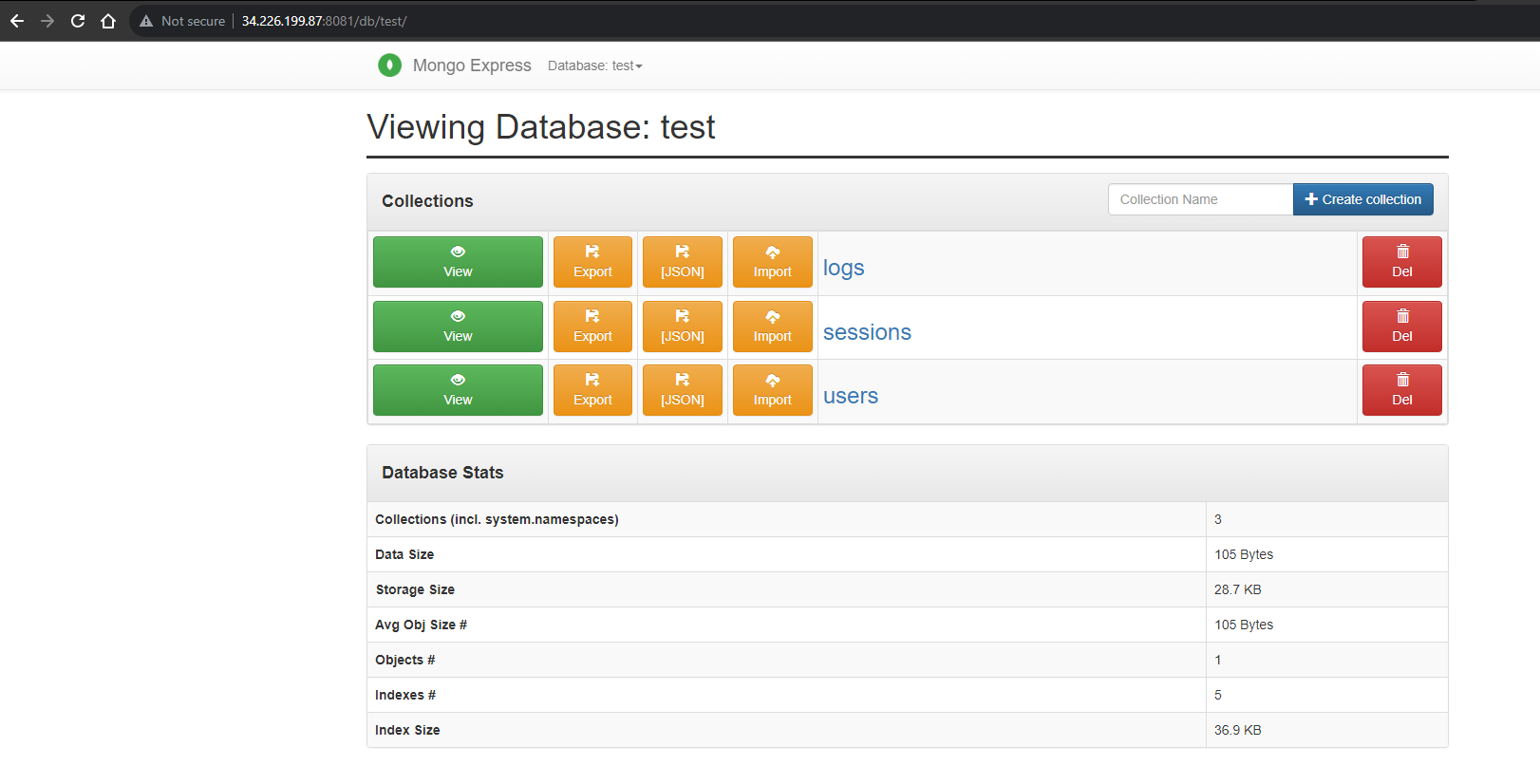
Description automatically generated with medium confidence

Running the mongo-express image will allow connection to the database. The ME\_CONFIG\_MONGODB\_URL is the same URI that was used in the frontend environment. mongodb://backend-database:27017 specifies the container name and the port the container is listening on. We also use port 8081 for mongo-express for the backend user interface.

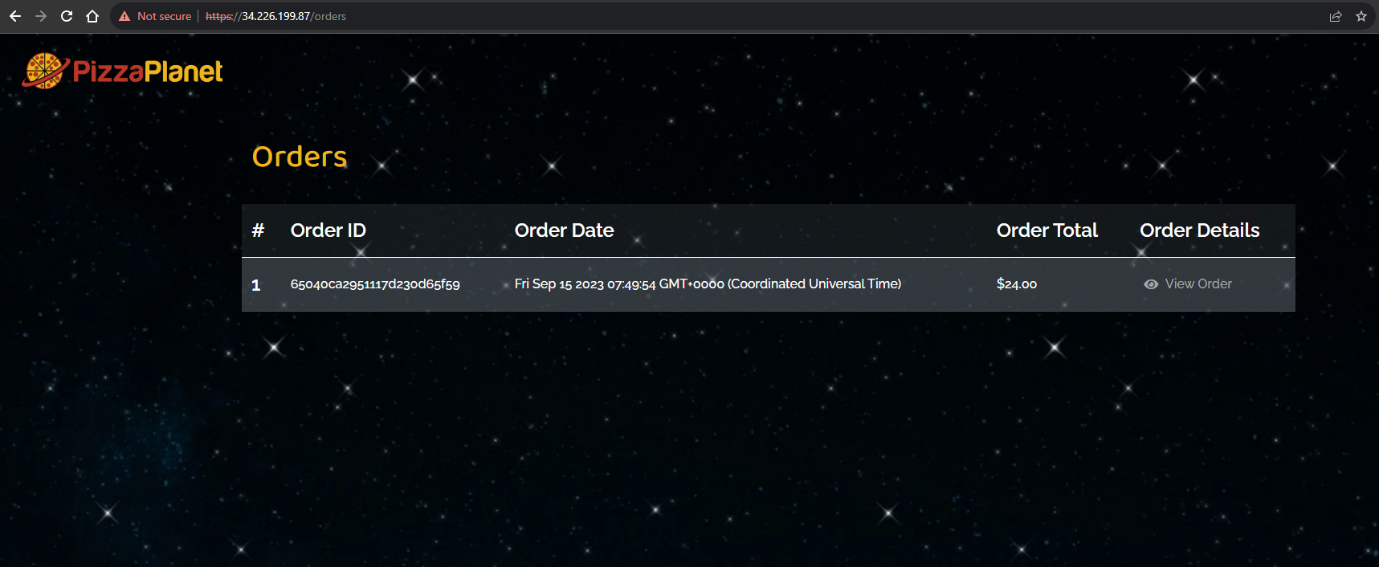
## Run Individual Containers



This is the frontend-app with a secure https connection running on port 443 which is redirected traffic from port 3000 and 80.



This is the mongo-express UI running on port 8081.



To ensure that the mongodb and frontend application is connected successfully we place an order for a pizza be creating a new user.

