**Travel Memory Application Deployment**

**Project Repository:**

Access the complete codebase of the Travel Memory application from the provided GitHub link: https://github.com/UnpredictablePrashant/TravelMemory

**Objective:**

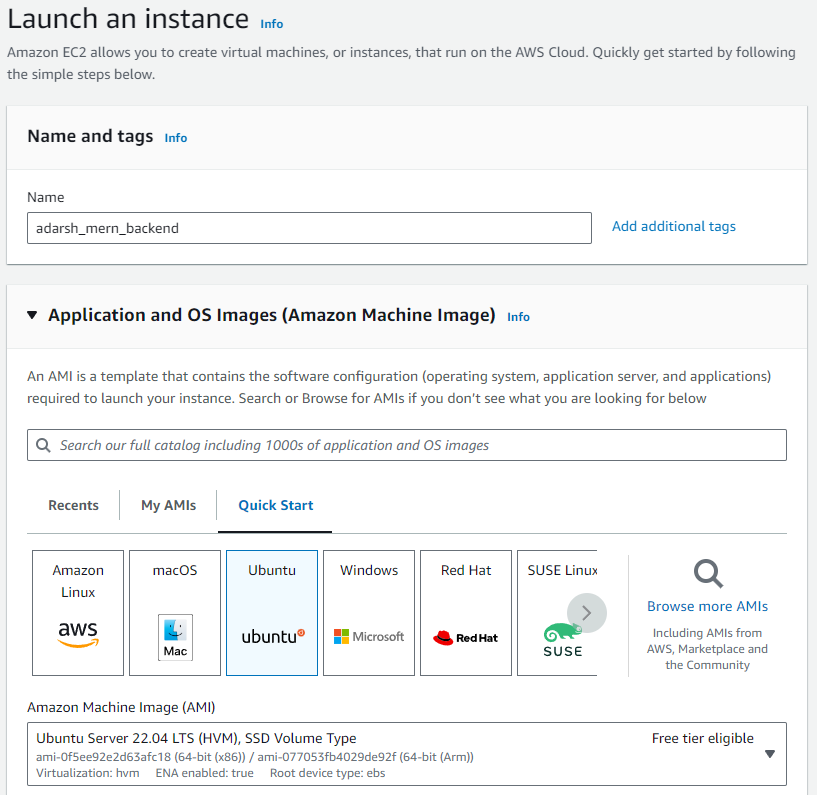
* Set up the backend running on Node.js.
* Configure the front end designed with React.
* Ensure efficient communication between the front end and back end.
* Deploy the full application on an EC2 instance.
* Facilitate load balancing by creating multiple instances of the application.
* Connect a custom domain through Cloudflare.

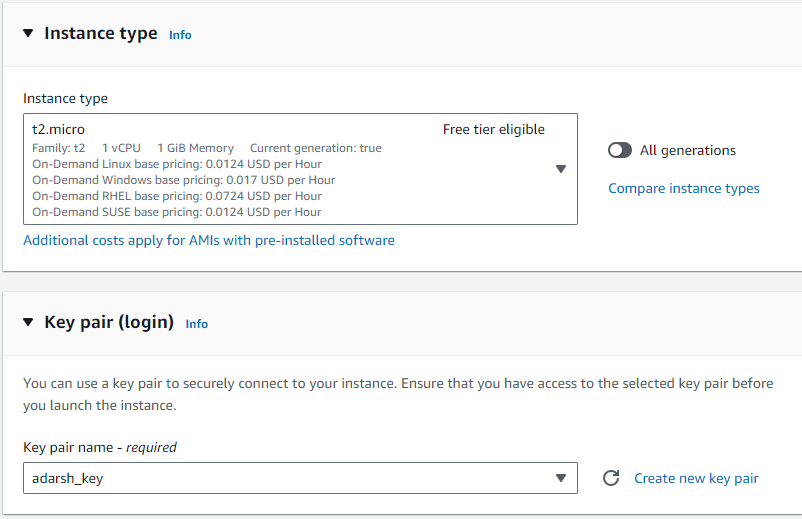
**Tasks:**

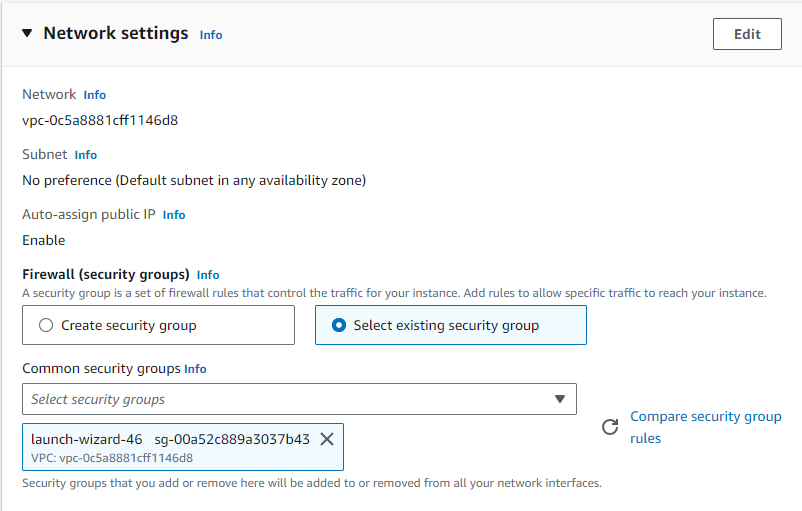
1. Backend Configuration:

* Clone the repository and navigate to the backend directory.
* The backend runs on port 3000. Set up a reverse proxy using nginx to ensure smooth deployment on EC2.
* Update the .env file to incorporate database connection details and port information.

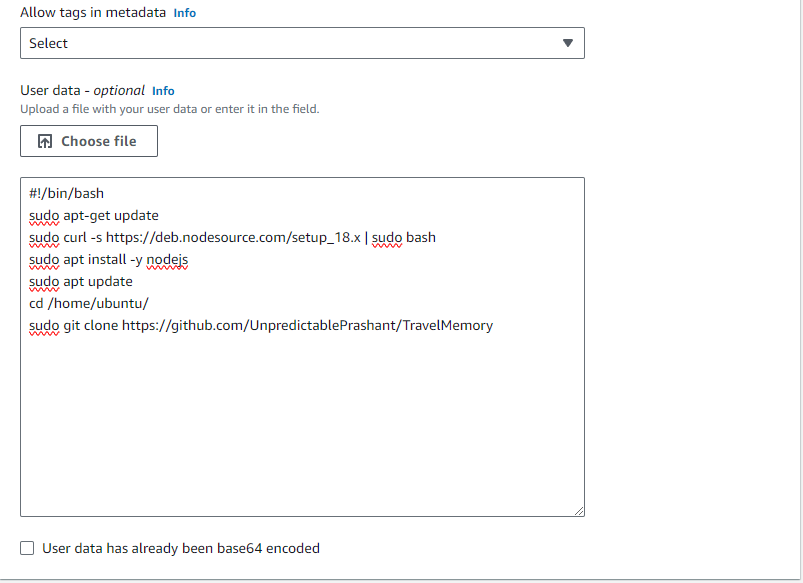
Launching the backend named “adarsh\_mern\_backend” having ubuntu as a base image, t2.micro as Instance type







Installing the Nodejs 18 and cloned the git repository of TravelMemory application.



#!/bin/bash

sudo apt update

sudo bash

curl -s https://deb.nodesource.com/setup\_18.x | sudo bash

sudo apt install nodejs -y

sudo cd /home/ubuntu/

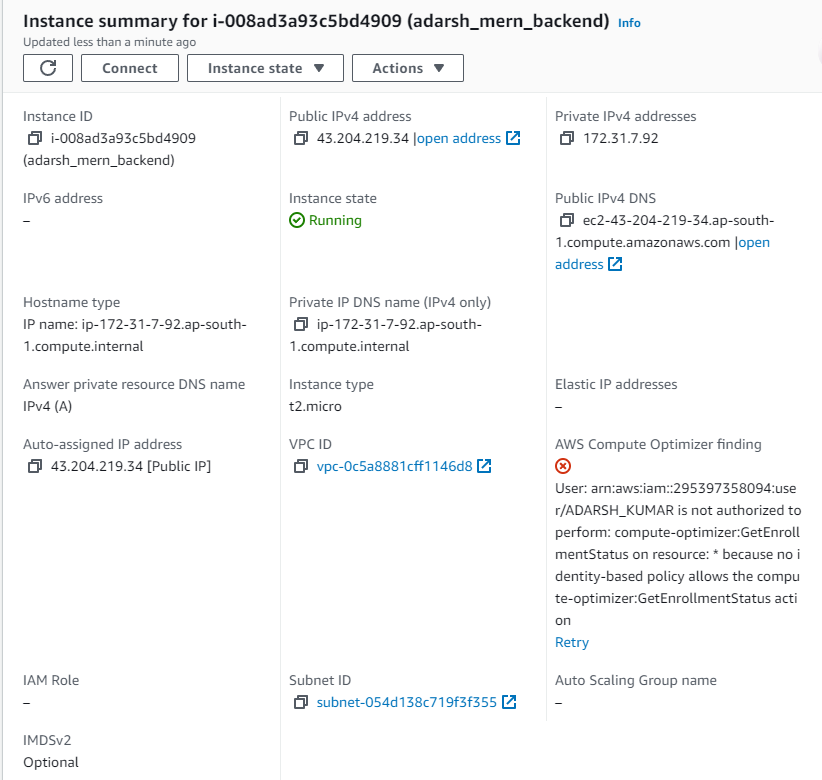
sudo git clone <https://github.com/UnpredictablePrashant/TravelMemory>

The Backend instances is up and running with

Public IP 43.204.219.34

Private IP 172.31.7.92

Availability zone: ap-south-1

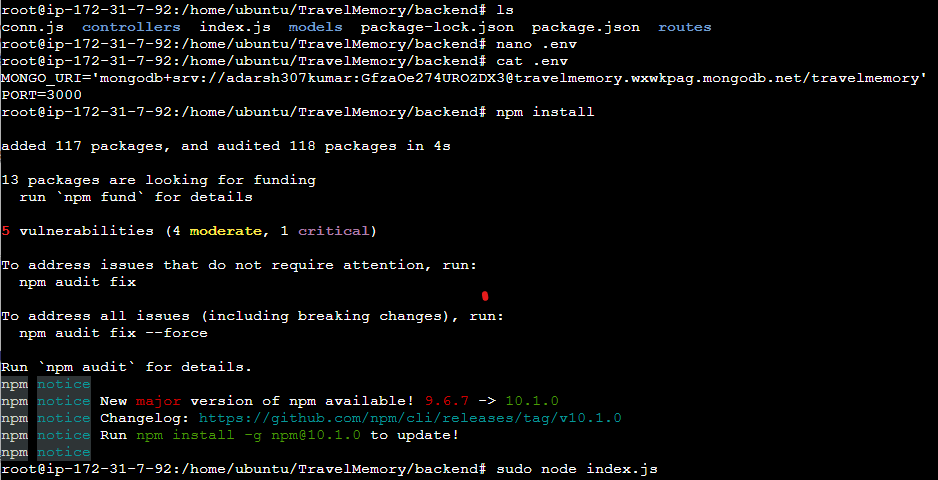


#nano .env -> add the database MongoDB credentials

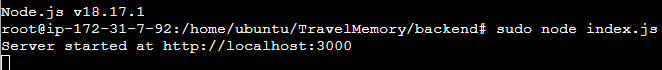
MONGO\_URI='mongodb+srv://adarsh307kumar:GfzaOe274UROZDX3@travelmemory.wxwkpag.mongodb.net/travelmemory'

