Travelmemory MERN Stack Application Deployment

This document provides a comprehensive guide to deploying the TravelMemory MERN stack application on Amazon EC2 instances. It includes steps to configure the backend and frontend servers, set up Nginx reverse proxy, and integrate the application with a load balancer and Cloudflare.

a by Ankita Lodha

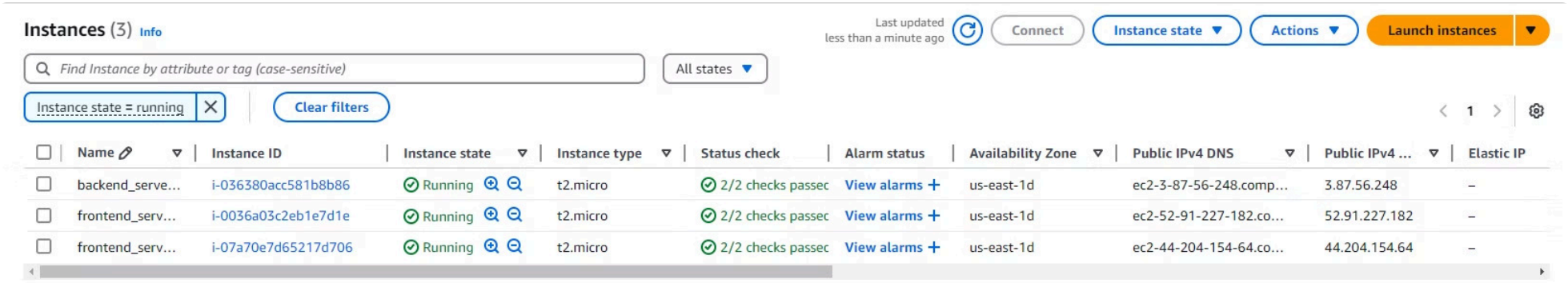
Step 1: EC2 Instances Setup

Launch 3 Ubuntu EC2 instances:

- Backend Server: backend_server001
- Frontend Servers: frontend_server001 and frontend_server002

Configure Security Groups:

- Backend Security Group (backendServerSG):
- Open ports: 22 (SSH), 80 (HTTP), 443 (HTTPS), 3000
- Frontend Security Group (frontendServerSG):
- Open ports: 22 (SSH), 80 (HTTP), 443 (HTTPS), 3000



Launched 3 instances, 1 instance for backend and 2 instances for frontend server.

Step 2: Backend Server Configuration

Connect to the backend server: ssh -i ubuntu@

Install prerequisites:

```
sudo apt update
sudo apt install nodejs
sudo apt install npm
```

Clone the repository:

git clone https://github.com/ankitalodha05/TravelMemory.git

Navigate to the backend folder: cd TravelMemory/backend

Create a .env file:

```
sudo nano .env
```

Add the following:

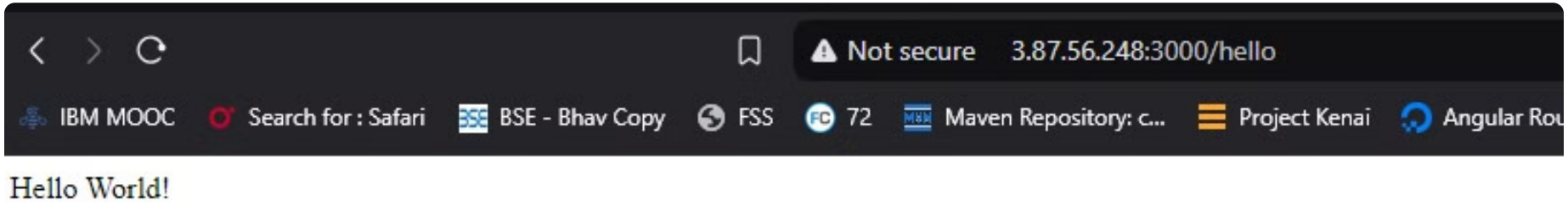
```
PORT=3000
MONGO_URI="mongodb+srv://ankitalodha05:HXbLCIGpRkRg9gNn@cluster10.as960.mongodb.net/ankitalodha"
```

Update index.js to replace 'localhost' with the backend server's public IP.

Install dependencies and start the server:

```
sudo npm install
sudo node index.js &
```

Verify the backend is running at http://3.87.56.248:3000/hello.