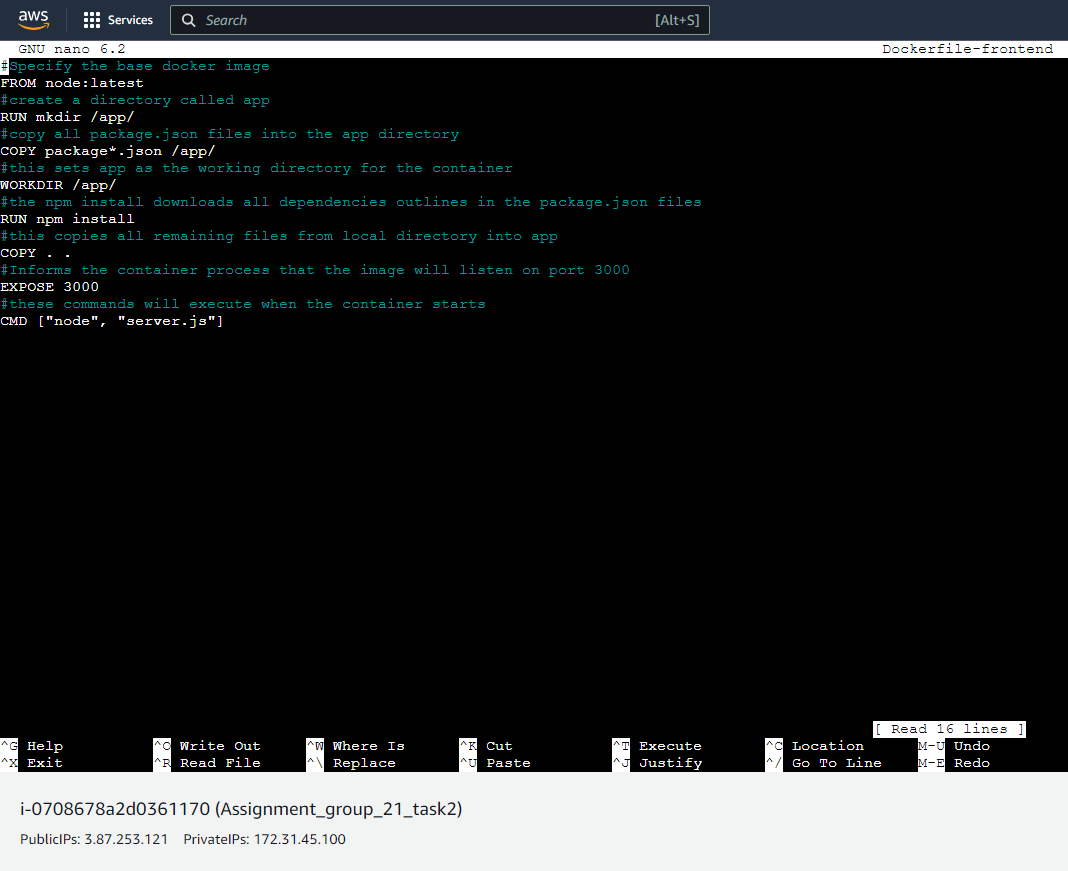


Inside the mongo-express UI we can see the created user manish that was used to place the pizza order inside the frontend-app.

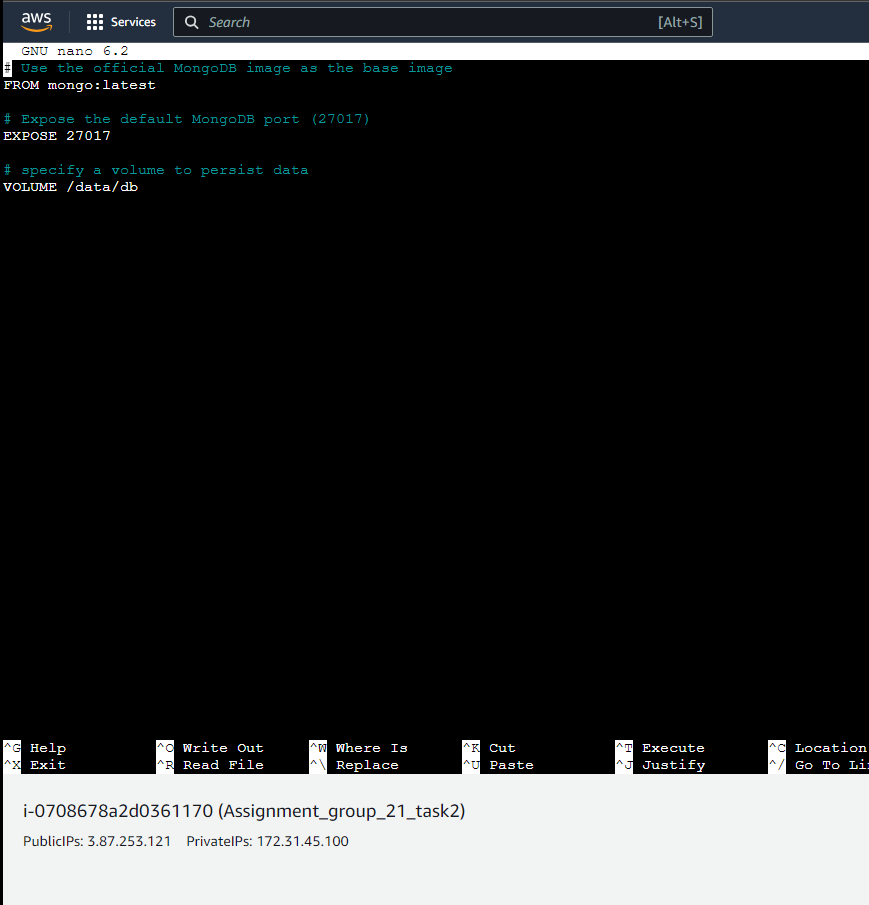
# Task 2

Task 2 requires us to create a docker compose file that will allow for an automated construction of the pizzeria project. We had created a new instance for task two, but the inbound security rules are the same as task 1. We will also install docker and mongo in the same fashion as task 1 except we will have to install docker compose. We have used the same docker files and nginx.conf from task one except for the frontend docker file and mongo-express dockerfile which had environment variables. Below are all docker files and files used for task 2:

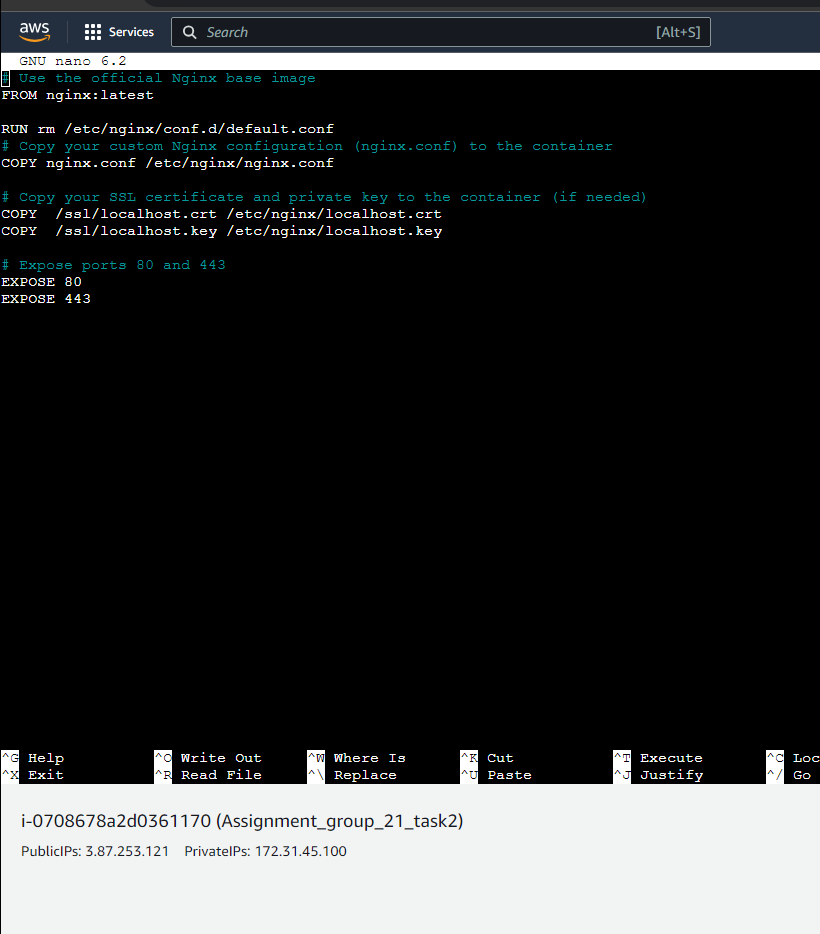
Dockerfile-frontend:



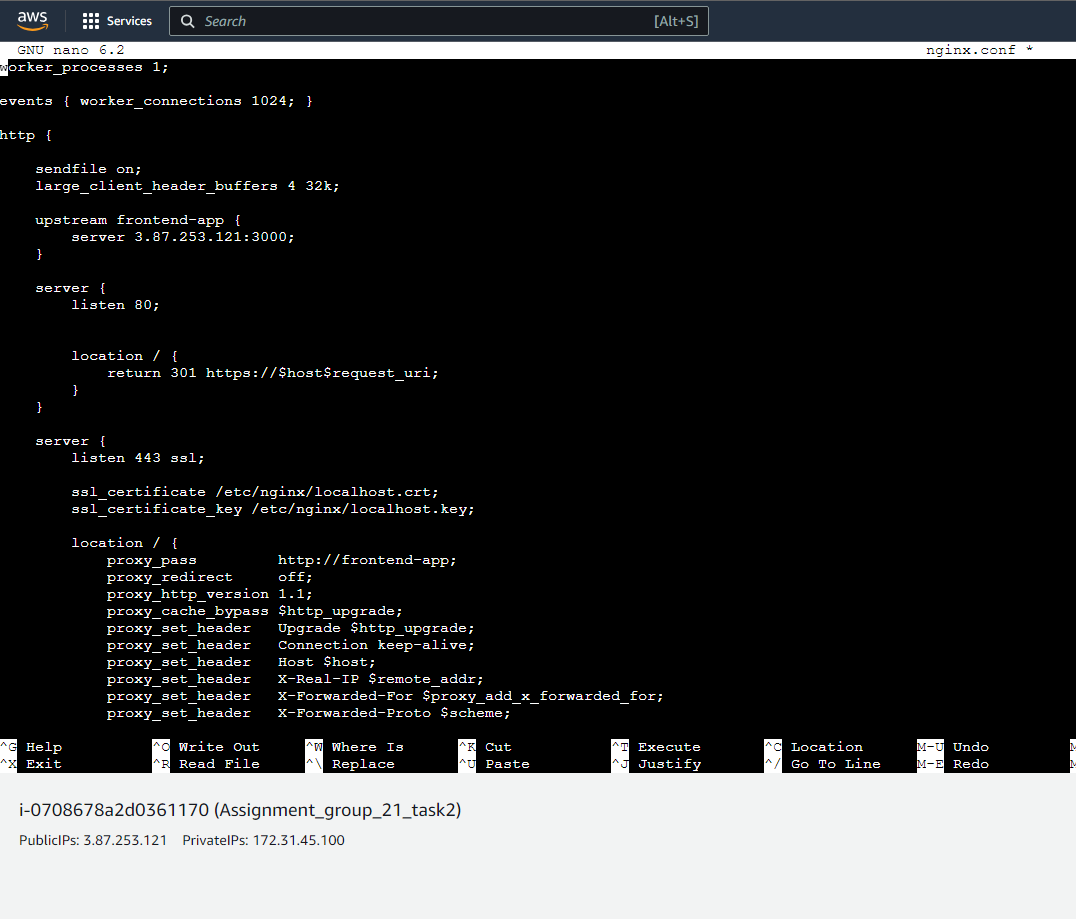
Docker file-backend:



Dockerfile-nginx:



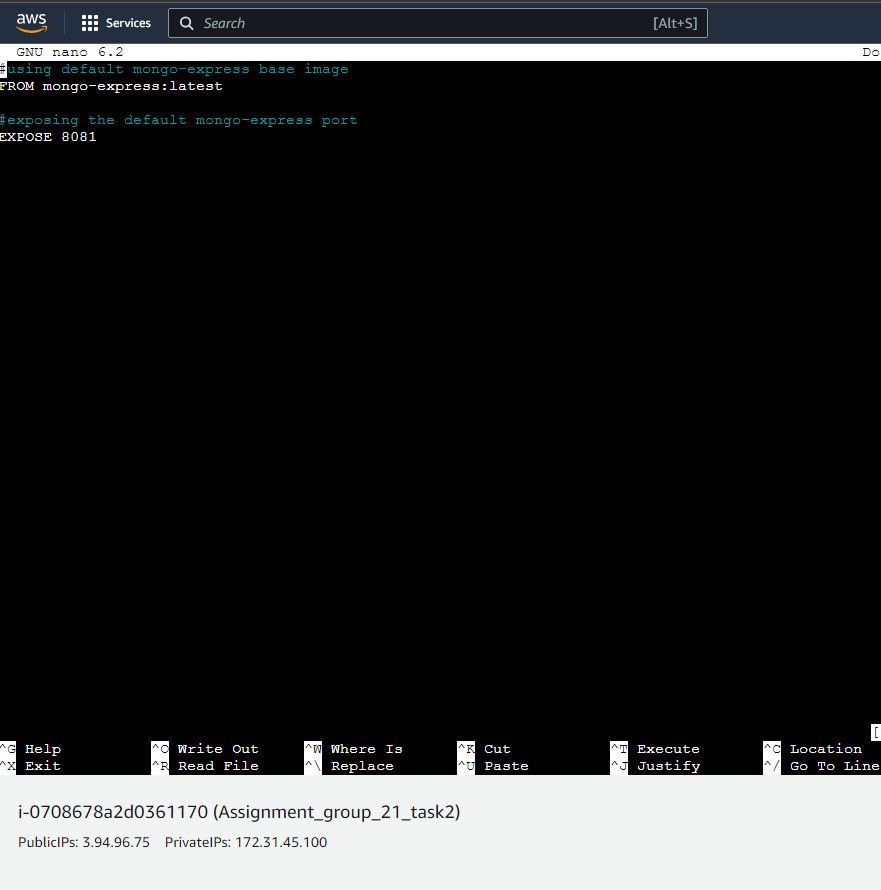
Nginx.conf:



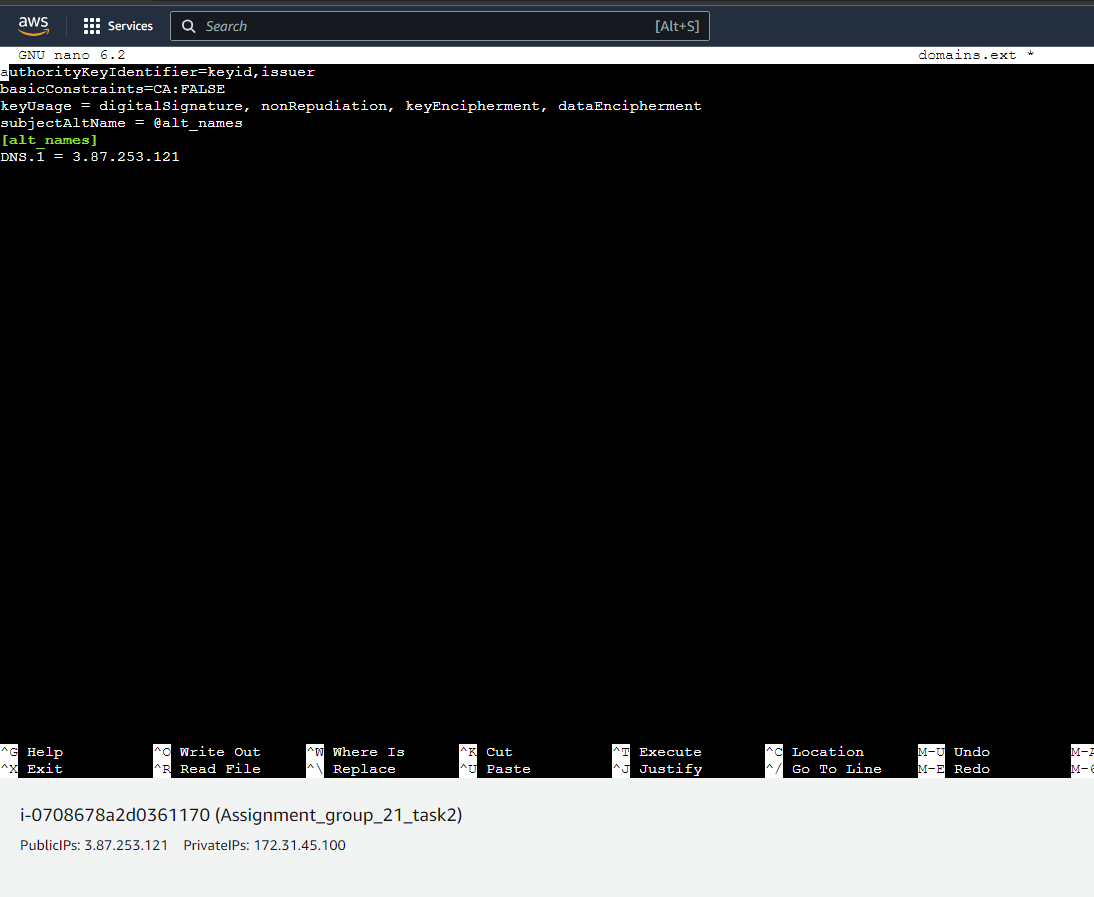
The rest of the Ngix.conf file:  
 proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_buffer\_size 128k;  
 proxy\_buffers 4 256k;  
 proxy\_busy\_buffers\_size 256k;  
 proxy\_read\_timeout 3600;  
 }  
 }  
 }