PORT=3000

# sudo npm install -> to install the dependies of the node application



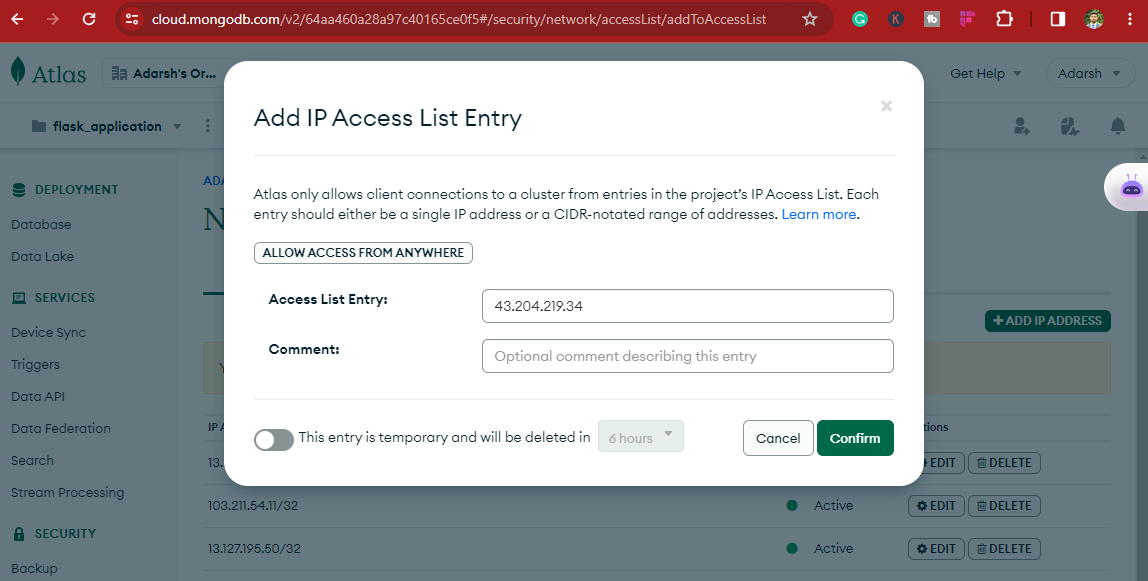
# sudo node index.js -> will start the backend at port 3000



We can access it browser by using the Public IP and port



Don’t forget to add the IP of backend to the Mongo DB IP access list so that backend instances can communicate to database

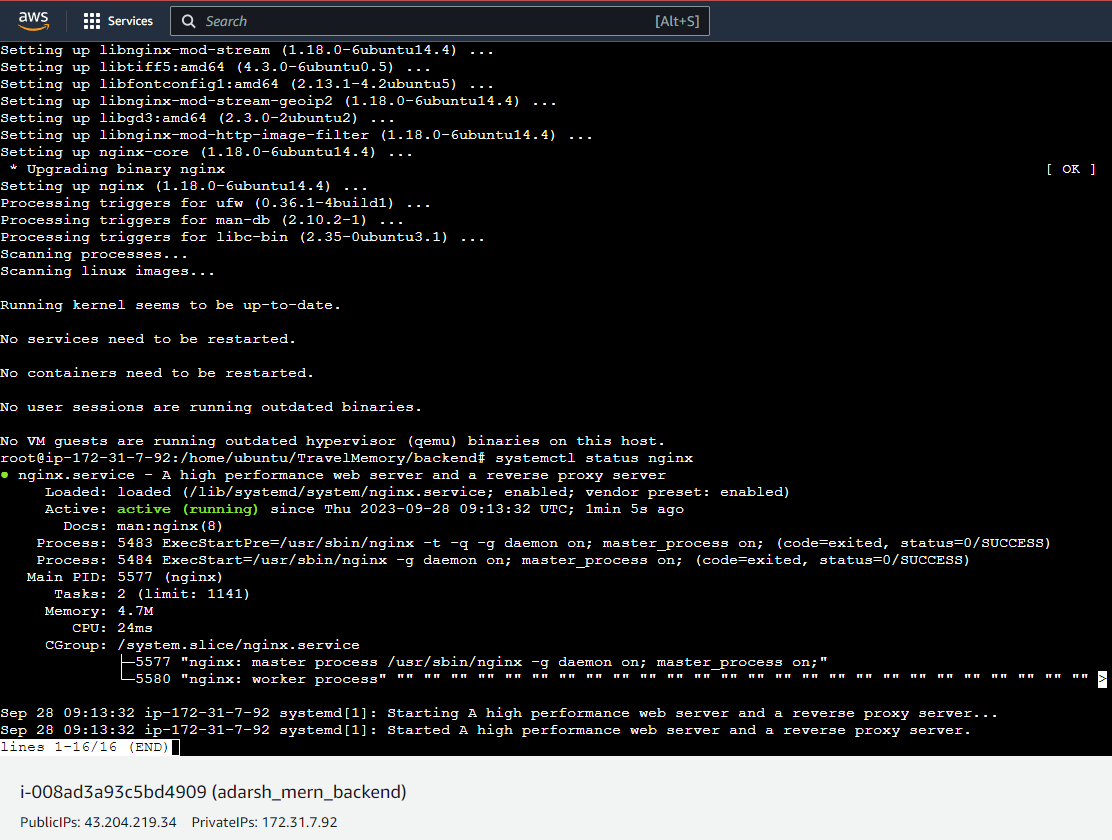




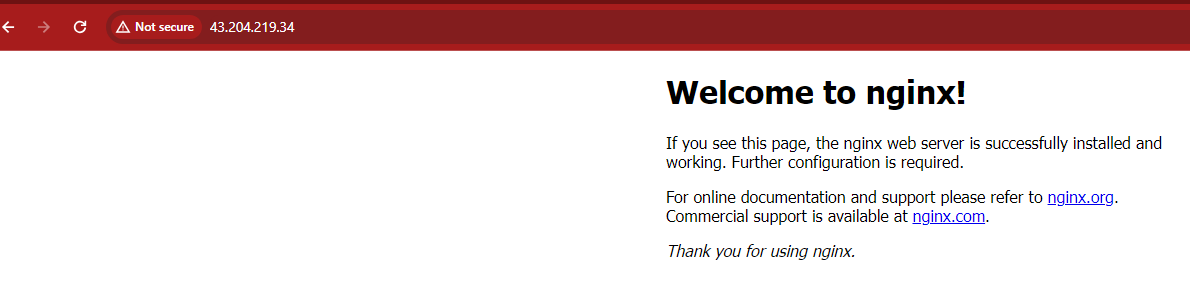
Now install the ngnix and check the status

# sudo apt install nginx

# sudo systemctl status nginx



We can check the nginx is running at port 80



Now for the reverse proxy

#sudo unlink /etc/nginx/sites-enabled/default

#cd /etc/nginx/sites-available/

#sudo nano custom\_server.conf

server {

listen 80;

location / {

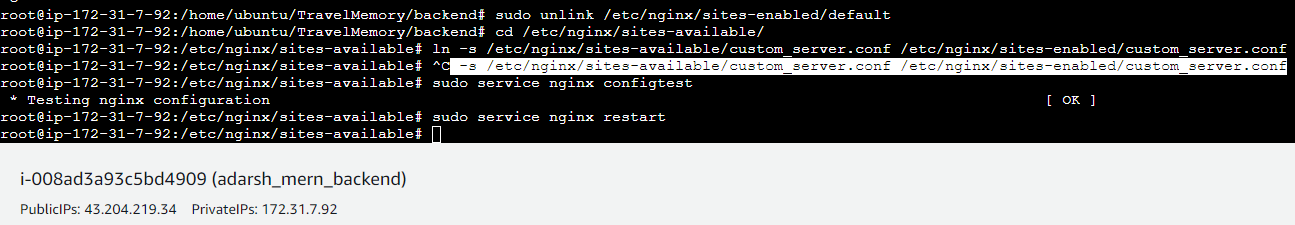
proxy\_pass http://my\_server;

}}

#ln -s /etc/nginx/sites-available/custom\_server.conf /etc/nginx/sites-enabled/custom\_server.conf

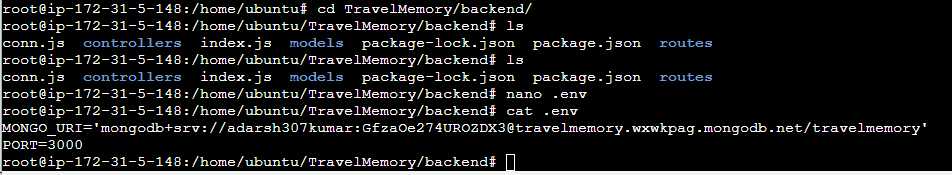
#sudo service nginx configtest

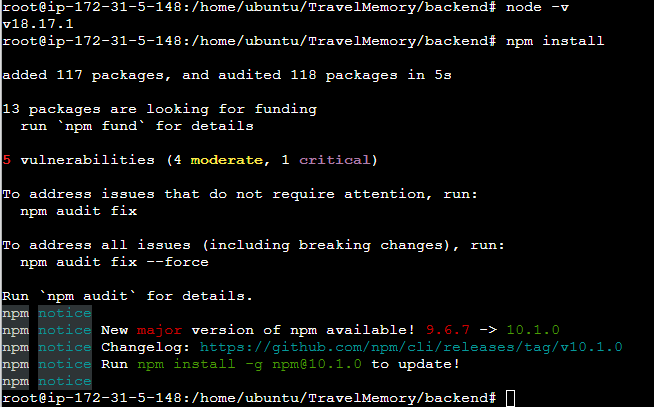
#sudo service nginx restart



Now we can access the backend at port 80 because of reverse proxy



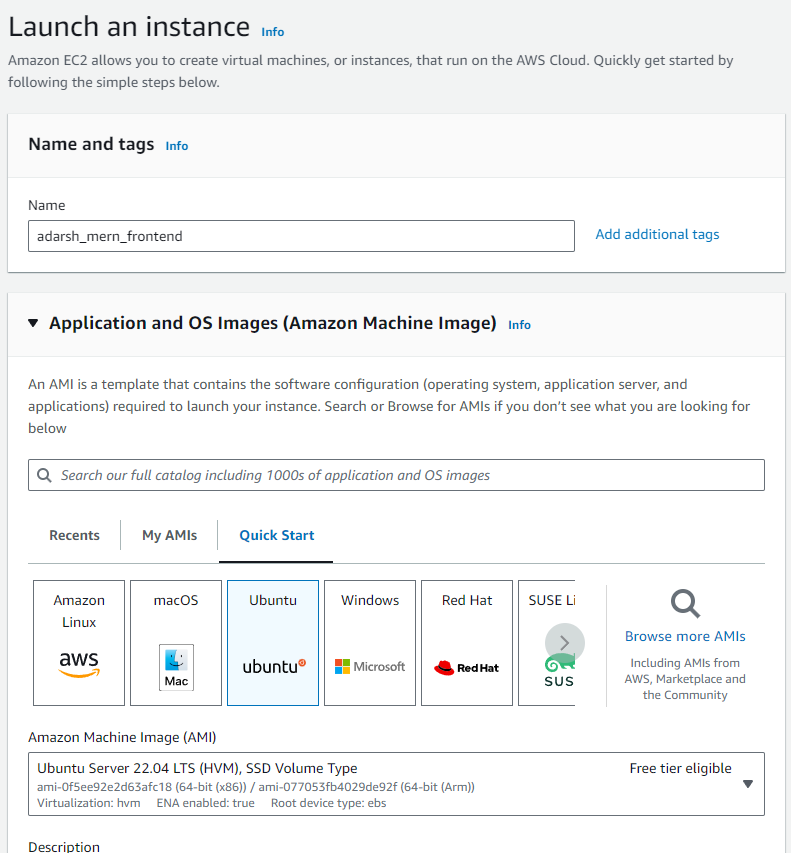




2. Frontend and Backend Connection:

* Navigate to the `urls.js` in the frontend directory.
* Update the file to ensure the frontend communicates effectively with the backend.