Backend server is running on port-3000

Step 4: Nginx Reverse Proxy Setup for Backend

Install Nginx:

```
sudo apt install -y nginx
```

Edit the Nginx default configuration:

```
sudo nano /etc/nginx/sites-available/default
```

Replace with the following:

```
server {
  listen 80;
  server_name 3.87.56.248;
  location / {
    proxy_pass http://127.0.0.1:3000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
  }
}
```

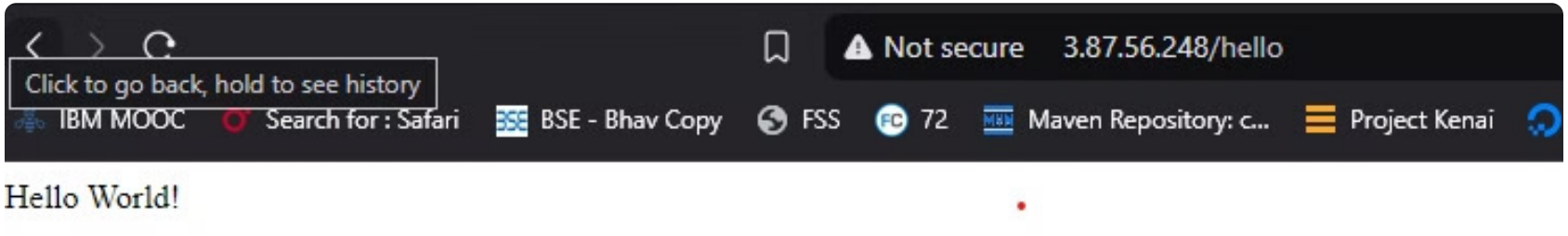
Test the configuration:

```
sudo nginx -t
```

Restart Nginx:

```
sudo systemctl restart nginx
```

Verify the backend is accessible at http://3.87.56.248



backend running successfully on port 80

Step 5: Frontend Server Configuration

Connect to the frontend001 server: ssh -i ubuntu@

Install prerequisites:

```
sudo apt update
sudo apt install npm
```

Clone the repository:

git clone https://github.com/ankitalodha05/TravelMemory.git

Navigate to TravelMemory/frontend/src and edit url.js to replace 'localhost' with the backend server's public IP.

Install dependencies and start the frontend server:

```
sudo npm install
sudo npm start
```

Verify the frontend001 is running at http://52.91.227.182:3000

Step 6: Nginx Reverse Proxy Setup for frontend

Install Nginx:

```
sudo apt install -y nginx
```

Edit the Nginx default configuration:

```
sudo nano /etc/nginx/sites-available/default
```

Replace with the following:

```
server {
  listen 80;
  server_name 52.91.227.182;
  location / {
    proxy_pass http://127.0.0.1:3000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
  }
}
```