Mongo-express dockerfile:



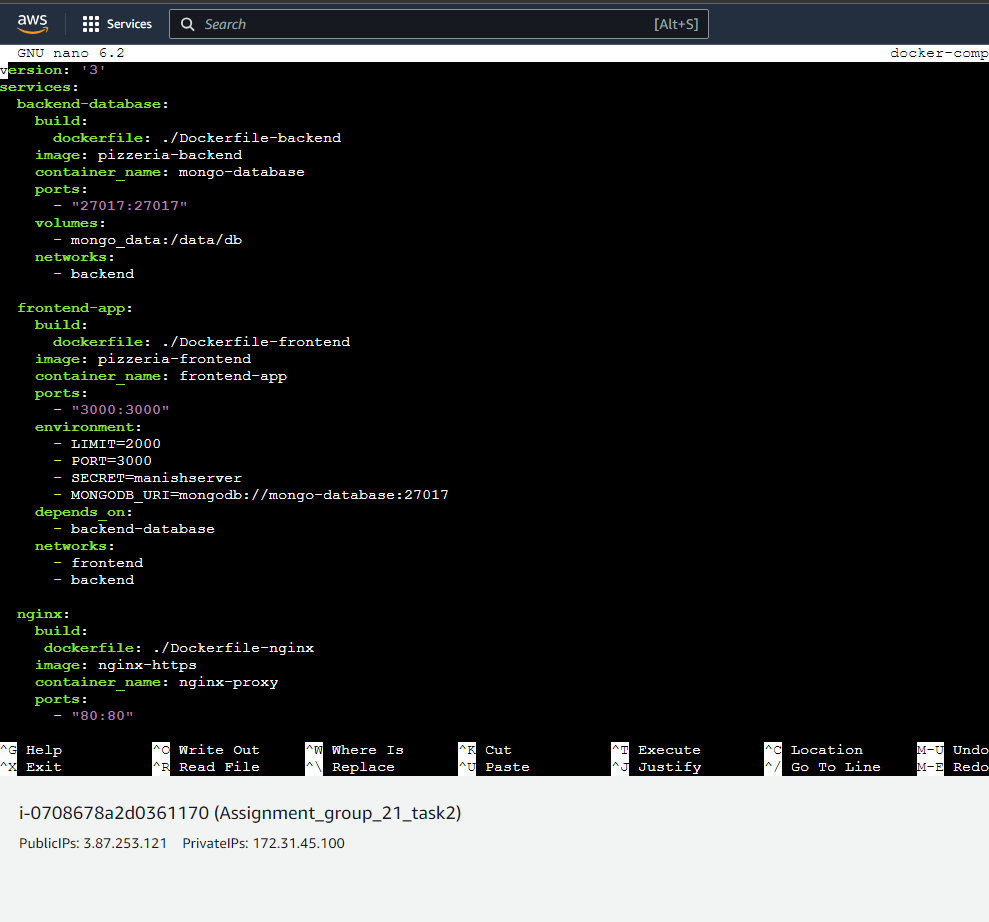
Domains.ext:

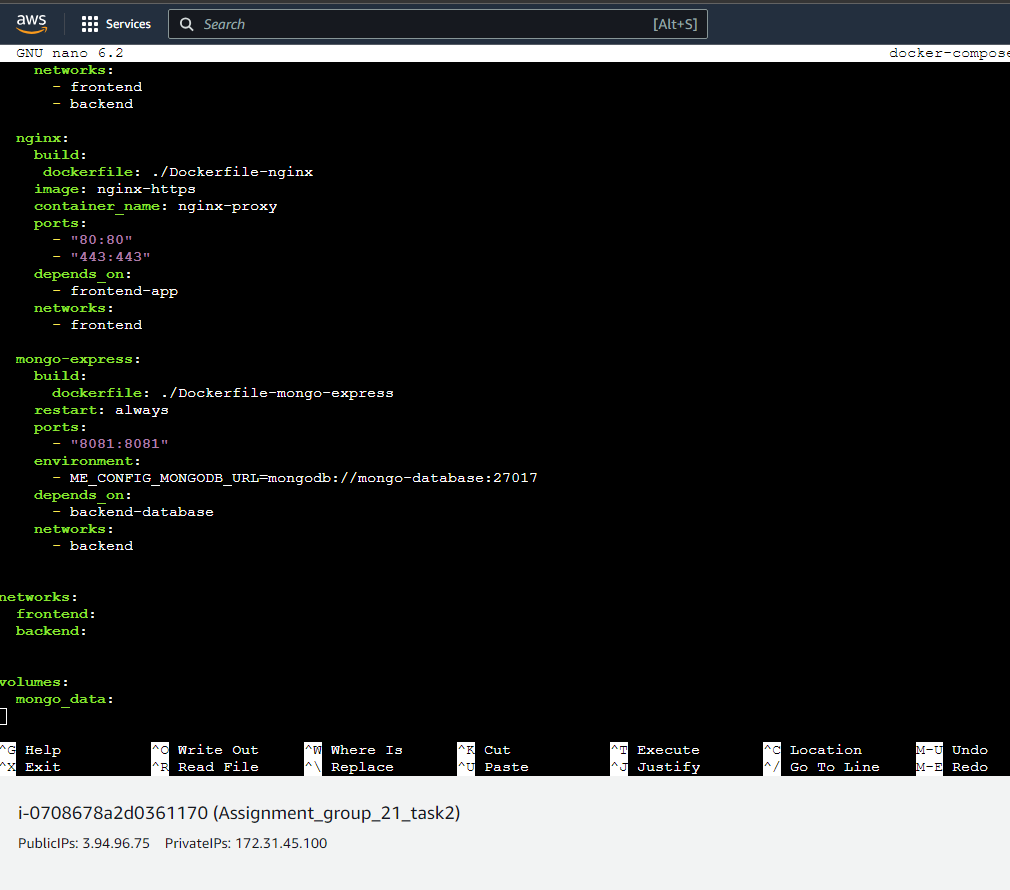


For the dns ip we will be using the public ip. For this example, it is listed here as 3.87.253.121.

## Docker Compose

The file below is the docker-compose.yaml file that we used. This file will allow us to define the services and ther respective configurations. First we have four services. These being frontend, backend, mongo-express, and nginx. We also configured some enviroment variables to connect the containers. These being in the mongo-express and the frontend app. We also configured volumes in order to persist data in mongodb container. This is achieved using the command “mongo\_data:/data/db”. For each of the services we exposed the relevant ports for them. This is seen in the use of the command port: -”3000:3000”. Exposing the relevant ports will allow for external access.

