Project: Deployment of 3-Tier Applications Using AWS Services

By: ***Harshika***

LinkedIn: ***Harshika***

Portfolio: ***harshika-devops.online***

Executive Summary

Project Title: End-to-End Deployment of a 3-Tier Web Architecture on AWS

Submitted by: Harshika

Date: 15 April 2026

1. Overview

This project involved the design, implementation, and successful deployment of a full-stack **3-Tier Architecture** on Amazon Web Services (AWS). The goal was to build a secure, highly available environment that separates the presentation layer (Web), logic layer (App), and data layer (Database) to ensure optimal performance and security.

2. Technical Implementation

The infrastructure was built within a custom **Virtual Private Cloud (VPC)**. Key technical milestones included:

- **Presentation Layer:** Configured **Nginx** as a high-performance web server and reverse proxy to serve a React frontend.
- **Logic Layer:** Deployed a **Node.js** backend managed by **PM2** to ensure persistent application uptime.

- **Data Layer:** Provisioned an **Amazon Aurora RDS** (MySQL-compatible) cluster for scalable and reliable data storage.
- **Traffic Management:** Implemented an **Application Load Balancer (ALB)** to handle external traffic, secured with **SSL/TLS certificates** via AWS Certificate Manager (ACM).
- **DNS Management:** Integrated a custom domain **harshika-devops.online** using **Amazon Route 53** to provide a user-friendly entry point to the application.

3. Key Successes & Problem Solving

A significant portion of the project focused on advanced troubleshooting in a Linux-based cloud environment. Challenges regarding **SELinux policies**, **cross-layer permissions**, and **Target Group health checks** were successfully identified and resolved. This demonstrated a deep understanding of networking protocols, security groups, and the interaction between Nginx and backend services.

4. Conclusion

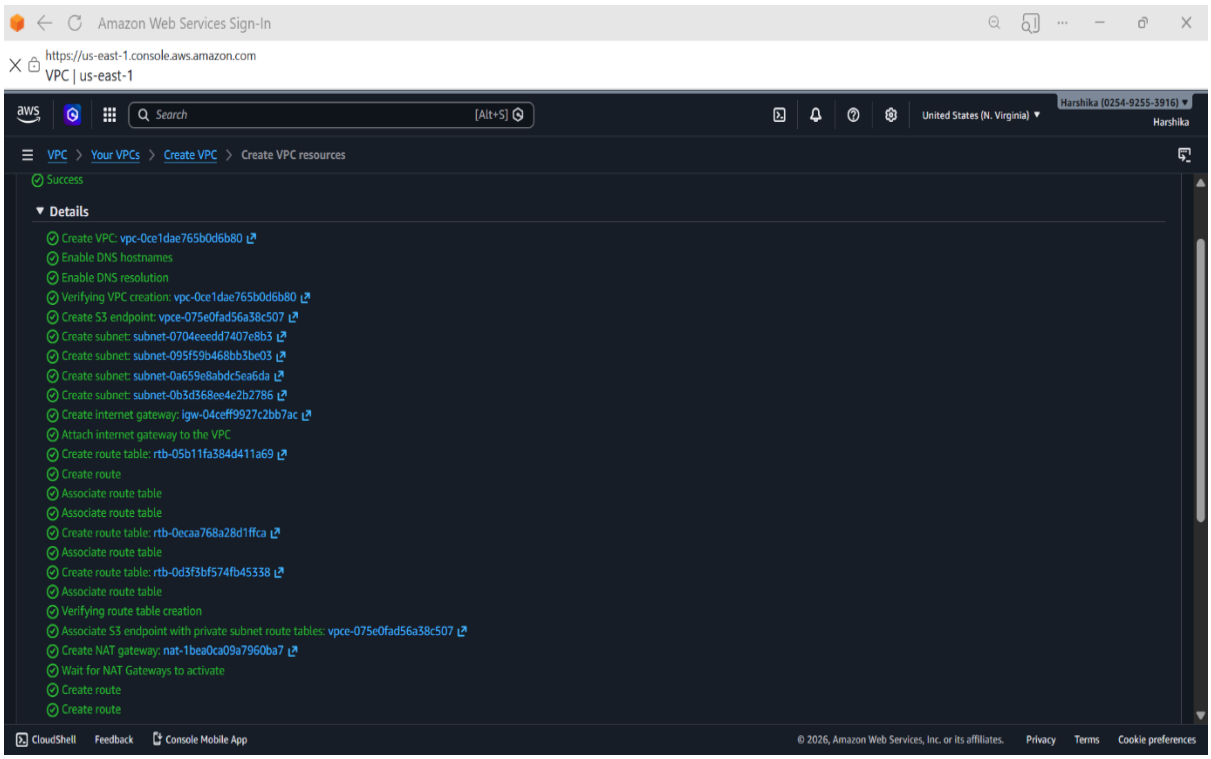
The project was successfully verified by accessing the application over a secure **HTTPS** connection, with full data persistence confirmed from the frontend to the Aurora database. The infrastructure was then decommissioned following the strict cleanup protocol to ensure cost-efficiency and security best practices.

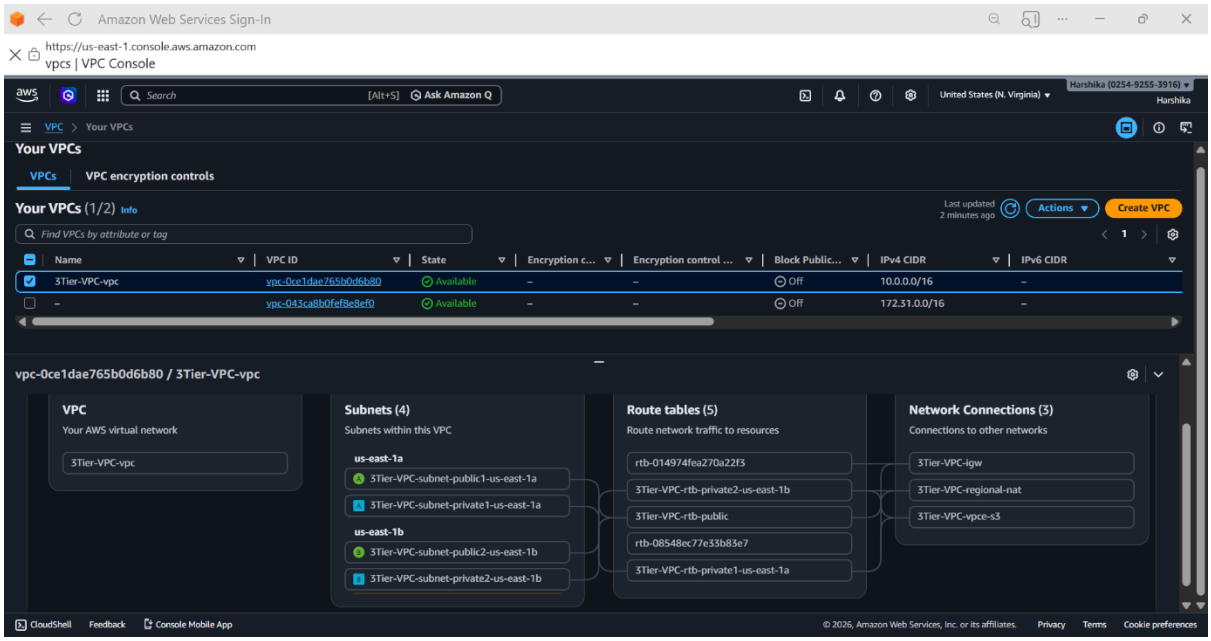
Phase 1: The Foundation (Networking & Database)

In this phase, we create the (VPC)and (RDS), where your data stays.

Step 1.1: Create the VPC

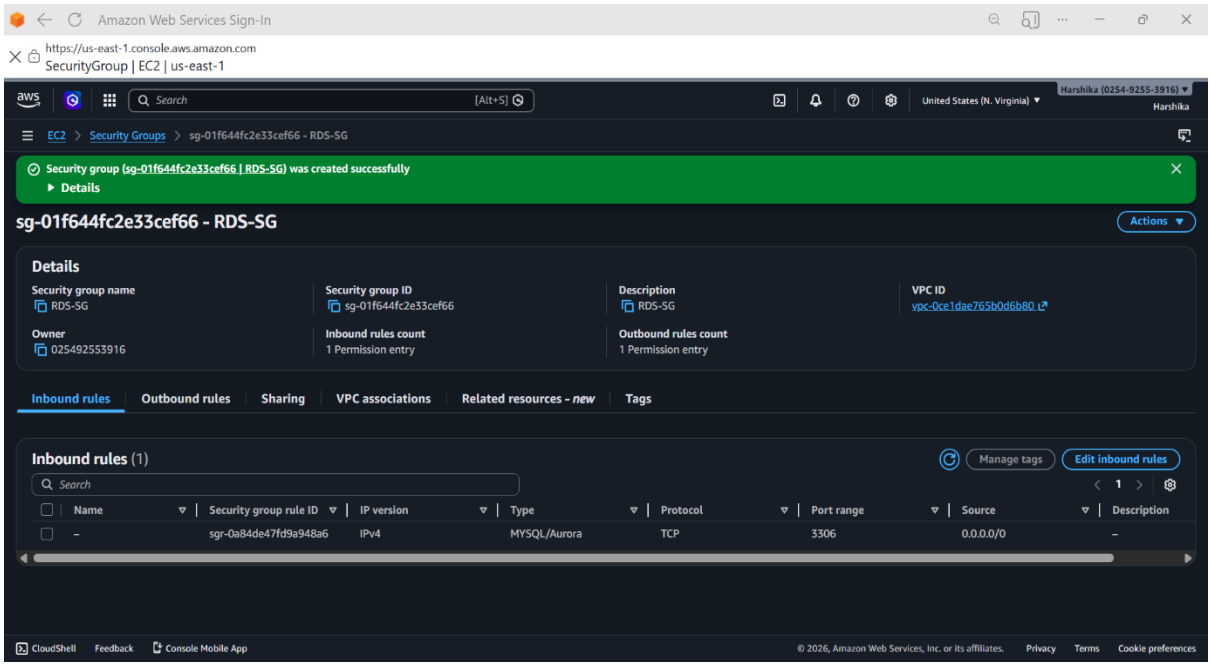
1. Go to the **VPC Dashboard** in the AWS Console.
2. Click **Create VPC**.
3. Select **"VPC and more"** (this is the easiest way to ensure subnets and Gateways are linked correctly).
4. **Name tag:** 3Tier-VPC
5. **IPv4 CIDR block:** 10.0.0.0/16
6. **Number of Availability Zones (AZs):** Select **2** (for High Availability).
7. **Number of Public Subnets: 2**
8. **Number of Private Subnets: 2**
9. **NAT Gateways:** Select **1 per AZ**.
10. Click **Create VPC**.





Step 1.2: Security Group for Database (RDS)

1. Go to **EC2 Dashboard** -> **Security Groups**.
2. Click **Create Security Group**.
3. **Name:** RDS-SG
4. **VPC:** Select your 3Tier-VPC.
5. **Inbound Rules:**
    - **Type:** MySQL/Aurora (Port 3306)
    - **Source:** For now, set to 0.0.0.0/0 (We will restrict this to the App-Tier SG later once we create it).
6. Click **Create**.

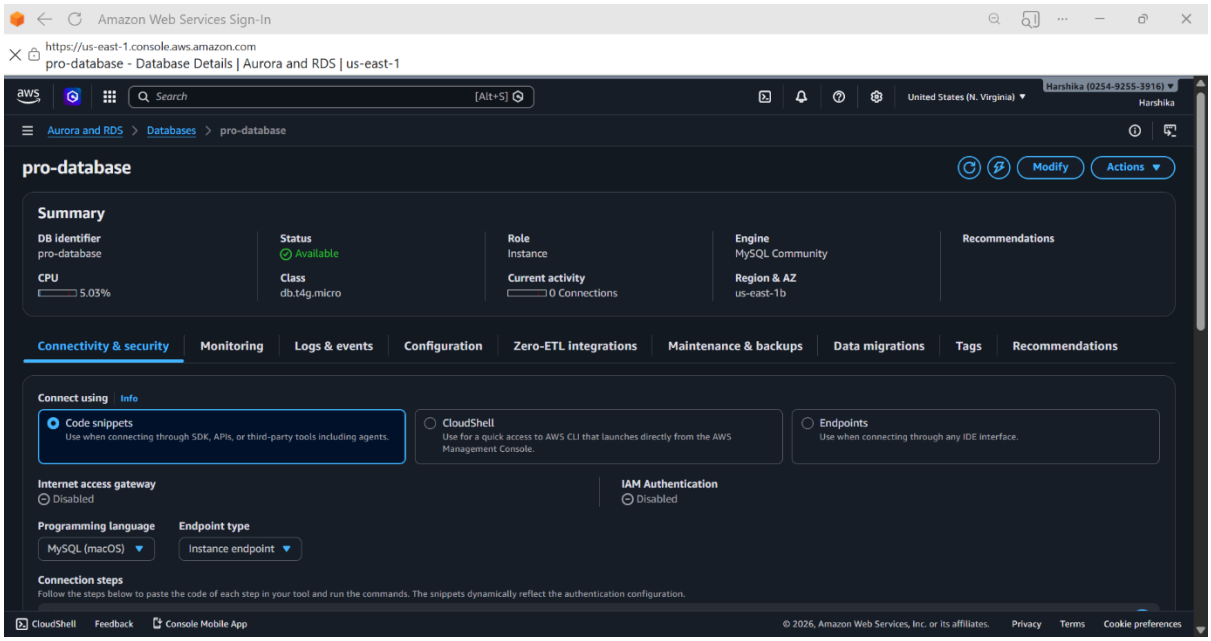


Step 1.3: Create the RDS Instance

1. Go to the **RDS Dashboard** -> **Databases** -> **Create database**.
2. **Choose a creation method:** Standard create.
3. **Engine options:** MySQL.
4. **Templates: Free Tier** (Crucial to avoid costs!).
5. **Settings:**
    - **DB instance identifier:** pro-database
    - **Master username:** admin
    - **Master password:** kastro2025
6. **Connectivity:**
    - **VPC:** 3Tier-VPC
    - **Public access: No** (It must stay in a private subnet).
    - **Existing VPC security groups:** Select RDS-SG (remove the 'default' one).
7. **Additional configuration:**

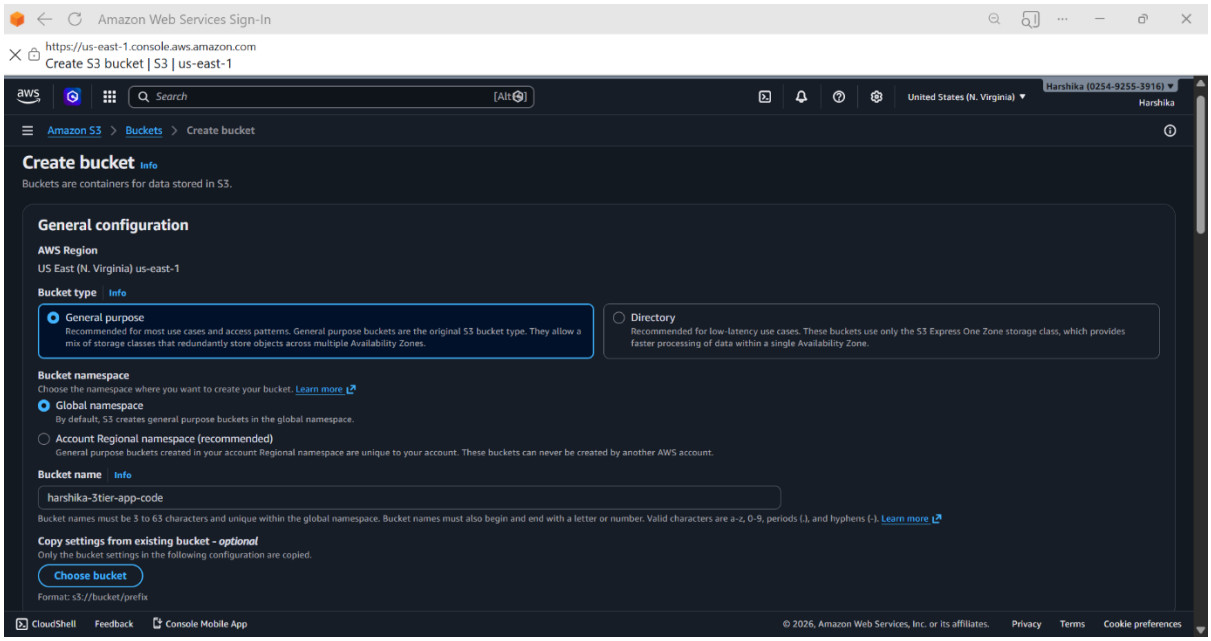
- **Initial database name:** webappdb

8. Click **Create database**. (Note: This takes about 5–10 minutes).



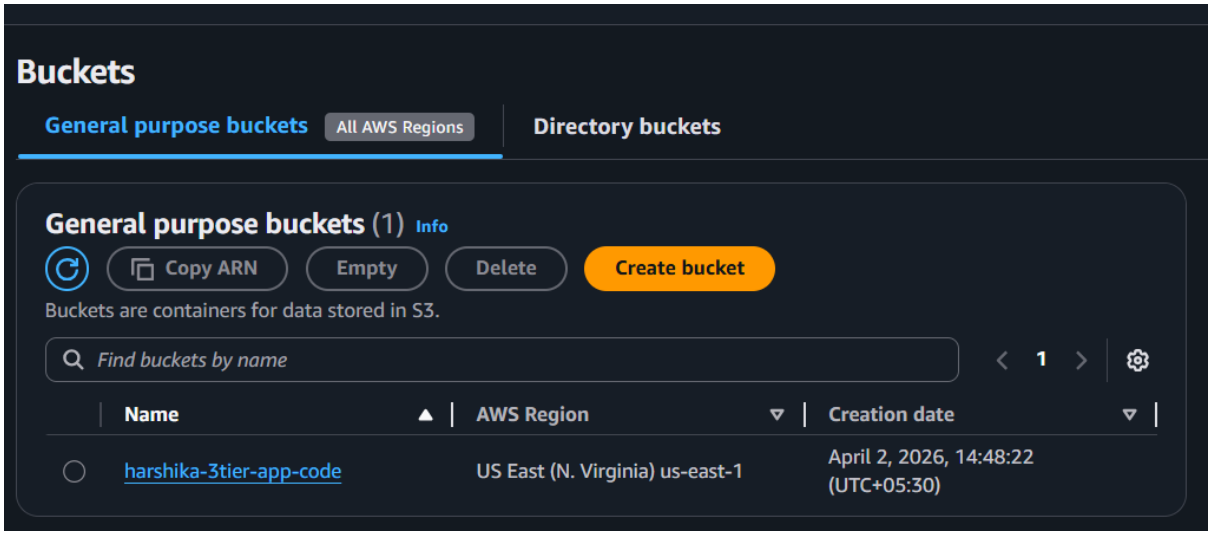
Step 1.4: Create the S3 Bucket

1. Go to **S3 Dashboard** -> **Create bucket**.
2. **Bucket name:** harshika-3tier-app-code (must be unique).



3. **Region:** Ensure it matches your VPC (e.g., ap-south-1).

4. **Object Ownership:** ACLs enabled.
5. **Block Public Access: Uncheck** all (we need this for the web-tier later).
6. Click **Create bucket**.

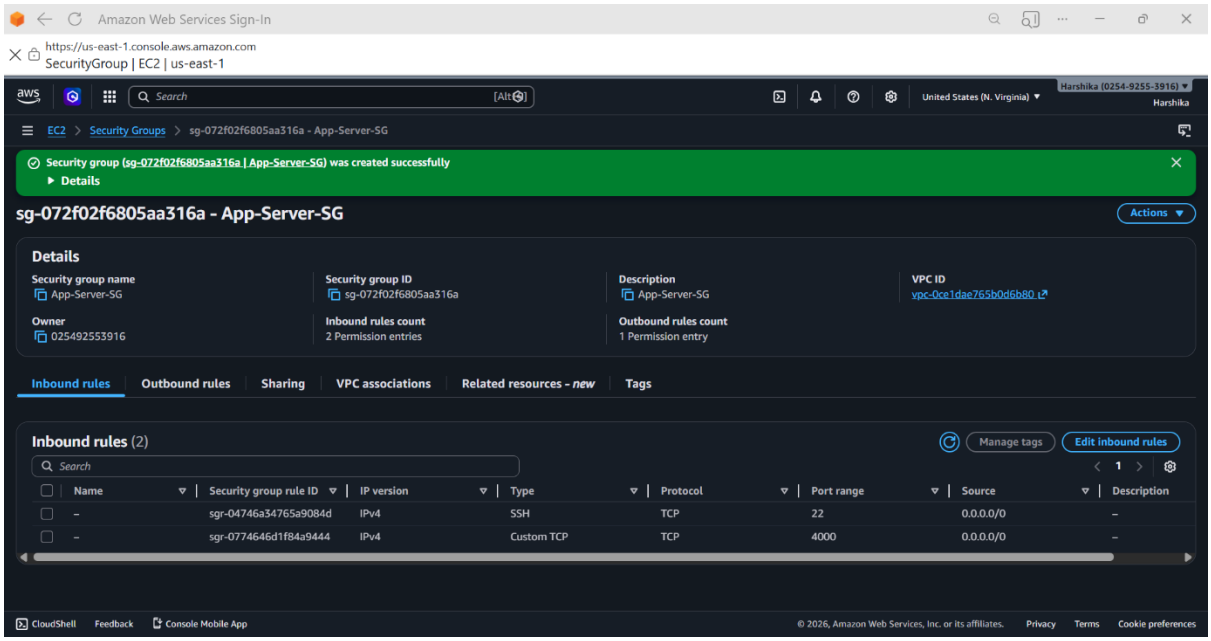


Phase 2: The Application Tier (Backend)

In this phase, we will launch an EC2 instance, install the backend code, and connect it to the database we created in phase 1.

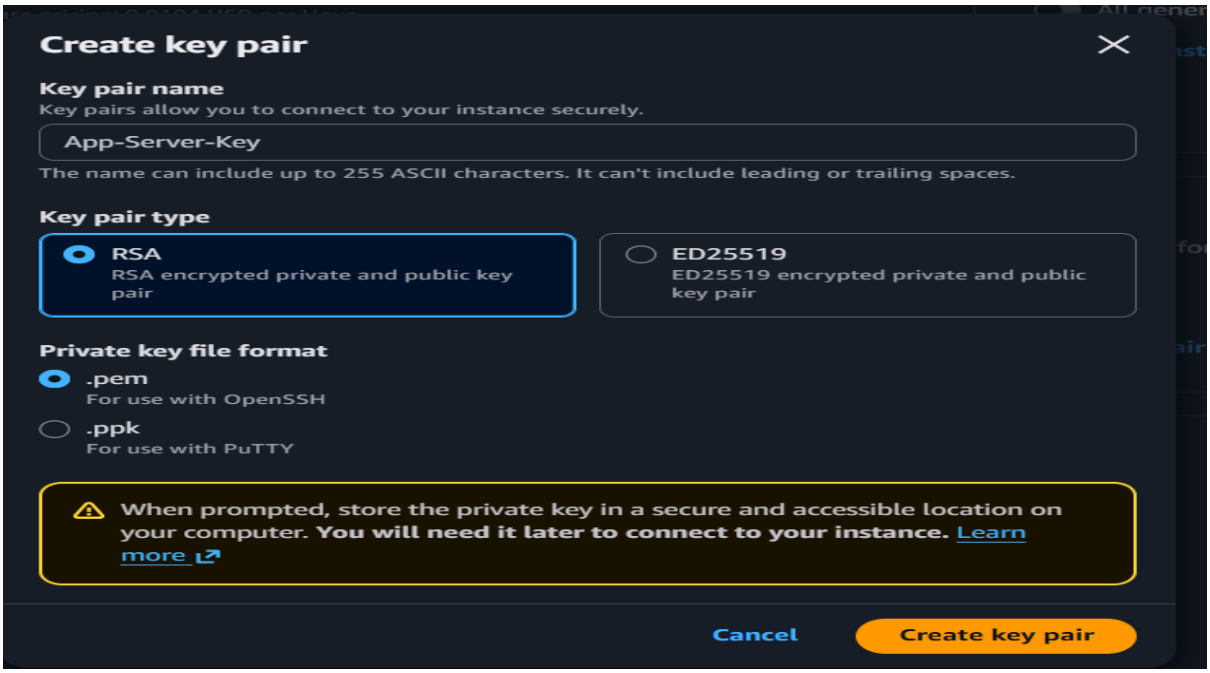
Step 2.1: Create App-Server Security Group

1. Go to **EC2 Dashboard** -> **Security Groups** -> **Create Security Group**.
2. **Name:** App-Server-SG
3. **VPC:** Select your 3Tier-VPC.
4. **Inbound Rules:**
    - **Rule 1:** Type: **SSH** (Port 22) | Source: **0.0.0.0/0** (So we can log in).
    - **Rule 2:** Type: **Custom TCP** (Port: **4000**) | Source: **0.0.0.0/0** (This is the port our Node.js app runs on).
5. Click **Create**.