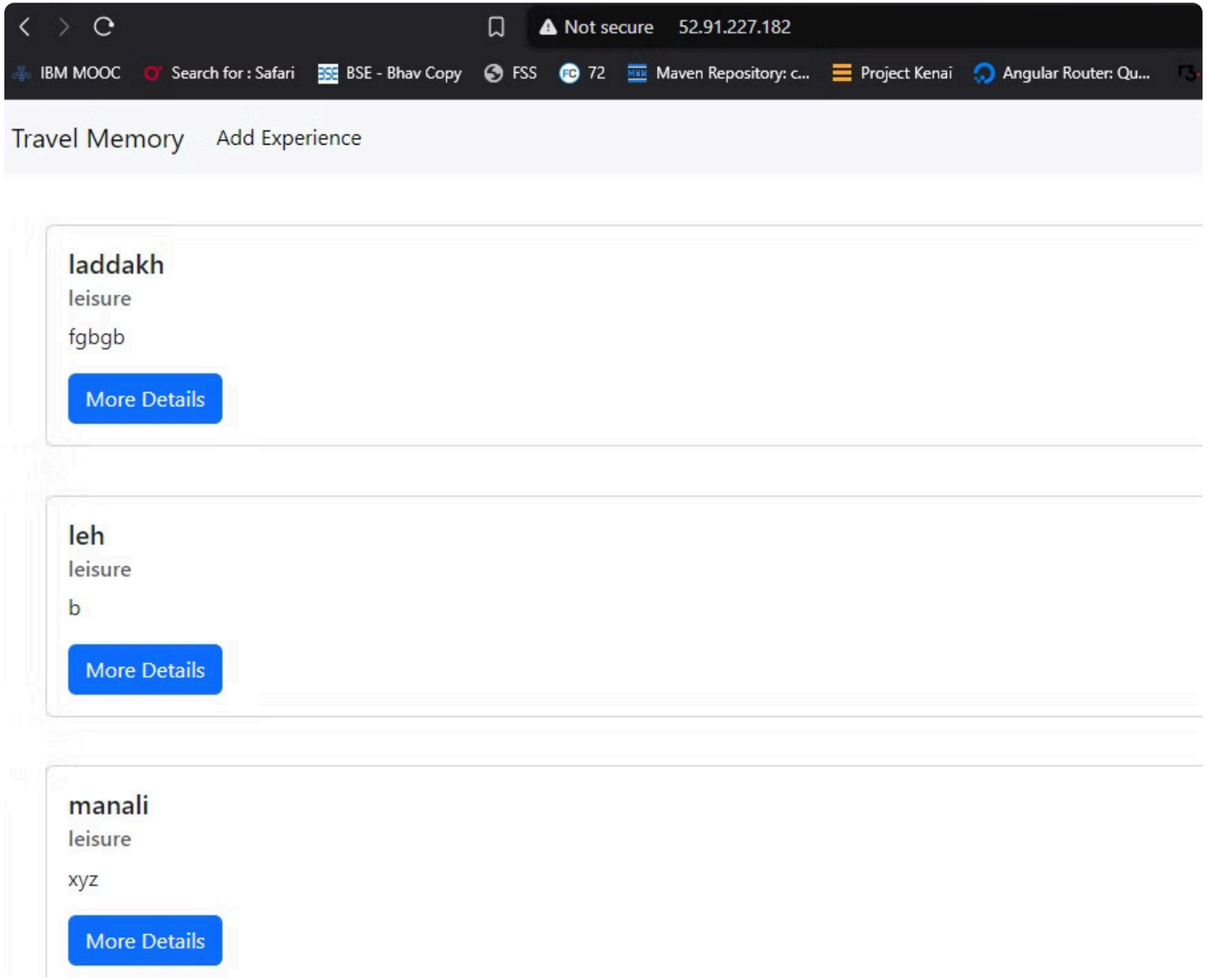
Test the configuration:

```
sudo nginx -t
```