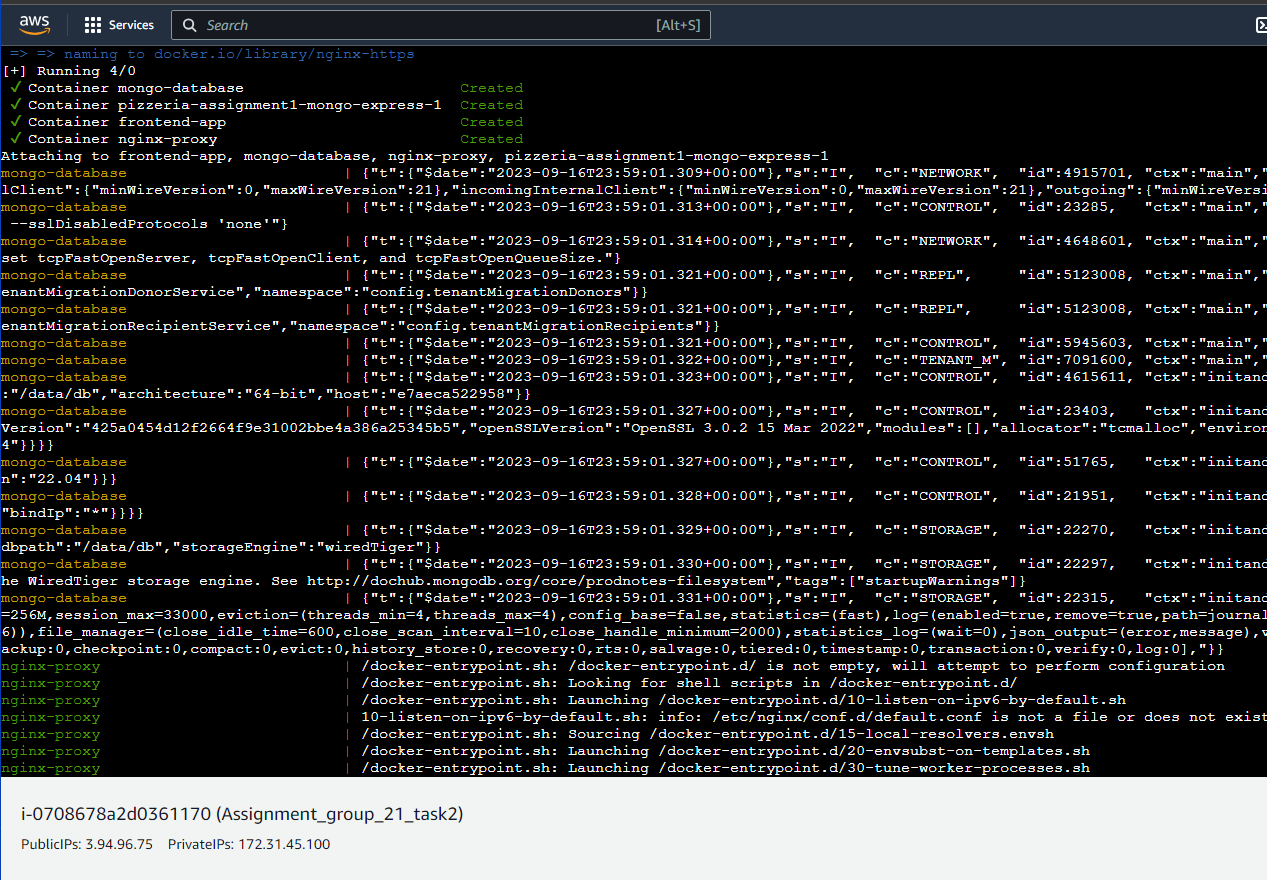


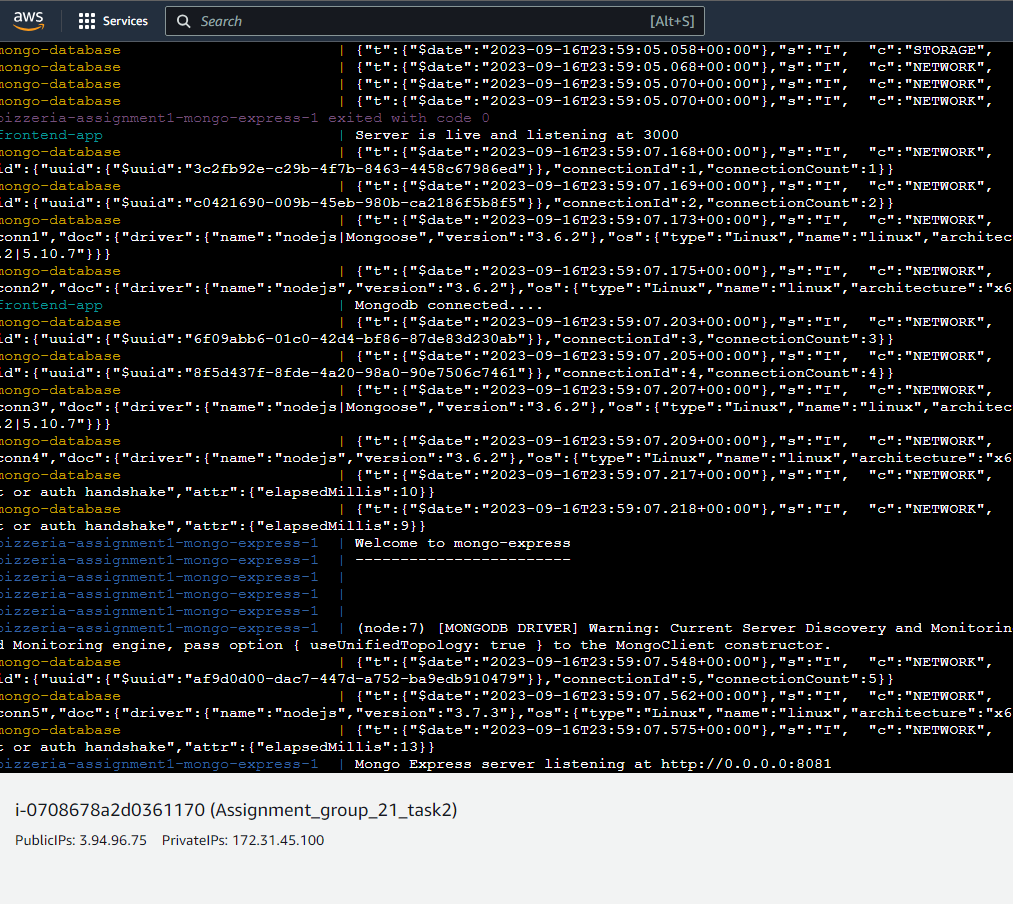


After this we use the command “docker-compose up –d" while in the directory of the docker compose file. This will run the docker compose file and the results of this can be seen below. The –d will run the file in detached mode.



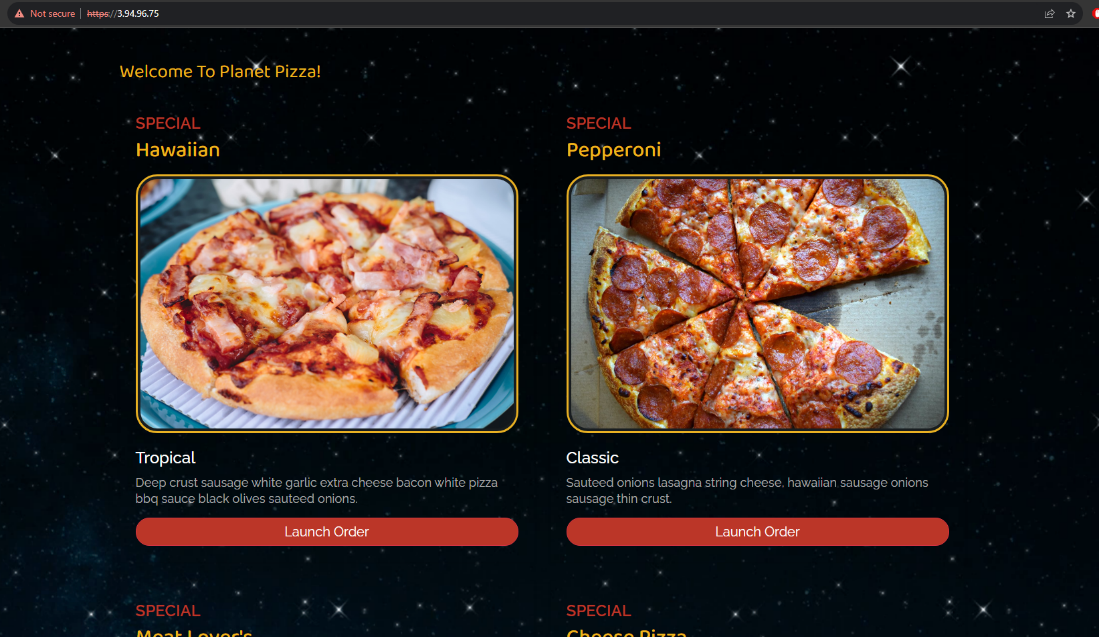
In the image below we can see the containers running



