# Graded Project

## on

# “Travel Memory” Application Deployment

**Course 6:** AWS Cloud Deployment | **DevOps B4**

SUBMITTED BY- **KINNAR CHOWDHURY, Hero Vired, DevOps, Batch 4**

Access the complete codebase of the TravelMemory application from the provided GitHub link: [GitHub - 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.

### 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.

### 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.

### 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.

### 5. Documentation:

- Prepare comprehensive documentation detailing each step of the deployment process. Include relevant screenshots to make the process clear and reproducible.
- Design a deployment architecture diagram using [draw.io](https://www.draw.io/) to visualize the flow and connections

**Expected Deliverables:**

1. A fully functional TravelMemory application hosted on an EC2 instance, accessible via a custom domain.
2. Detailed documentation of the deployment process with appropriate screenshots.
3. A deployment architecture diagram.

# TravelMemory Application Deployment

## Capture and cherish your travel adventures.

The **TravelMemory** application is one of the learning session materials that helped us in the learning process for DevOps from Hero Vired. Prashant Kumar Dey [Profile Link: https://github.com/UnpredictablePrashant] develops this application.
TravelMemory appears to be an application designed to help users document and relive their travel experiences. This app lets you create a digital journal of your trips, complete with photos, locations, expenses, and descriptions. You can even share your travel stories with others!

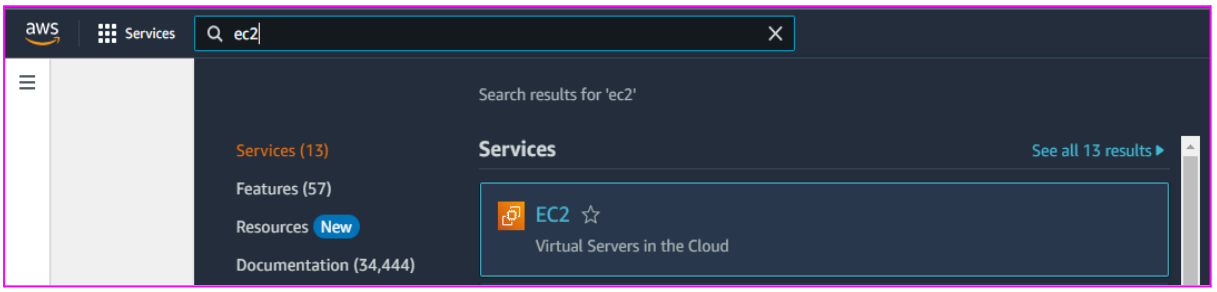
### Detailed information on the deployment process of the TravelMemory app is referenced below;

Initially, for this project, I have taken two EC2 machines from Amazon Web Services (AWS).

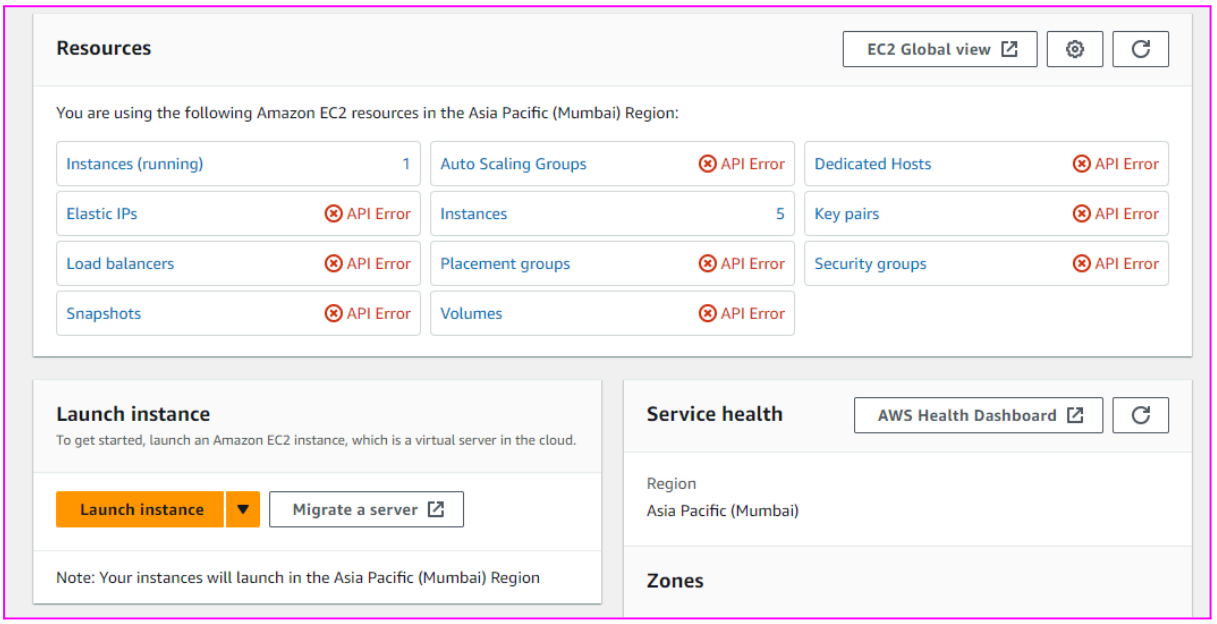
The process regarding commissioning an EC2 instance is mentioned as follows:

To set up an AWS EC2. first, you have to log in to your AWS account and then proceed to create an EC2 Instance

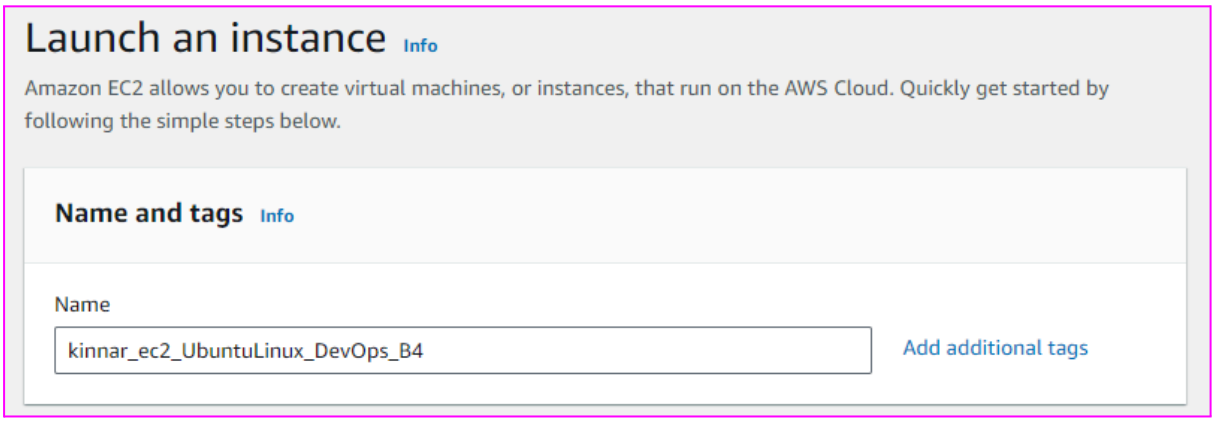
1. Search for EC2 in the AWS console dashboard and click on it.