Restart Nginx:

```
sudo systemctl restart nginx
```

Verify the frontend001 is accessible at http://52.91.227.182



Frontend001 running on port 80

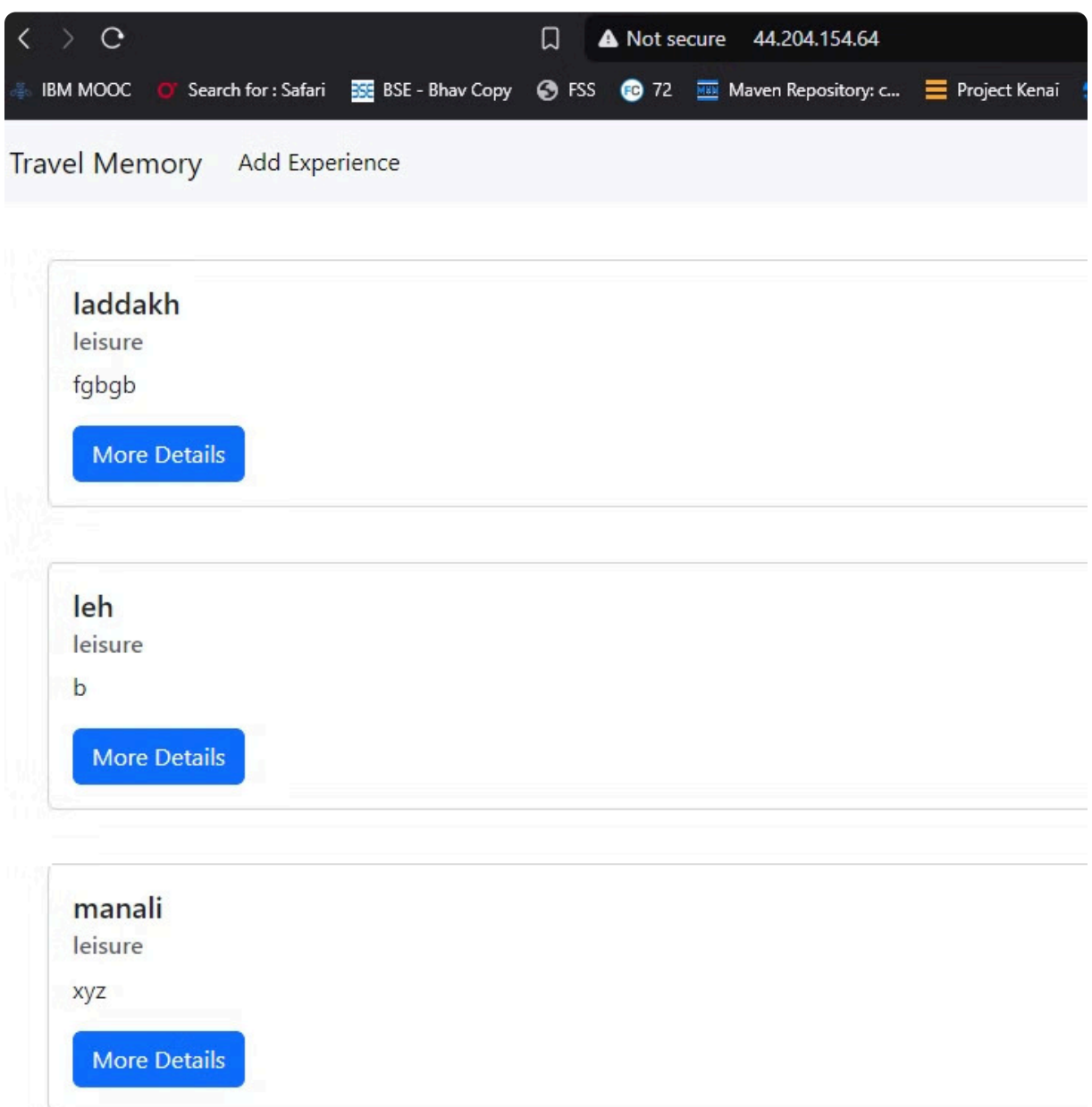
Step 7 :Frontend server002

To configure the frontend server002 - same process as frontend server 001

```
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.31.94.125:3000

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

webpack compiled successfully
```

i-07a70e7d65217d706 (frontend_server002)

PublicIPs: 44.204.154.64 PrivateIPs: 172.31.94.125



frontendServer002 is running on port 80

Step 5: Load Balancer Configuration

Create a target group:

- Include both frontend servers (frontend_server001 and frontend_server002).

Create an Application Load Balancer (ALB):

- Attach the target group.

Verify the ALB DNS name and ensure it routes traffic correctly to the frontend servers.

Access the application via the ALB DNS name.

loadbalancer

Actions

Details

Load balancer type	Status	VPC	Load balancer IP address type
Application	Active	vpc-00cb05b21565275ae	IPv4
Scheme	**Hosted zone**	**Availability Zones**	**Date created**
Internet-facing	Z35SXDOTRQ7X7K	subnet-05c37f01e67e39c98 us-east-1f (use1-az5) subnet-03d1526a3103910f0 us-east-1b (use1-az6) subnet-0c6d584984b3ba3ac us-east-1e (use1-az3) subnet-004d842af0341cece us-east-1c (use1-az1) subnet-05ae42201ba7fe590 us-east-1a (use1-az4) subnet-0dc6855caf36a4d9c us-east-1d (use1-az2)	November 29, 2024, 23:00 (UTC+05:30)

Load balancer ARN
arn:aws:elasticloadbalancing:us-east-1:746669215817:loadbalancer/app/loadbalancer/7f799808635e638d

DNS name Info
loadbalancer-1072278344.us-east-1.elb.amazonaws.com (A Record)

Target-Group

Actions

Details

arn:aws:elasticloadbalancing:us-east-1:746669215817:targetgroup/Target-Group/8a370e35fcfaa5b3

Target type	Protocol : Port	Protocol version	VPC
Instance	HTTP: 80	HTTP1	vpc-00cb05b21565275ae
IP address type	**Load balancer**		
IPv4	None associated		

2	0	0	2	0	0
Total targets	Healthy	Unhealthy	Unused	Initial	Draining
	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 (2) 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.