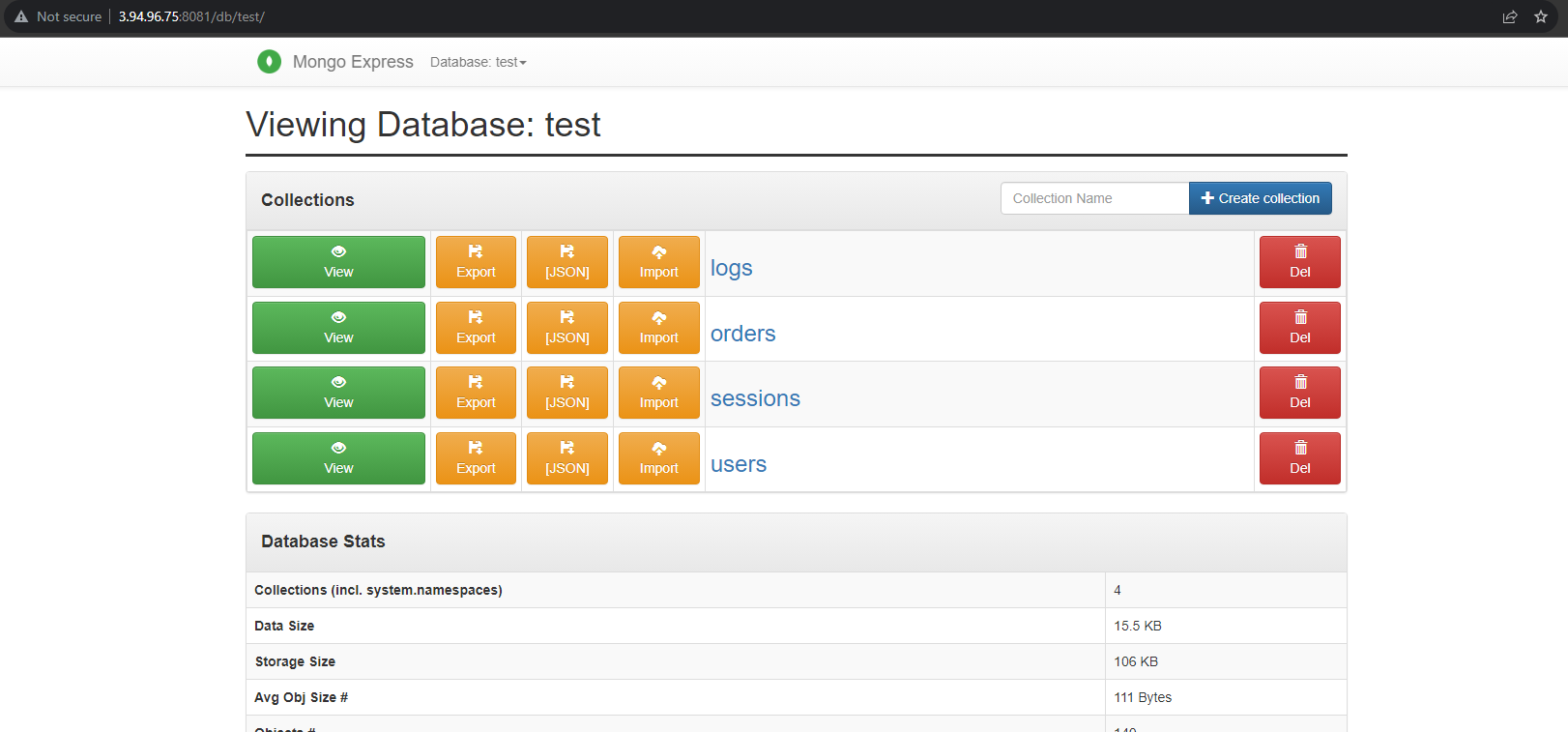


## Run Containers with Docker Compose

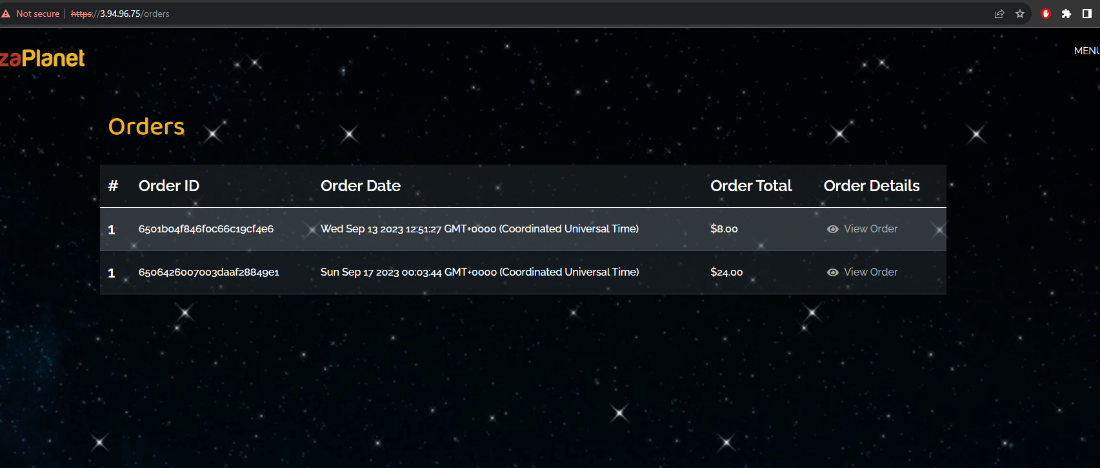
In the screenshots below we can see the site working as intended. In the first screenshot we can see the site is displaying the pizzas this shows that the front end app is connected to the pizzeria zip file.