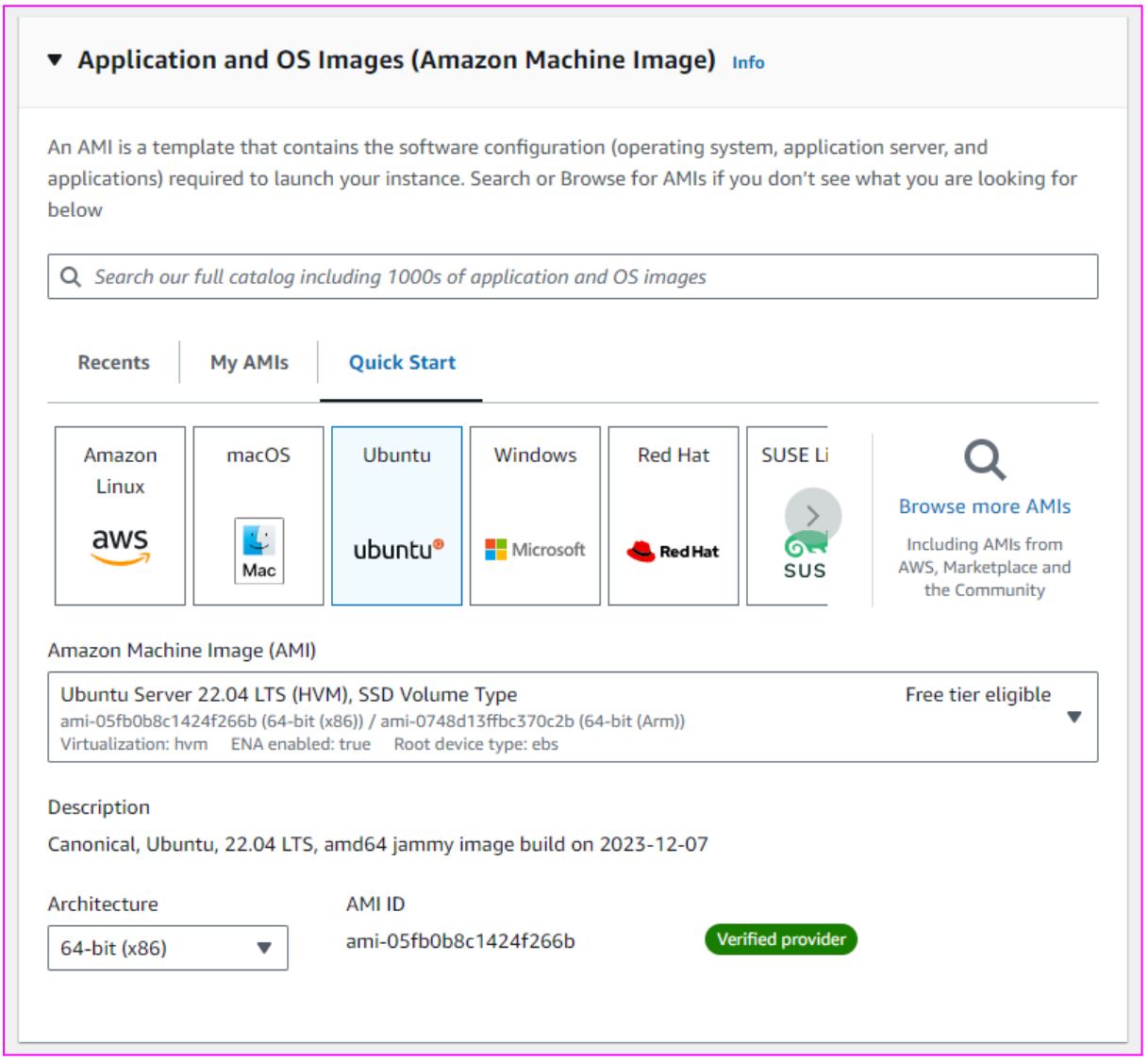

2. Click on Launch EC2 Instances



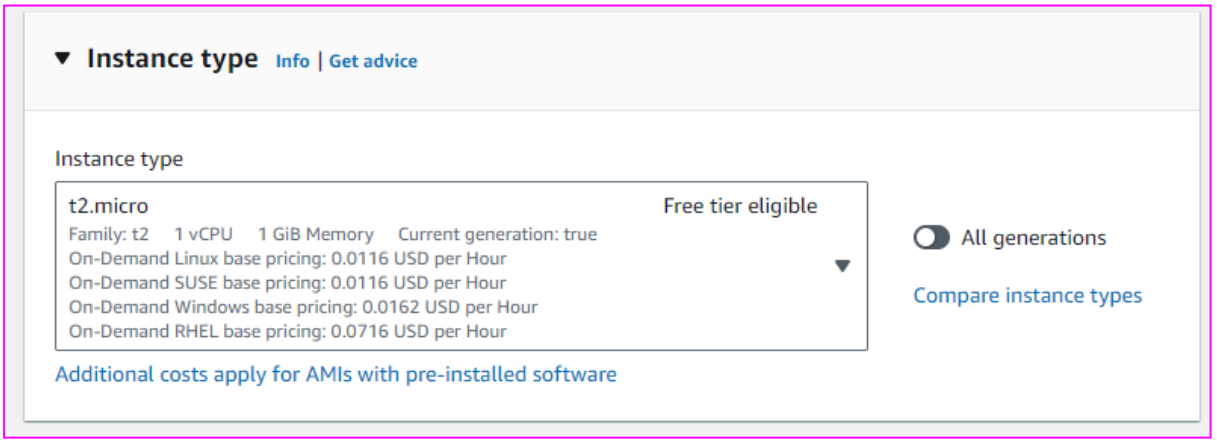

3. Provide Name and Tags.



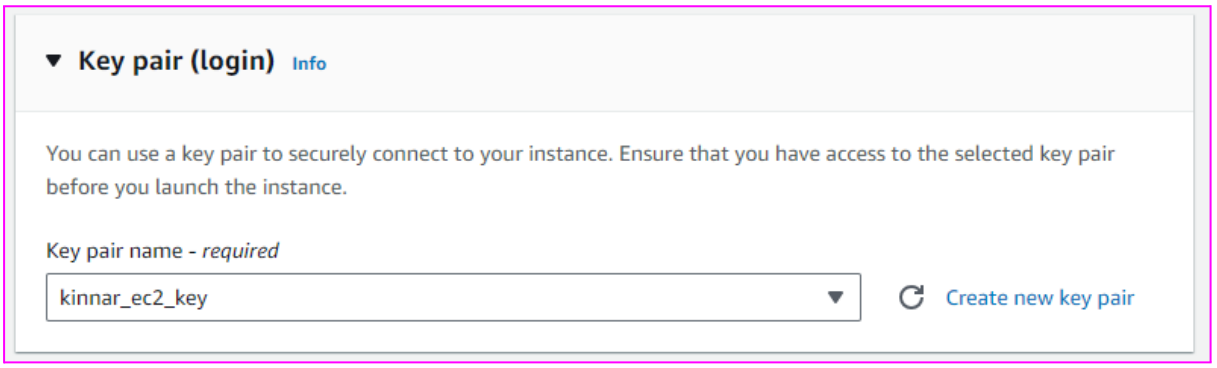
4. Choose an OS Image for Amazon Machine Image.