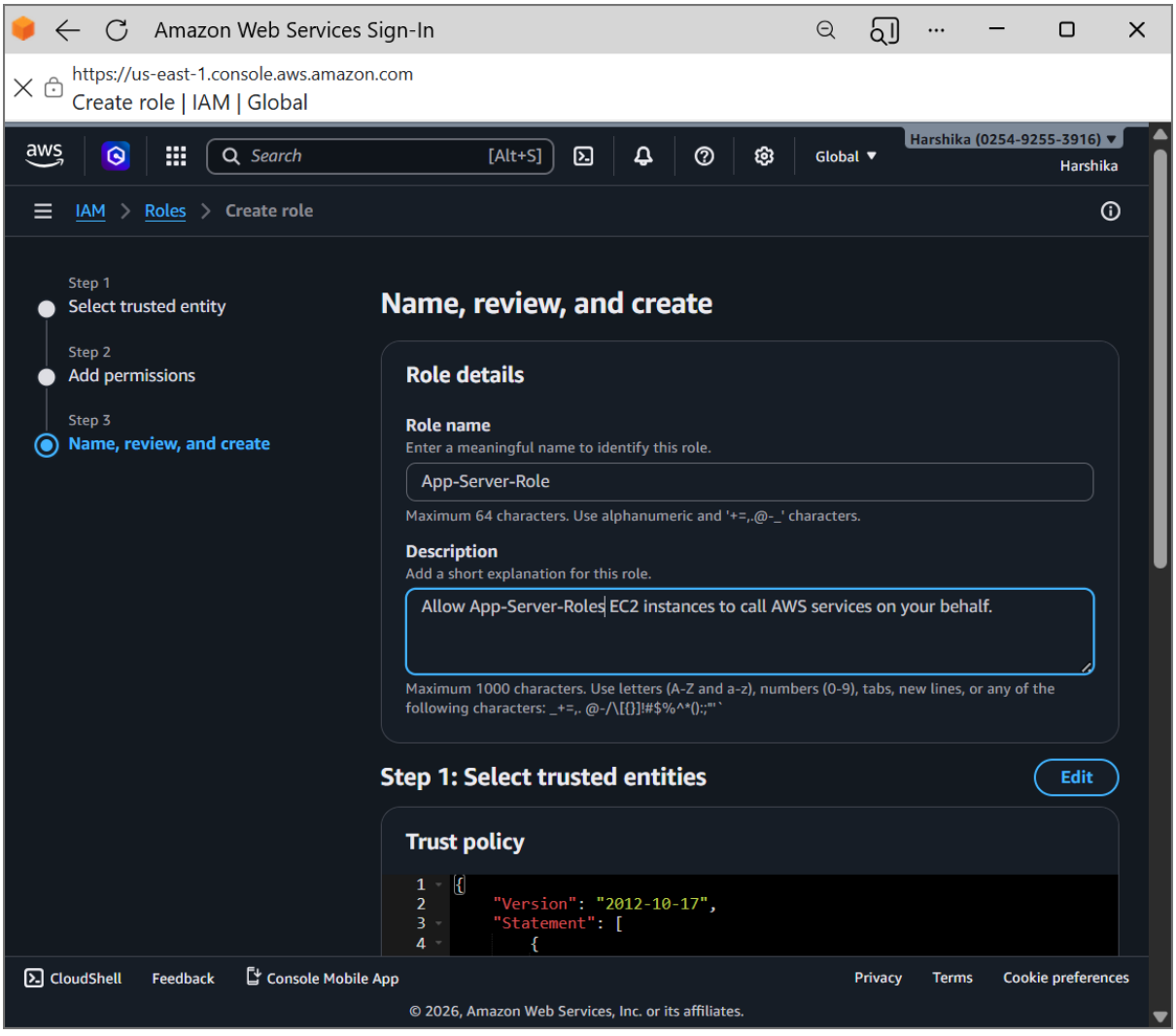


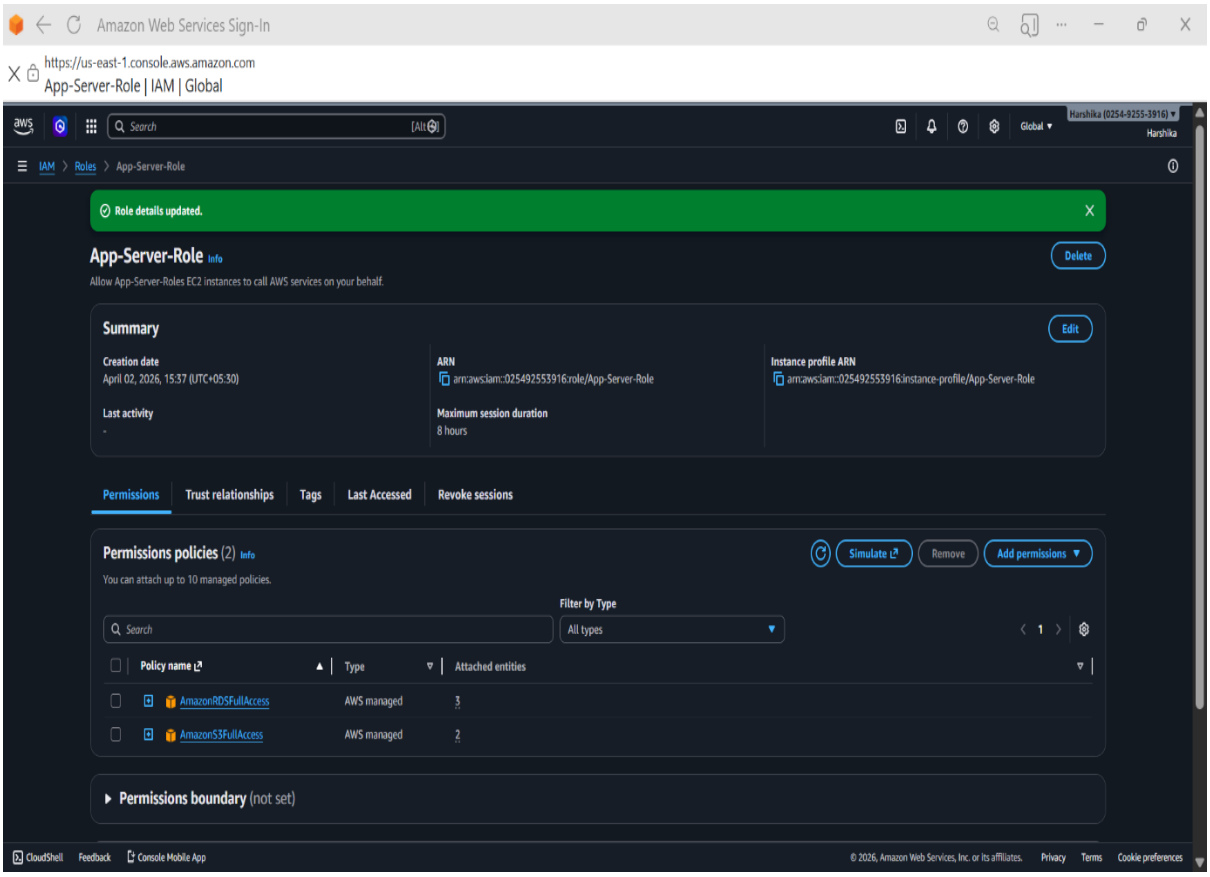
Step 2.2: Launch the App-Server EC2

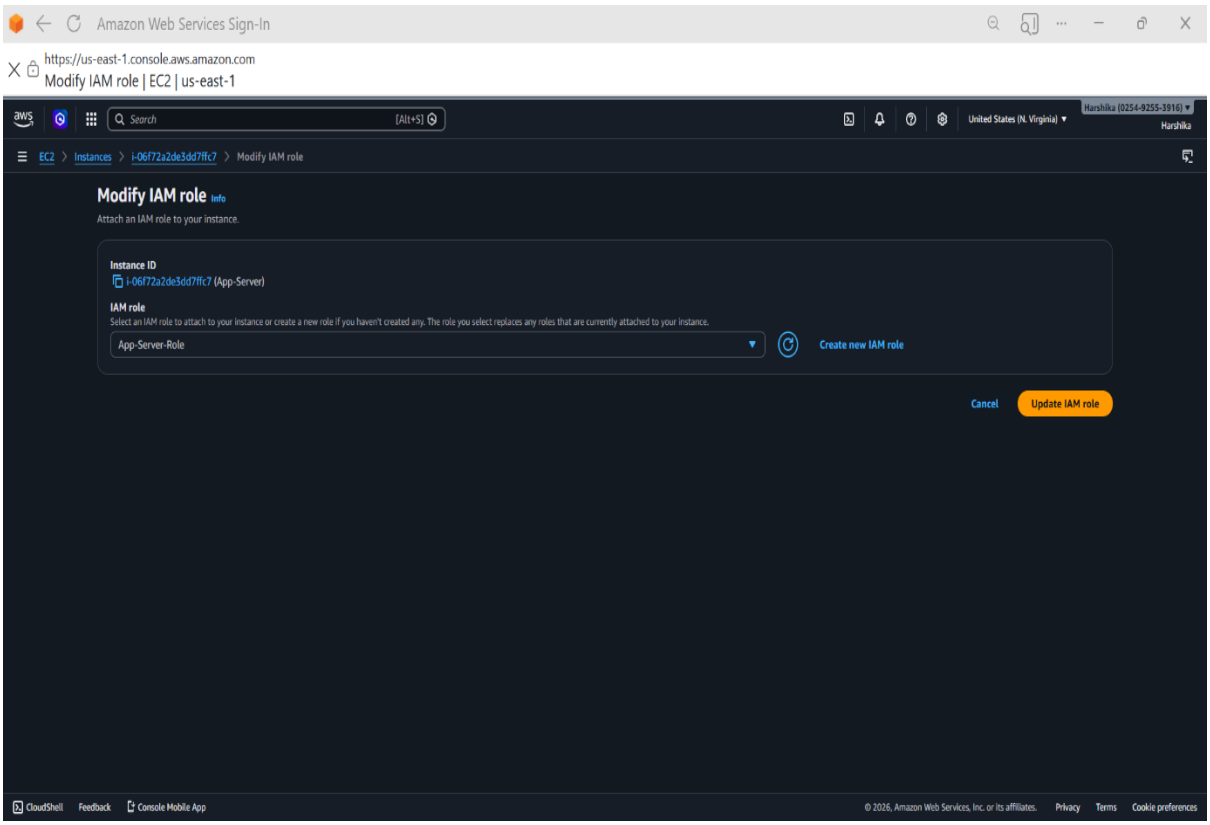
1. Go to **EC2 Dashboard** -> **Instances** -> **Launch Instance**.
2. **Name:** App-Server
3. **AMI:** Amazon Linux 2023 (Free Tier).
4. **Instance Type:** t3.micro (Free Tier).
5. **Key Pair:** Create a new key pair.



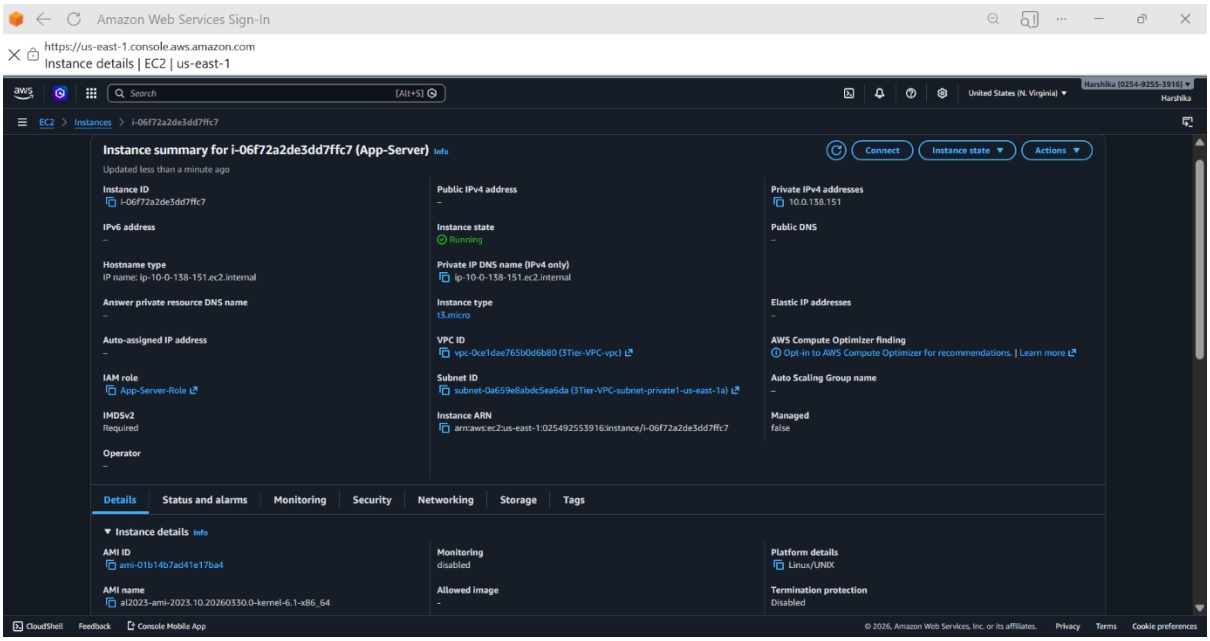
6. **Network Settings:**
    - **VPC:** 3Tier-VPC
    - **Subnet:** Select a **Private Subnet** (e.g., 3Tier-VPC-subnet-private1).
    - **Auto-assign Public IP: Disable** (Private instances don't need public IPs).
    - **Security Group:** Select App-Server-SG.
7. **IAM Instance Profile:** Create an IAM Role for EC2 with **S3FullAccess** and **AmazonRDSFullAccess** and attach it here.




Amazon Web Services Sign-In
https://us-east-1.console.aws.amazon.com
App-Server-Role | IAM | Global
Search
Global
Harshika (0254-9255-3916)
Harshika
IAM > Roles > App-Server-Role
Role details updated.
App-Server-Role Info
Delete
Allow App-Server-Roles EC2 instances to call AWS services on your behalf.
Summary
Edit
Creation date
April 02, 2026, 15:37 (UTC+05:30)
ARN
arn:aws:iam::025492553916:role/App-Server-Role
Instance profile ARN
arn:aws:iam::025492553916:instance-profile/App-Server-Role
Last activity
Maximum session duration
8 hours
Permissions
Trust relationships
Tags
Last Accessed
Revoke sessions
Permissions policies (2) Info
Simulate
Remove
Add permissions
You can attach up to 10 managed policies.
Filter by Type
Search
All types
Policy name
Type
Attached entities
AmazonRDSFullAccess
AWS managed
3
AmazonS3FullAccess
AWS managed
2
Permissions boundary (not set)
CloudShell
Feedback
Console Mobile App
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

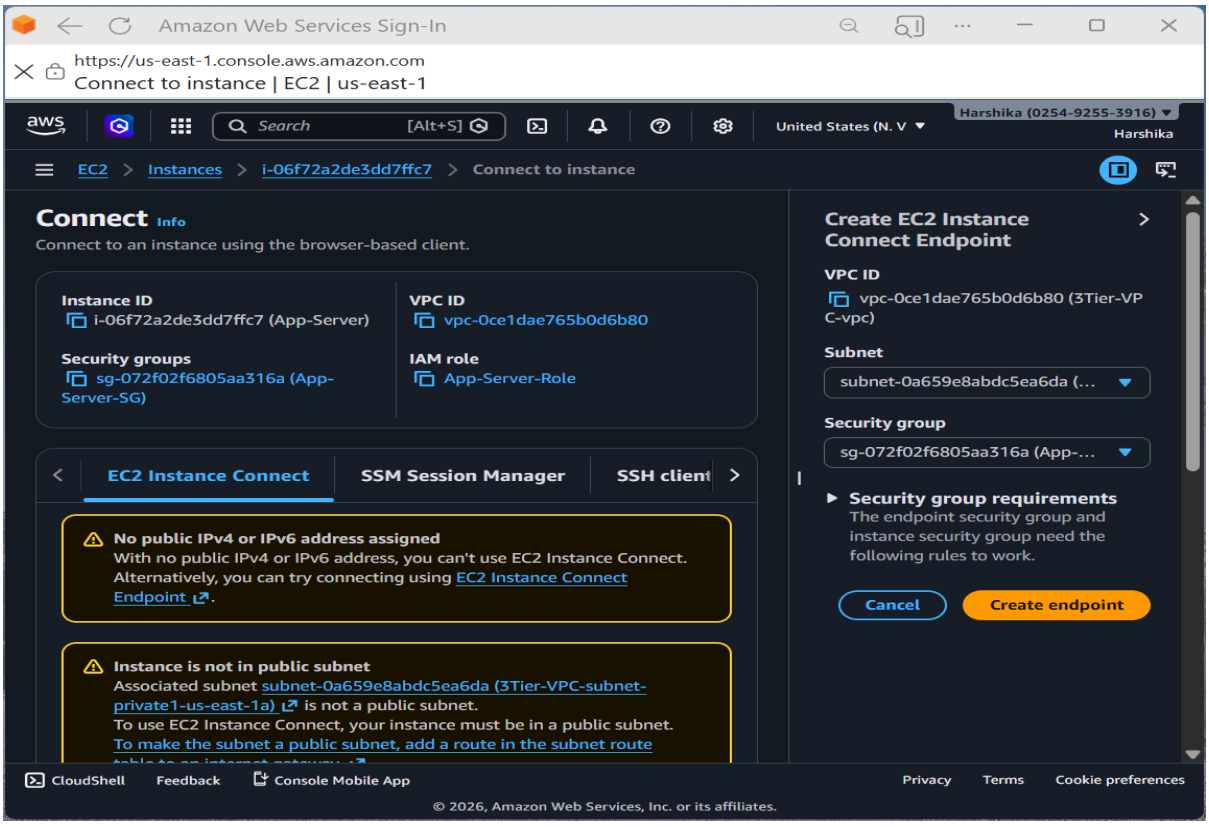

Amazon Web Services Sign-In
https://us-east-1.console.aws.amazon.com
Modify IAM role | EC2 | us-east-1
Search
United States (N. Virginia)
Harshika (0254-9255-3916)
Harshika
EC2 > Instances > i-06f72a2de3dd7ffc7 > Modify IAM role
Modify IAM role Info
Attach an IAM role to your instance.
Instance ID
i-06f72a2de3dd7ffc7 (App-Server)
IAM role
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.
App-Server-Role
Create new IAM role
Cancel
Update IAM role
CloudShell
Feedback
Console Mobile App
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

8. Click **Launch**.

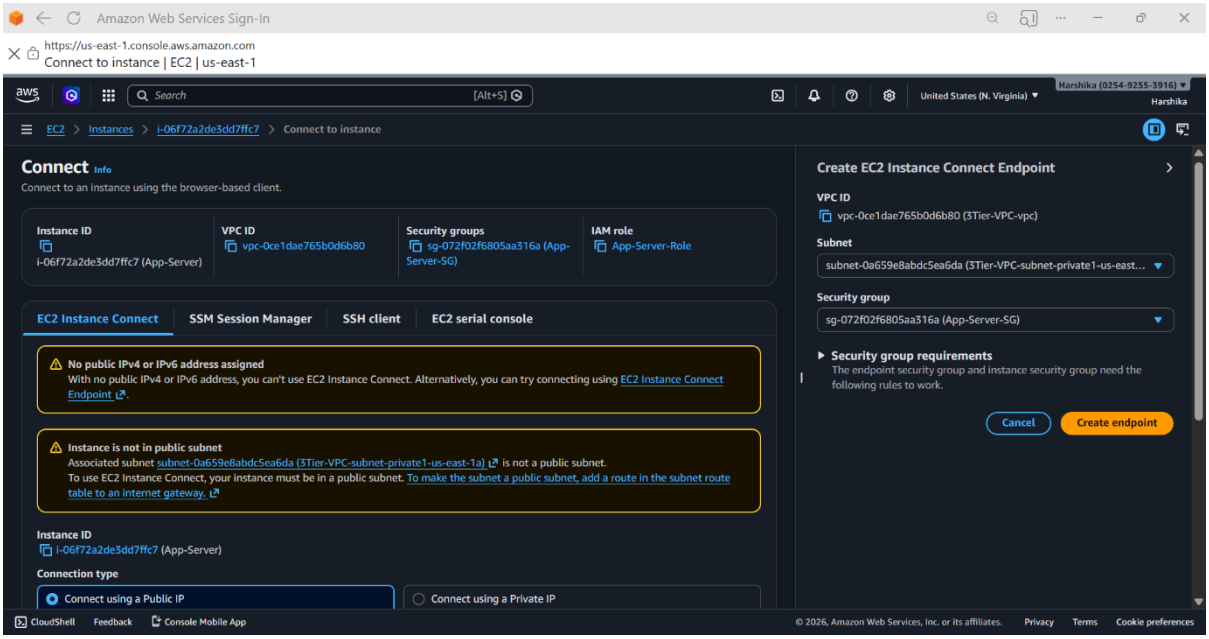


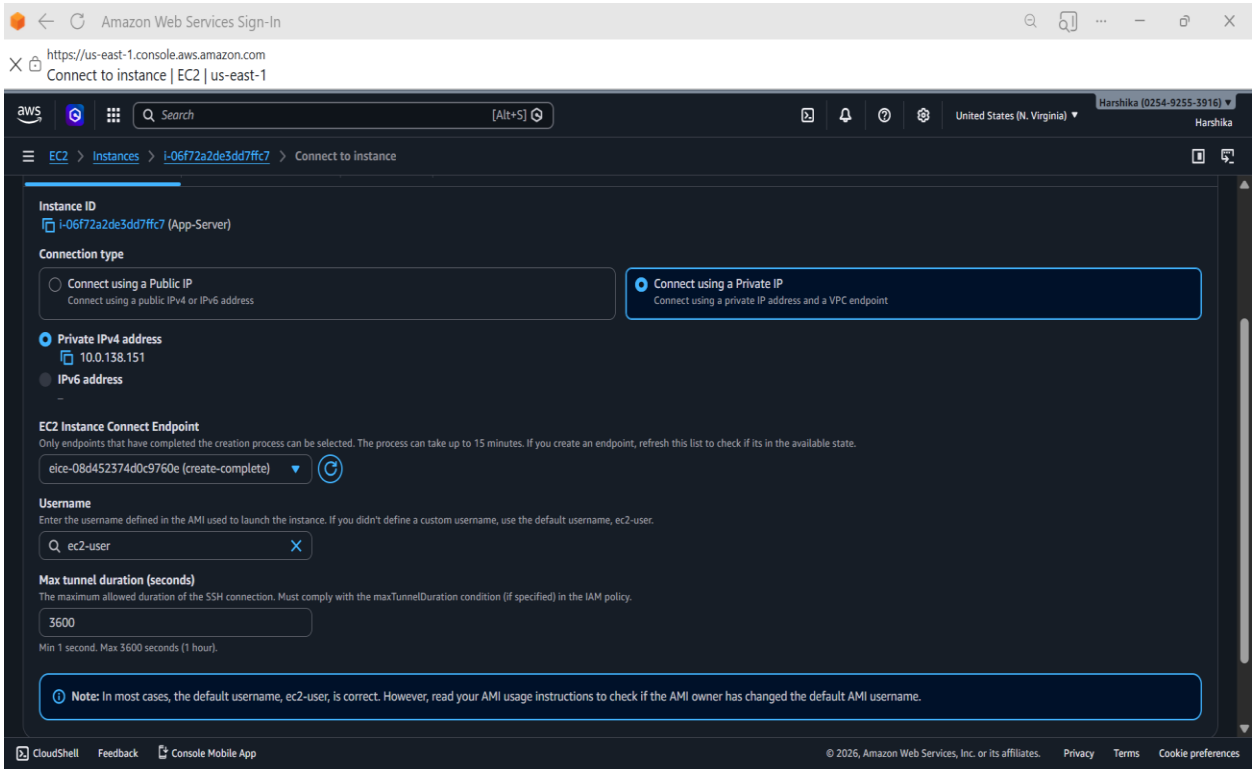
Step 2.2.1: Create the Correct Endpoint

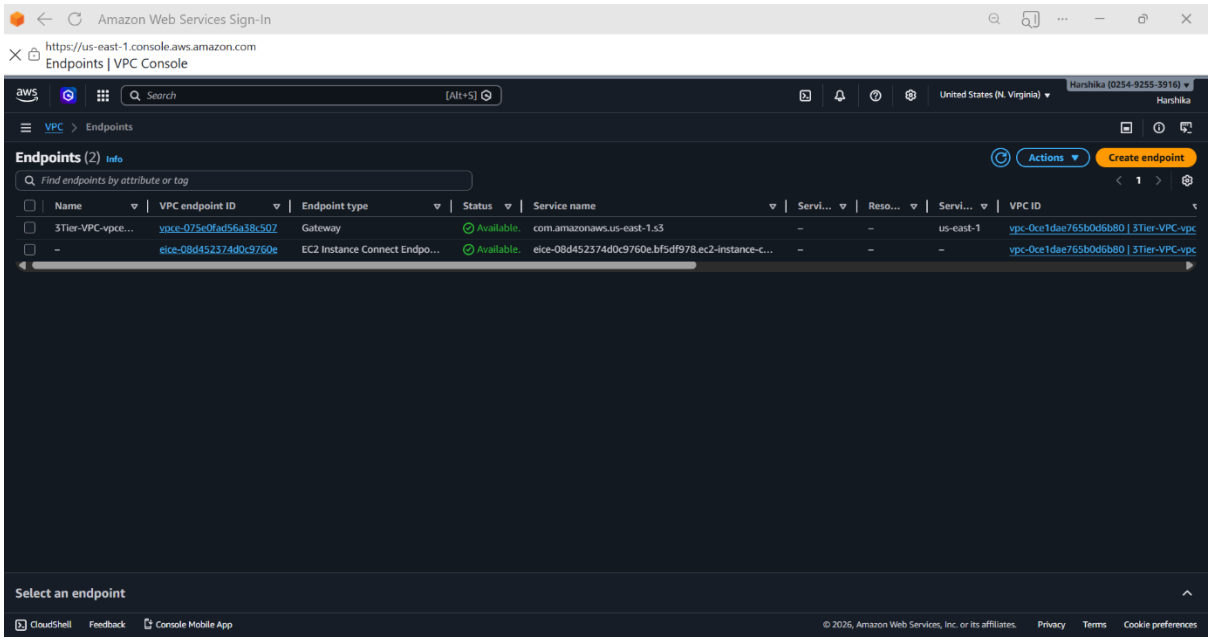
1. **Service category:** Select the radio button for **EC2 Instance Connect Endpoint**



2. **VPC:** Select your 3Tier-VPC.
3. **Security Groups:** Select your App-Server-SG.
4. **Subnet:** Select any **Private Subnet** from your VPC list.
5. Click **Create endpoint**.