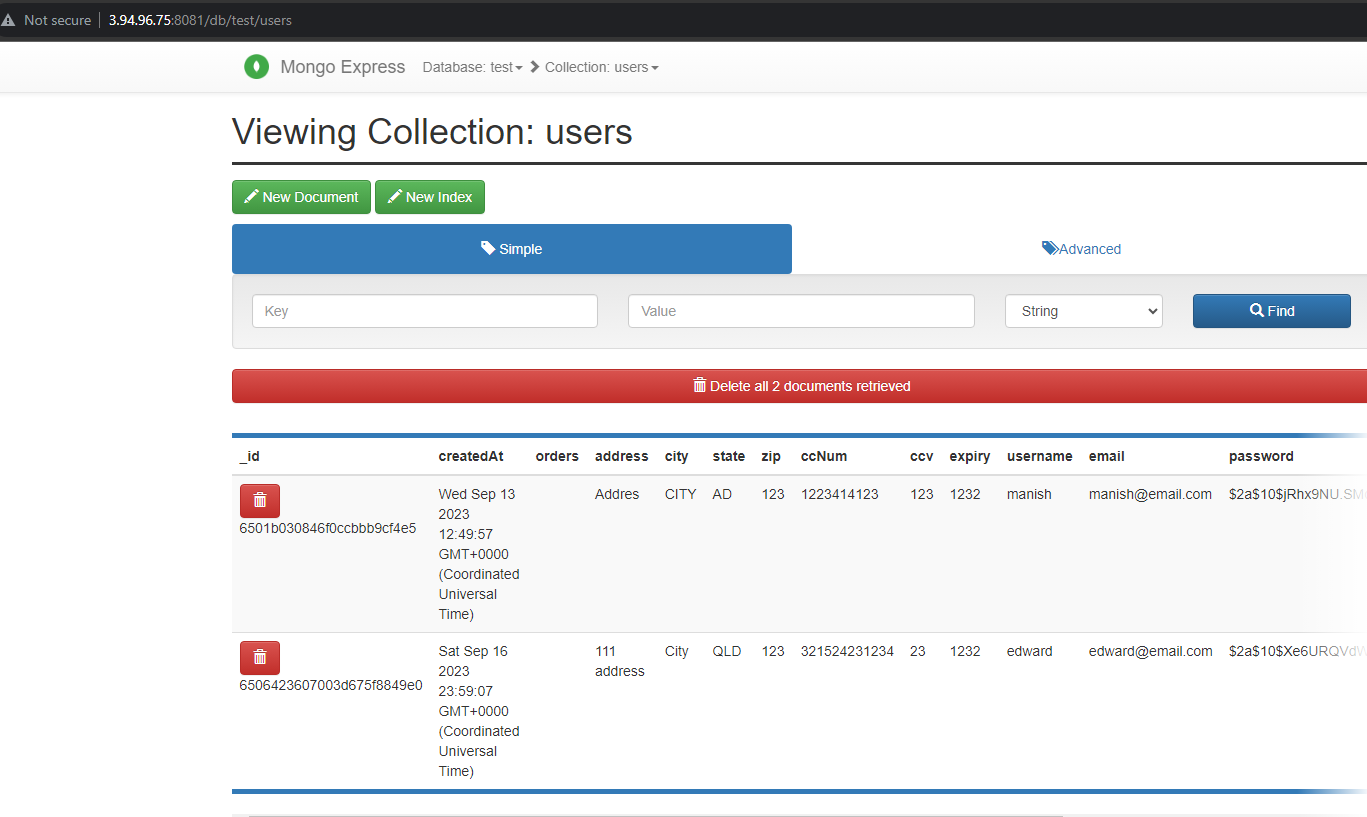
We connect to the same ip using port 8081 and we can see the mongo express is working and we can view the data bases.



Here we created a new user and ordered a pizza.



After creating a new users the data base updated and showed the new user. This can be seen below.



This means that the front end application was successful in updating the mongodb and the backend files.

# Task 3

Task 3 was not able to be successfully completed mainly due to time constraints. However we were able to create the pods and the files below.

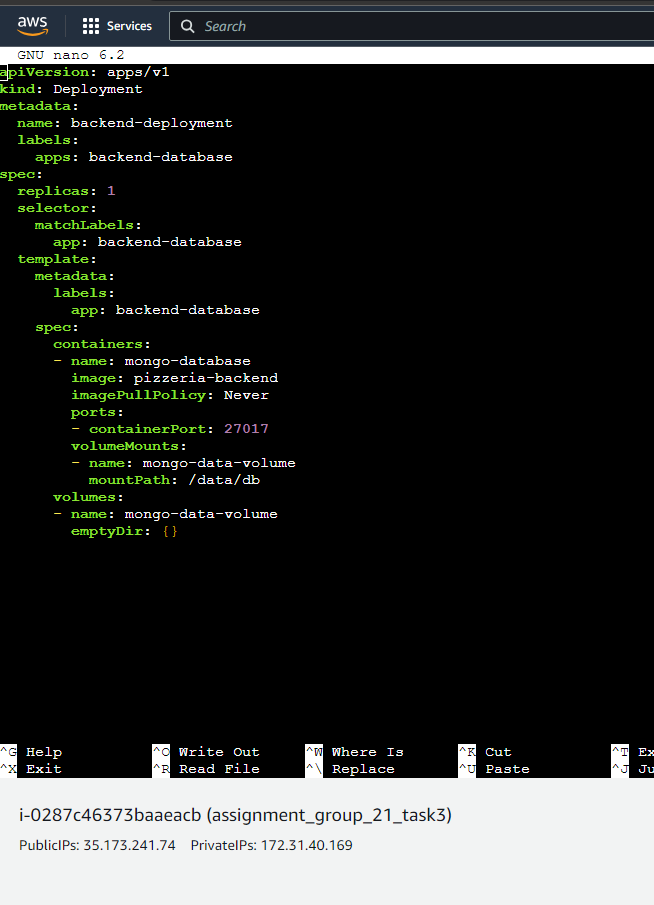
## Create Pods

Firstly we will create Kubernetes YAML files for each of the services. These being frontend, backend, mongo-express and nginx. These will also use –deployment to avoid confusion with the dockerfiles in tasks 1 and 2.