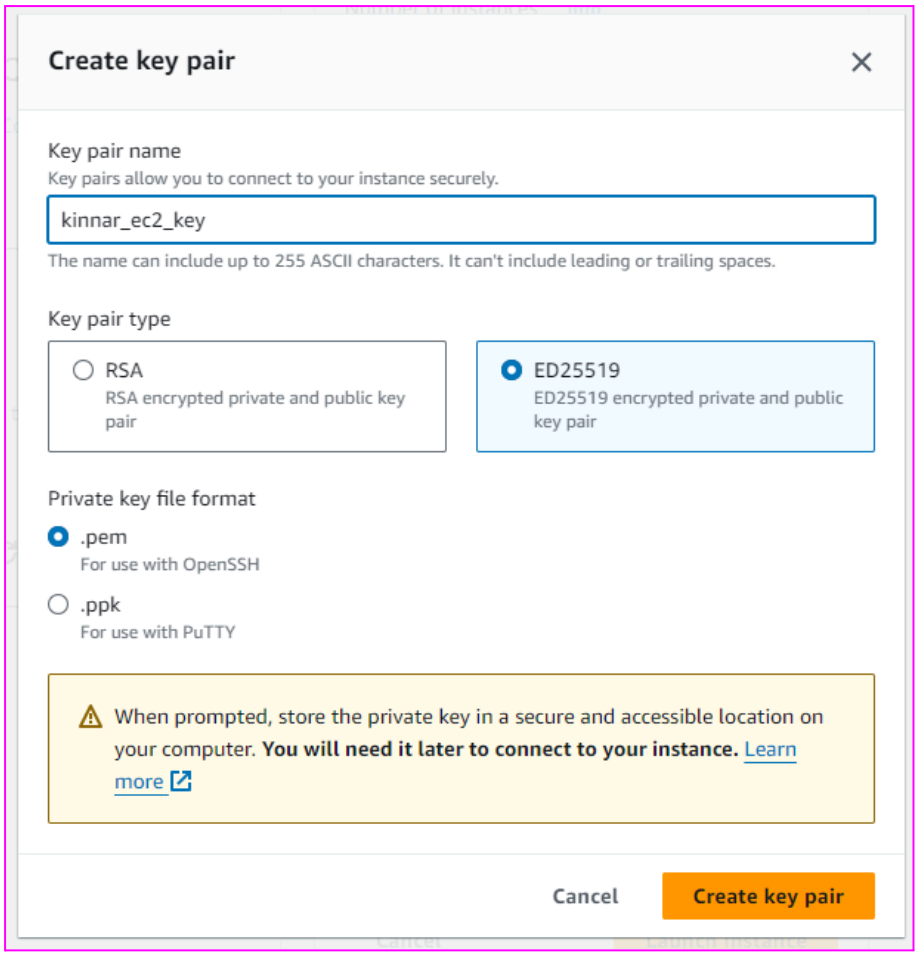



5. Choose Instance Type.

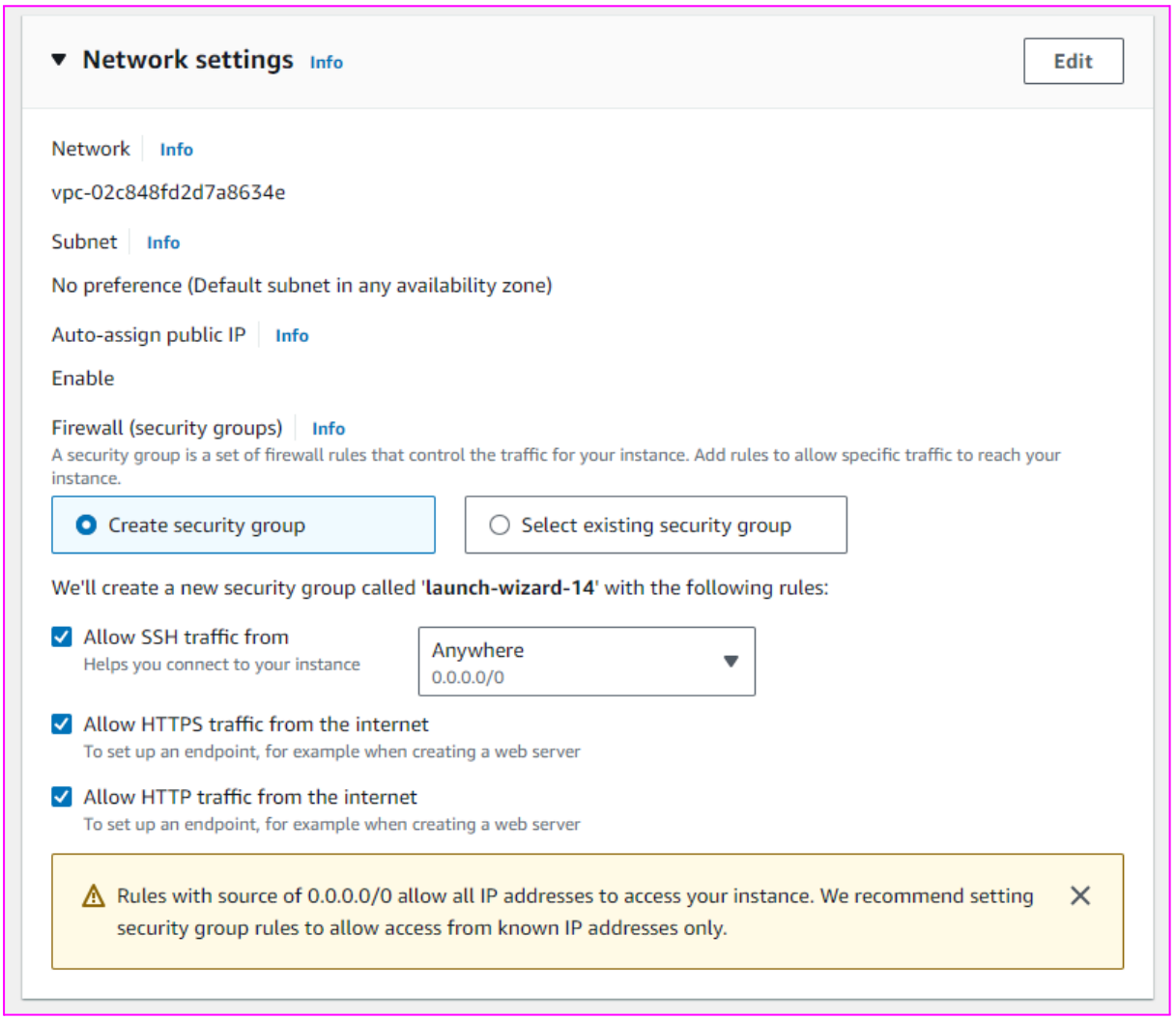


6. You can use an existing key pair or create an EC2 key pair