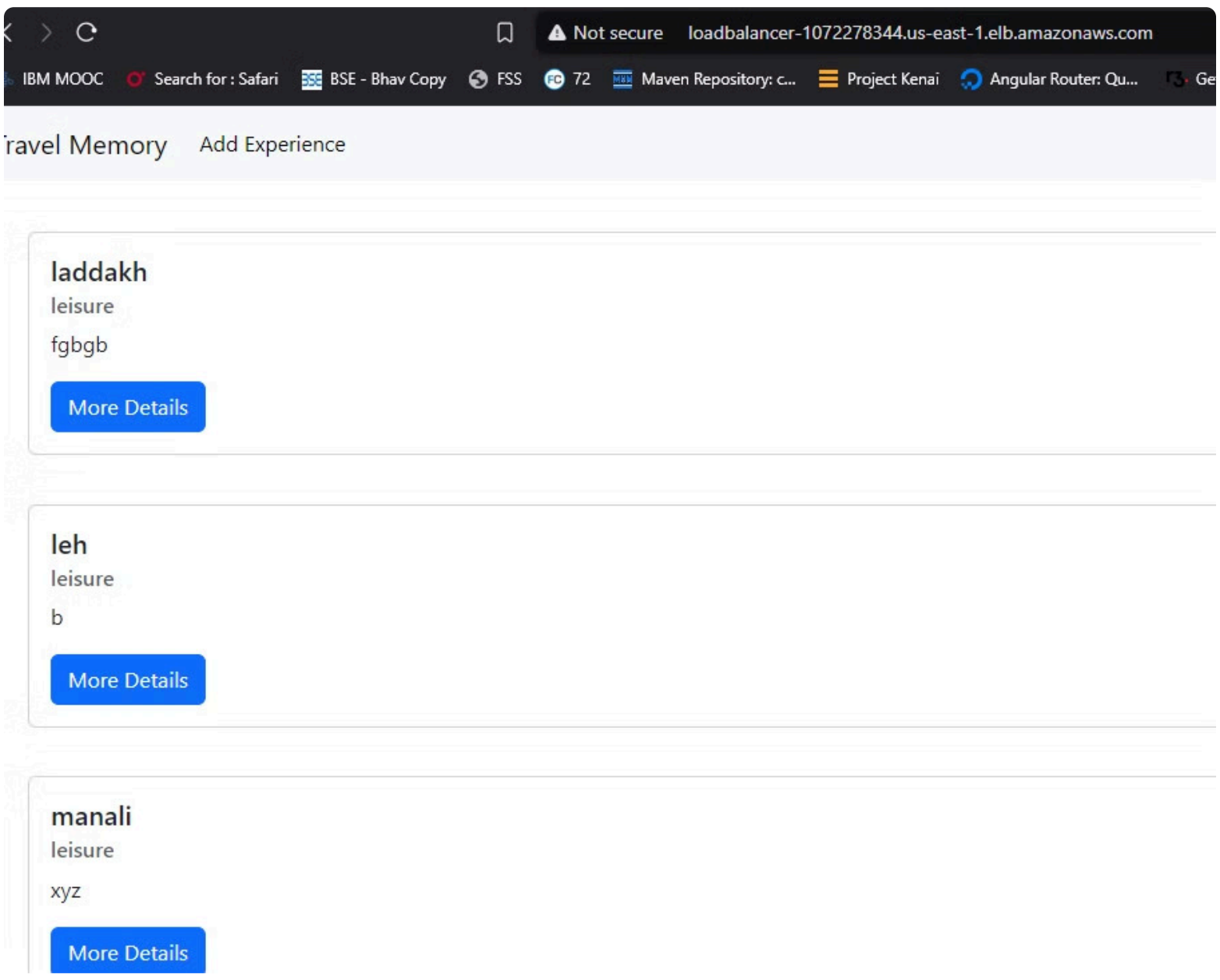
Filter targets

Instance ID	Name	Port	Zone	Health status	Health status details	Admini...	Overri...
i-0036a03c2eb1e7d1e	frontend_serv...	80	us-east-1d (us...	Unused	Target group is not co...	-	-
i-07a70e7d65217d706	frontend_serv...	80	us-east-1d (us...	Unused	Target group is not co...	-	-

select both the frontend instances as include pending below on port 80

created a load balancer

DNS — loadbalancer-1072278344.us-east-1.elb.amazonaws.com



loadbalencer DNS in browser.

Step 6: Cloudflare Integration

1. Update Nameservers

- Replace your domain registrar's nameservers with:
  - audrey.ns.cloudflare.com
  - ignat.ns.cloudflare.com
- Wait for propagation (up to 24-48 hours).