Launching the backend named “adarsh\_mern\_frontend” having ubuntu as a base image, t2.micro as Instance type

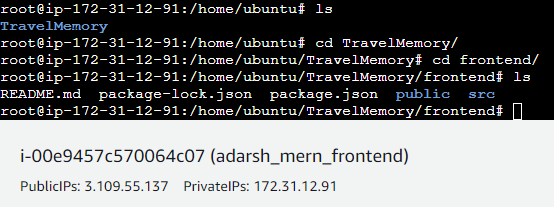


The Frontend instances is up and running with

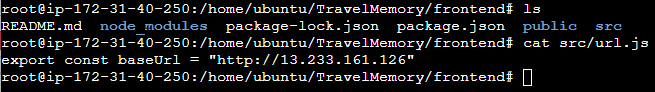
Public IP 3.109.55.137

Private IP 172.31.12.91

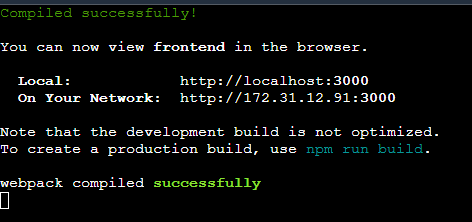
Availability zone: ap-south-1



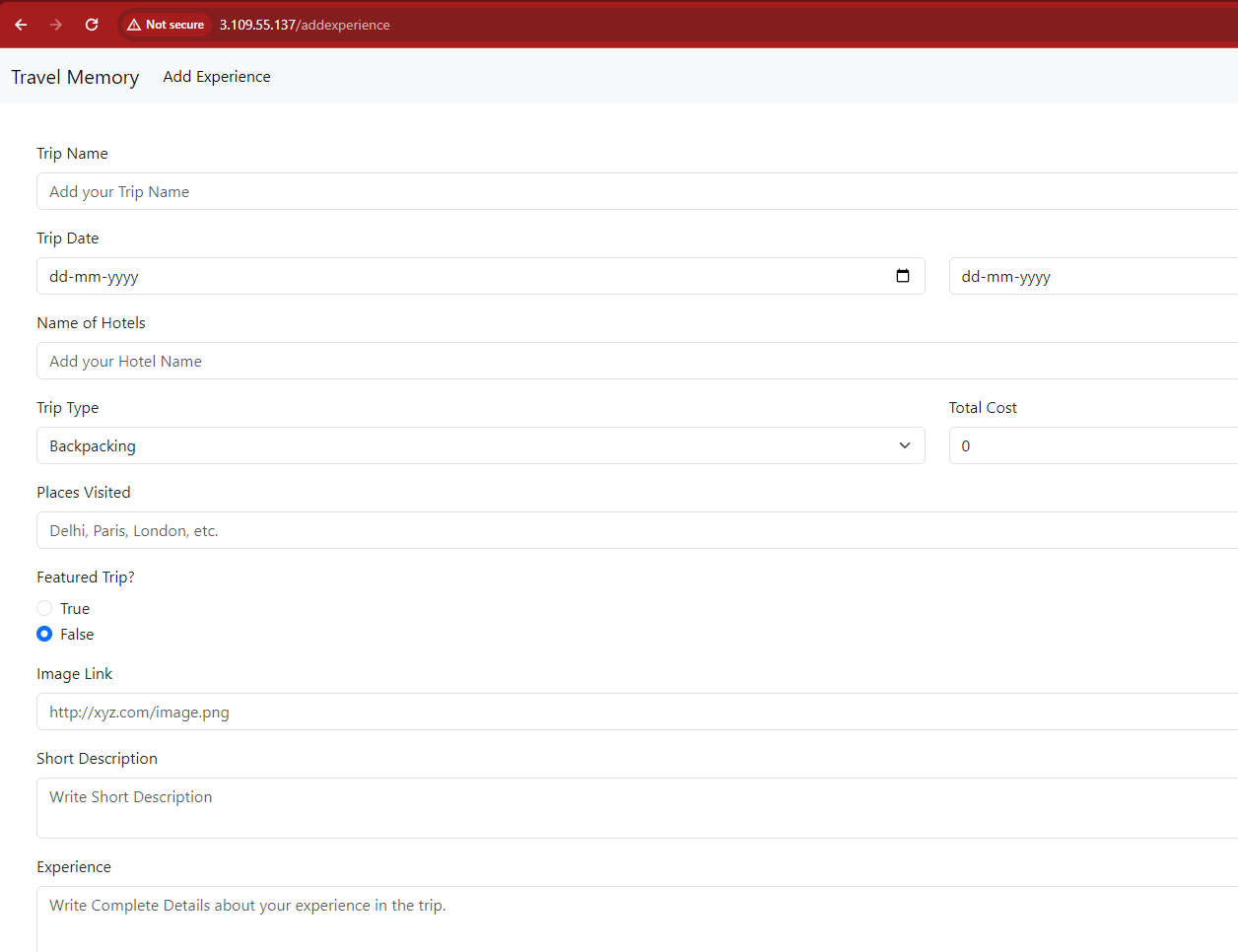
update the url.js file with the backend IP address

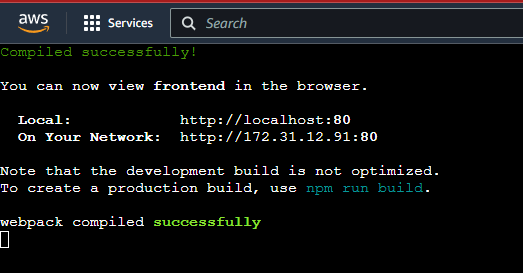


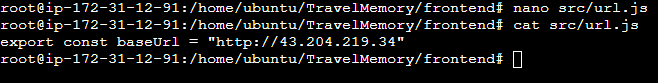
# sudo npm start -> the nodejs frontend application is running at port 3000



After doing the reverse proxy frontend will be running at port 80



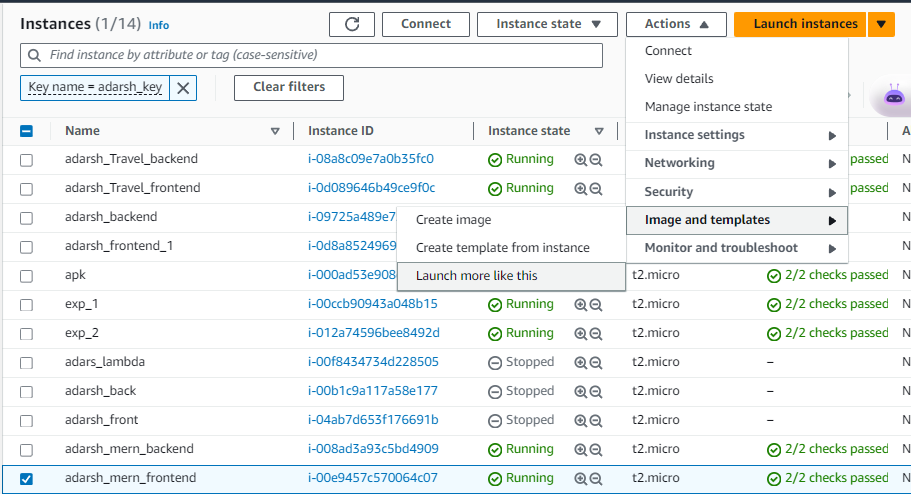


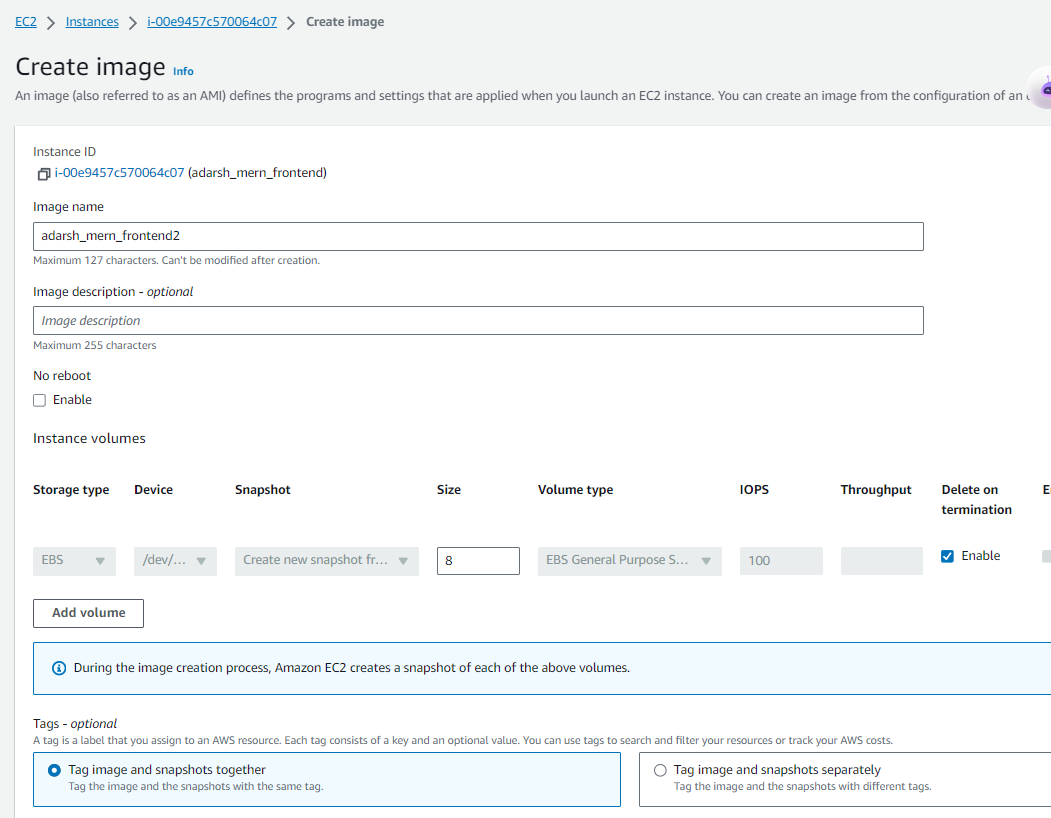


**3. Scaling the Application:**

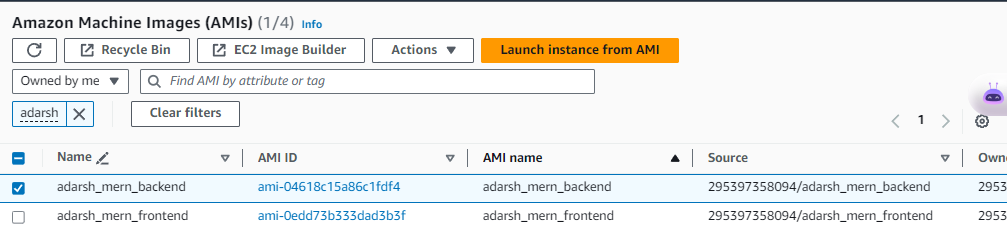
* Create multiple instances of both the frontend and backend servers.
* Add these instances to a load balancer to ensure efficient distribution of incoming traffic.