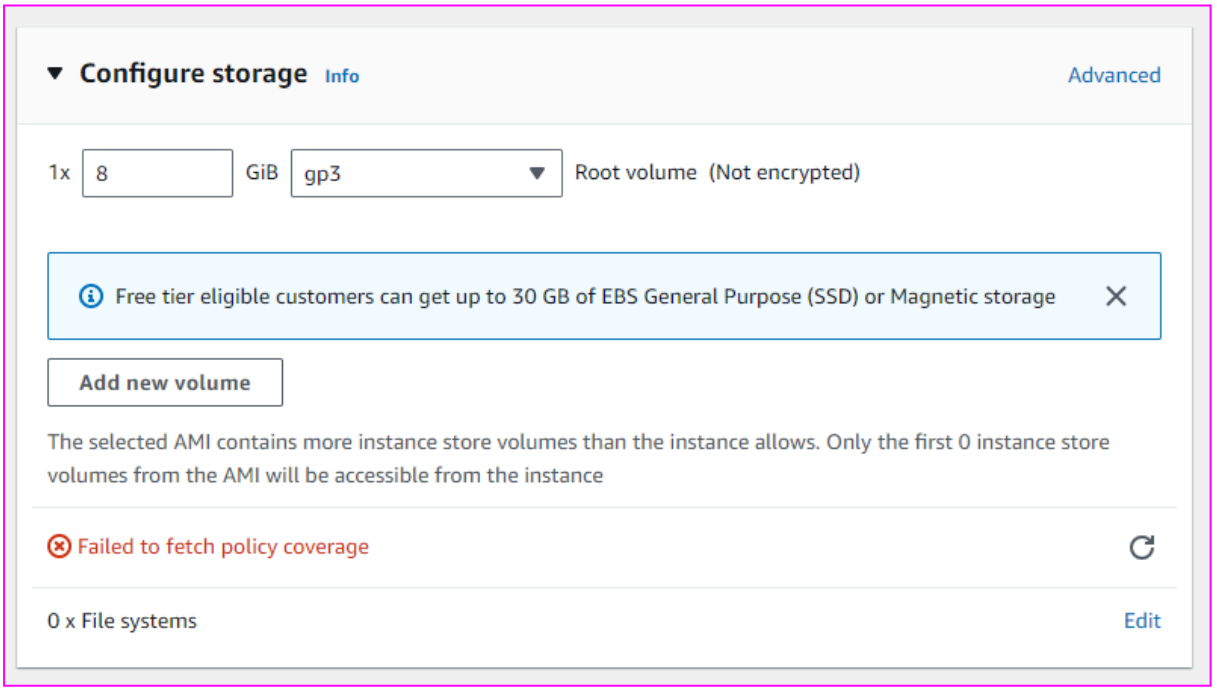


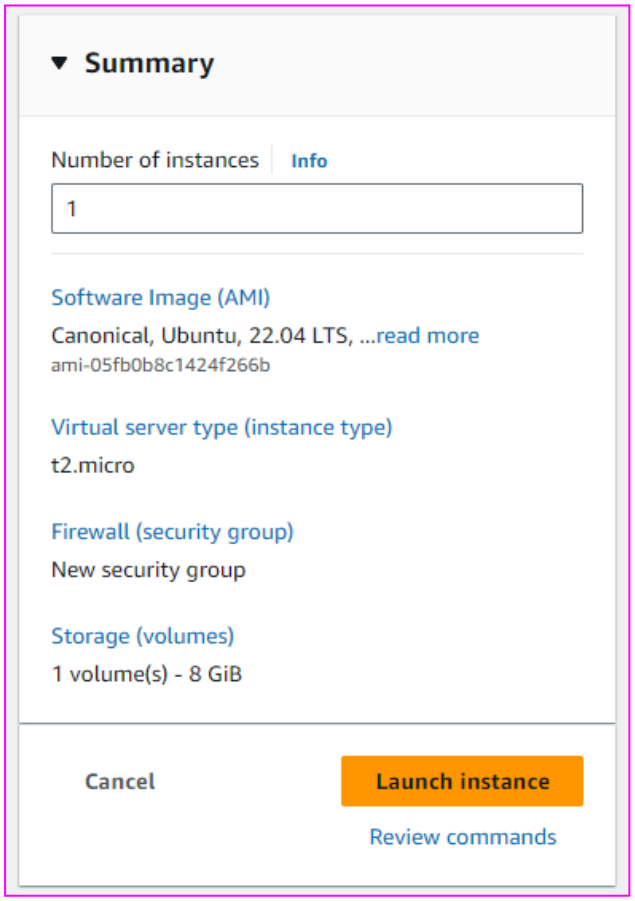
7. Set up Network settings as per your requirements.