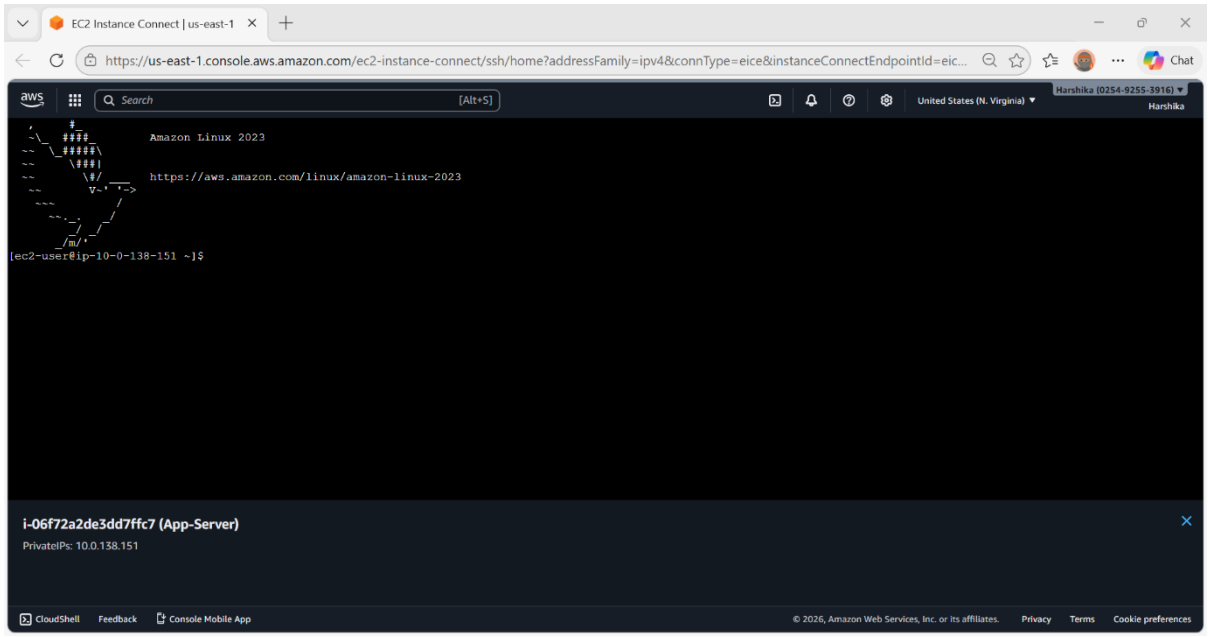




Once the Black Terminal Opens:

We are now "inside" our private App-Server.

To get the backend running, run these commands one by one:



1. Install Node.js:

Bash

sudo yum update -y

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh | bash

source ~/.bashrc

nvm install 16

nvm use 16

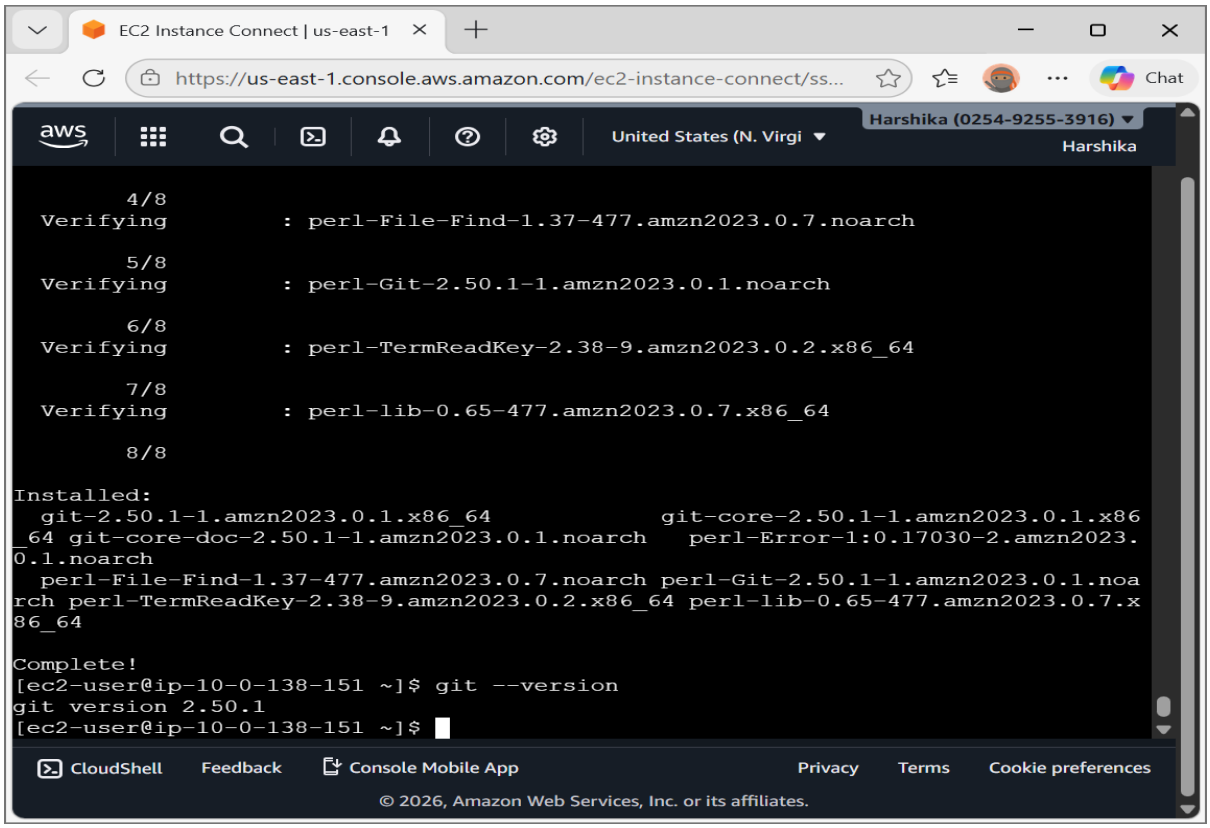
npm install -g pm2

2. Install Git

Run this command to give your server "Git" powers:

Bash

sudo yum install git -y



3. Clone the Repository

Now that Git is installed, run the clone command again:

```
git clone https://github.com/harshika369/AWS3TierApp.gi
```

```
cd AWS3TierApp/application-code/app-tier/
```

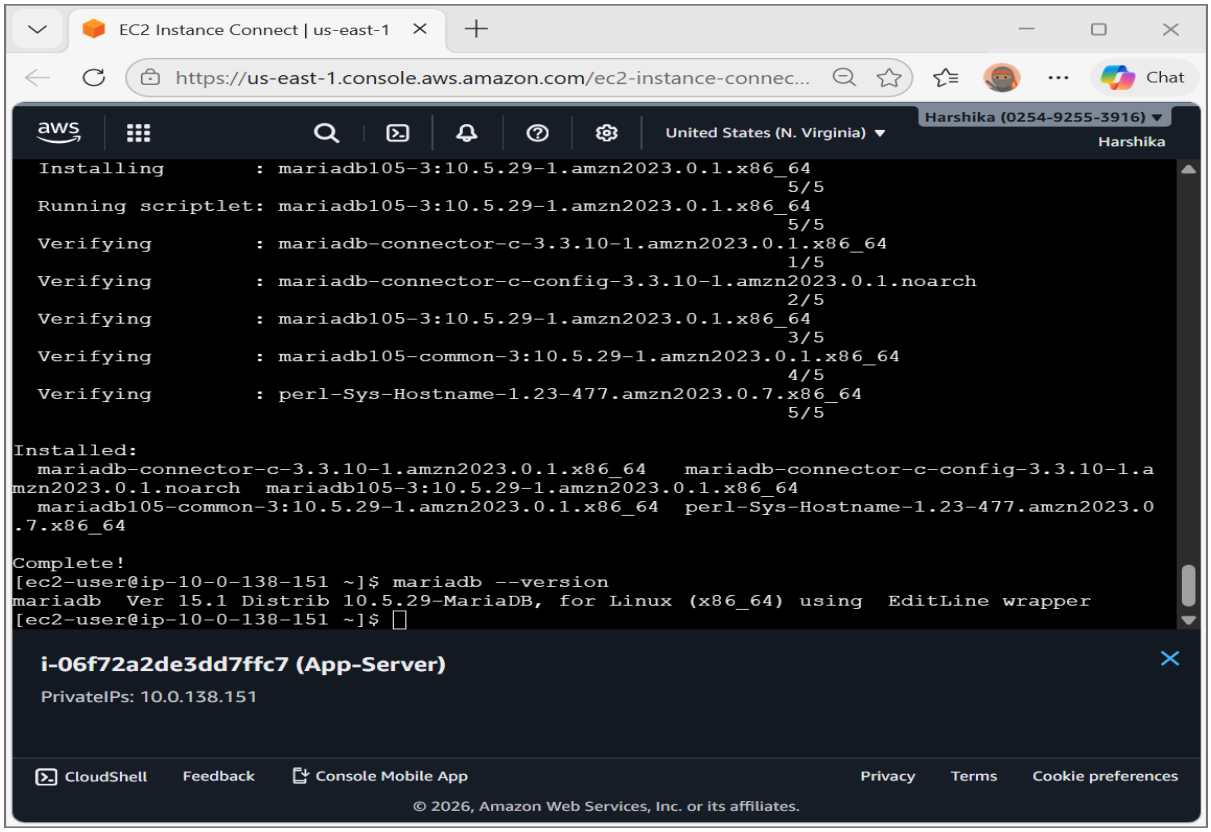
```
[ec2-user@ip-10-0-138-151 ~]$ git clone https://github.com/harshika369/AWS3TierA
pp.git                        git clone https://github.com/harshika369/AWS3TierA
pp.git
Cloning into 'AWS3TierApp'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 59 (delta 7), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (59/59), 357.49 KiB | 23.83 MiB/s, done.
Resolving deltas: 100% (7/7), done.
[ec2-user@ip-10-0-138-151 ~]$
```

4. Environment Preparation

First, install the necessary tools to communicate with the database and manage the Node.js process.

```
# Update and install MariaDB (for manual DB checks)
```

```
sudo dnf install mariadb105 -y
```



Install Node.js Process Manager (PM2)

sudo npm install -g pm2

5. Directory & Dependencies

cd /home/ec2-user/AWS3TierApp/application-code/app-tier/

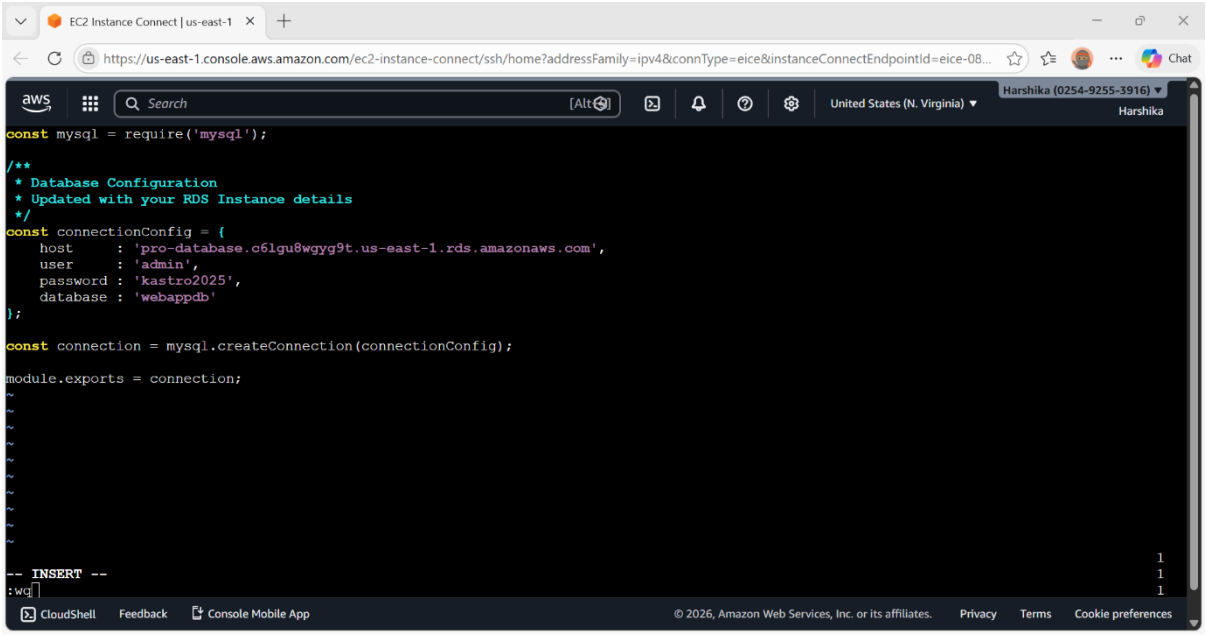
npm install mysql2

6. Update DbConfig.js

Vi DbConfig.js

```
ec2-user@ip-10-0-138-151 ~]$ cd AWS3TierApp/application-code/app-tier/
ec2-user@ip-10-0-138-151 app-tier]$ vi DbConfig.js
```

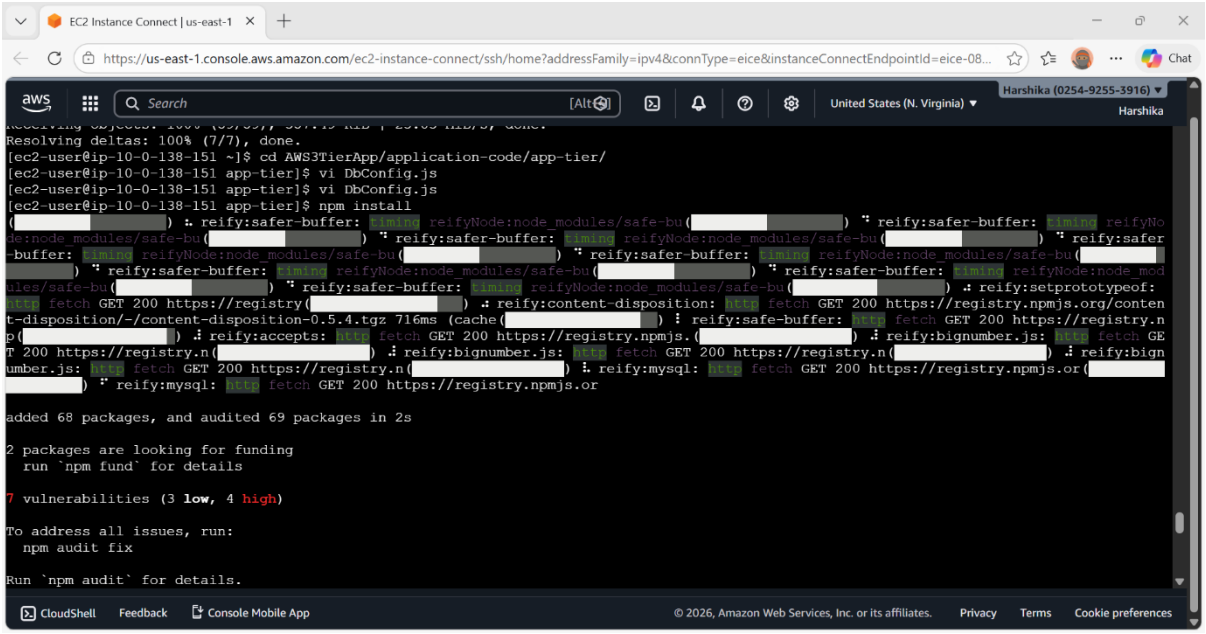
```
module.exports = Object.freeze({
   DB_HOST: 'pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com',
   DB_USER: 'admin',
   DB_PWD: 'kastro2025',
   DB_DATABASE: 'webappdb'
});
```



7. Update TransactionService.js

```
cat <<EOF > /home/ec2-user/AWS3TierApp/application-code/app-tier/TransactionService.js

const dbcreds = require('./DbConfig');

const mysql = require('mysql2');


const con = mysql.createConnection({
  host: dbcreds.DB_HOST,
  user: dbcreds.DB_USER,
  password: dbcreds.DB_PWD,
  database: dbcreds.DB_DATABASE
});
function addTransaction(amount,desc){
  var sql = "INSERT INTO transactions (amount, description) VALUES (?, ?)";
  con.query(sql, [amount, desc], function(err,result){
    if (err) throw err;
  });
  return 200;
}
function getAllTransactions(callback){
  var sql = "SELECT * FROM transactions";
  con.query(sql, function(err,result){
    if (err) throw err;
    return(callback(result));
  });
```

```
}
function findTransactionById(id,callback){
   var sql = "SELECT * FROM transactions WHERE id = ?";
   con.query(sql, [id], function(err,result){
      if (err) throw err;
      return(callback(result));
   });
}
function deleteAllTransactions(callback){
   var sql = "DELETE FROM transactions";
   con.query(sql, function(err,result){
      if (err) throw err;
      return(callback(result));
   });
}
function deleteTransactionById(id, callback){
   var sql = "DELETE FROM transactions WHERE id = ?";
   con.query(sql, [id], function(err,result){
      if (err) throw err;
      return(callback(result));
   });
}
module.exports = {addTransaction, getAllTransactions, deleteAllTransactions,
findTransactionById, deleteTransactionById};
EOF
```

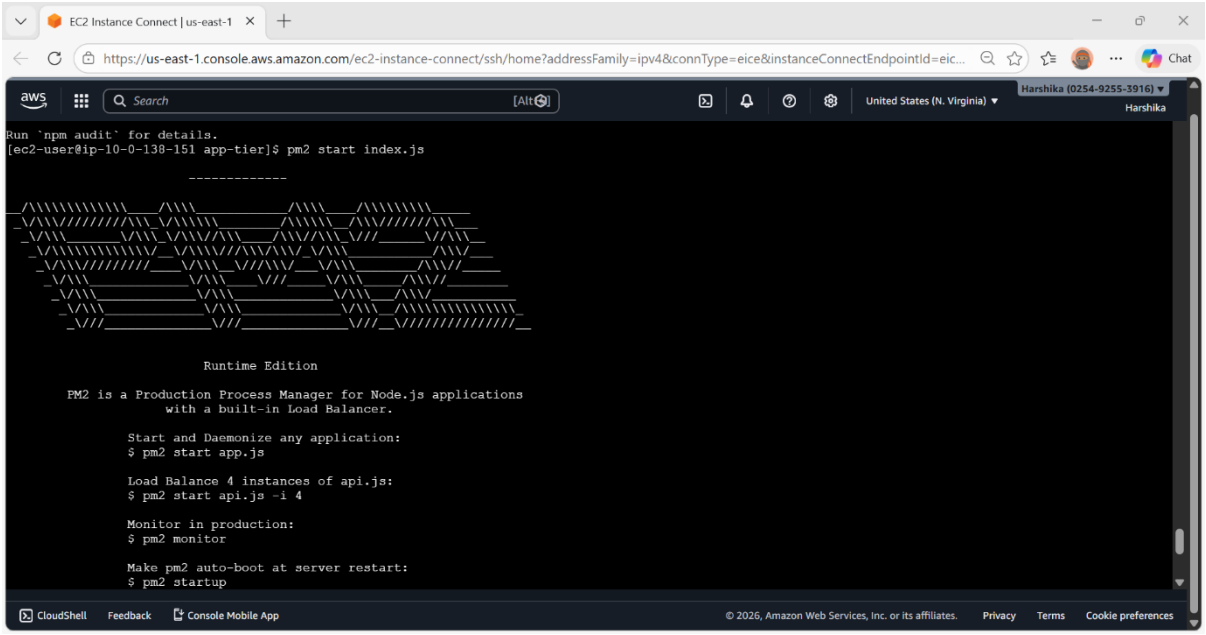


8. Execution & Final Validation

Now, clear the process memory and start the server.

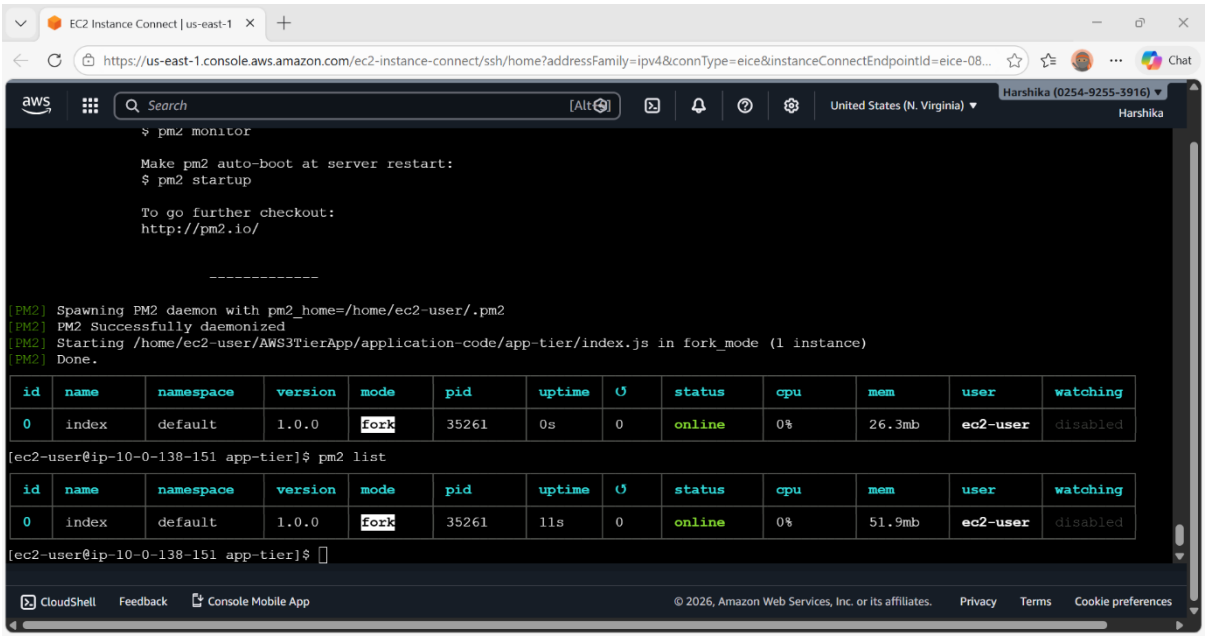
pm2 delete all

pm2 start index.js



Wait 5 seconds for the DB handshake

sleep 5



The Final Test

curl http://localhost:4000/transaction

Success Indicators

- **PM2 Status:** Should show online in green.
- **JSON Output:** You should see {"result":[{"id":1,"amount":"400.00","description":"groceries"}...]}.