Launching the extra instances to distribute the load using taking snapshot of frontend & backend both.

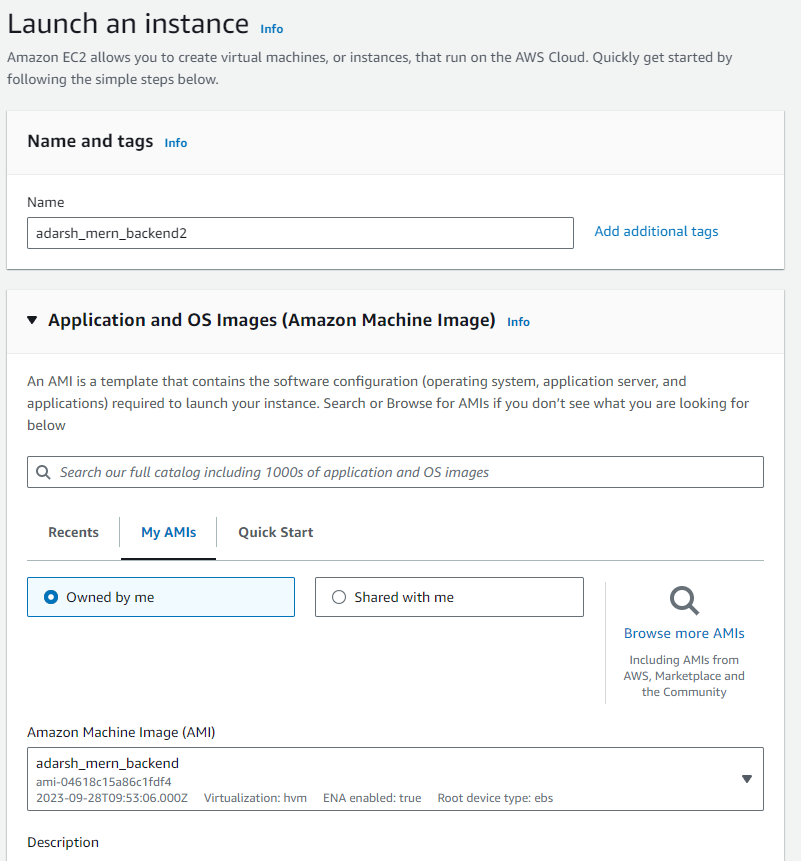


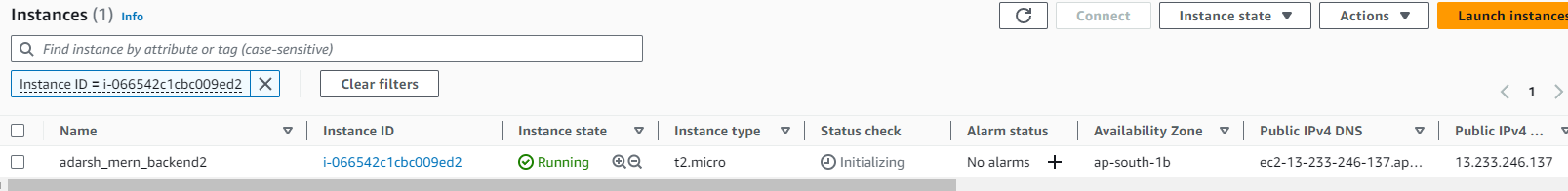


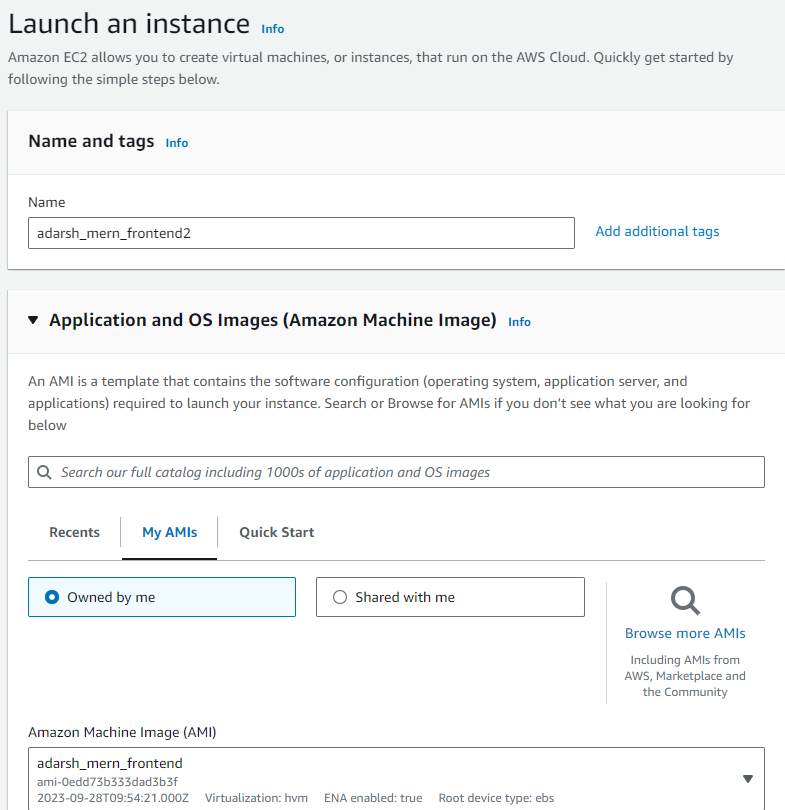
You can find the snapshots in AMIs (Amazon Machine Images)

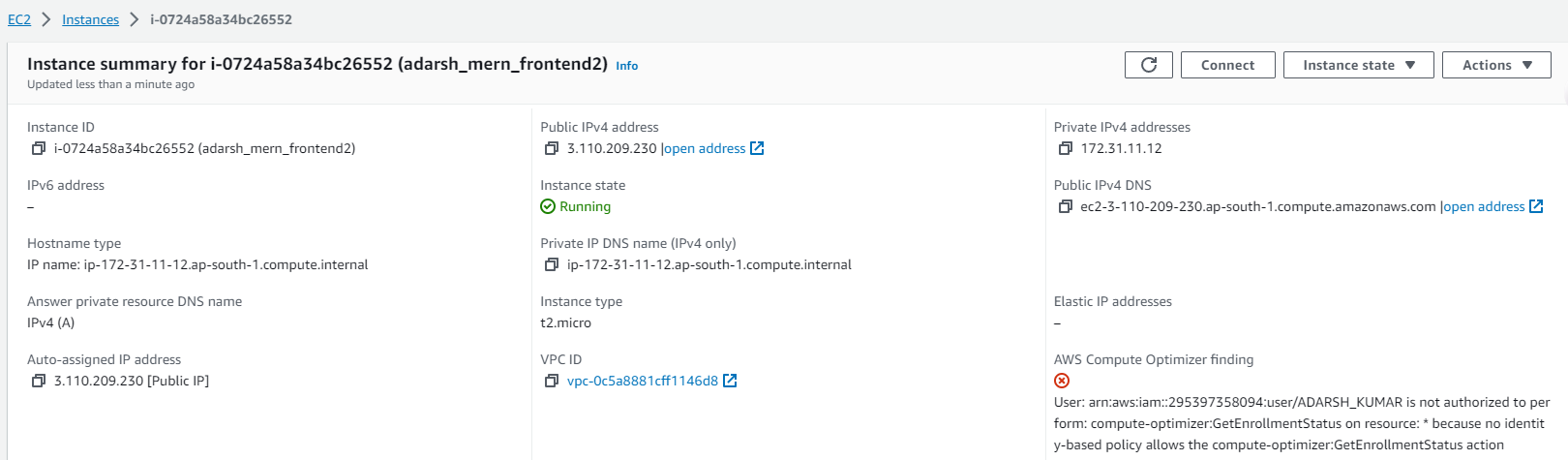


Then will be launching the instance using the AMIs we have saved earlier.