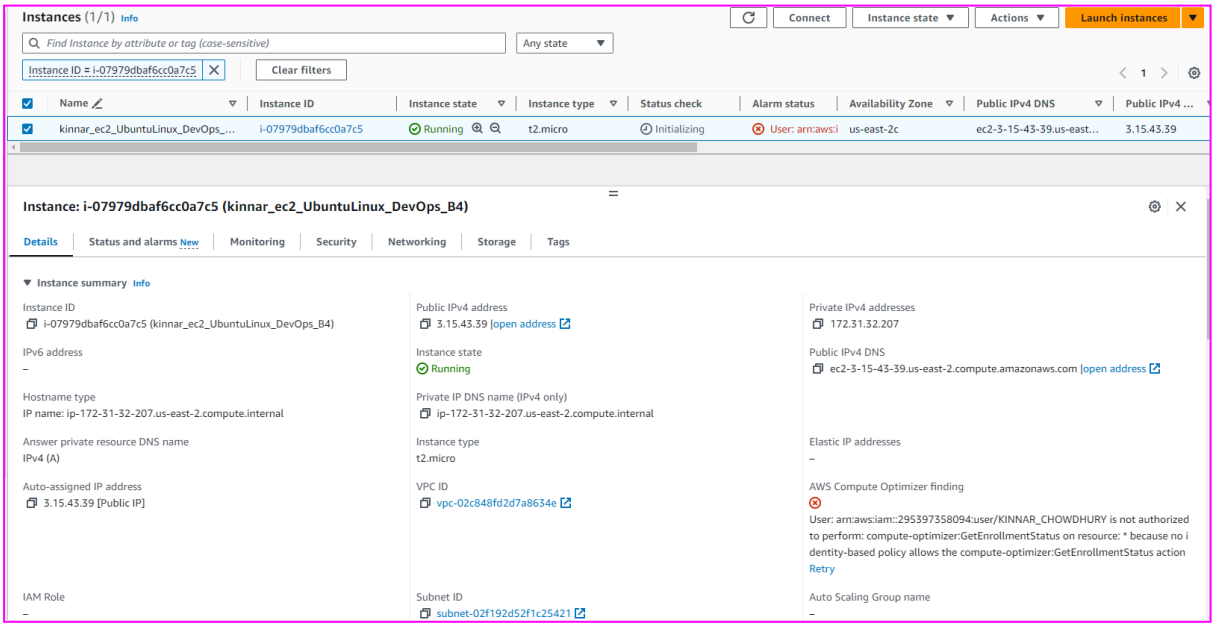

8. Allocate storage for your ec2



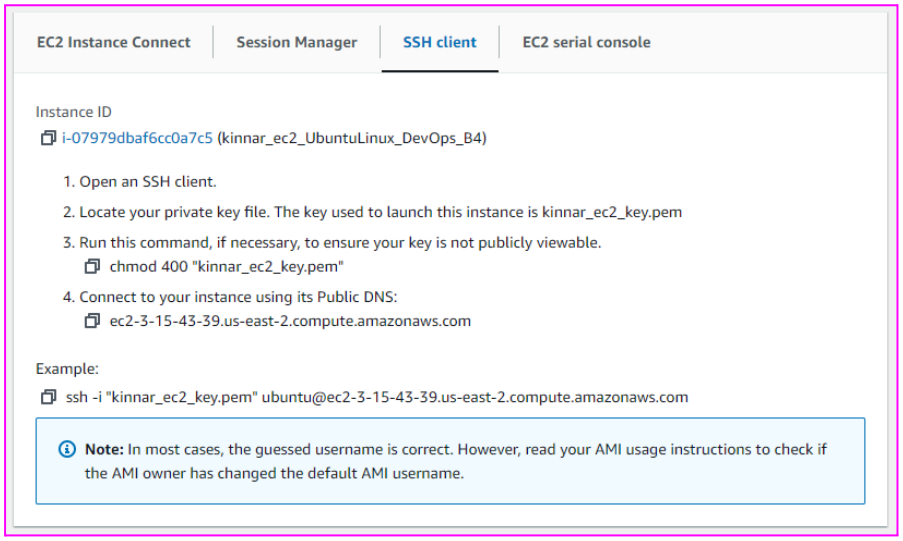
9. Now review your settings and click on Launch Instance and wait for some time to be provisioned by AWS.



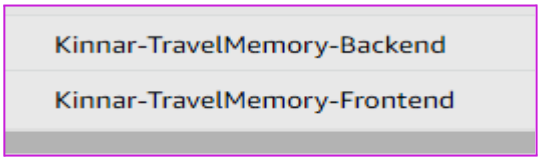

10. After successfully launching, Check the EC2 running dashboard to get the EC2 IP and other details.

11. Now connect to your AWS instance as follows.




So, for this project, I have taken two ec2 instances to deploy this application. One ec2 will have the backend deployment, and the other one will be used to host the frontend. Now, I have to set up the backend server that will be connected to the front end of the TravelMemory application.




## Application System Requirements

| | |
|---|---|
| Operating System | - Ubuntu Linux (22.04) |
| Application | - Node (Version 18.x) |
| Database | - Mongo DB |
| Web Server | - Nginx |

Now, we will install Node JS in the both servers. Please follow these steps sequentially:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install -y curl
$ curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
$ sudo apt install -y nodejs
After finish check node version:
$ node --version
```

```
   37  sudo apt update
   38  sudo apt upgrade
   39  sudo apt install -y curl
   40  curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
   41  sudo apt install -y nodejs
   42  node -v
   43  clear
   44  history
ubuntu@ip-172-31-7-122:~$ node --version
v18.20.2
ubuntu@ip-172-31-7-122:~$
```

**1. Backend Configuration:**

After logging into the ec2 here mentioned as Kinnar-TravelMemory-Backend through ssh, we will first update the apt repository of the ec2 instance.

```
$ sudo apt update
```

We can also upgrade the packages to the latest version.