2.Frontend (CNAME Record):

- Name: vaanilodha.com
- Content: loadbalancer-1072278344.us-east-1.elb.amazonaws.com
- Proxy Status: **Proxied**

3. Enable Proxying and SSL

- **Proxying**: Ensure both DNS records are set to **Proxied**.
- **SSL**:
  - Go to the **SSL/TLS** tab in Cloudflare.
  - Set the SSL mode to:
    - **Flexible**: If backend/frontend servers don't have SSL.
    - **Full (Strict)**: If valid SSL certificates are installed on servers.

4. Verify Domain Setup

- Test frontend: https://vaanilodha.com

DNS management for **vaanilodha.com**

Review, add, and edit DNS records. Edits will go into effect once saved.

DNS Setup: Full ⓘ Import and Export Dashboard Display Settings

Search DNS Records

Add filter Search Add record

	Type ⓘ	Name ⓘ	Content ⓘ	Proxy status ⓘ	TTL ⓘ	Actions
⚠	CNAME	vaanilodha.com	loadbalancer-1072278344.us...	Proxied	Auto	Edit

Cloudflare Nameservers

To use Cloudflare, change your nameservers, or authoritative DNS servers. These are your assigned Cloudflare nameservers.

Type	Value
NS	audrey.ns.cloudflare.com
NS	ignat.ns.cloudflare.com

Not secure vaanilodha.com

IBM MOOC Search for : Safari BSE - Bhav Copy FSS 72 Maven Repository: c... Project Ke

Travel Memory Add Experience

laddakh

leisure

fgbgb

More Details

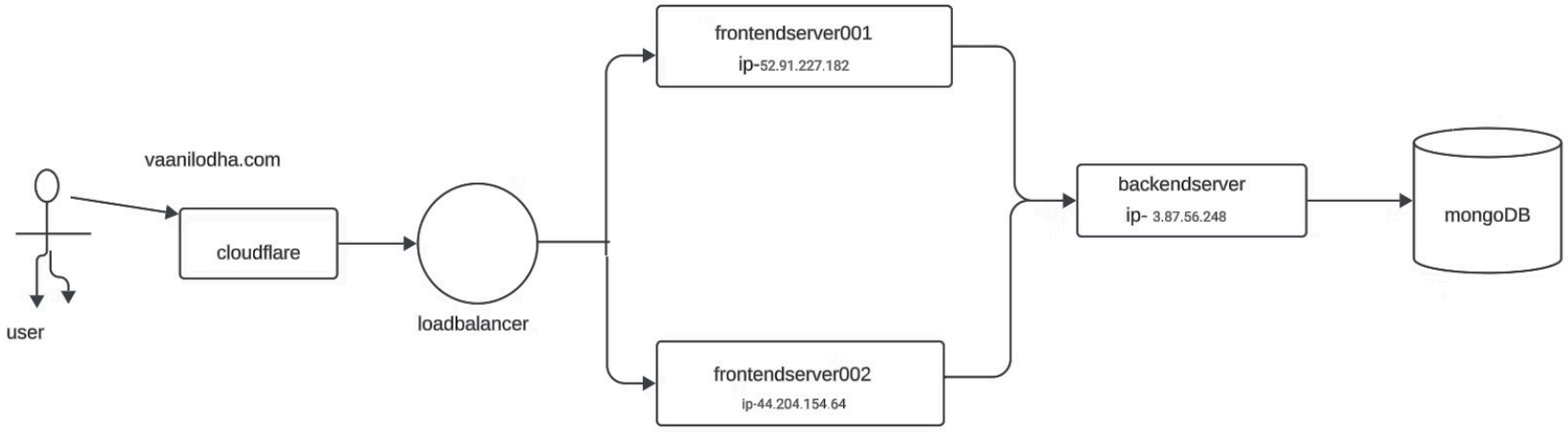
leh

leisure

b

More Details

As you can see my website is working well for frontend.— vaanilodha.com



deployment architecture diagram

Conclusion

This document provides a detailed procedure for deploying the TravelMemory MERN stack application on Amazon EC2 instances. By following these steps, you can successfully configure your backend and frontend servers, implement Nginx reverse proxy, integrate with a load balancer, and secure your application with Cloudflare. With this setup, you can enjoy a robust, scalable, and reliable application architecture for your TravelMemory platform.