Troubleshooting:

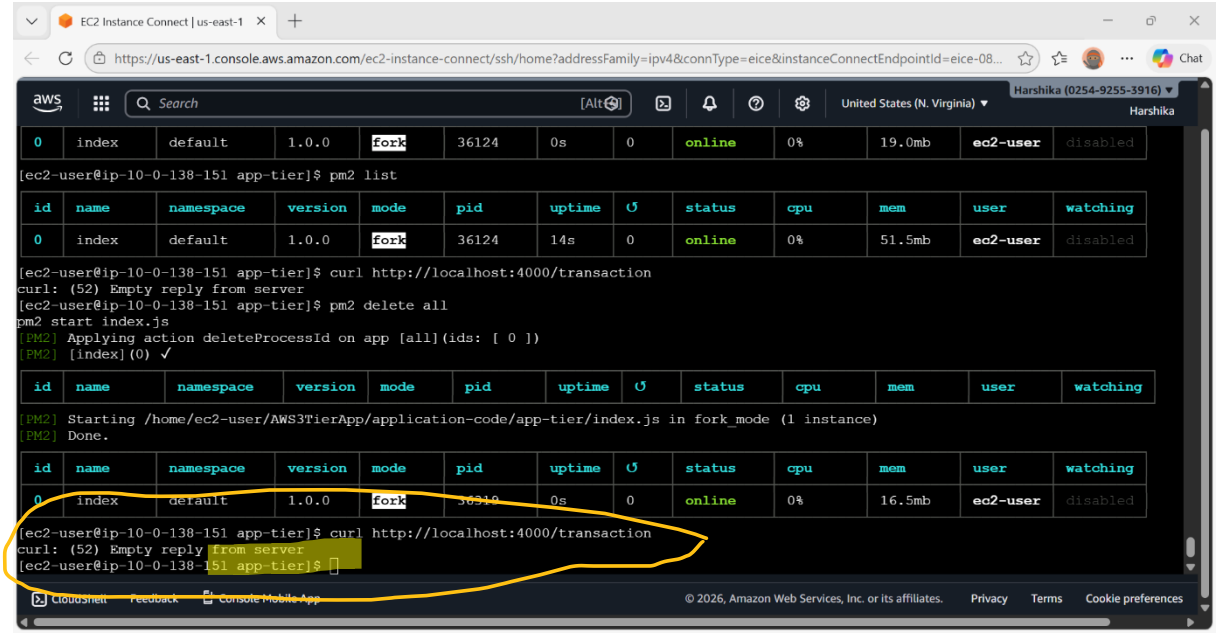
[ec2-user@ip-10-0-138-151 app-tier]$ curl http://localhost:4000/health

"This is the health check"[ec2-user@ip-10-0-138-151 app-tier]$ curl http://localcurl http://localhost:4000/transaction

curl: (52) Empty reply from server

[ec2-user@ip-10-0-138-151 app-tier]$

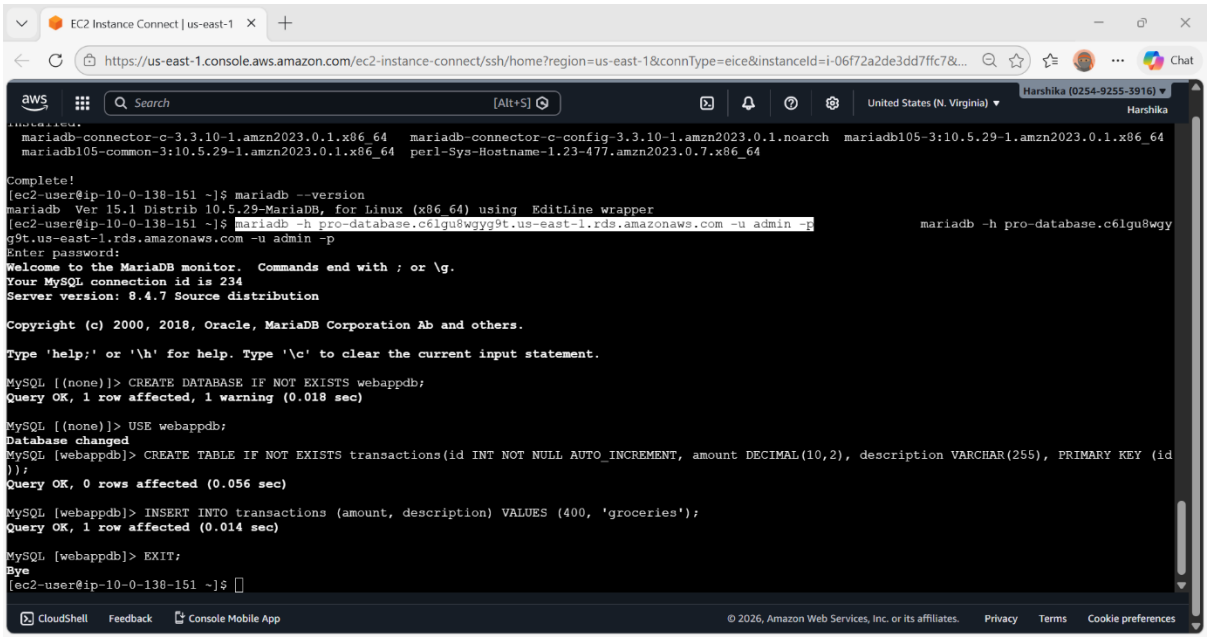
The empty **reply from the server** on the /transaction route means the app is crashing the moment it tries to talk to the RDS database.



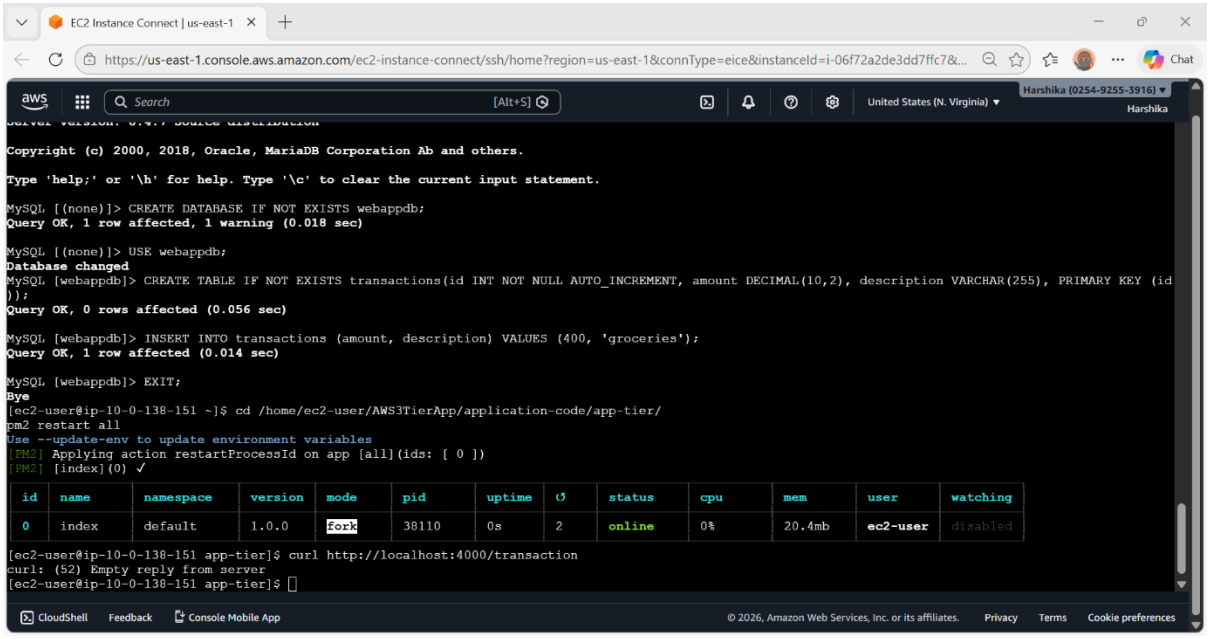
1. The Quick Fix: RDS Security Group

The Database has its own "shield" called RDS-SG (sg-01f644fc2e33cef66). We need to make sure it allows traffic from the App Server.

1. Go to the **AWS Console** > **VPC** > **Security Groups**.
2. Select **RDS-SG**.
3. Click **Edit inbound rules**.
4. Check your current rule. It likely shows port 3306 with source 0.0.0.0/0.
5. **Change the Source:** Delete 0.0.0.0/0 and instead start typing sg-. Select your **App-Server-SG** (sg-072f02f6805aa316a) from the list.
6. **Log into your RDS Database:** (When it asks for the password, use kastro2025.)



7. mariadb -h pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com -u admin -p
8. CREATE DATABASE IF NOT EXISTS webappdb;
9. USE webappdb;
10. CREATE TABLE IF NOT EXISTS transactions(id INT NOT NULL AUTO_INCREMENT, amount DECIMAL(10,2), description VARCHAR(255), PRIMARY KEY (id));
11. INSERT INTO transactions (amount, description) VALUES (400, 'groceries');
12. EXIT;



3.Restart your App Tier:

cd /home/ec2-user/AWS3TierApp/application-code/app-tier/

pm2 restart all

4. The Final Verification:

curl http://localhost:4000/transaction

Step 1: Check Database & Table Manually

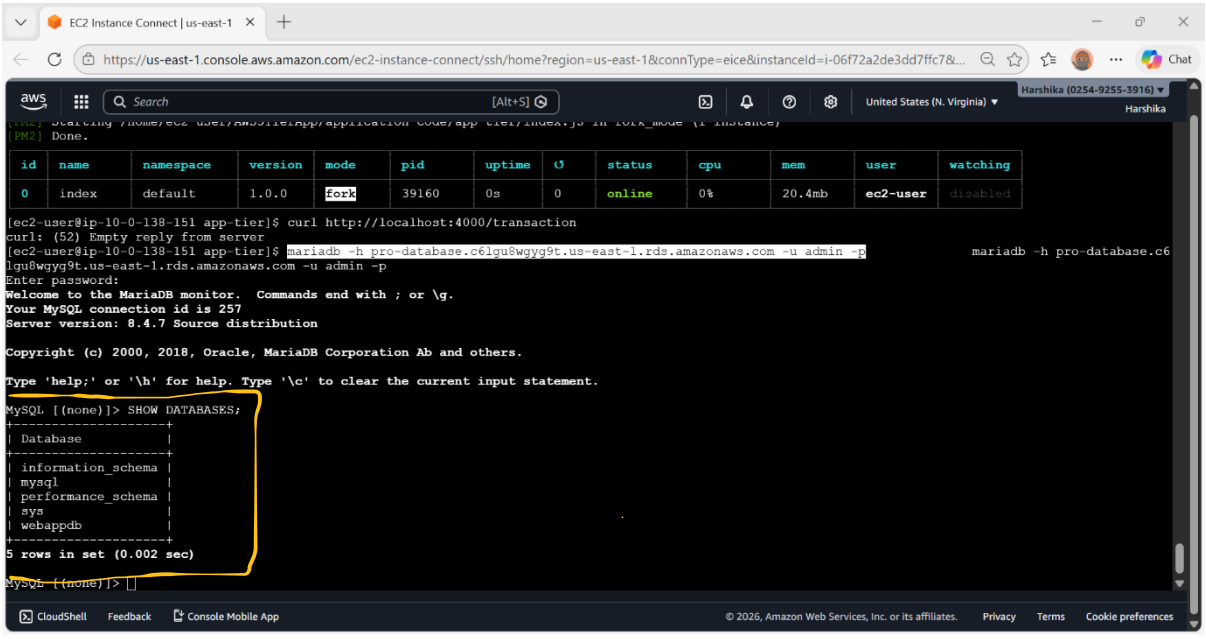
mariadb -h pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com -u admin -p

(Enter your password: kastro2025)

Once you see the MariaDB [(none)]> prompt, run these **three commands**.

1. **See if the database exists:**

SHOW DATABASES;



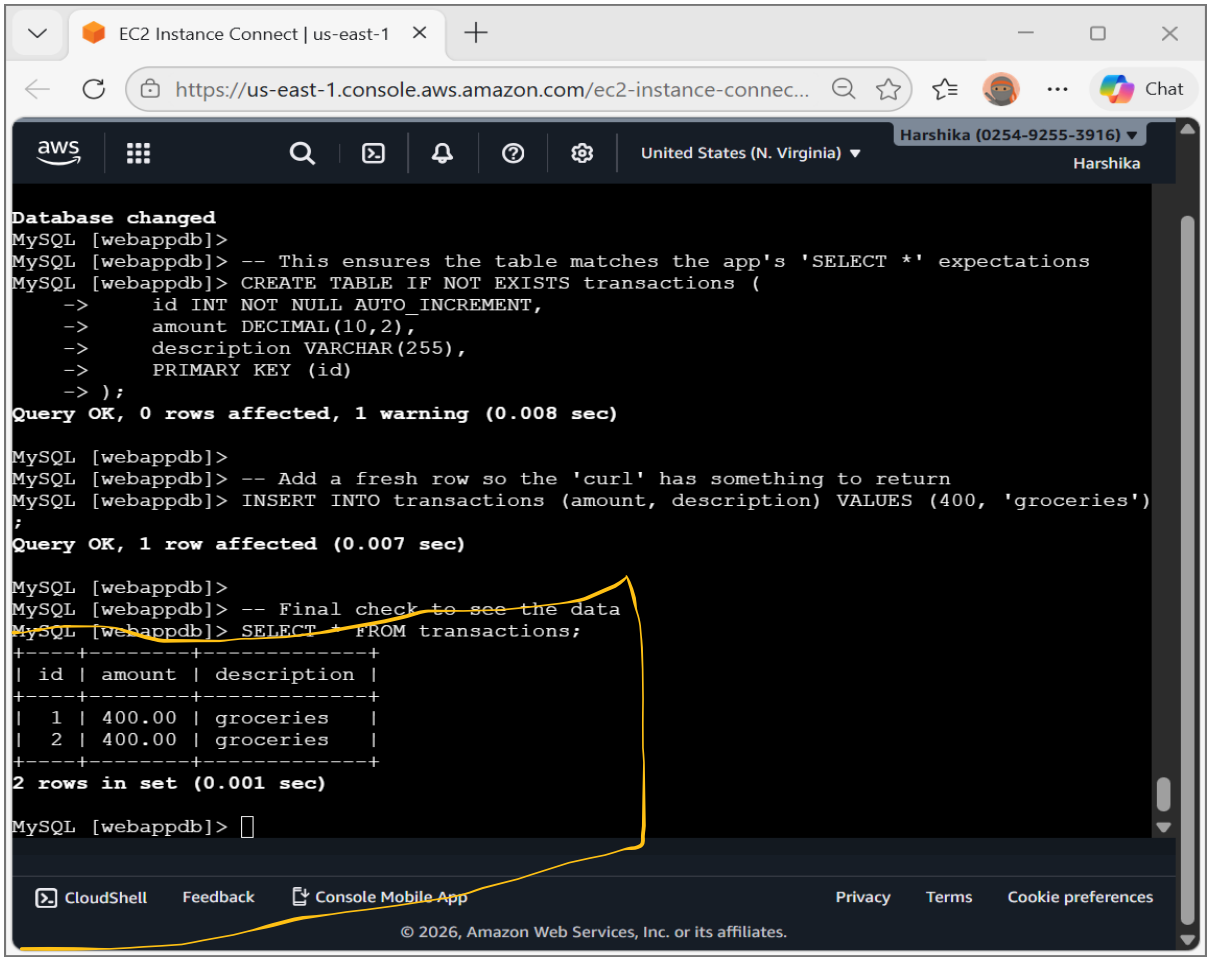
USE webappdb;

SHOW TABLES;

We can see transactions in the list.

SELECT * FROM transactions;

It shows the "groceries" row.



Step 2: Restart and Verify

pm2 delete all

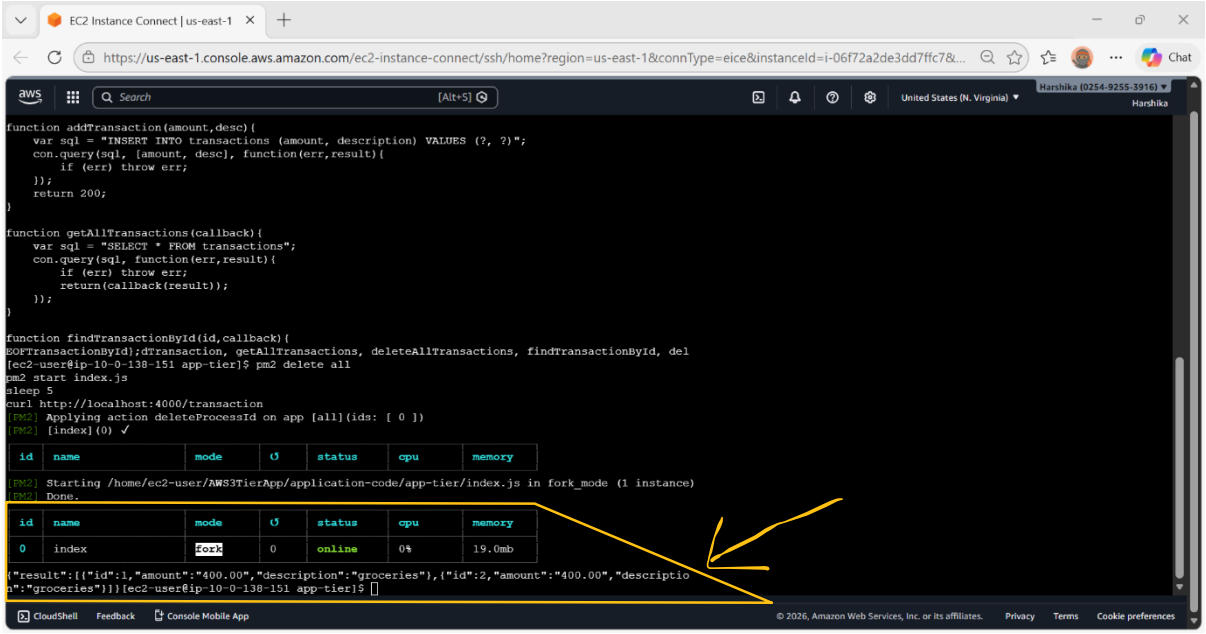
pm2 start index.js

sleep 5

curl http://localhost:4000/transaction

Success Indicators

- **PM2 Status:** Should show online in green.
- **JSON Output:** You should see {"result":[{"id":1,"amount":"400.00","description":"groceries"}...]}

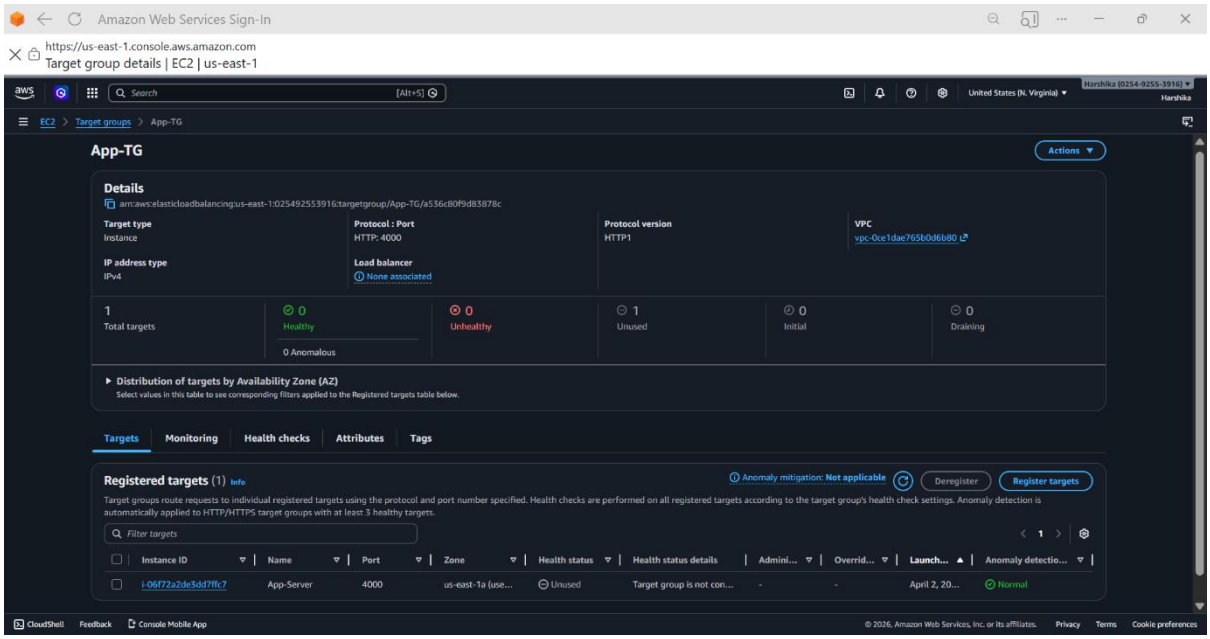


Phase 3: Internal Load Balancer Setup

Step 1: Create a Target Group

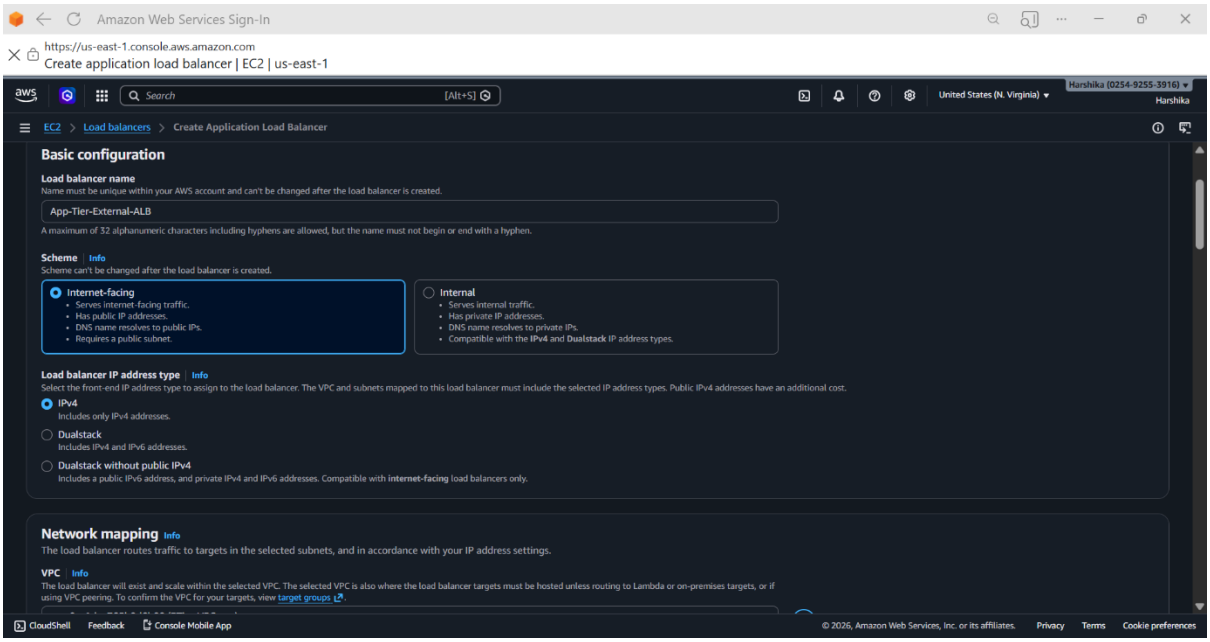
The Target Group specifies where to send traffic (to the App Server).

1. Go to **EC2 Console** -> **Target Groups** -> **Create target group**.
2. **Target type:** Instances.
3. **Target group name:** App-TG.
4. **Protocol:** HTTP | **Port:** 4000 (Because the Node app runs on 4000).
5. **VPC:** Select your 3-Tier VPC.
6. **Health checks:** Path should be /health (or just / if you haven't set a health route).
7. Click **Next**, select your **App-Tier Instance**, click **Include as pending below**, and then **Create target group**.