```
$ sudo apt upgrade -y
```

As of now we already have accessed the complete codebase of the TravelMemory app from the provided GitHub link
https://github.com/UnpredictablePrashant/TravelMemory

```
$ git clone https://github.com/UnpredictablePrashant/TravelMemory.git
```

```
ubuntu@ip-172-31-7-122:~/travel_memory_backend$ git clone https://github.com/UnpredictablePrashant/TravelMemory.git
Cloning into 'TravelMemory'...
remote: Enumerating objects: 92, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 92 (delta 8), reused 5 (delta 5), pack-reused 70
Receiving objects: 100% (92/92), 217.75 KiB | 3.40 MiB/s, done.
Resolving deltas: 100% (23/23), done.
ubuntu@ip-172-31-7-122:~/travel_memory_backend$ ls
TravelMemory
ubuntu@ip-172-31-7-122:~/travel_memory_backend$ cd TravelMemory/backend/
```

Created an environment file (.env) in the below location:
/home/ubuntu/travel_memory_backend/TravelMemory/backend

```
$ cd /home/ubuntu/travel_memory_backend/TravelMemory/backend/
$ sudo nano .env
```

**MONGO_URI='mongodb+srv://development:xxxxx-kc@cluster0-kc.kodulgi.mongodb.net/travelmemory'**
**PORT=3000**

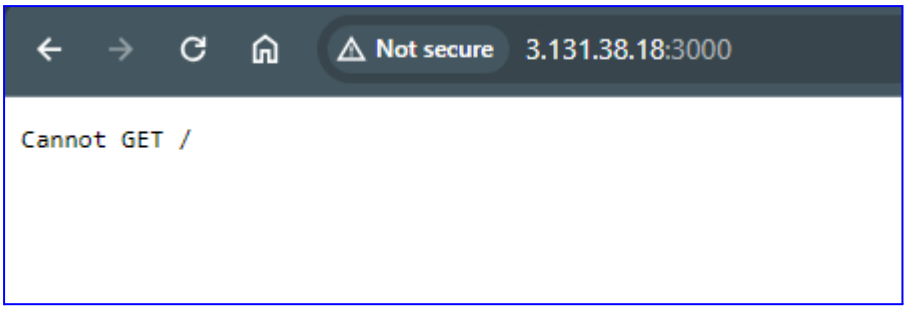
```
ubuntu@ip-172-31-7-122:~$ cd /home/ubuntu/travel_memory_backend/TravelMemory/backend/
ubuntu@ip-172-31-7-122:~/travel_memory_backend/TravelMemory/backend$ pwd
/home/ubuntu/travel_memory_backend/TravelMemory/backend
ubuntu@ip-172-31-7-122:~/travel_memory_backend/TravelMemory/backend$ ls -a
.  ..  .env  conn.js  controllers  index.js  models  package-lock.json  package.json  routes
ubuntu@ip-172-31-7-122:~/travel_memory_backend/TravelMemory/backend$ sudo nano .env
ubuntu@ip-172-31-7-122:~/travel_memory_backend/TravelMemory/backend$ cat .env
MONGO_URI='mongodb+srv://development:Dev2024-kc@cluster0-kc.kodulgi.mongodb.net/travelmemory'
PORT=3000
ubuntu@ip-172-31-7-122:~/travel_memory_backend/TravelMemory/backend$
```

After the successful installation of node version 18, we will proceed for the backend server to be ready for use. For this process, we will simply move to the backend directory located at **/home/ubuntu/travel_memory_backend/TravelMemory/backend/** and install the required dependencies using Node Package Manager from **package.json**.

```
$ cd /home/ubuntu/travel_memory_backend/TravelMemory/backend/
$ npm install
$ sudo node index.js
```

```
ubuntu@ip-172-31-7-122:~/travel_memory_backend/TravelMemory/backend$ sudo node index.js
Server started at http://localhost:3000
```

Now check the application in the browser:




**Note:** If you are seeing this page, it means the backend application is running healthy.

Now We need to configure a reverse proxy to ensure smooth running of application deployment. So, we install nginx in the backend server and configure a reverse proxy in the /etc/nginx/sites-available/default file. We have added proxy_pass method in the location directives in the default file.

```
location / {
    proxy_pass http://127.0.0.1:3001/;
}
```

Another Twick:- sudo nohup node index.js > /home/ubuntu/travel_memory_backend/backend.log &

**2. Frontend Configuration:**

After logging into the ec2, here mentioned as Kinnar-TravelMemory-Frontend through ssh, we will first update the apt repository of the ec2 instance.

```
$ sudo apt update
```

We can also upgrade the packages to the latest version.

```
$ sudo apt upgrade -y
```

As of now we already have accessed the complete codebase of the TravelMemory app from the provided GitHub link
https://github.com/UnpredictablePrashant/TravelMemory

```
$ git clone https://github.com/UnpredictablePrashant/TravelMemory.git
```

```
ubuntu@ip-172-31-7-122:~/travel_memory_backend$ git clone https://github.com/UnpredictablePrashant/TravelMemory.git
Cloning into 'TravelMemory'...
remote: Enumerating objects: 92, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 92 (delta 8), reused 5 (delta 5), pack-reused 70
Receiving objects: 100% (92/92), 217.75 KiB | 3.40 MiB/s, done.
Resolving deltas: 100% (23/23), done.
ubuntu@ip-172-31-7-122:~/travel_memory_backend$ ls
TravelMemory
ubuntu@ip-172-31-7-122:~/travel_memory_backend$ cd TravelMemory/backend/
```

We need to modify url.js file inside src folder in the below location:
/home/ubuntu/travel_memory_backend/TravelMemory/frontend/