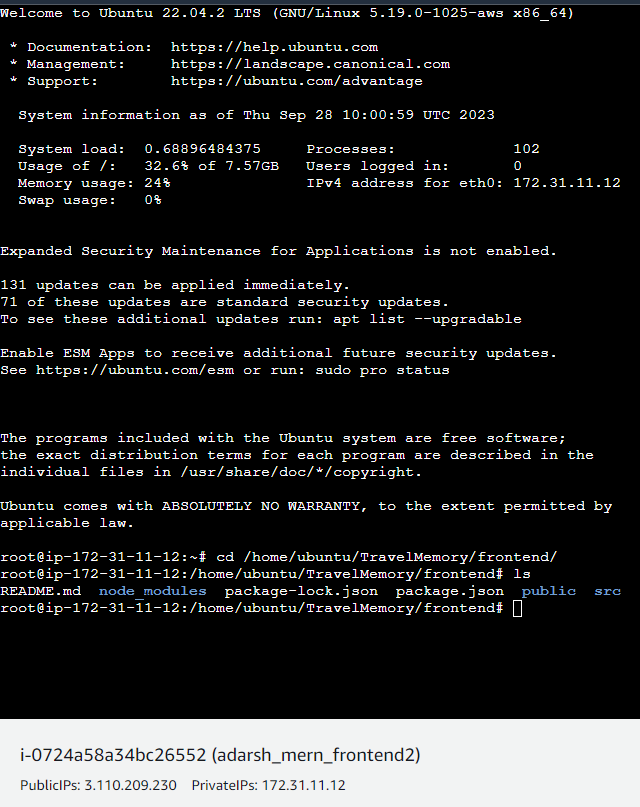






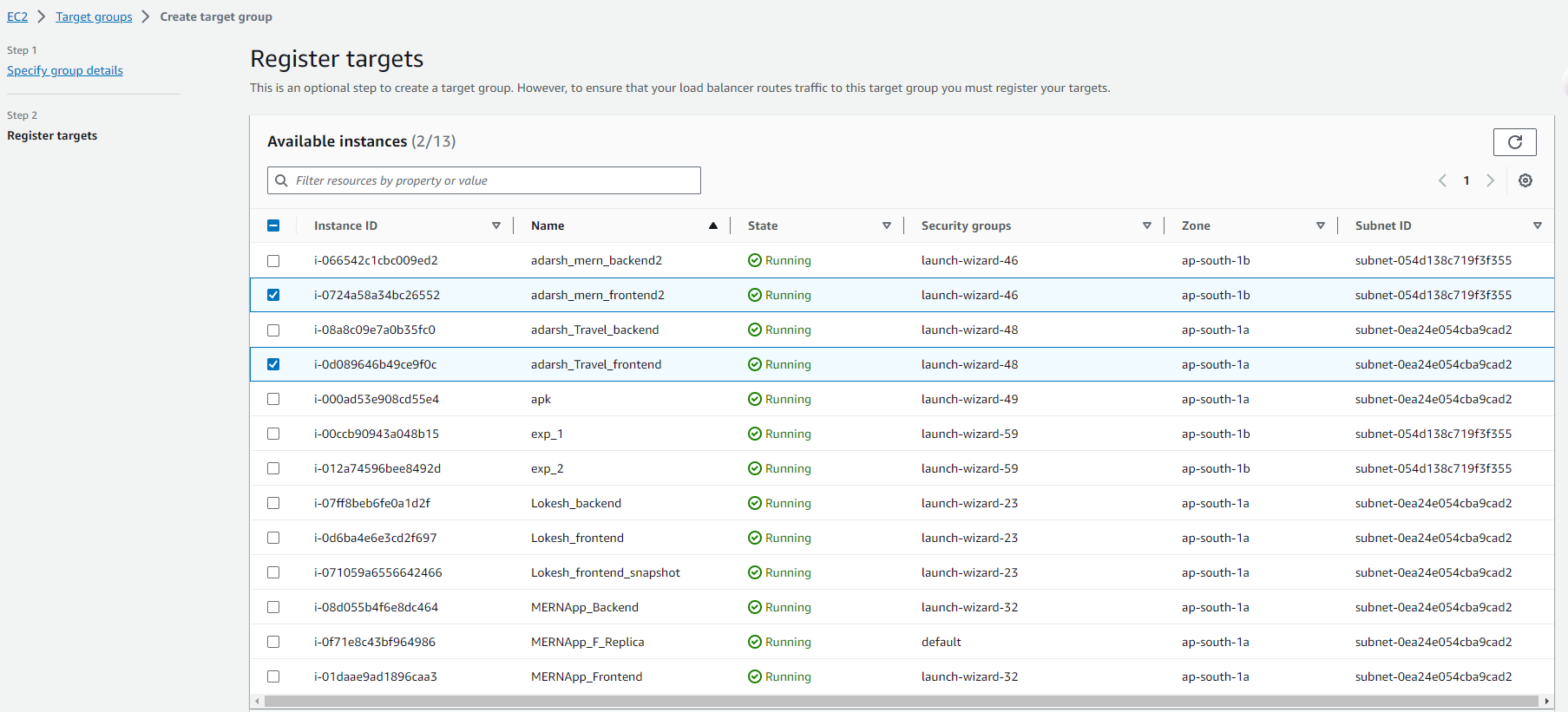


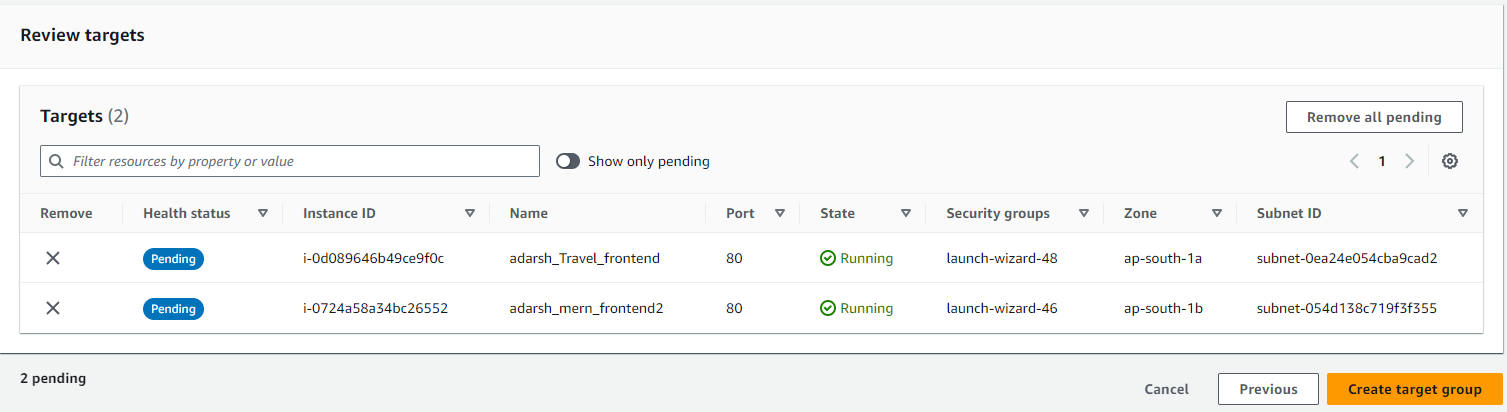
We are able to see the same data and configuration.

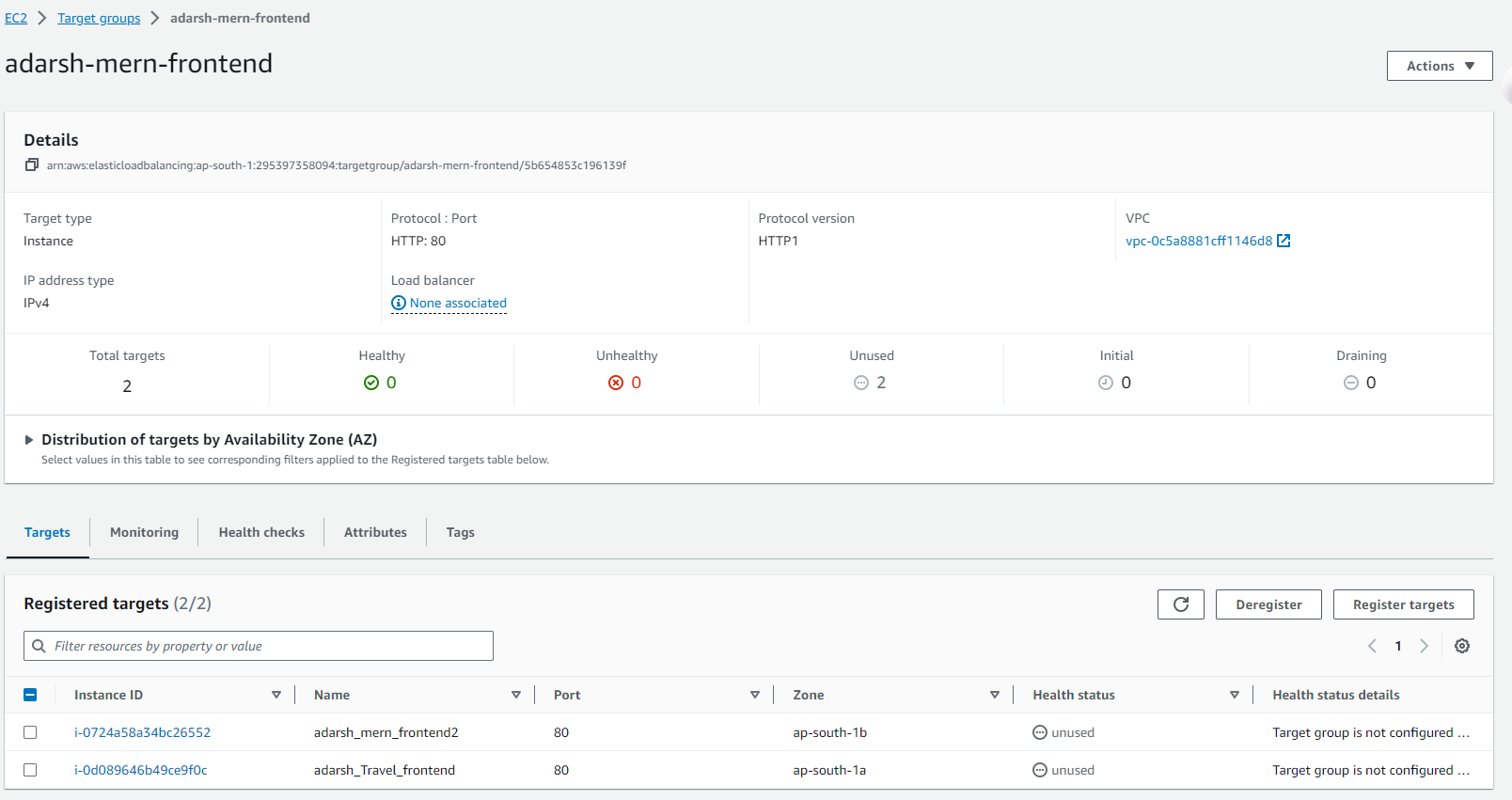


For Load balancing we need to create Target Groups to collect the instances of frontend ( i.e adarsh\_mern\_frontend & adarsh\_mern\_frontend2) and backend (adarsh\_mern\_backend & adarsh\_mern\_backend2)