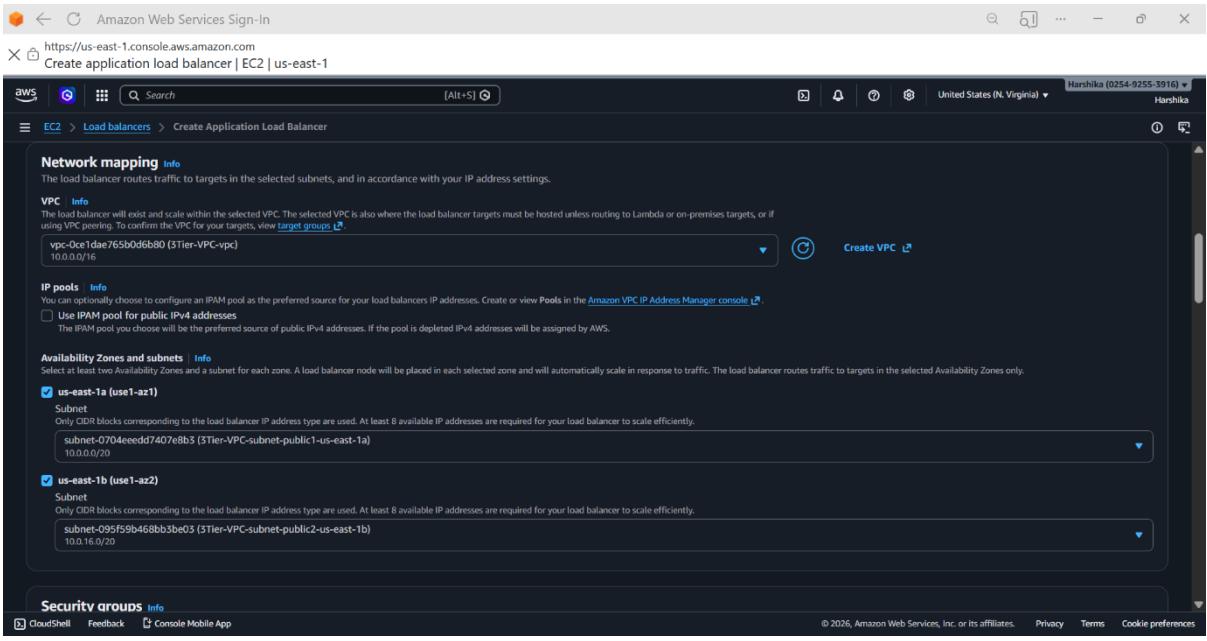


Step 2: Create the Internal Load Balancer

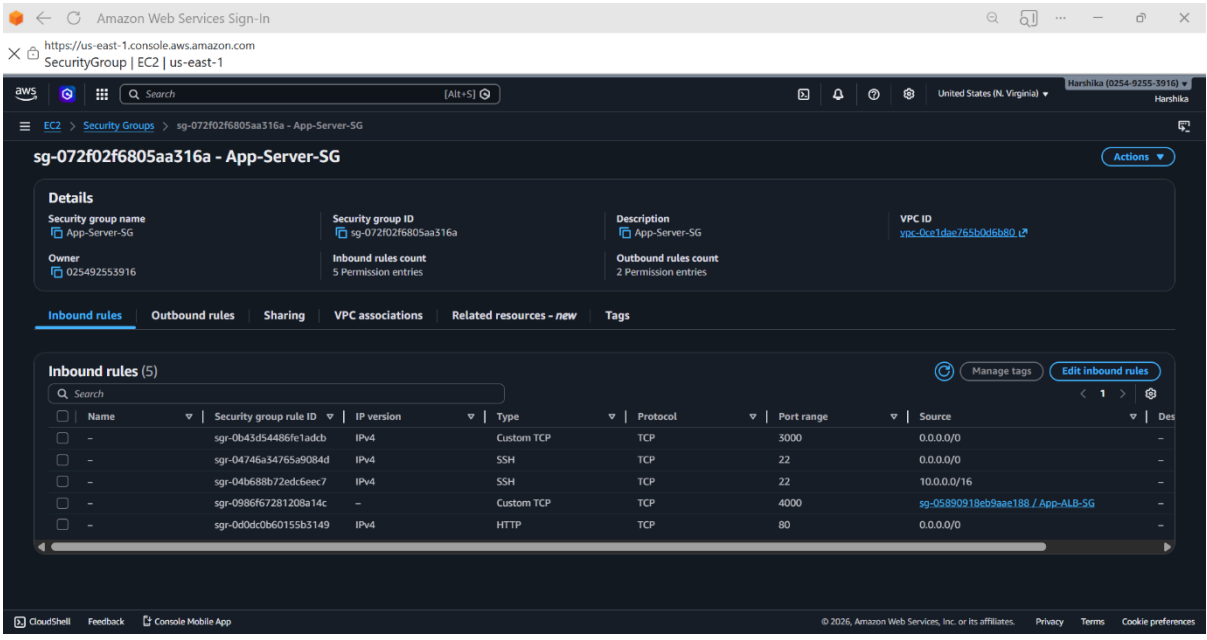
1. **Go to:** EC2 Dashboard -> **Load Balancers** -> **Create Load Balancer**.
2. **Select: Application Load Balancer**.
3. **Basic Configuration:**
    - **Name:** App-Tier-External-ALB
    - **Scheme:** Internet-facing (This is the most important setting).
    - **IP address type:** IPv4.



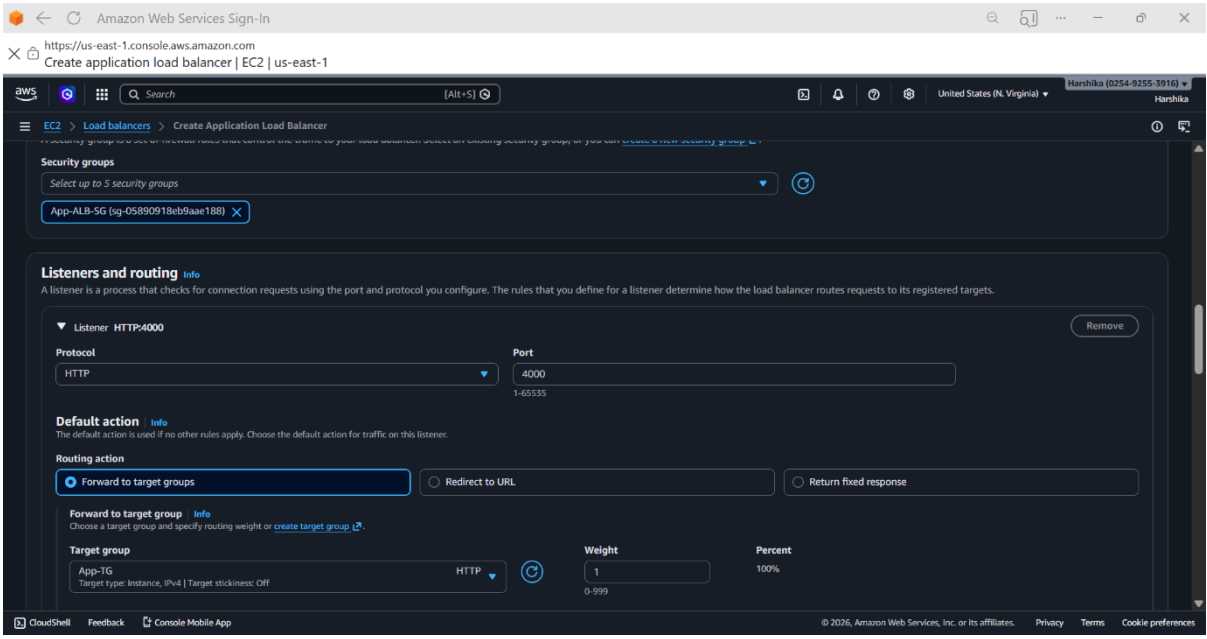
4. **Network Mapping:**
    - **VPC:** Select your 3-Tier VPC.
    - **Subnets:** Select your **Public Subnets**. (Even though it's for the App Tier, the browser needs a public entry point to send the "Save" request).



5. **Security Groups:**
    - **Choose:** A Security Group that allows **Inbound Port 4000** from **0.0.0.0/0**.



6. **Listeners and Routing:**
    - **Protocol:** HTTP | **Port:** 4000.
    - **Default action:** Forward to your existing **App-Tier-TG**.

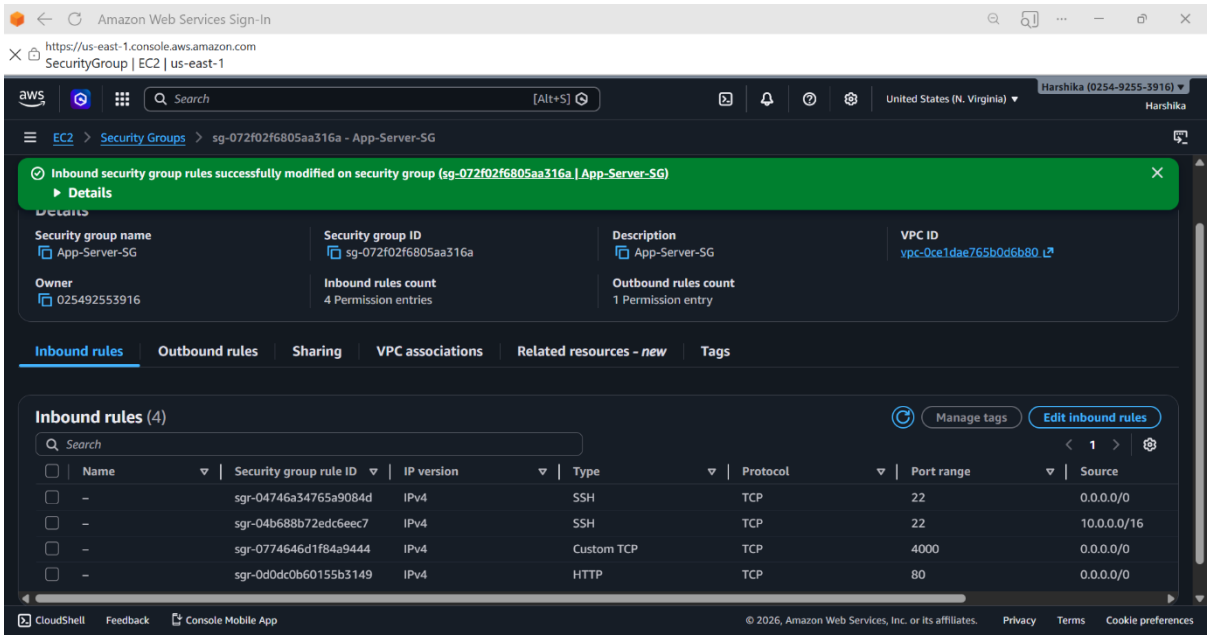


7. **Create:** Click Create and wait for the status to show **Active**.

Step 3: Edit the App-Server-SG

We must edit the App-Server-SG **after** or **during** the creation of the ALB. The App Server needs to "trust" the new Load Balancer.

1. **Go to:** Security Groups -> Select **sg-072f02f6805aa316a (App-Server-SG).**
2. **Action:** Edit **Inbound Rules**.
3. **Add Rule:**
    - **Type:** Custom TCP.
    - **Port:** 4000.
    - **Source:** Select the **Security Group ID** of the Load Balancer you just created in Step 1.
4. **Save Rules.**



The Final Step: Connecting the Web Tier

Now we need to tell the **Web Server** to send its requests to the **Load Balancer** instead of trying to find the database on its own.

Step 1: Get the New DNS Name

1. Go to your **App-Tier-External-ALB** details.
2. Copy the **DNS name**: App-Tier-External-ALB-1927615714.us-east-1.elb.amazonaws.com.

Step 2: Update the Backend Address

Run this to tell the React website where the App Server lives.

Bash

1. Copy your App ALB DNS name and paste it below

export NEW_APP_DNS="App-Tier-External-ALB-1927615714.us-east-1.elb.amazonaws.com"

2. Update the configuration file with the URL

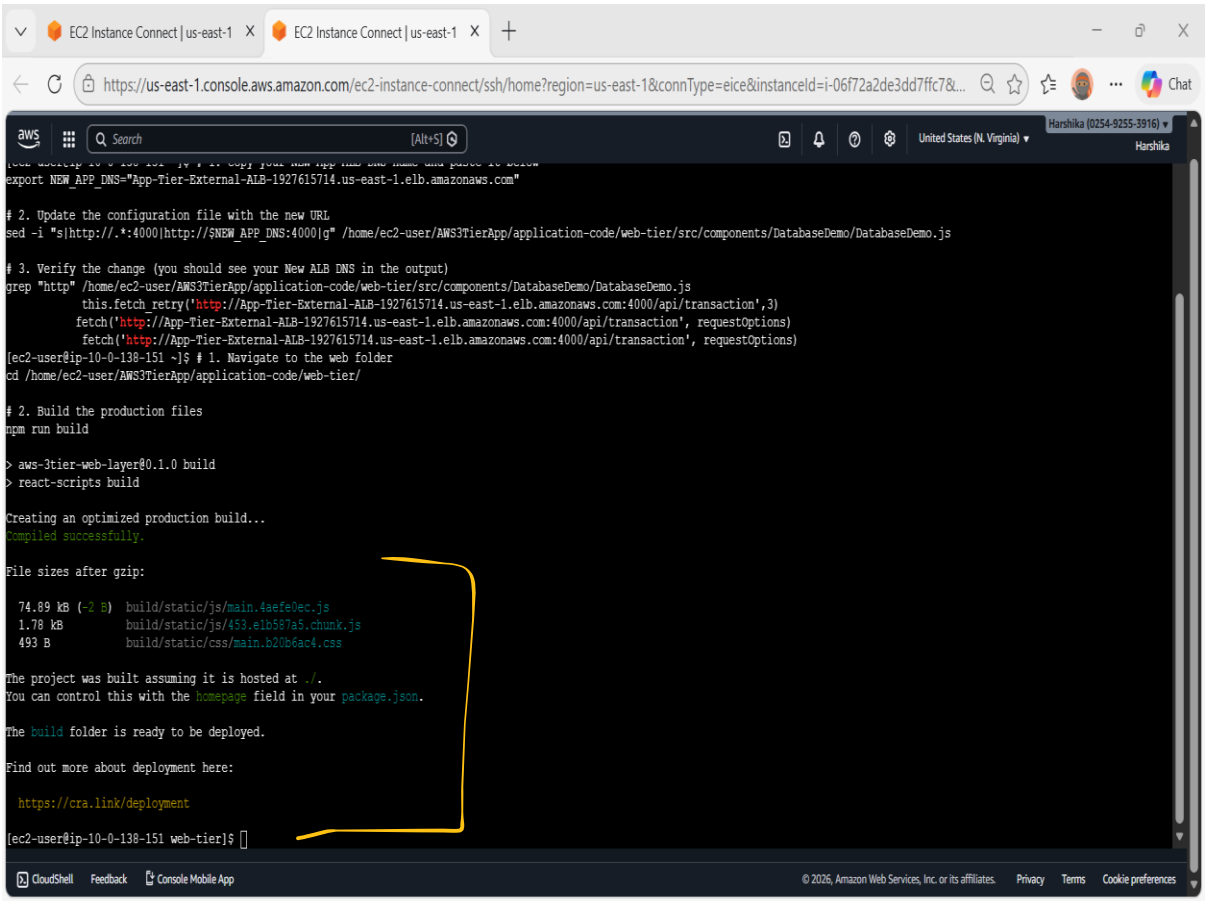
sed -i "s|http://.*:4000|http://$NEW_APP_DNS:4000|g" /home/ec2-user/AWS3TierApp/application-code/web-tier/src/components/DatabaseDemo/DatabaseDemo.js

Step 3: Build the Web Application

```
# 1. Navigate to the web folder

cd /home/ec2-user/AWS3TierApp/application-code/web-tier/

# 2. Build the production files

npm run build
```

Step 4: Start the Web Service

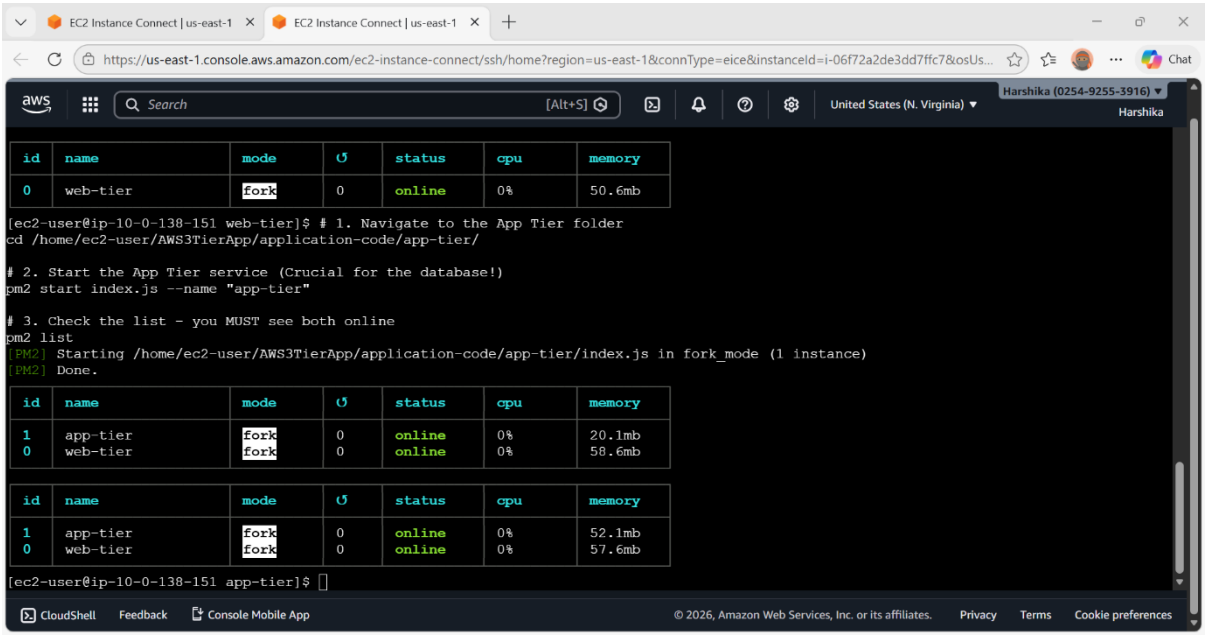
```
# 1. Start the web tier using PM2

pm2 start npm --name "web-tier" – start
```



2. Check the status

pm2 list

*We should see the web-tier and app-tier both showing **online** in green.*

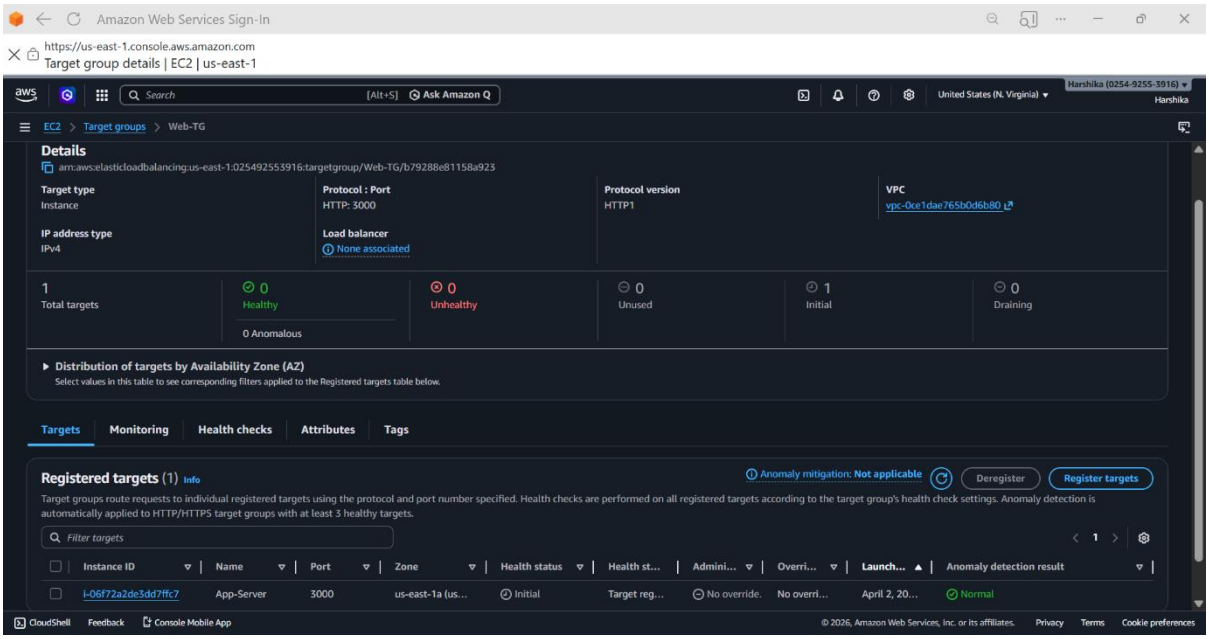


Phase 4: The External "Front Door" (ALB)

1. Create a Web Target Group

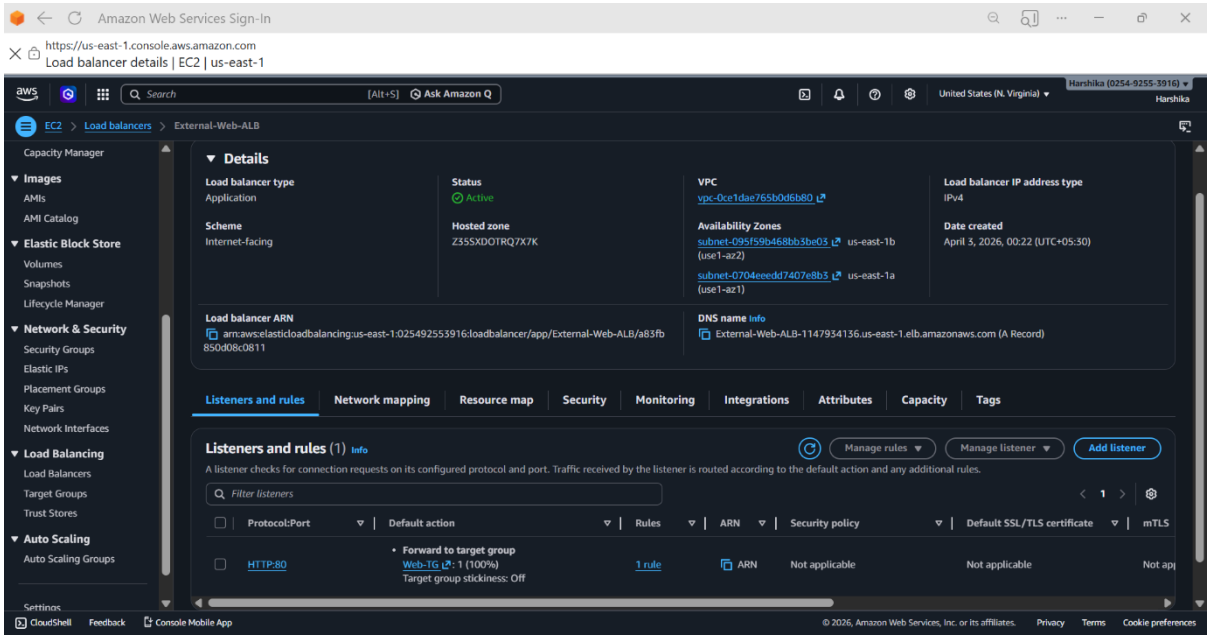
This tells the new Load Balancer to send traffic to your Web Server.

- **Target type:** Instances.
- **Name:** Web-TG.
- **Protocol**: HTTP | **Port:** 3000 (since React runs on 3000).
- **VPC:** Your 3-Tier VPC.
- **Health Check Path:** /
- **Register Targets:** Select your **Web-Tier Instance** and click **Include as pending**.



2. Create the External Load Balancer

- **Name:** External-Web-ALB.
- **Scheme: Internet-facing** (This is the only one that should be public).
- **Mappings:** Select your **Public Subnets** (usually 2 of them).
- **Security Group:** Create one that allows **HTTP (Port 80)** from **0.0.0.0/0** (Everywhere).
- **Listeners:** Port 80 -> Forward to Web-TG.