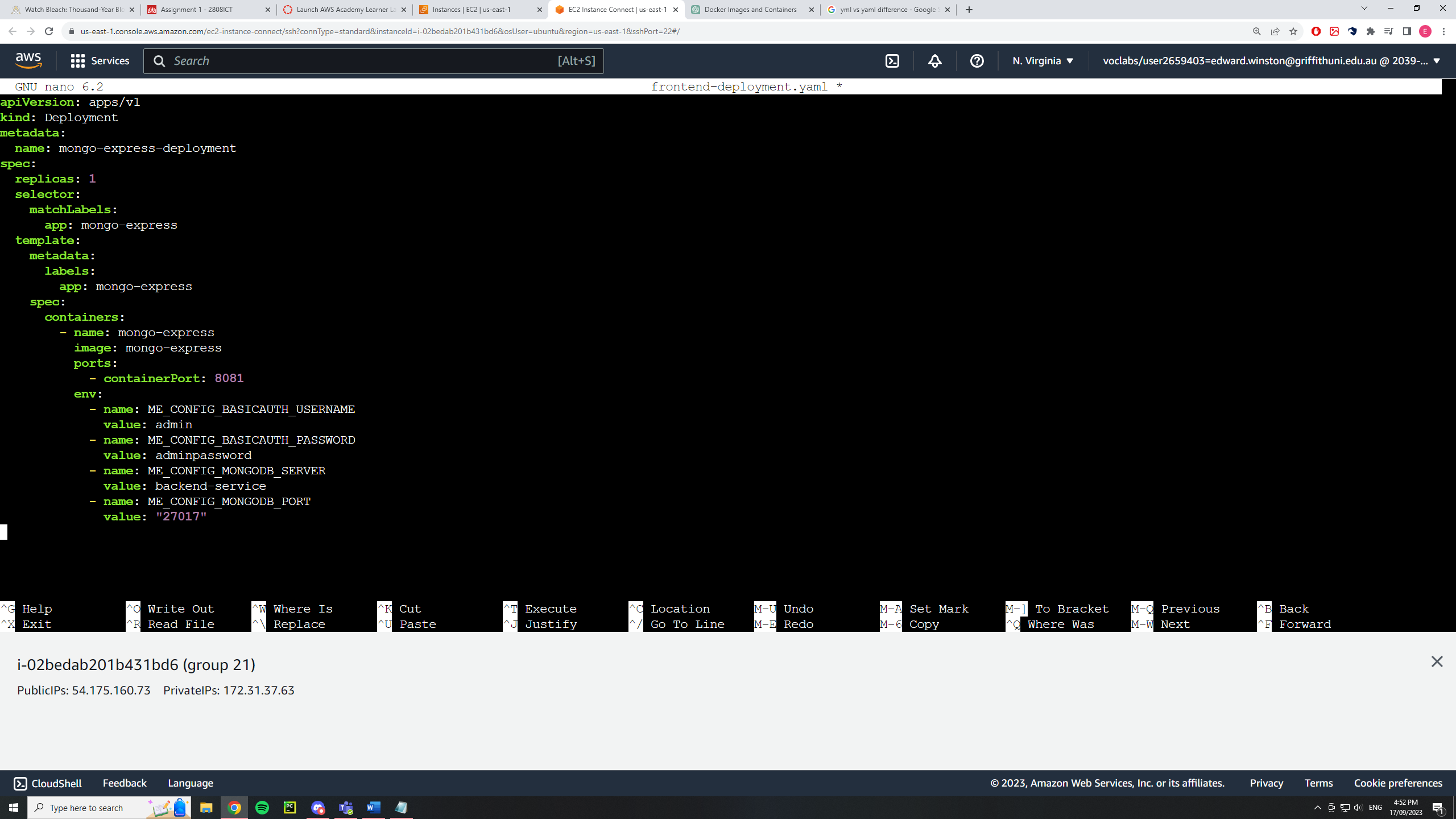
**frontend-deployment.yaml:**



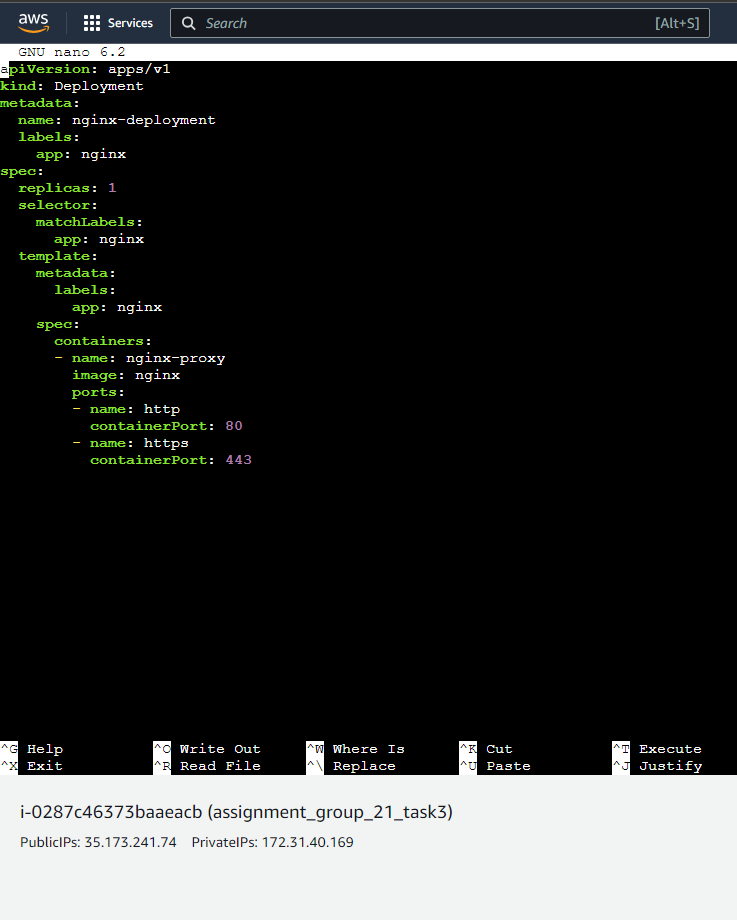
**backend-deployment.yaml**



**mongo-express-deployment.yaml**



**nginx-deployment.yaml**



Next we will need to deploy these kubernetes using the following commands

kubectl apply -f frontend-deployment.yaml

kubectl apply -f backend-deployment.yaml

kubectl apply -f mongo-express-deployment.yaml

kubectl apply -f nginx-deployment.yaml

After this we will need to expose the services to allow for external access of the application and to facilitate pod communication. This will need to be done for the Frontend service and the back end service.

**frontend-service.yaml**



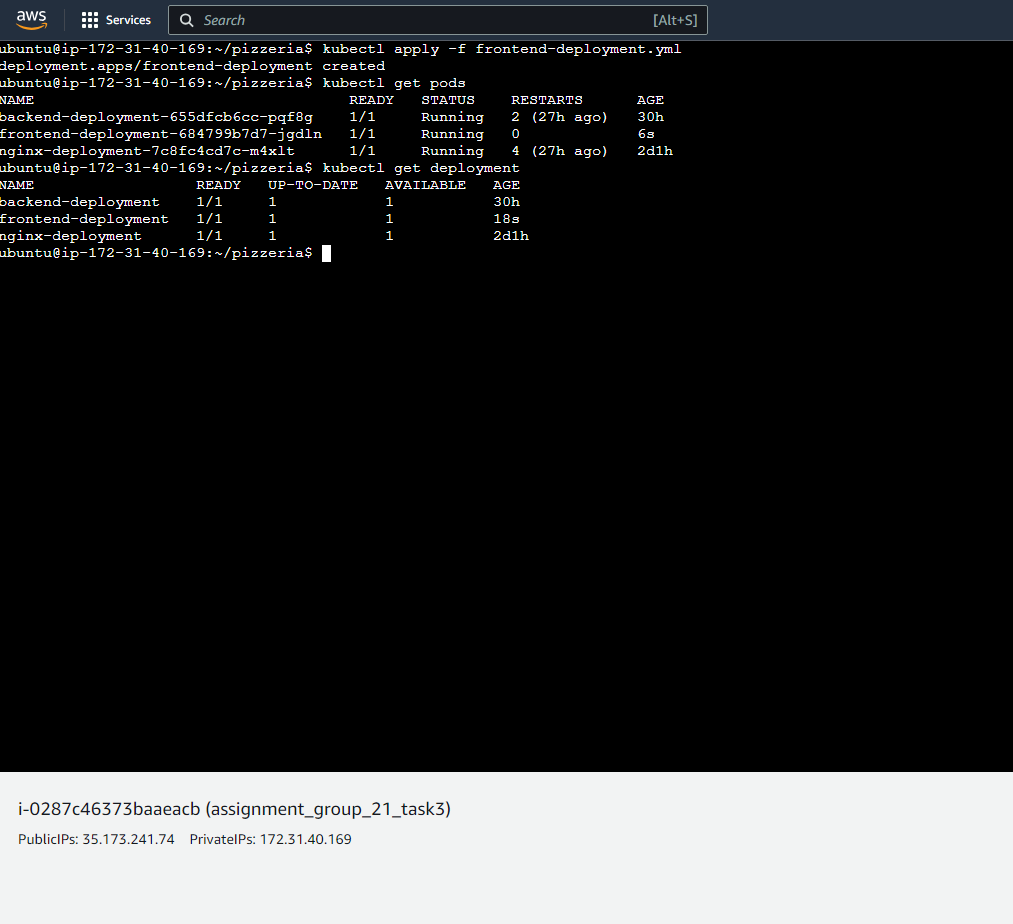
**backend-service.yaml**



Then we need to apply these services using the commands below:

kubectl apply -f frontend-service.yaml

kubectl apply -f backend-service.yaml



## Run Cluster with Minikube

# Self-Learning

Stack overflow was used in self-learning throughout this assignment as it provided many solutions to the problems and errors we were encountering. Any time an error or a command we didn't understand popped up we could enter it into stack overflow and understand more about it through the forums.

The youtube video [Node js NGINX Reverse Proxy On Docker | NGINX Reverse Proxy Setup | Thetips4you](https://youtu.be/I3LpAVcScWs?si=rVzFsf-X_2W8CMoH) was useful to understand the steps that needed to be taken to complete task 1. The youtube video [Dockerizing a Node.js, Express, MongoDB App with NGINX Reverse Proxy using Docker Compose](https://www.youtube.com/watch?v=4zUQEkDdNR0&ab_channel=Thetips4you) was useful to understand the steps that needed to be taken to complete task 2. These two videos outlined how each of the files functions and how they linked to one another. I found these youtube videos while searching how to configure nginx files for docker containers.

# References

[How to Install Docker and Docker Compose on Linux (howtogeek.com)](https://www.howtogeek.com/devops/how-to-install-docker-and-docker-compose-on-linux/)

[*Node js NGINX Reverse Proxy On Docker | NGINX Reverse Proxy Setup | Thetips4you*](https://youtu.be/I3LpAVcScWs?si=rVzFsf-X_2W8CMoH)

[*Dockerizing a Node.js, Express, MongoDB App with NGINX Reverse Proxy using Docker Compose*](https://www.youtube.com/watch?v=4zUQEkDdNR0&ab_channel=Thetips4you)

<https://stackoverflow.com/>