EC2 > Target groups > Create target group

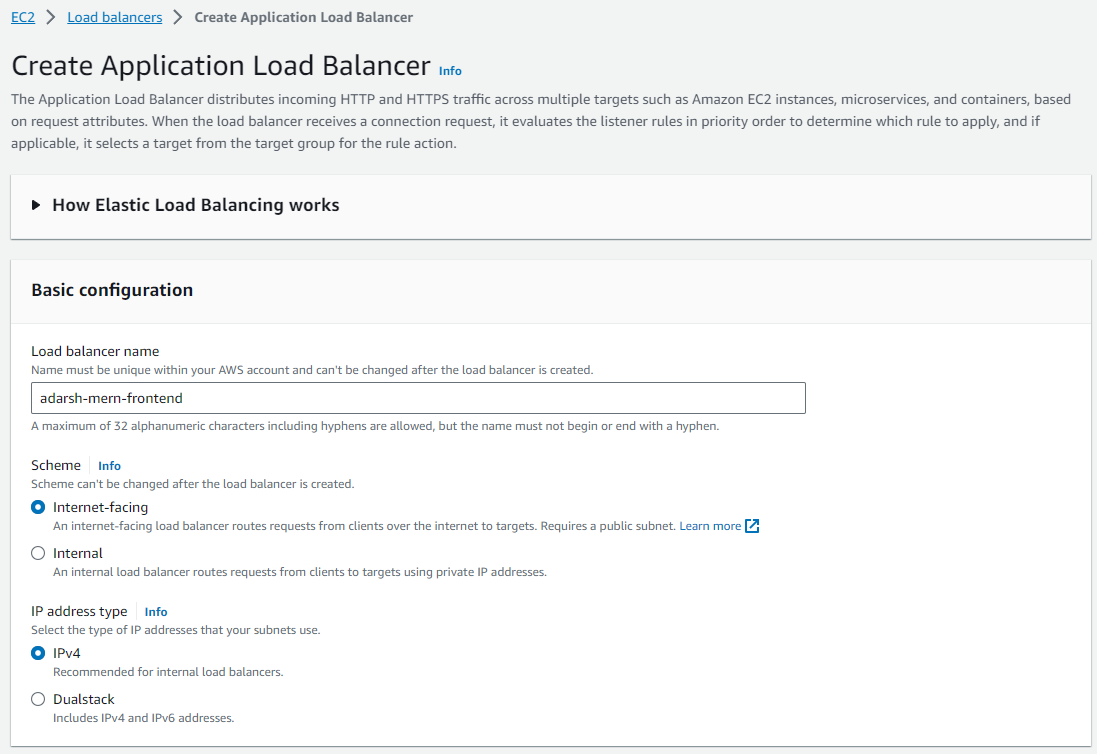




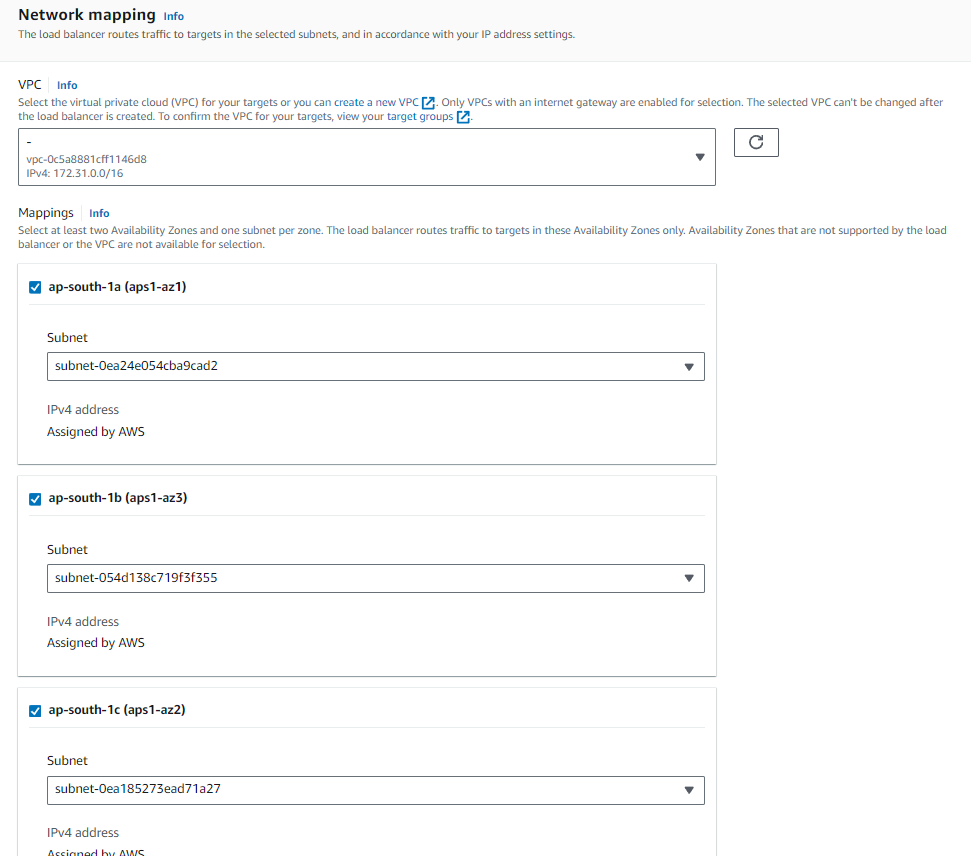


For creating the load balancer based on target groups

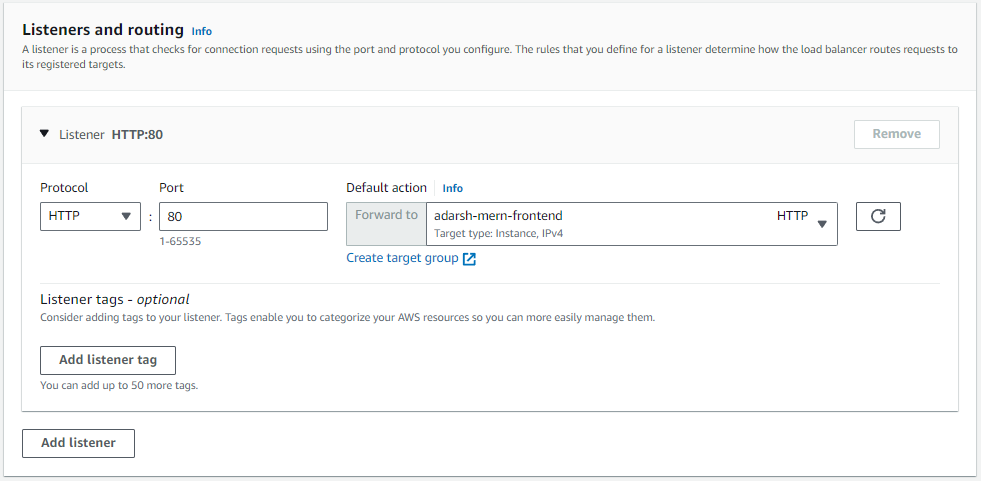
EC2 > Load balancers > Create Application Load Balancer

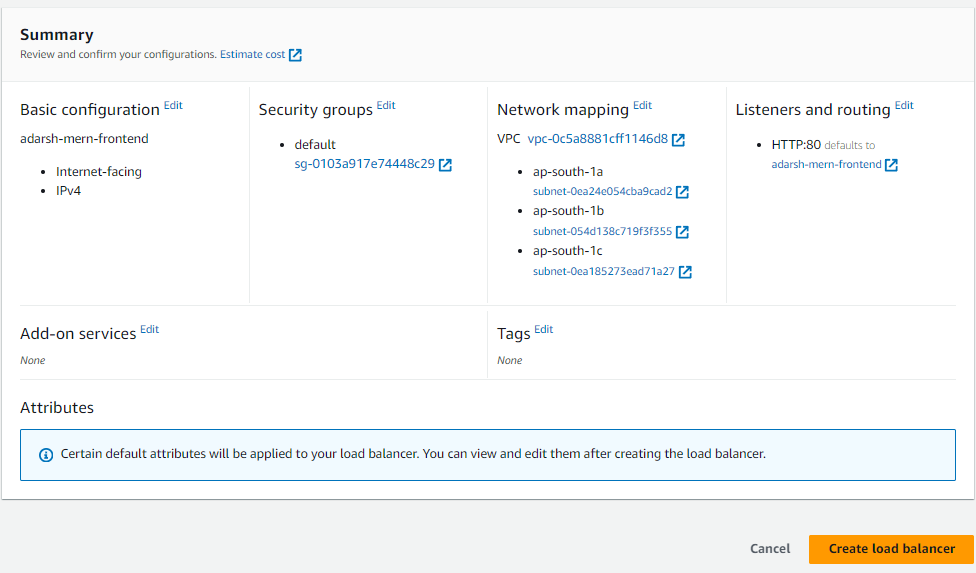


Make sure the Load balancer is mapped to multi AZs

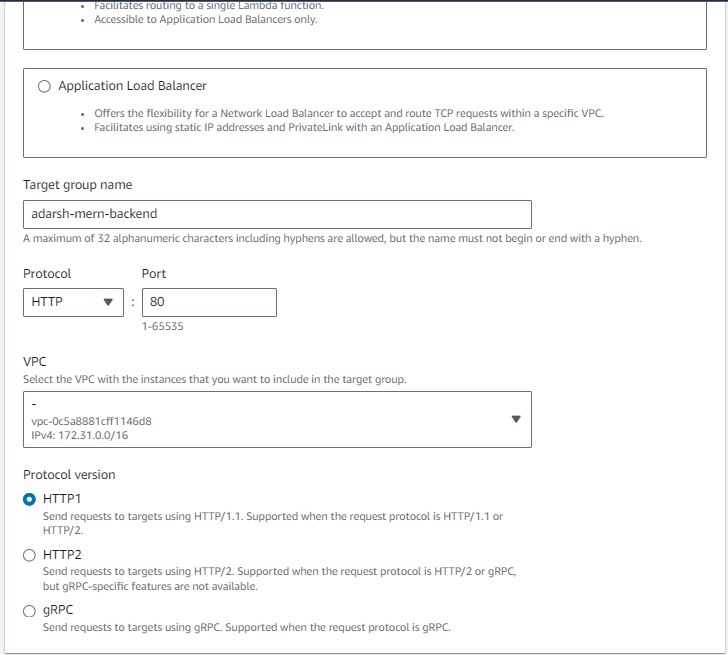


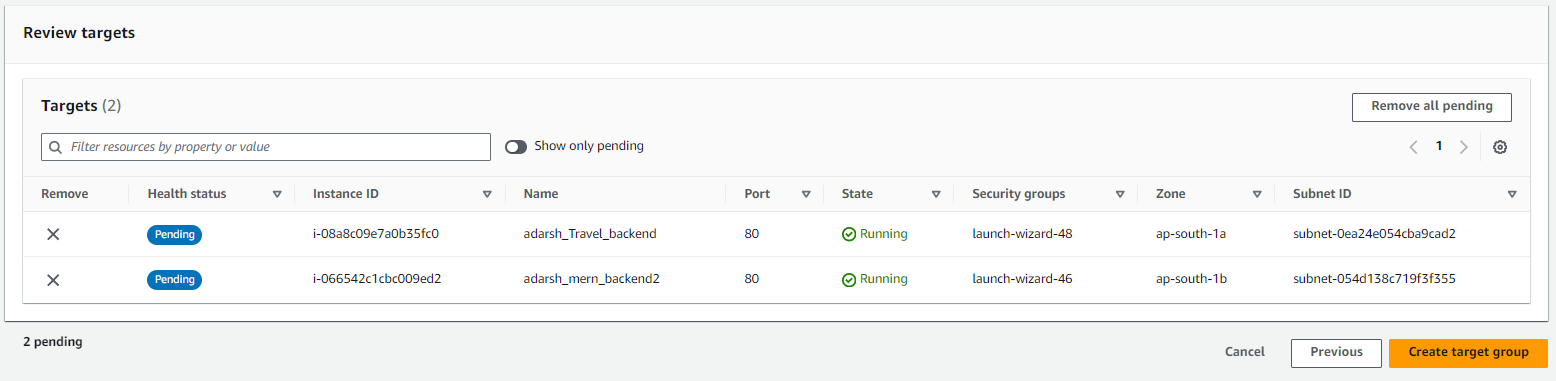
Adding the Target group we have created for the frontend instances.