```
$ cd /home/ubuntu/travel_memory_frontend/TravelMemory/frontend/
$ cd src && sudo nano url.js
```

**export const baseUrl = "http://3.139.60.175:3001"**

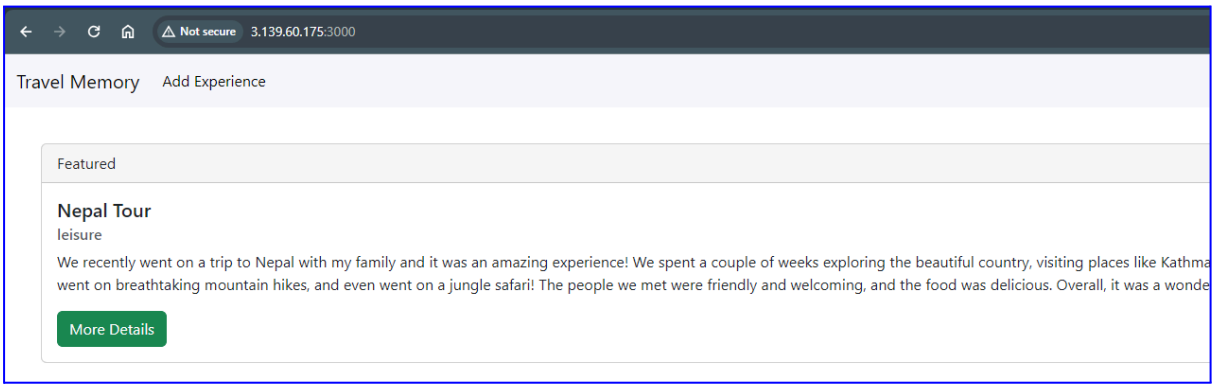
```
ubuntu@ip-172-31-34-120:~$ cd /home/ubuntu/travel_memory_frontend/TravelMemory/frontend/
ubuntu@ip-172-31-34-120:~/travel_memory_frontend/TravelMemory/frontend$ cd src/ && sudo nano url.js
ubuntu@ip-172-31-34-120:~/travel_memory_frontend/TravelMemory/frontend/src$ ls
App.css  App.js  App.test.js  components  index.css  index.js  logo.svg  reportWebVitals.js  setupTests.js  url.js
ubuntu@ip-172-31-34-120:~/travel_memory_frontend/TravelMemory/frontend/src$ cd ..
ubuntu@ip-172-31-34-120:~/travel_memory_frontend/TravelMemory/frontend$ ls
README.md  node_modules  package-lock.json  package.json  public  src
ubuntu@ip-172-31-34-120:~/travel_memory_frontend/TravelMemory/frontend$ cd src/
ubuntu@ip-172-31-34-120:~/travel_memory_frontend/TravelMemory/frontend/src$ cat url.js
export const baseUrl = "http://52.14.109.168:3001"
ubuntu@ip-172-31-34-120:~/travel_memory_frontend/TravelMemory/frontend/src$ cd ..
ubuntu@ip-172-31-34-120:~/travel_memory_frontend/TravelMemory/frontend$
```

After the successful installation of node version 18, we will proceed for the backend server to be ready for use. For this process, we will simply move to the backend directory located at **/home/ubuntu/travel_memory_frontend/TravelMemory/frontend/** and install the required dependencies using Node Package Manager from **package.json**.

```
$ cd /home/ubuntu/travel_memory_frontend/TravelMemory/frontend/
$ npm install
$ sudo npm start
```

```
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3001
  On Your Network:  http://172.31.34.120:3001

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Now check the application in the browser:




**Note:** If you are seeing this page, it means the frontend application is running healthy.

Now We need to configure another reverse proxy for the frontend application to ensure smooth running of application deployment. So, we install nginx in the frontend server and configure a reverse proxy in the /etc/nginx/sites-available/default file. We have added proxy_pass method in the location directives in the default file.

```
location / {
    proxy_pass http://127.0.0.1:3000/;
}
```

Another Twick:- sudo nohup npm start > /home/ubuntu/travel_memory_frontend/frontend.log &

## 3. Scaling the Application:

We have registered two new application load balancers for Backend & Frontend to scale up the application.

| Name | DNS name | State | VPC ID | Availability Zones | Type | Date created |
|---|---|---|---|---|---|---|
| Kinnar-TravelMemory-Frontend-LB | Kinnar-TravelMemory-Fron... | Active | vpc-7faf5614 | 3 Availability Zones | application | April 27, 2024, 13:42 (UTC+05:30) |
| Kinnar-TravelMemory-Backend-LB | Kinnar-TravelMemory-Bac... | Active | vpc-7faf5614 | 3 Availability Zones | application | April 27, 2024, 13:31 (UTC+05:30) |

## Kinnar-TravelMemory-Frontend-LB

Kinnar-TravelMemory-Frontend-LB

Actions

**Details**

Load balancer type: Application
Status: Active
VPC: vpc-7faf5614
IP address type: IPv4
Scheme: Internet-facing
Hosted zone: Z3AADJGX6KTTL2
Availability Zones: subnet-6c683716 us-east-2b (use2-az2); subnet-b00483fc us-east-2c (use2-az3); subnet-63fa1b08 us-east-2a (use2-az1)
Date created: April 27, 2024, 13:42 (UTC+05:30)
Load balancer ARN: arn:aws:elasticloadbalancing:us-east-2:515210271098:loadbalancer/app/Kinnar-TravelMemory-Frontend-LB/13d4426c25279a01
DNS name Info: Kinnar-TravelMemory-Frontend-LB-1191351255.us-east-2.elb.amazonaws.com (A Record)

Listeners and rules | Network mapping | Resource map - new | Security | Monitoring | Integrations | Attributes | Tags

**Listeners and rules** (1) Info

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

Manage rules | Manage listener | Add listener

Filter listeners

| Protocol:Port | Default action | Rules | ARN | Security policy | Default SSL/TLS certificate | mTLS | Trust store | Tags |
|---|---|---|---|---|---|---|---|---|
| HTTP:80 | Forward to target group<br>• Kinnar-TravelMemory-Frontend-TG: 1 (100%)<br>• Group-level stickiness: Off | 1 rule | ARN | Not applicable | Not applicable | Not applicable | Not applicable | 2 tag |

## Kinnar-TravelMemory-Frontend-TG

Kinnar-TravelMemory-Frontend-TG

Actions

**Details**

arn:aws:elasticloadbalancing:us-east-2:515210271098:targetgroup/Kinnar-TravelMemory-Frontend-TG/404f92b4d61a3851

Target type: Instance
Protocol : Port: HTTP: 80
Protocol version: HTTP1
VPC: vpc-7faf5614
IP address type: IPv4
Load balancer: Kinnar-TravelMemory-Frontend-LB

| Total targets | Healthy | Unhealthy | Unused | Initial | Draining |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 |

0 Anomalous

**Distribution of targets by Availability Zone (AZ)**
Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets | Monitoring | Health checks | Attributes | Tags