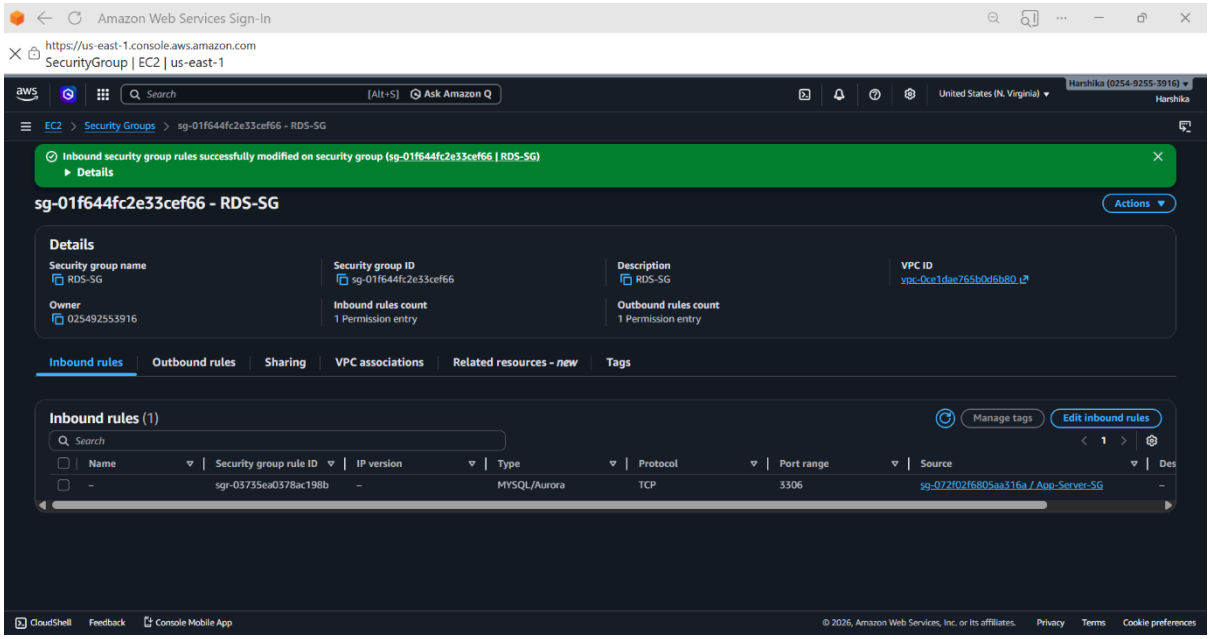


2. The Correct Security Group Setup

To keep your 3-tier lab secure but working, your rules should look like this:

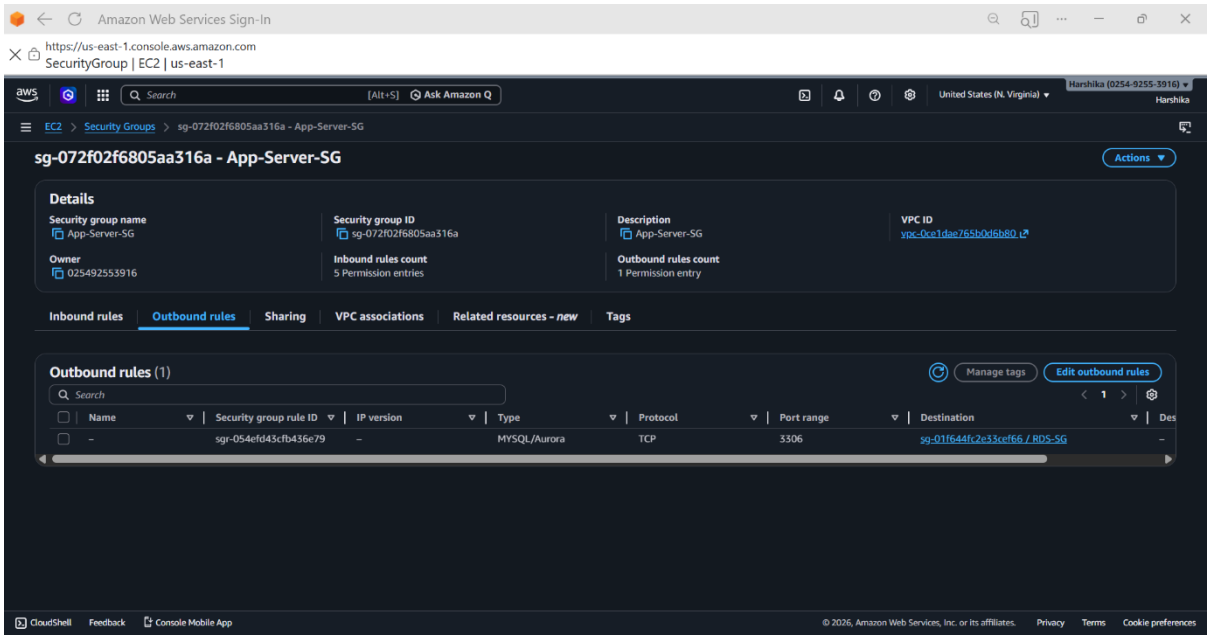
RDS-SG (Inbound Rules)

Type	Port	Source
MySQL/Aurora	**3306**	**App-Server-SG (ID)**



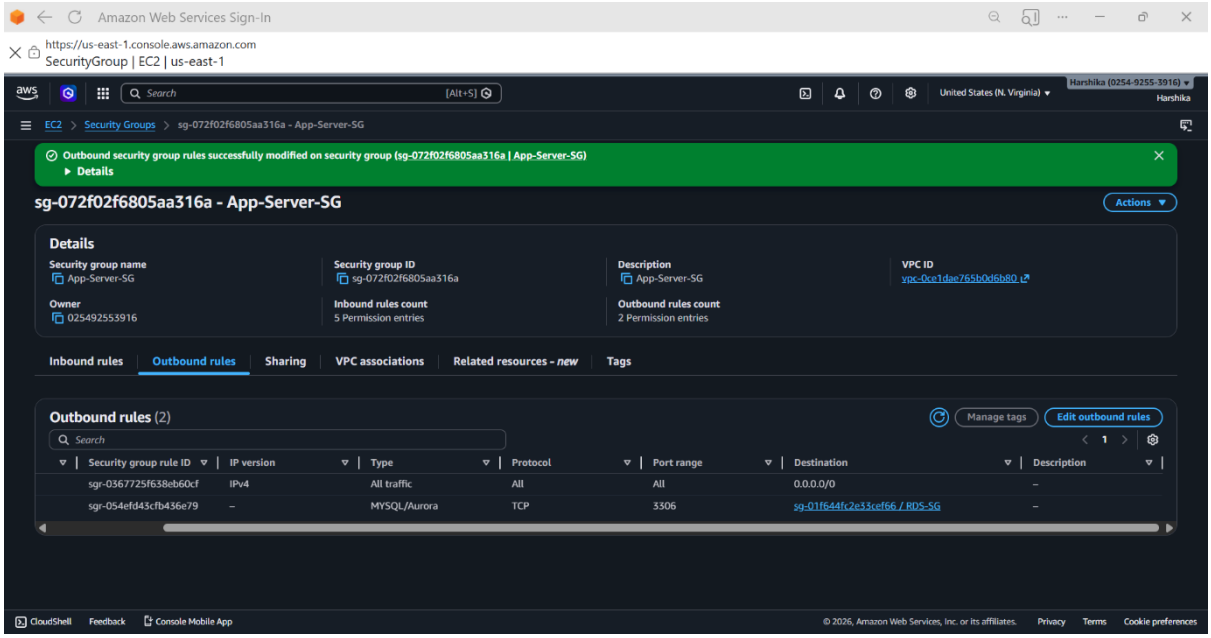
App-Server-SG (Inbound Rules)

Type	Port	Source
SSH	**22**	**0.0.0.0/0** (or your IP)
Custom TCP	**4000**	**Internal-ALB-SG (ID)**



App-Server-SG (Outbound Rules

Type	Port	Destination
All Traffic	**All**	**0.0.0.0/0**



1. Navigate to the web folder

cd /home/ec2-user/AWS3TierApp/application-code/web-tier/

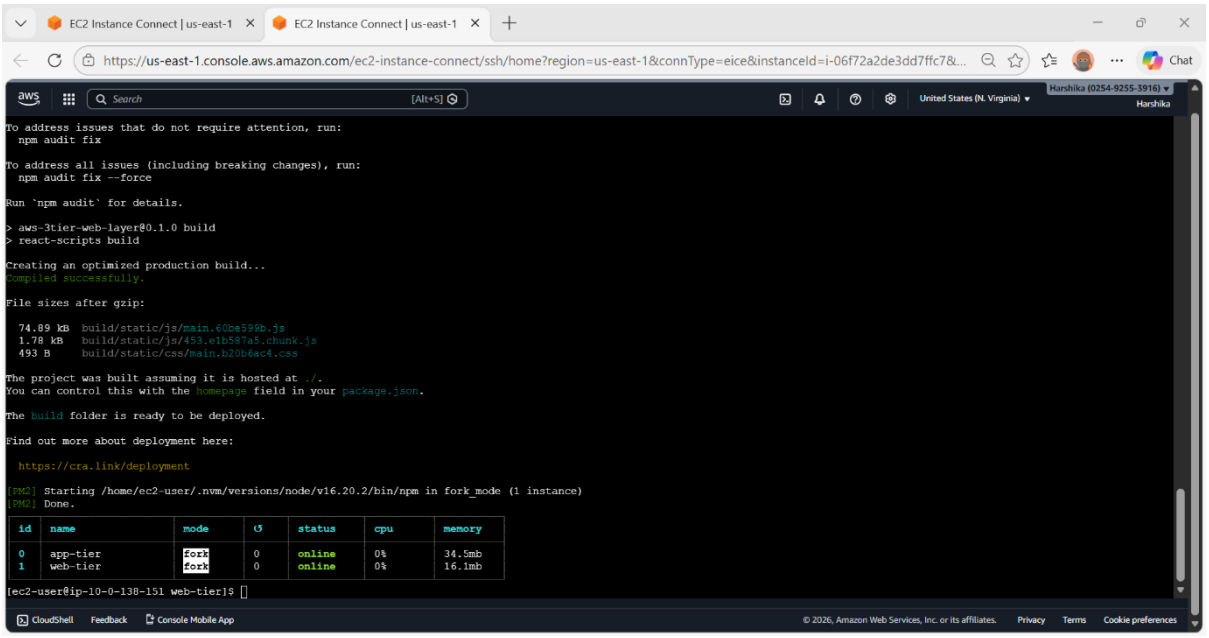
2. Build the production files

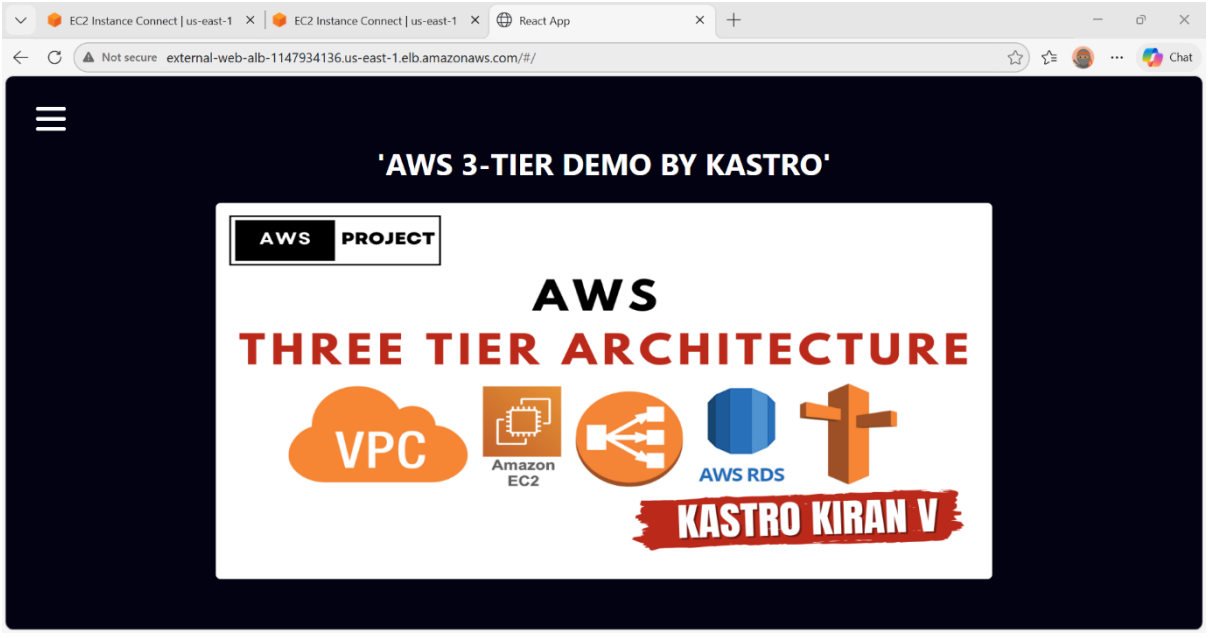
npm run build

3. Start the web tier using PM2

pm2 start npm --name "web-tier" – start

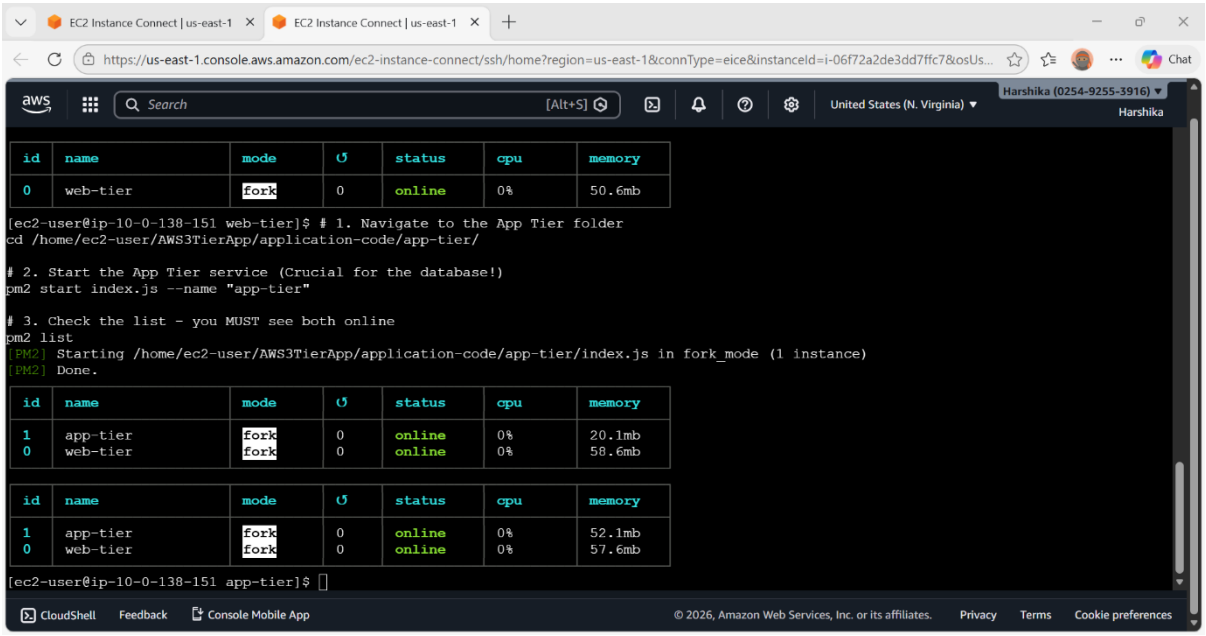
pm2 list





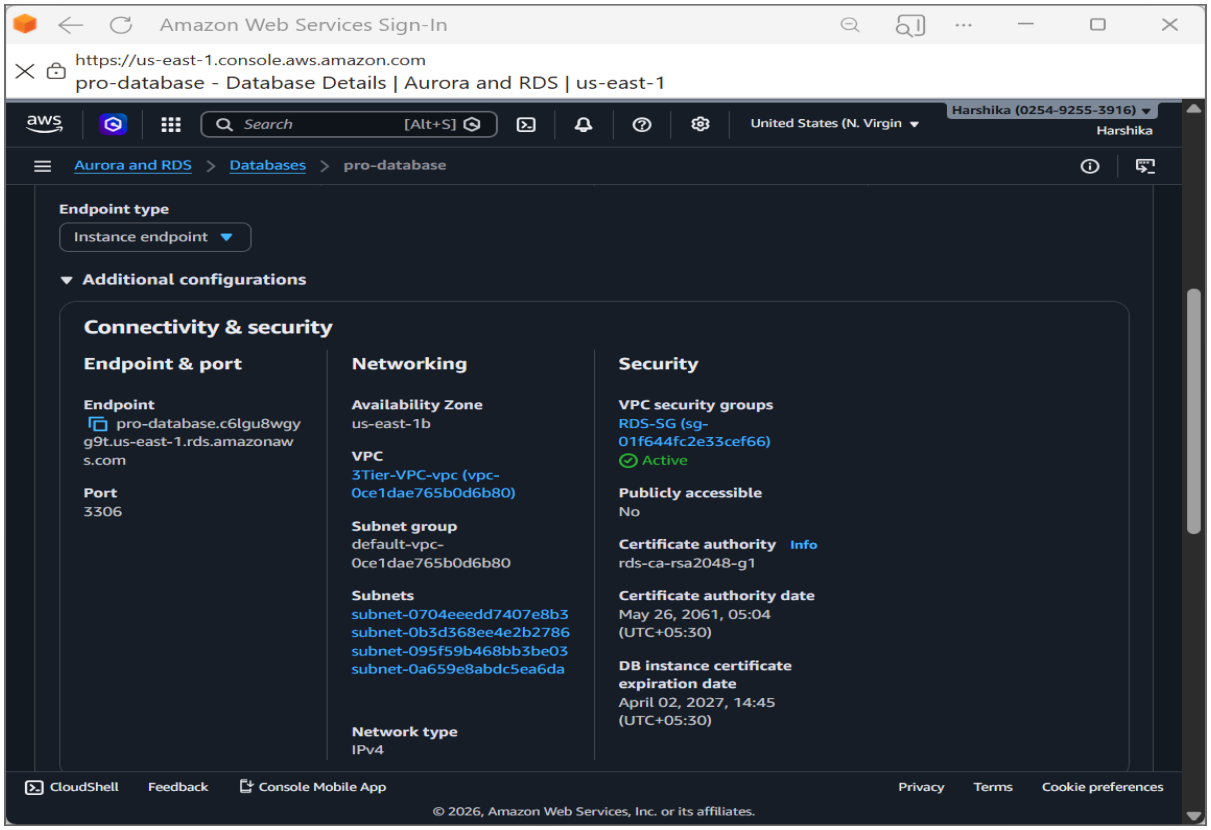
Troubleshooting: We can able to see the web page, but are not able to save an entry in the table.

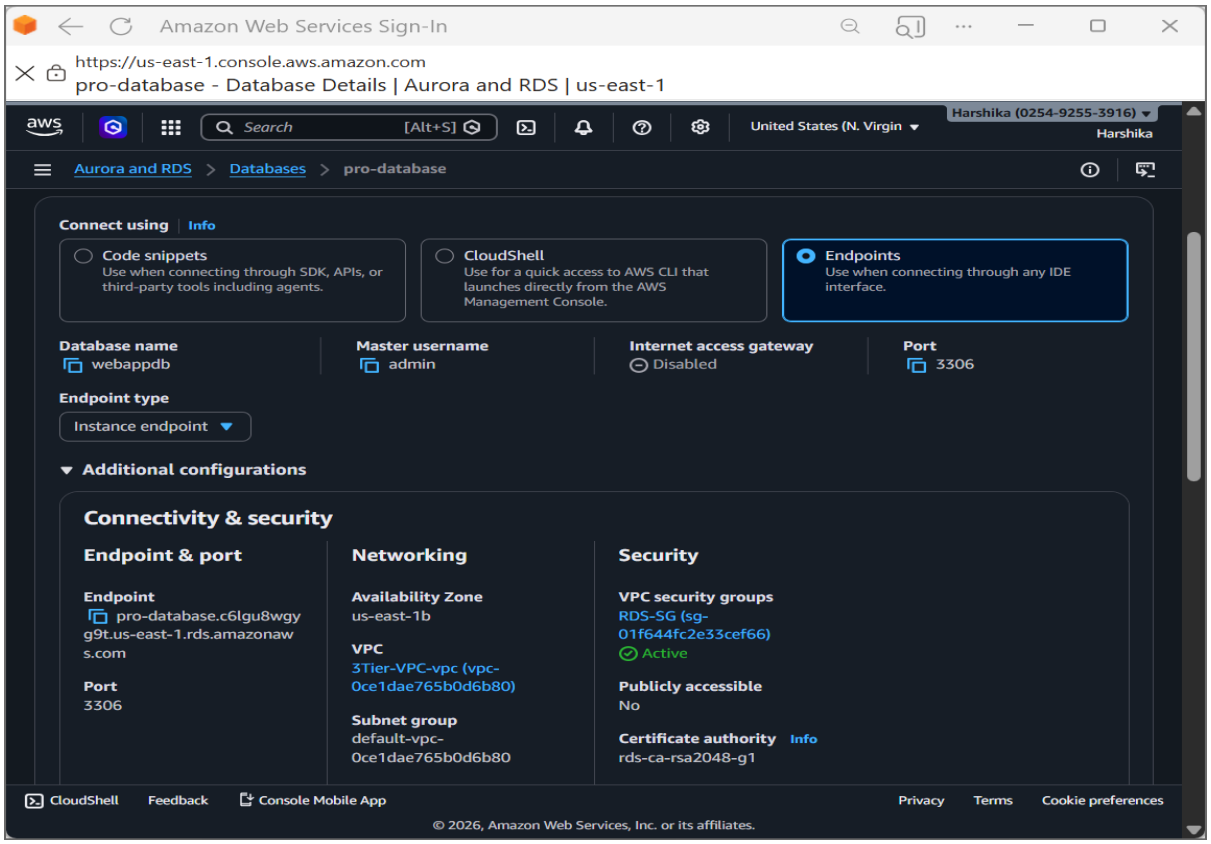
```
# 1. Navigate to the App Tier folder
cd /home/ec2-user/AWS3TierApp/application-code/app-tier/

# 2. Start the App Tier service (Crucial for the database!)
pm2 start index.js --name "app-tier"

# 3. Check the list - you MUST see both online
pm2 list
```



1. Get your App Load Balancer DNS

- Go to the **EC2 Console** -> **Load Balancers**.
- Select your **App-Tier-External-ALB**.
- Copy the **DNS name**.


Amazon Web Services Sign-In
https://us-east-1.console.aws.amazon.com
pro-database - Database Details | Aurora and RDS | us-east-1
Search [Alt+S]
United States (N. Virgin
Harshika (0254-9255-3916)
Harshika
Aurora and RDS > Databases > pro-database
Endpoint type
Instance endpoint
Additional configurations
Connectivity & security
Endpoint & port
Endpoint
pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com
Port
3306
Networking
Availability Zone
us-east-1b
VPC
3Tier-VPC-vpc (vpc-0ce1dae765b0d6b80)
Subnet group
default-vpc-0ce1dae765b0d6b80
Subnets
subnet-0704eeedd7407e8b3
subnet-0b3d368ee4e2b2786
subnet-095f59b468bb3be03
subnet-0a659e8abdc5ea6da
Network type
IPv4
Security
VPC security groups
RDS-SG (sg-01f644fc2e33cef66)
Active
Publicly accessible
No
Certificate authority Info
rds-ca-rsa2048-g1
Certificate authority date
May 26, 2061, 05:04 (UTC+05:30)
DB instance certificate expiration date
April 02, 2027, 14:45 (UTC+05:30)
CloudShell
Feedback
Console Mobile App
Privacy
Terms
Cookie preferences
© 2026, Amazon Web Services, Inc. or its affiliates.