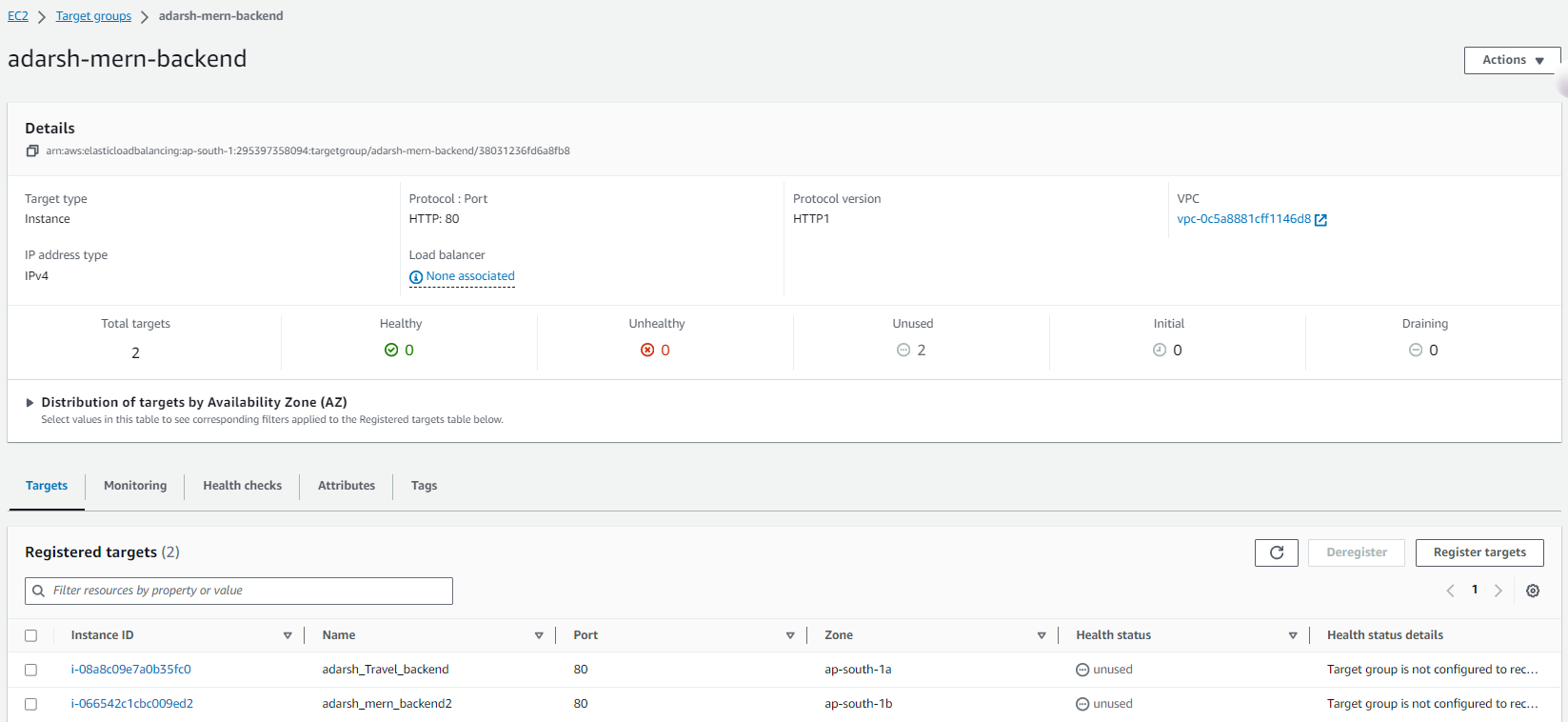


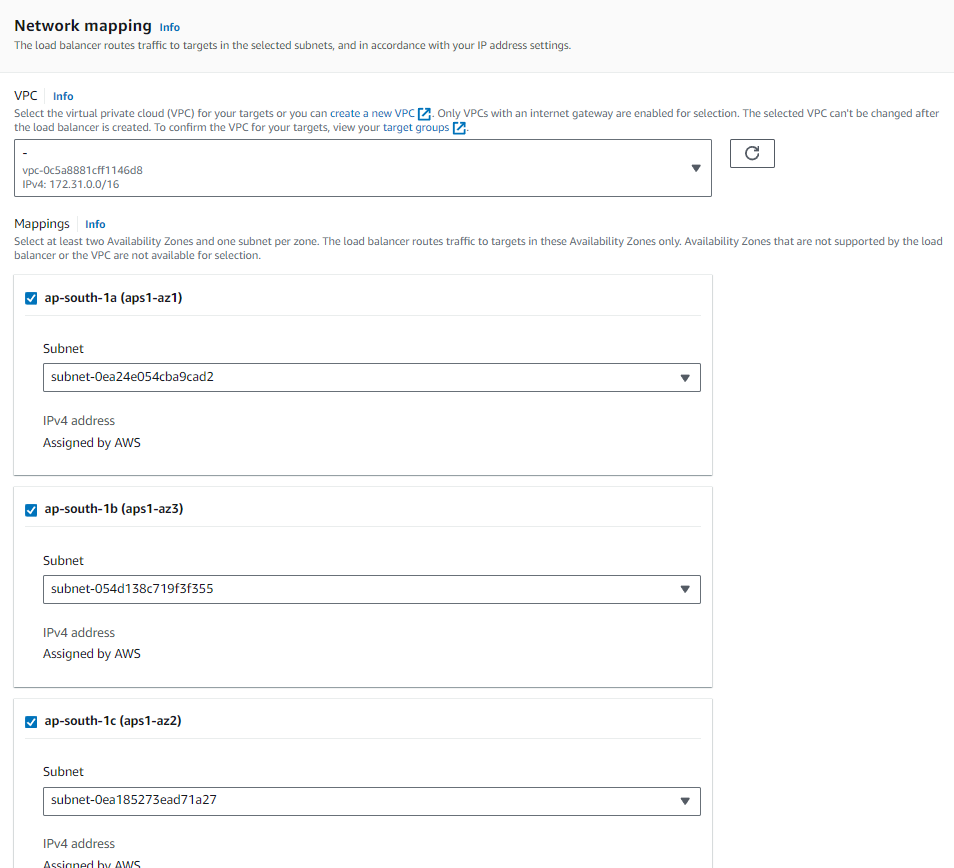


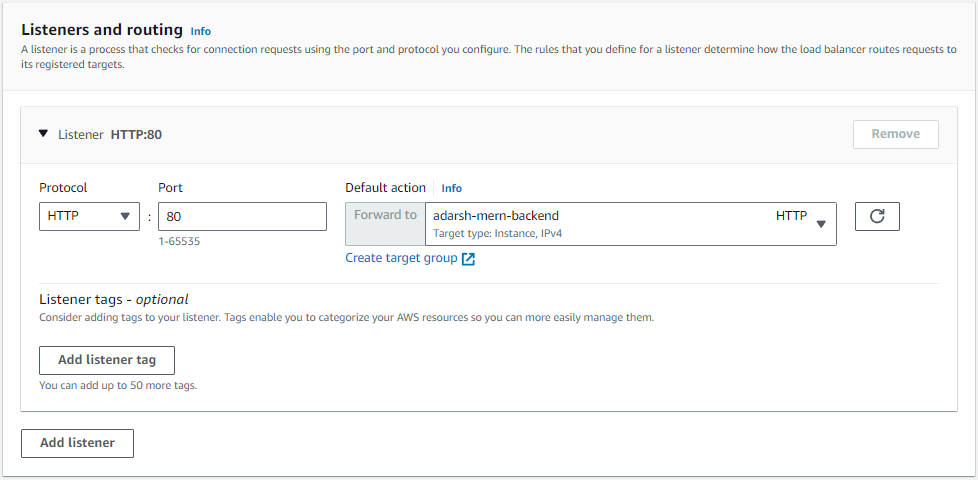
Same for the Backend Loadbalancer.

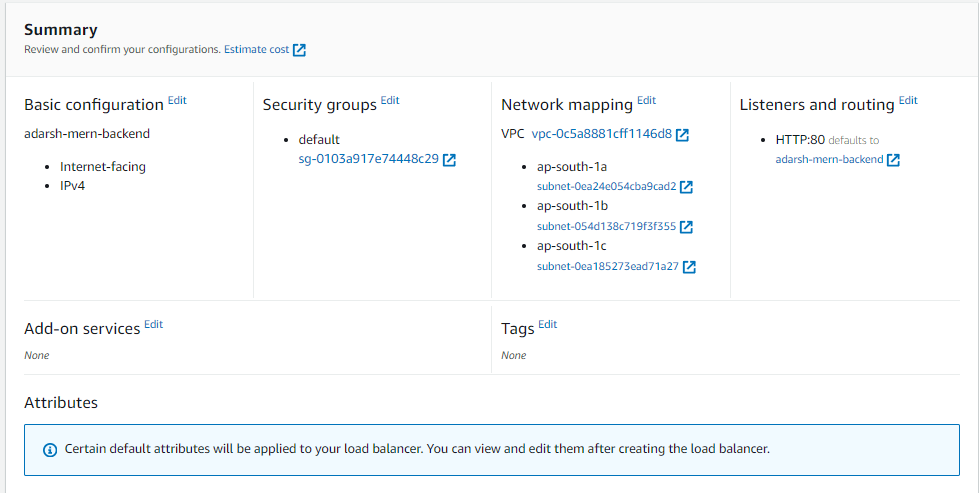




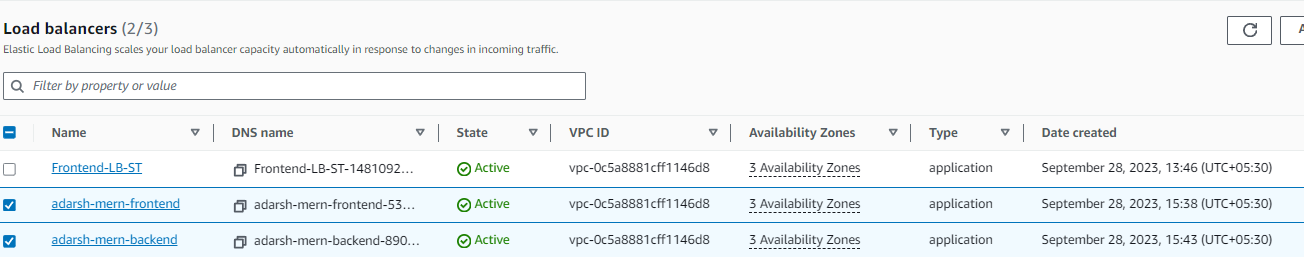








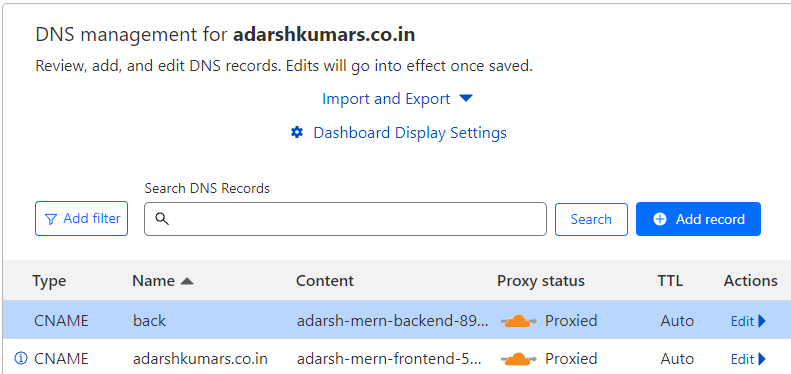
Both the Load Balancers are active.



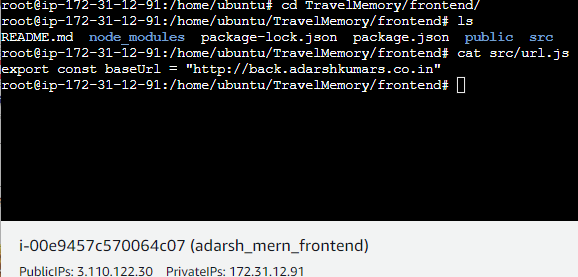
4. Domain Setup with Cloudflare:

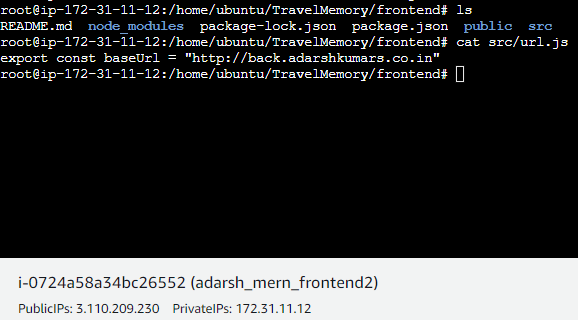
* Connect your custom domain to the application using Cloudflare.
* Create a CNAME record pointing to the load balancer endpoint.
* Set up an A record with the IP address of the EC2 instance hosting the frontend.