**Registered targets** (1) Info

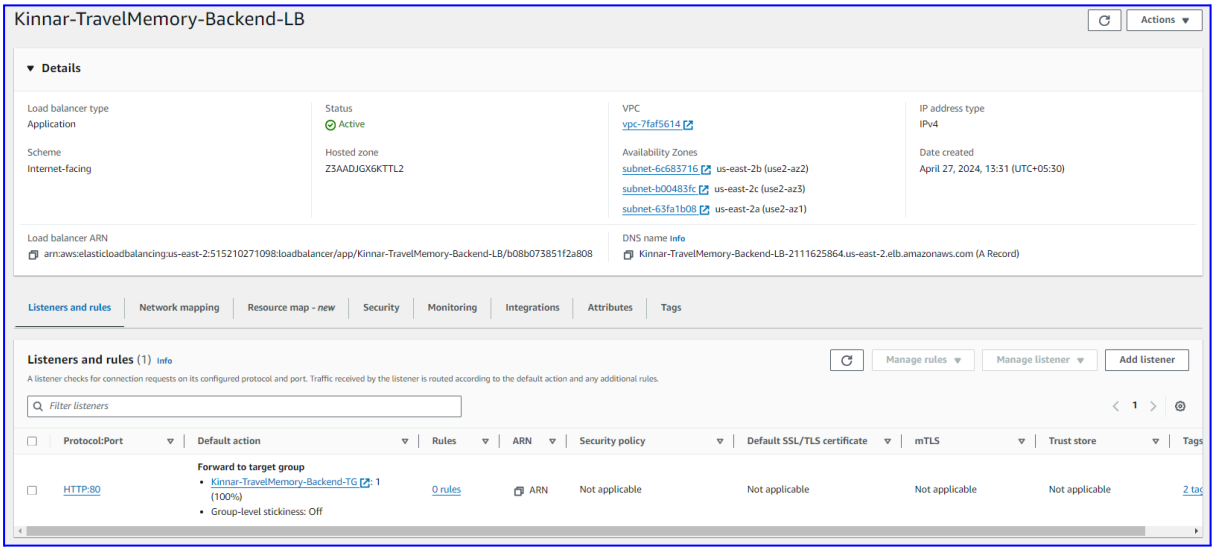
Anomaly mitigation: Not applicable | Deregister | Register targets

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

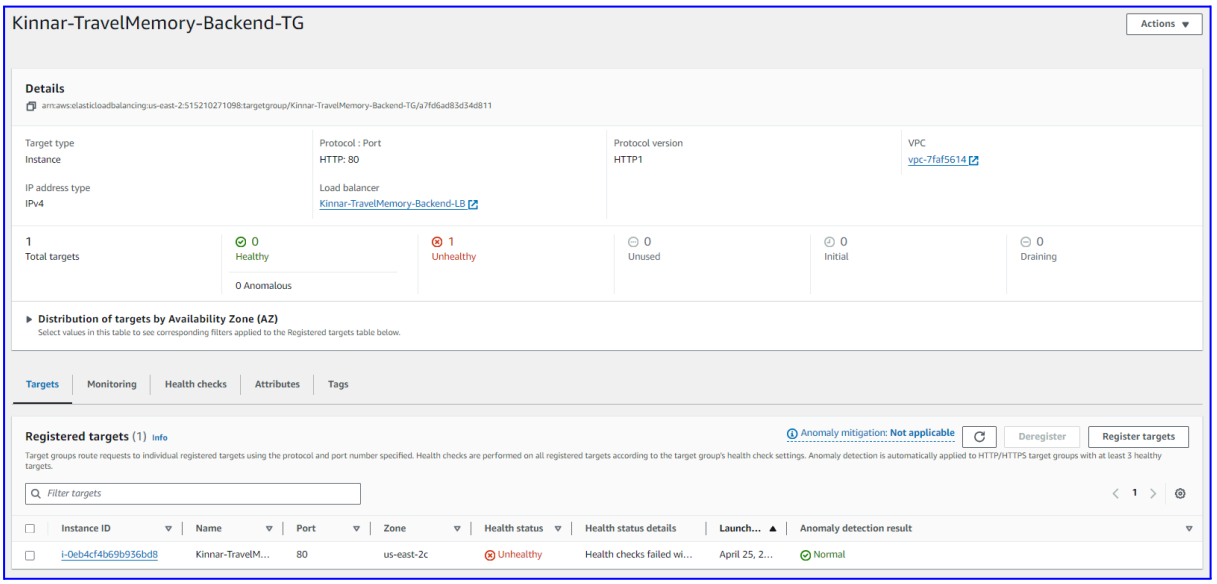
Filter targets

| Instance ID | Name | Port | Zone | Health status | Health status details | Launch... | Anomaly detection result |
|---|---|---|---|---|---|---|---|
| i-0e7b81bb2517016d0 | Kinnar-TravelM... | 80 | us-east-2c | Healthy | - | April 25, 2... | Normal |

## Kinnar-TravelMemory-Backend-LB



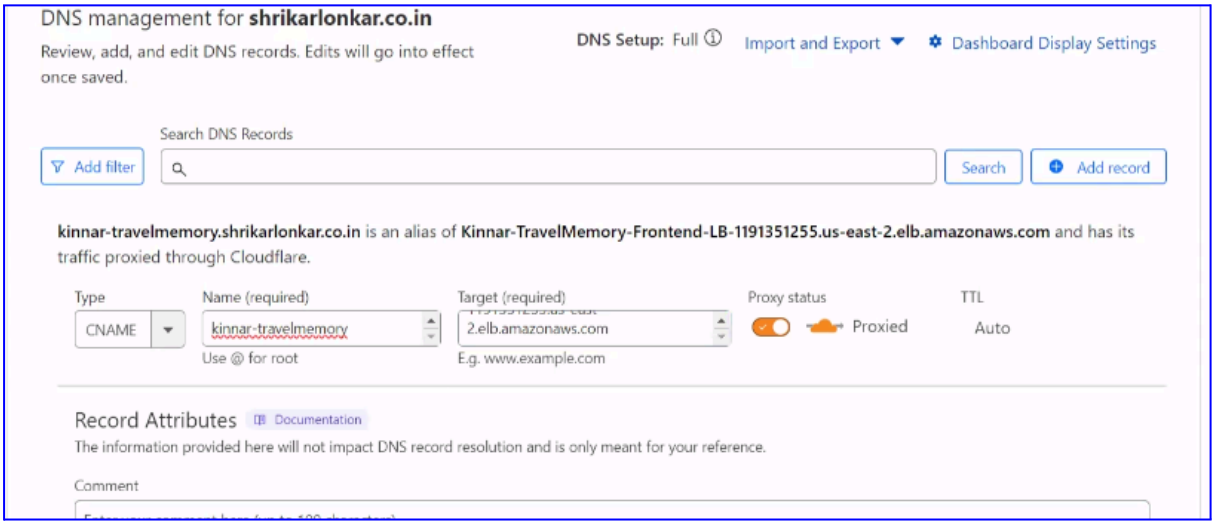

## Kinnar-TravelMemory-Backend-TG



## 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.

Setting up DNS for **https://kinnar-travelmemory.shrikarlonkar.co.in/**



**Reference:**


1. https://stackoverflow.com/questions/76318653/how-can-i-install-node-js-version-18-on-ubuntu-22-04
2.


# THANK YOU

==========


SUBMITTED BY- **KINNAR CHOWDHURY, Hero Vired, DevOps, Batch 4**