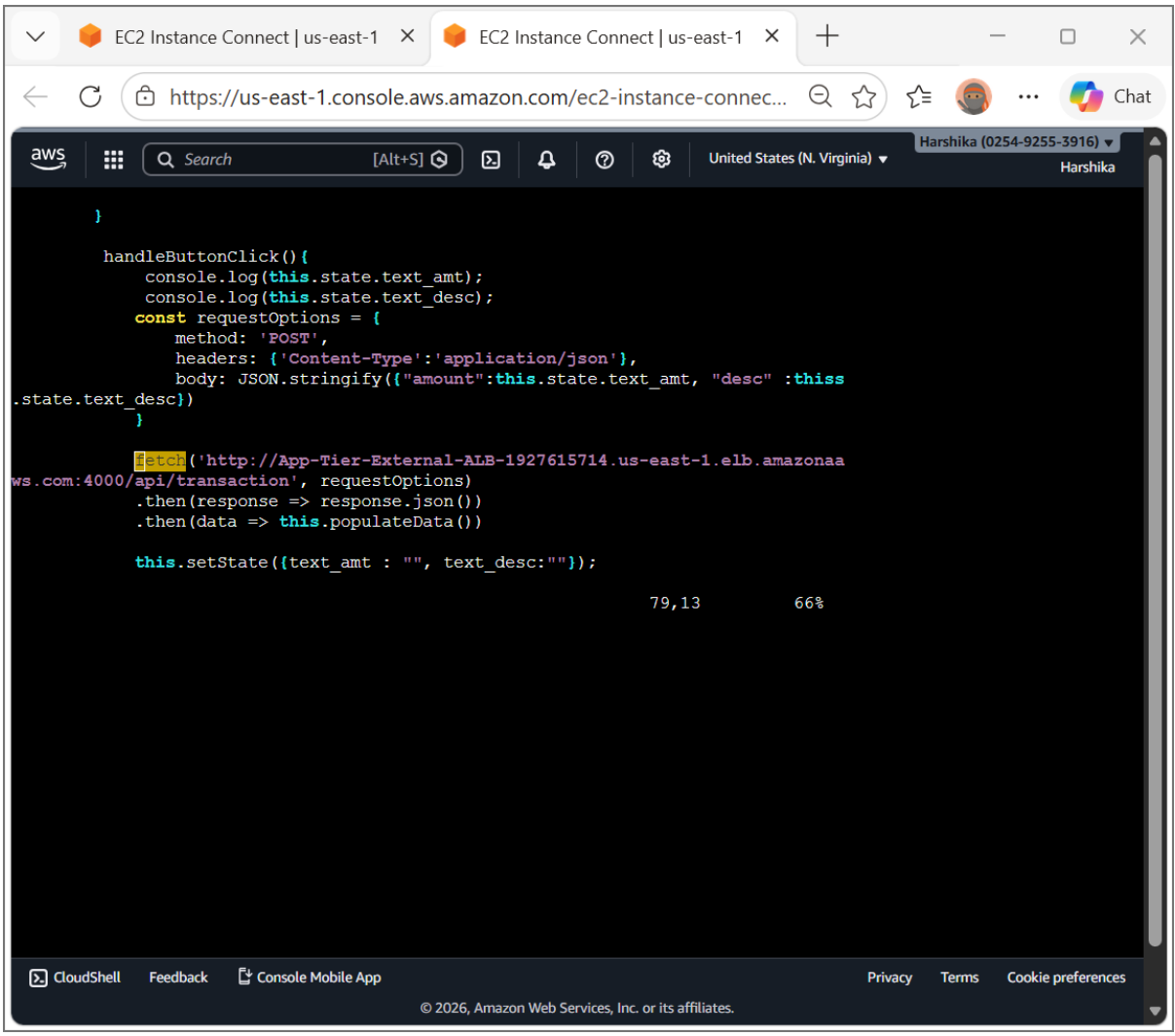

Amazon Web Services Sign-In
https://us-east-1.console.aws.amazon.com
pro-database - Database Details | Aurora and RDS | us-east-1
Search [Alt+S]
United States (N. Virgin
Harshika (0254-9255-3916)
Harshika
Aurora and RDS > Databases > pro-database
Connect using Info
Code snippets
Use when connecting through SDK, APIs, or third-party tools including agents.
CloudShell
Use for a quick access to AWS CLI that launches directly from the AWS Management Console.
Endpoints
Use when connecting through any IDE interface.
Database name
webappdb
Master username
admin
Internet access gateway
Disabled
Port
3306
Endpoint type
Instance endpoint
Additional configurations
Connectivity & security
Endpoint & port
Endpoint
pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com
Port
3306
Networking
Availability Zone
us-east-1b
VPC
3Tier-VPC-vpc (vpc-0ce1dae765b0d6b80)
Subnet group
default-vpc-0ce1dae765b0d6b80
Security
VPC security groups
RDS-SG (sg-01f644fc2e33cef66)
Active
Publicly accessible
No
Certificate authority Info
rds-ca-rsa2048-g1
CloudShell
Feedback
Console Mobile App
Privacy
Terms
Cookie preferences
© 2026, Amazon Web Services, Inc. or its affiliates.

Update the Frontend Code

1. Open your frontend file again:

vi /home/ec2-user/AWS3TierApp/application-code/web-tier/src/components/DatabaseDemo/DatabaseDemo.js

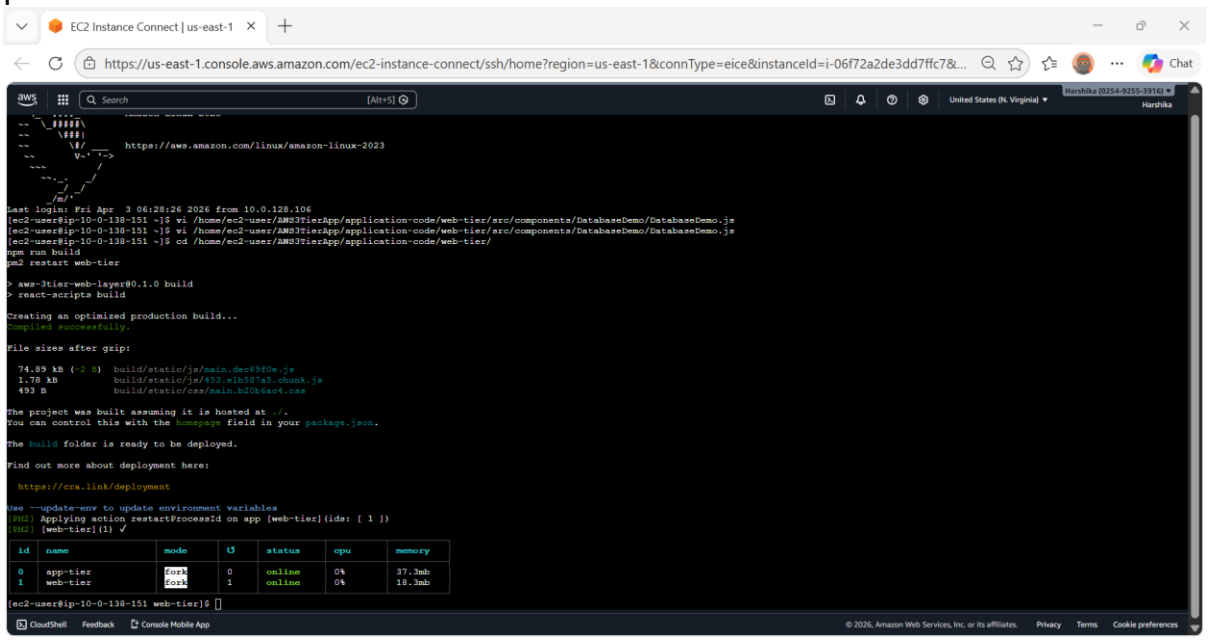
2. Find the lines where we pasted the Load Balancer DNS.
3. **Remove the /api part** from the URL.
    - **Change this:** http://...alb...:4000/api/transaction



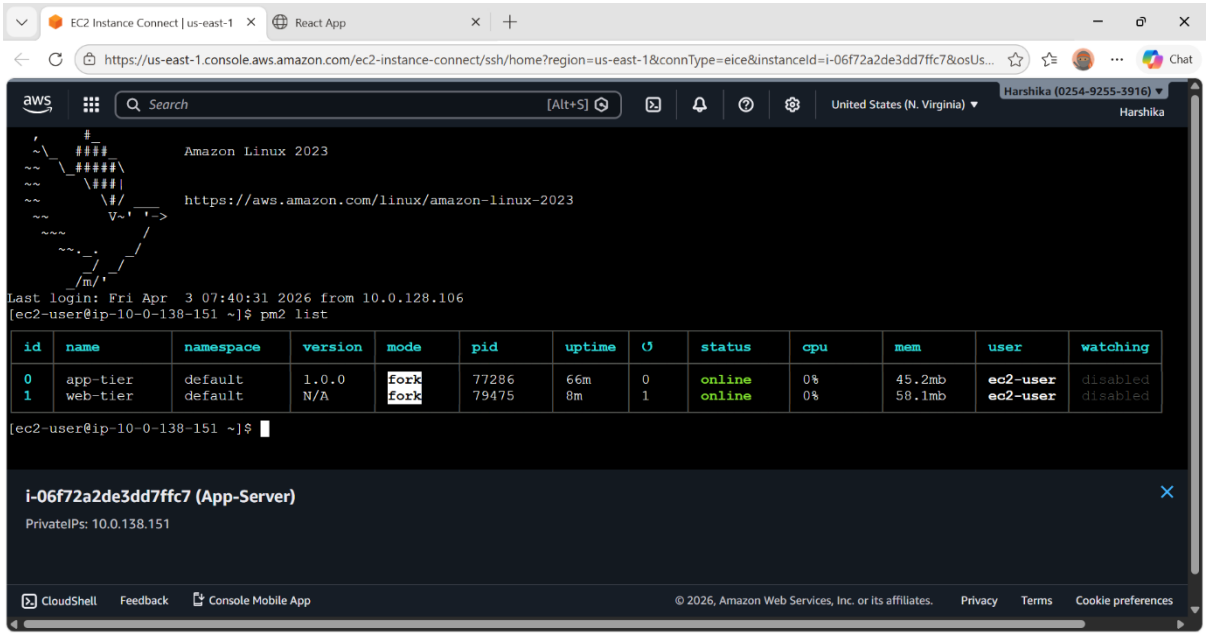
    - **To this:** http://...alb...:4000/transaction

4. **Save and Exit** (:wq).
5. **Rebuild the frontend**:

cd /home/ec2-user/AWS3TierApp/application-code/web-tier/
npm run build
pm2 restart web-tier

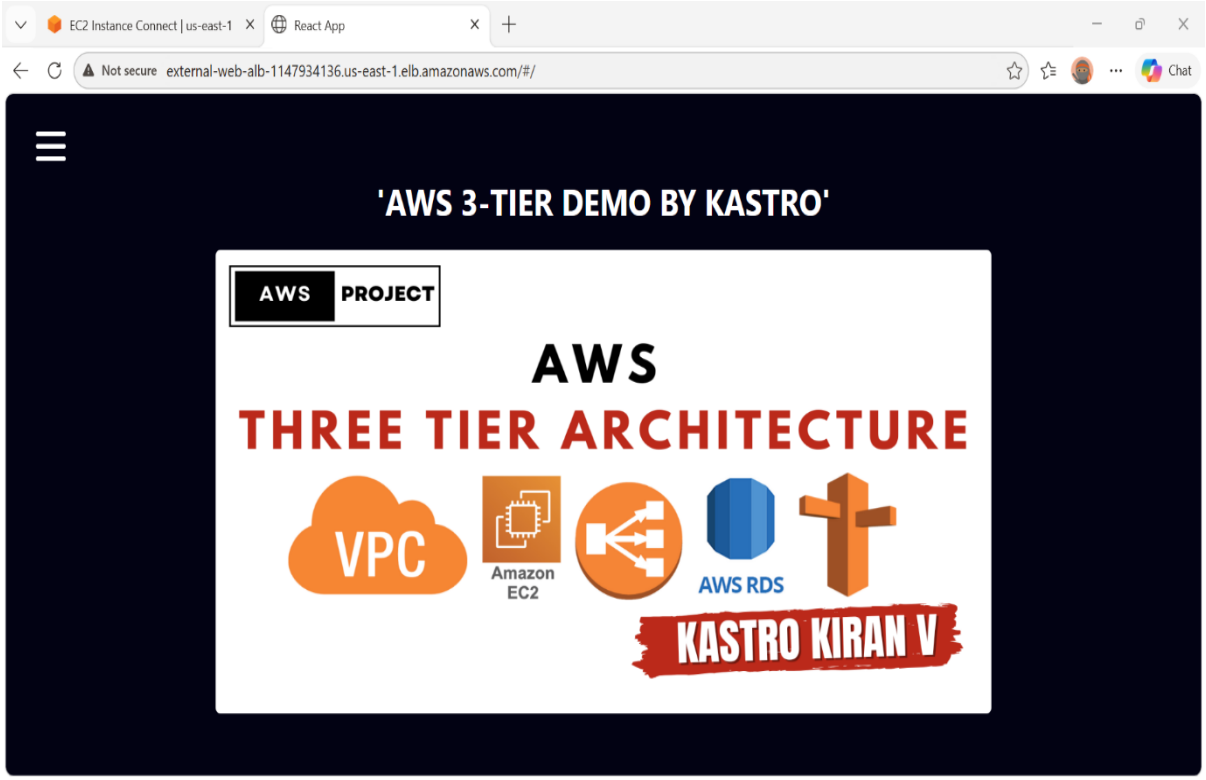


- **Command:** pm2 list

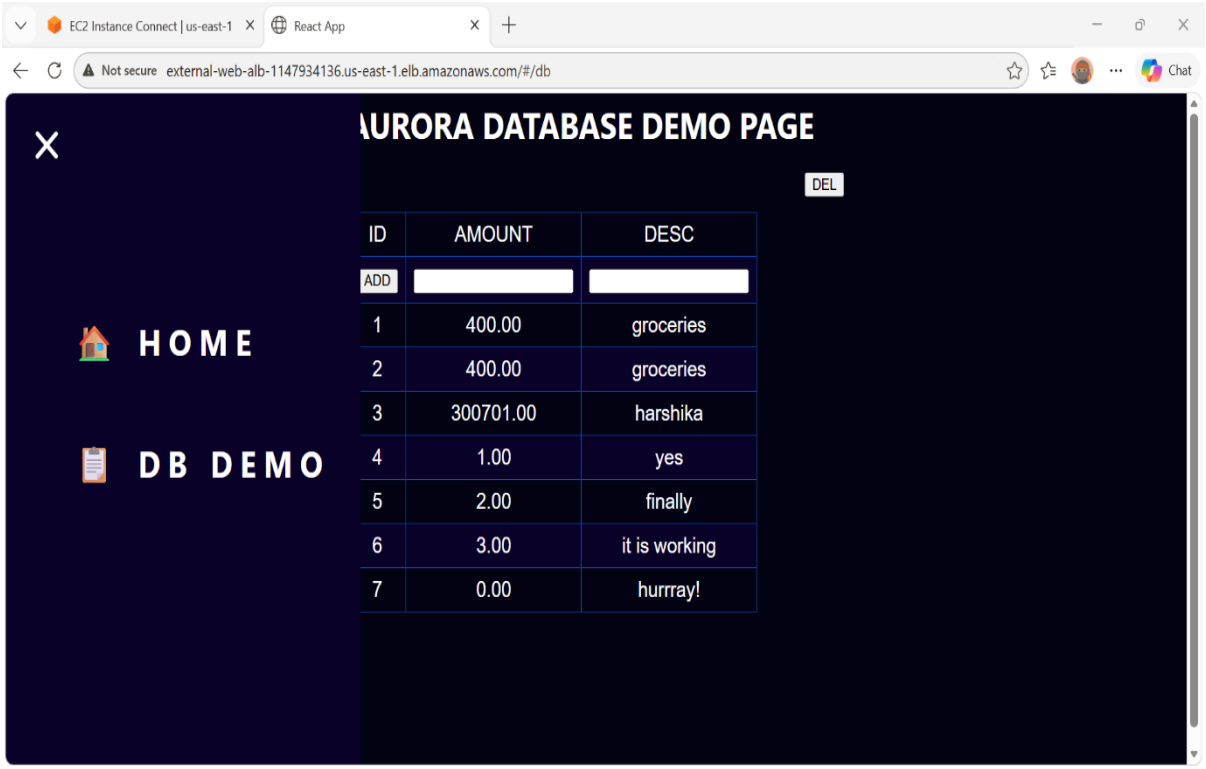


- Copy the URL and paste it into any browser.

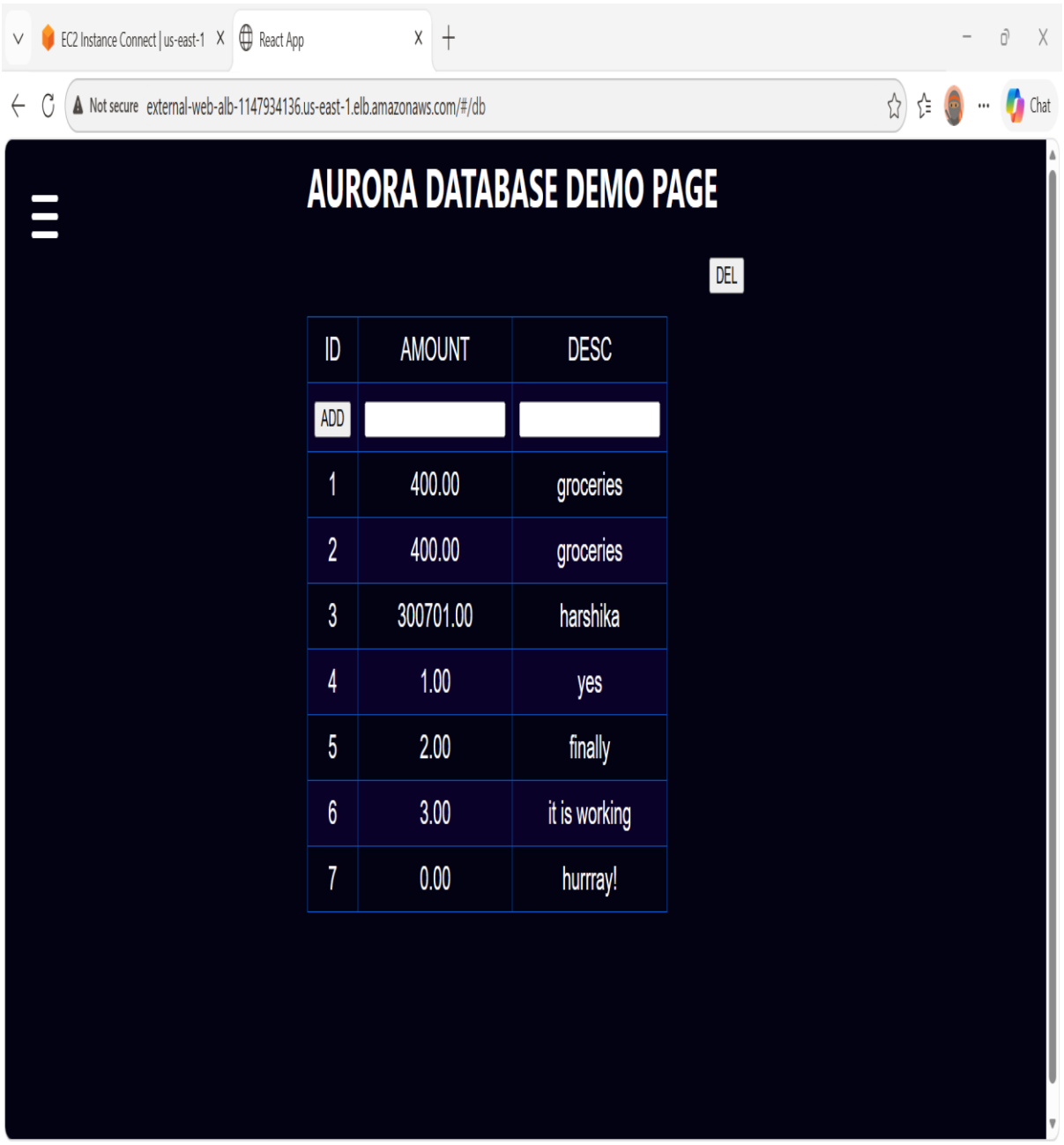
Successful!



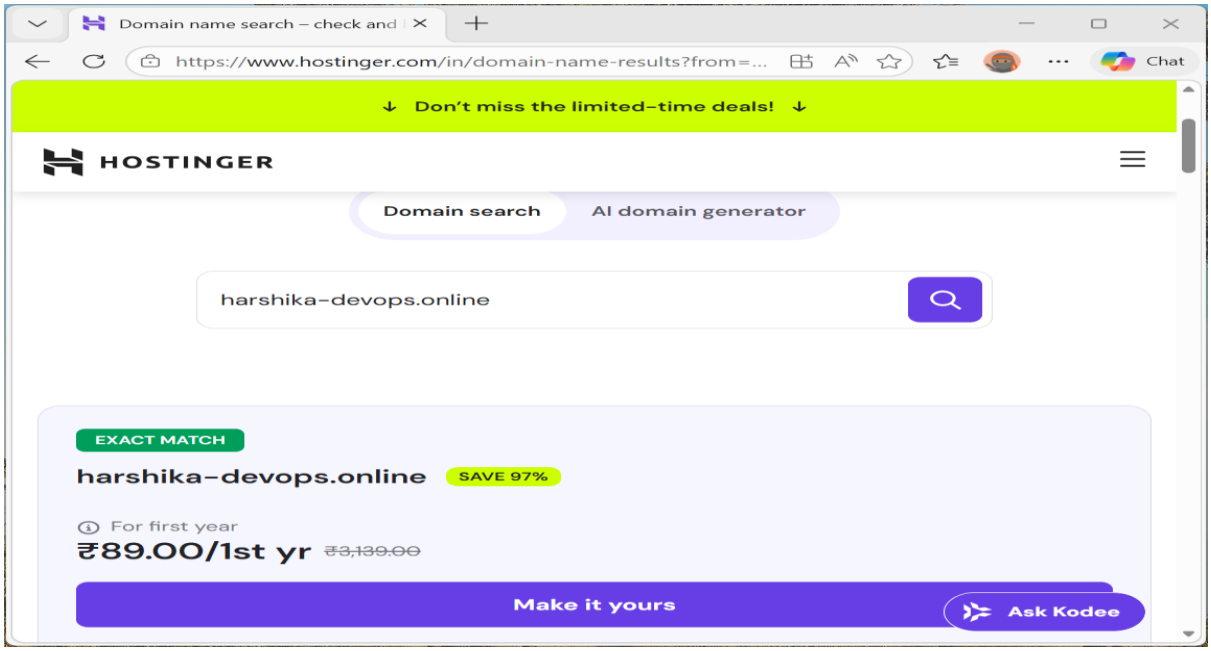
Saving the entry in the table.