Used the domain [adarshkumars.co.in] to point towards the Frontend Load Balancer by adding a CNAME entry in Cloudflare and used the subdomain [back.adarshkumars.co.in].

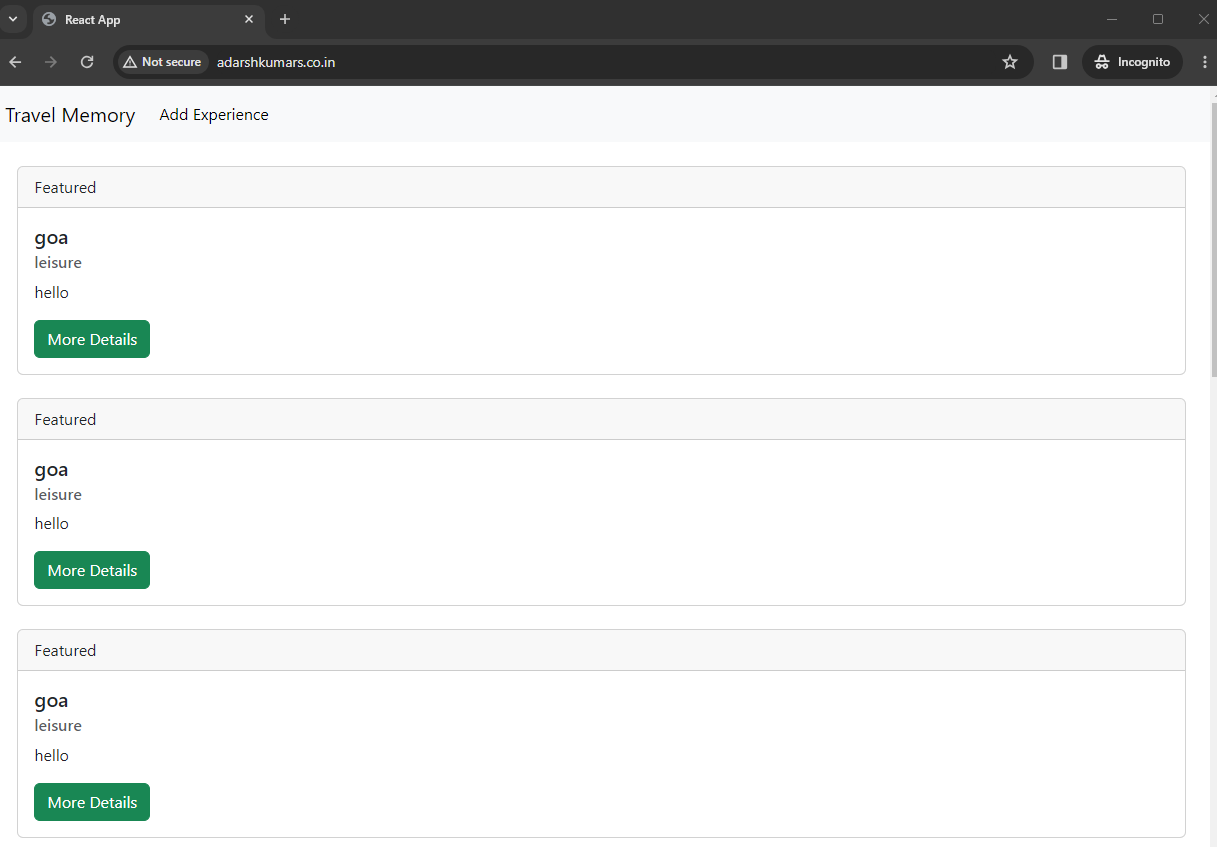


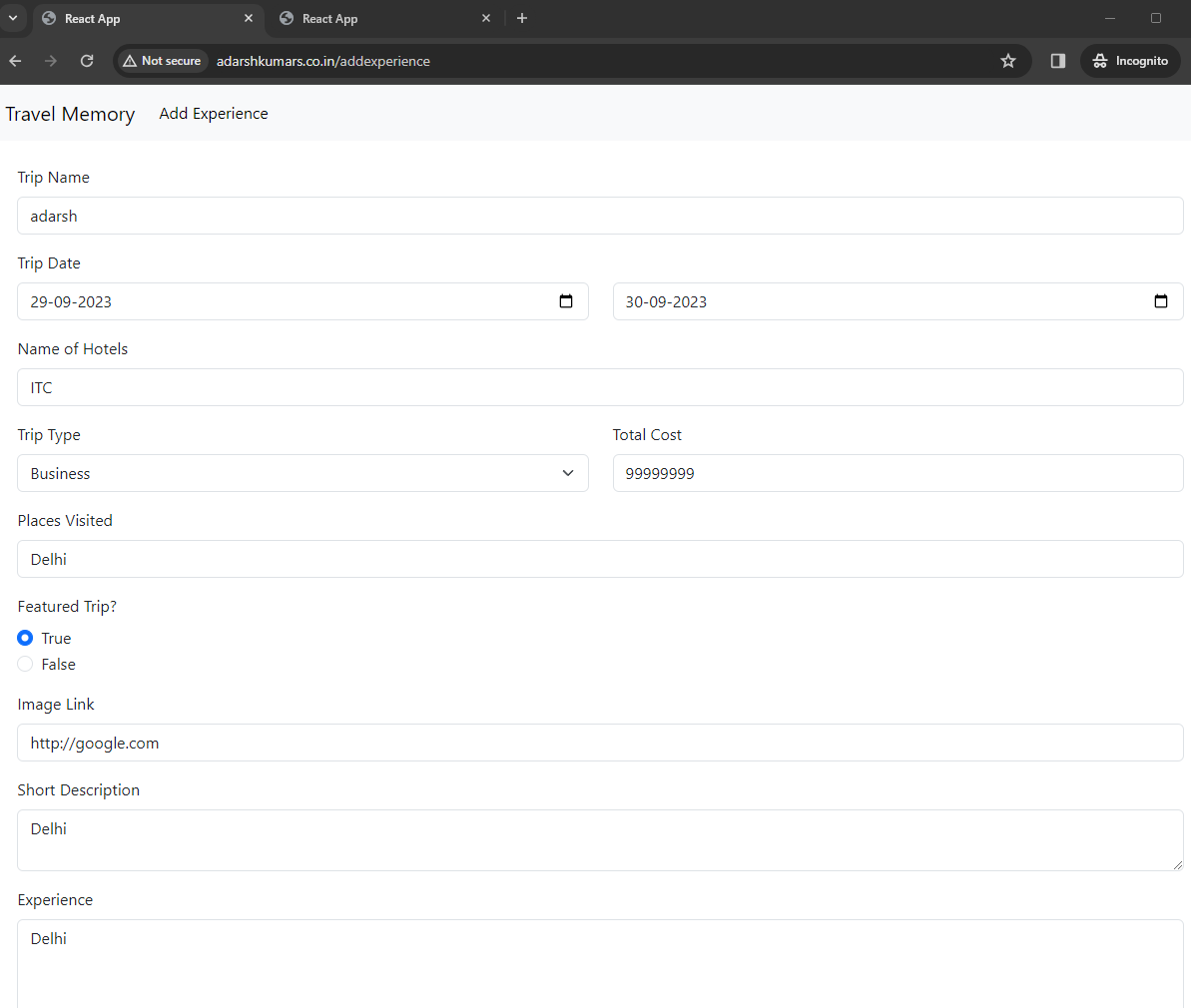
Now making the backend accessible to the subdomain we have created for both the backend and putting in url.js file of frontend instances.

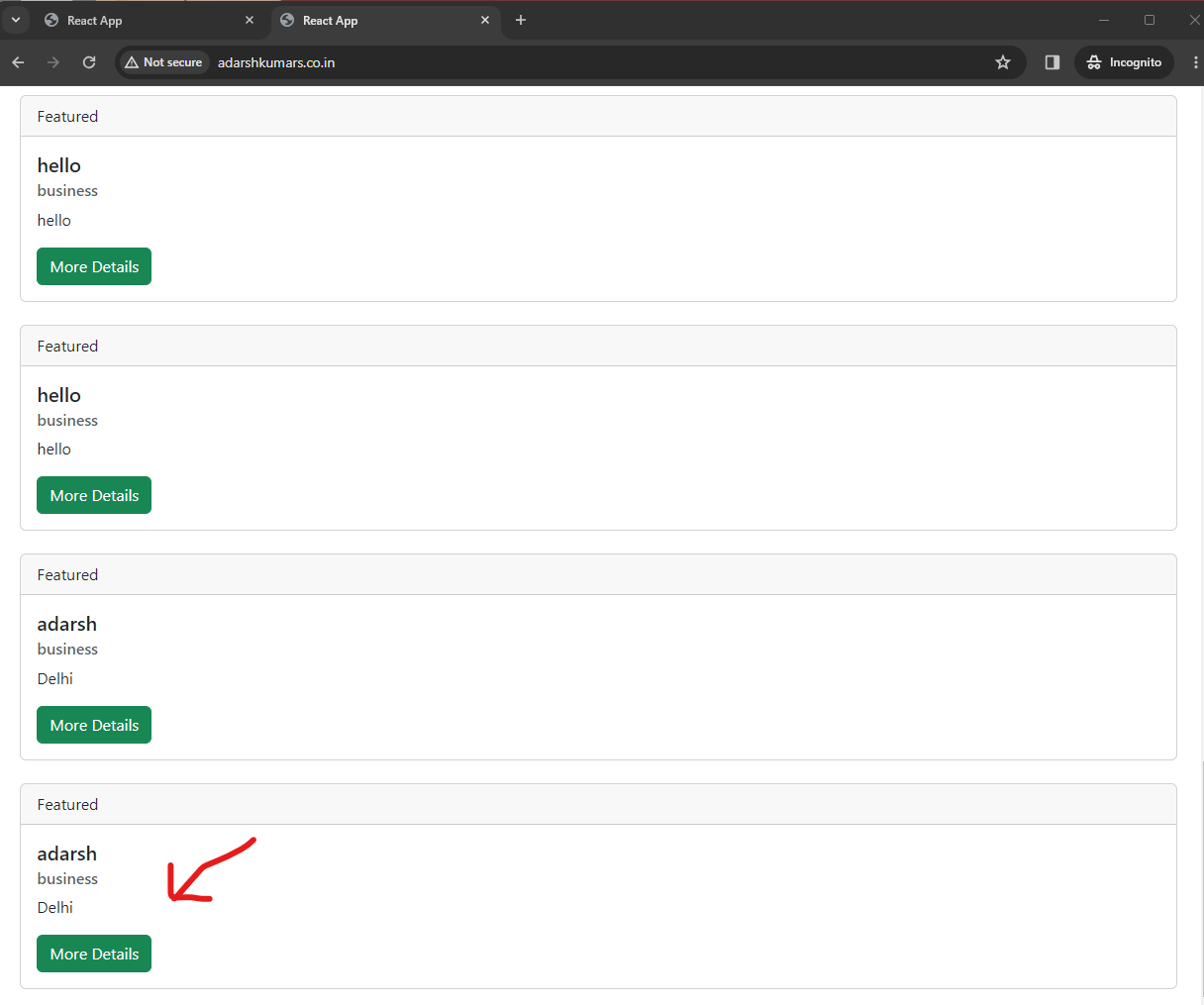




Lets run the application using domain name







The data get stored in the MongoDB database.