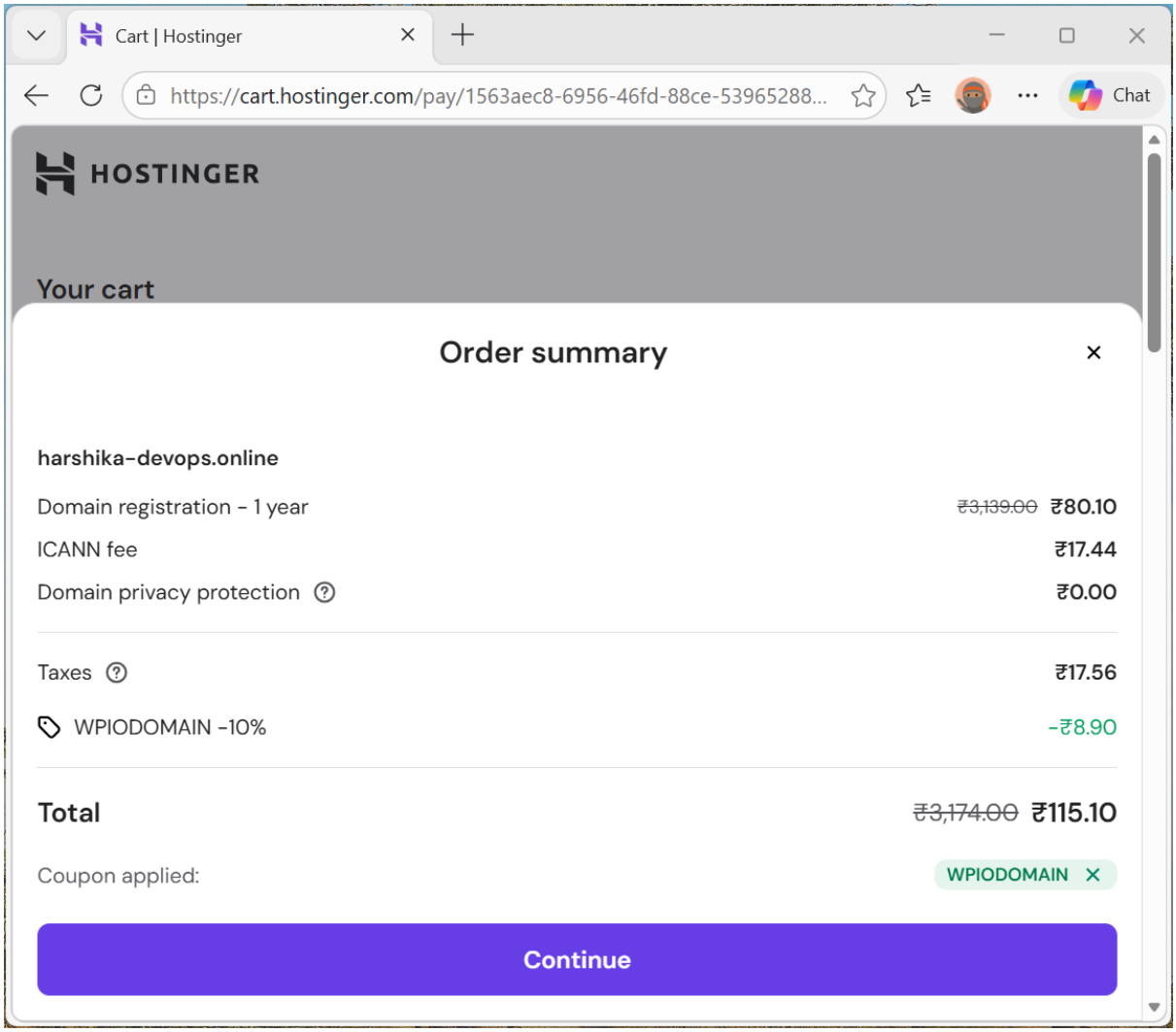


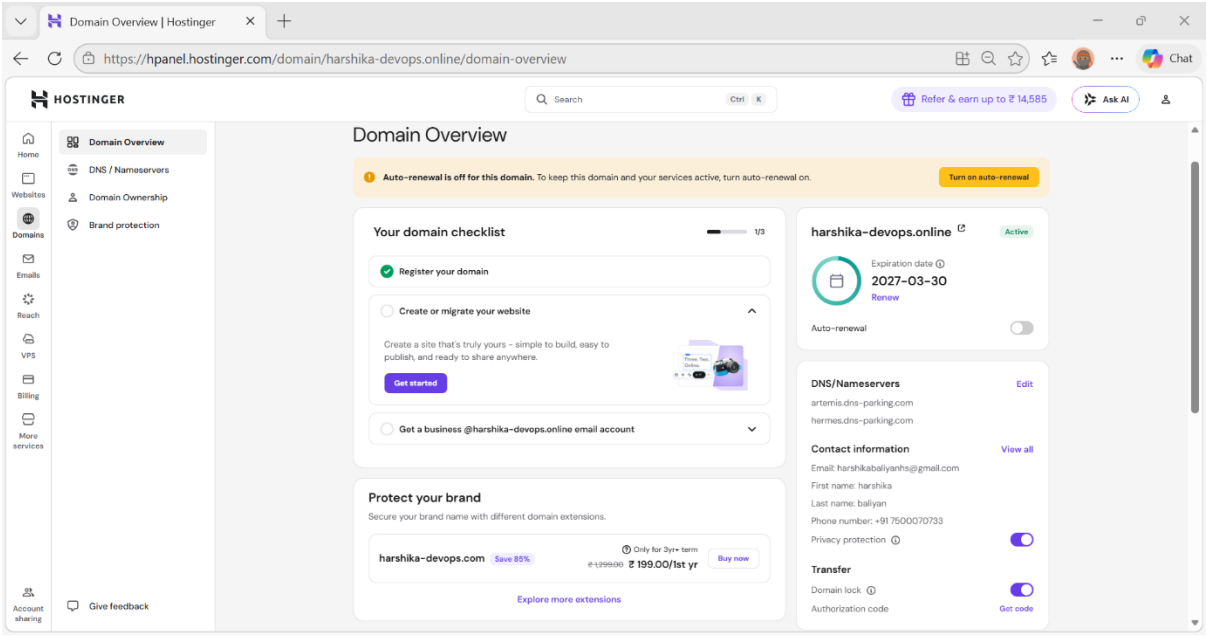
Saved entries!



Purchasing the Domain Name from Hostinger: Harshika-devops.online



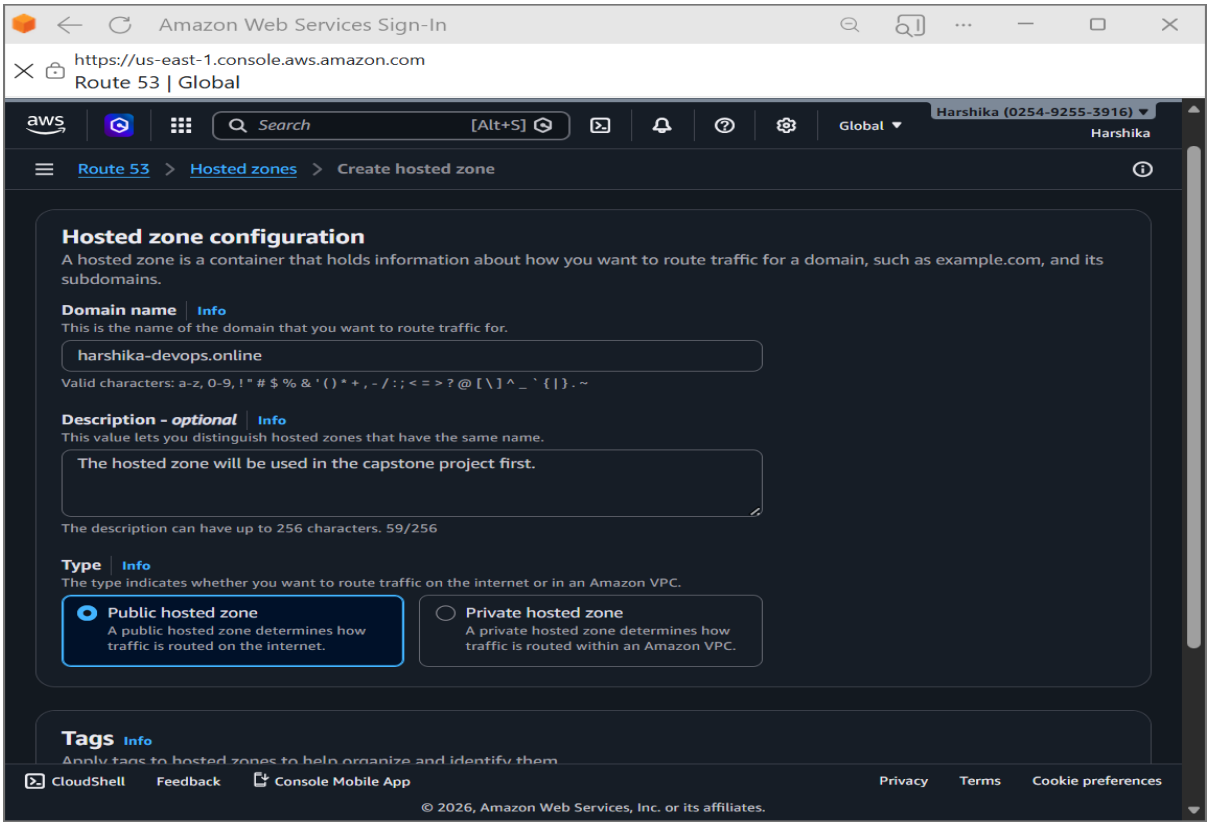




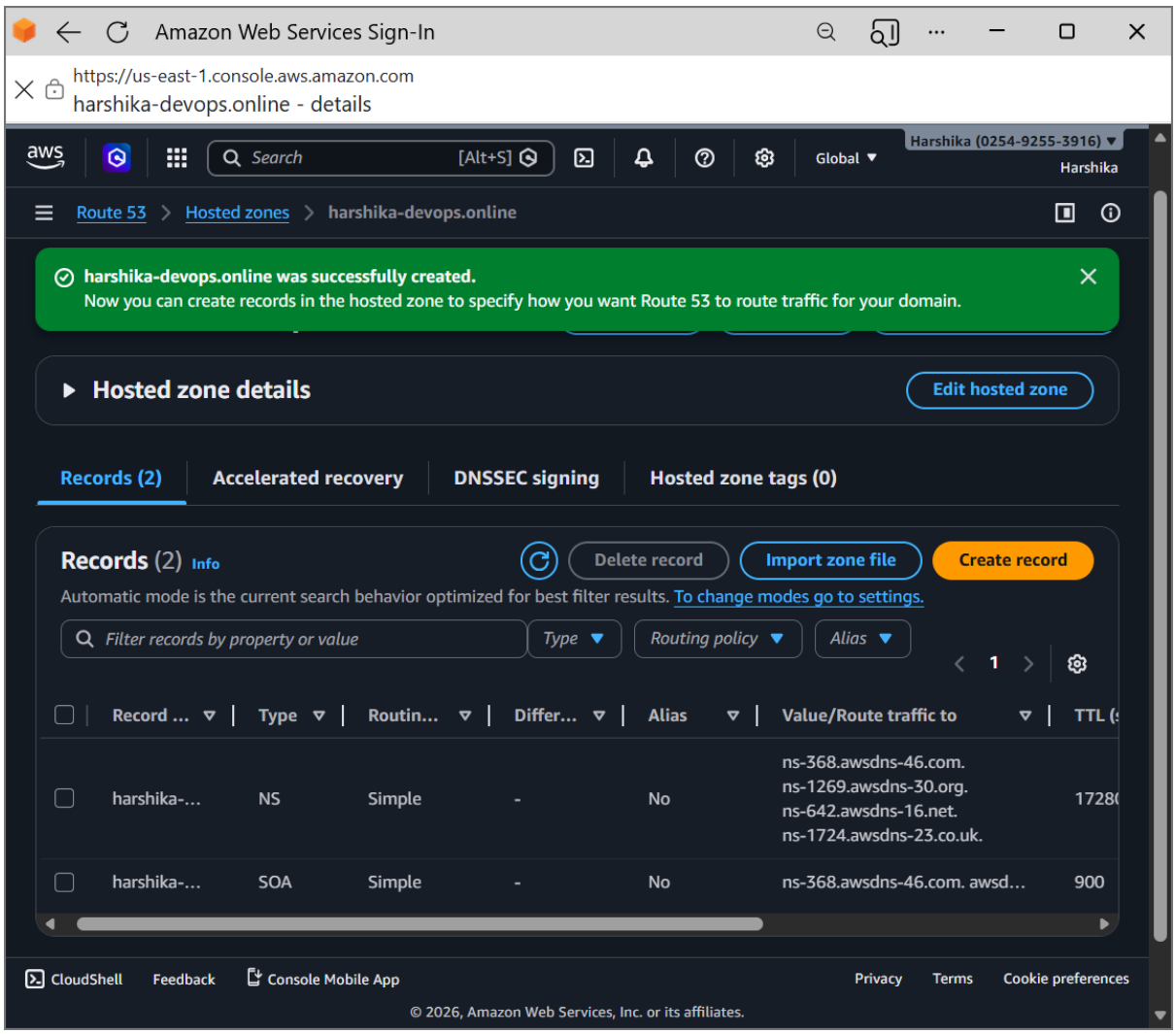
Step 1: Create a Hosted Zone

First, AWS needs a place to manage the records for the domain.

1. Go to **Route 53** in the AWS Console.
2. Click **Create hosted zone**.



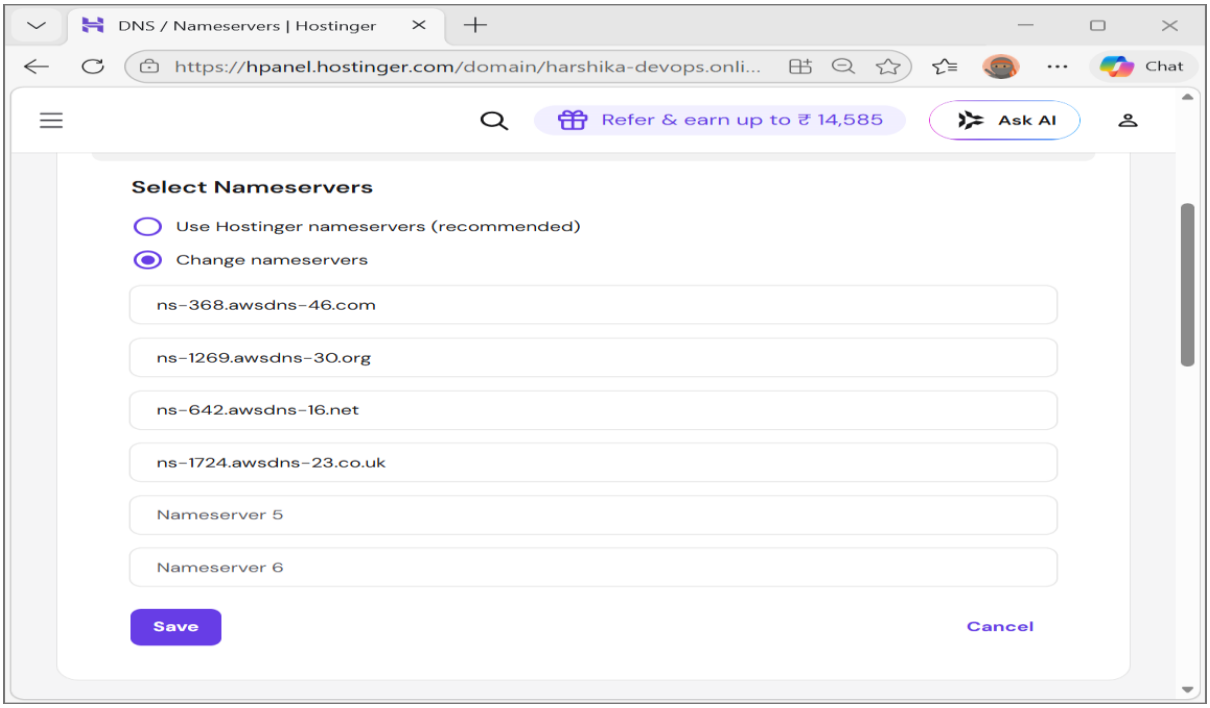
3. **Domain name:** Enter harshika-devops.online
4. **Type:** Public hosted zone.
5. Click **Create**.

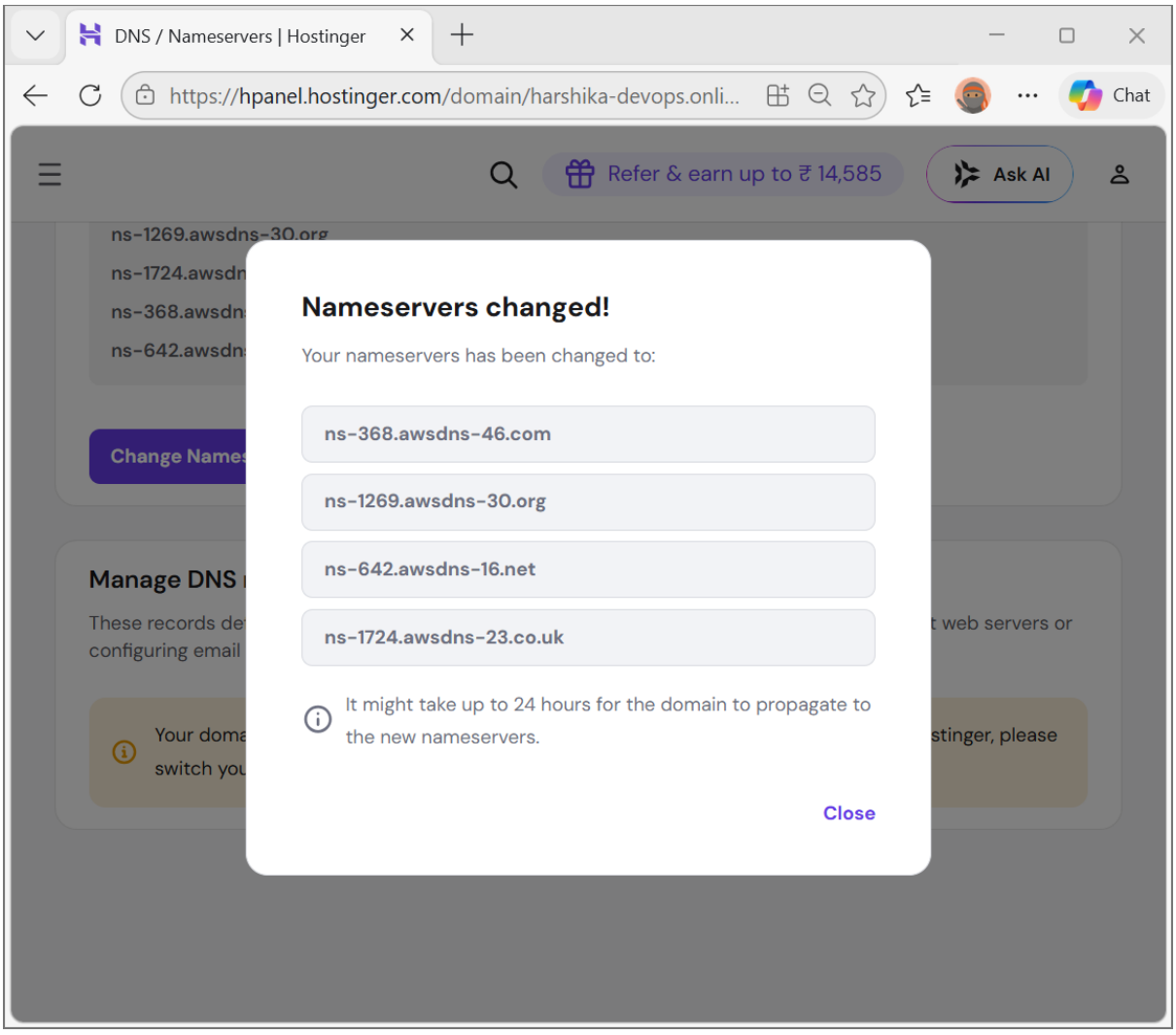


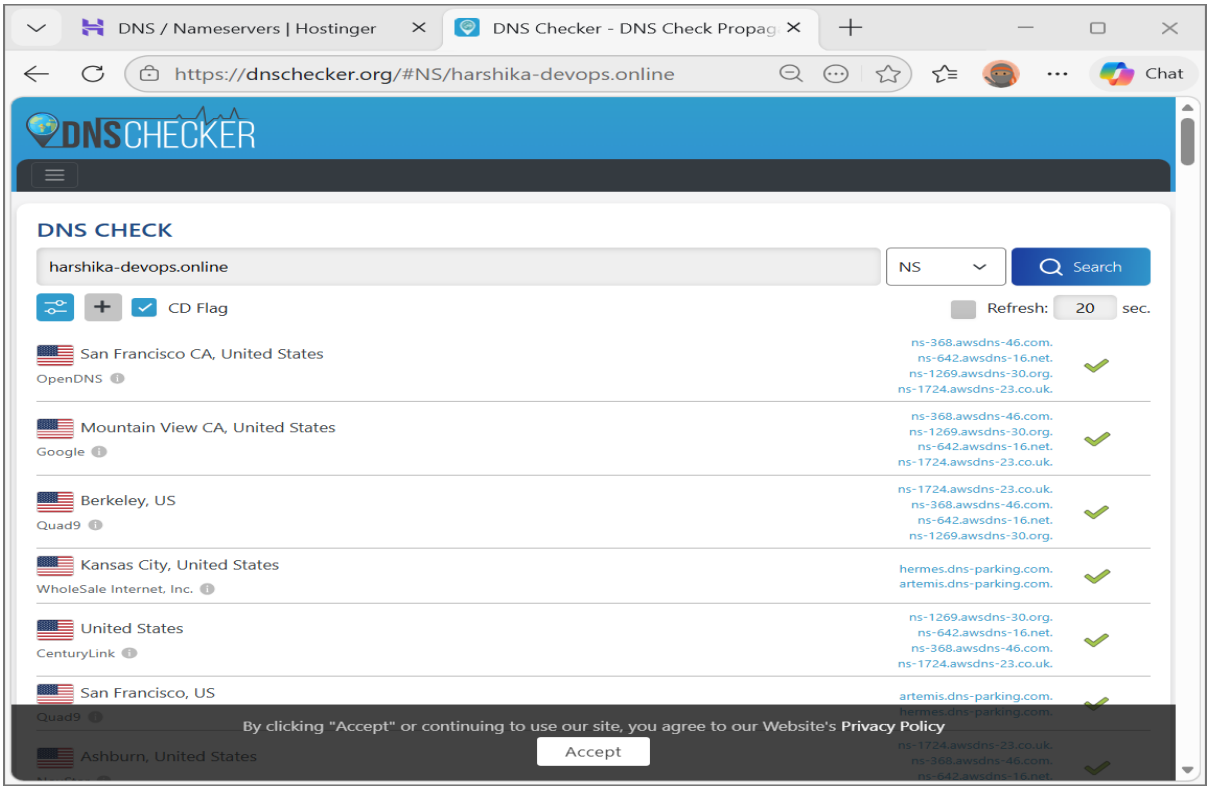
Step 2: Update Name Servers

Once the zone is created, we will see 4 **Name Servers (NS)** (e.g., ns-123.awsdns-01.com).

1. Copy these four addresses.
2. Go to the website where you bought the domain (like Namecheap, GoDaddy, or Hostinger).
3. Find the **DNS Management** or **Name Servers** section for your domain.


DNS / Nameservers | Hostinger
https://hpanel.hostinger.com/domain/harshika-devops.onli...
Chat
Refer & earn up to ₹ 14,585
Ask AI
Select Nameservers
Use Hostinger nameservers (recommended)
Change nameservers
ns-368.awsdns-46.com
ns-1269.awsdns-30.org
ns-642.awsdns-16.net
ns-1724.awsdns-23.co.uk
Nameserver 5
Nameserver 6
Save
Cancel


DNS / Nameservers | Hostinger
https://hpanel.hostinger.com/domain/harshika-devops.onli...
Chat
Refer & earn up to ₹ 14,585
Ask AI
ns-1269.awsdns-30.org
ns-1724.awsdn
ns-368.awsdn
ns-642.awsdn
Change Names
Manage DNS
These records de
configuring email
t web servers or
Your doma
switch you
stinger, please
Nameservers changed!
Your nameservers has been changed to:
ns-368.awsdns-46.com
ns-1269.awsdns-30.org
ns-642.awsdns-16.net
ns-1724.awsdns-23.co.uk
It might take up to 24 hours for the domain to propagate to the new nameservers.
Close



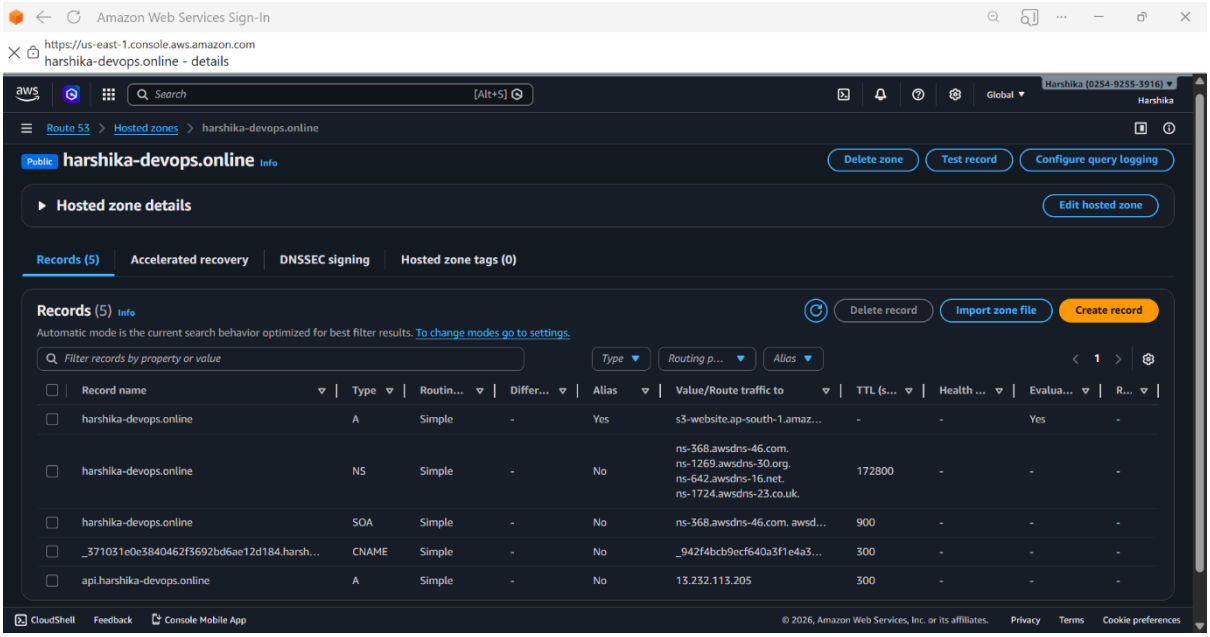
4. Change the "Default" name servers to the **Custom** AWS ones you just copied.

Note: It can take 5 minutes to an hour for this change to "propagate" across the internet.

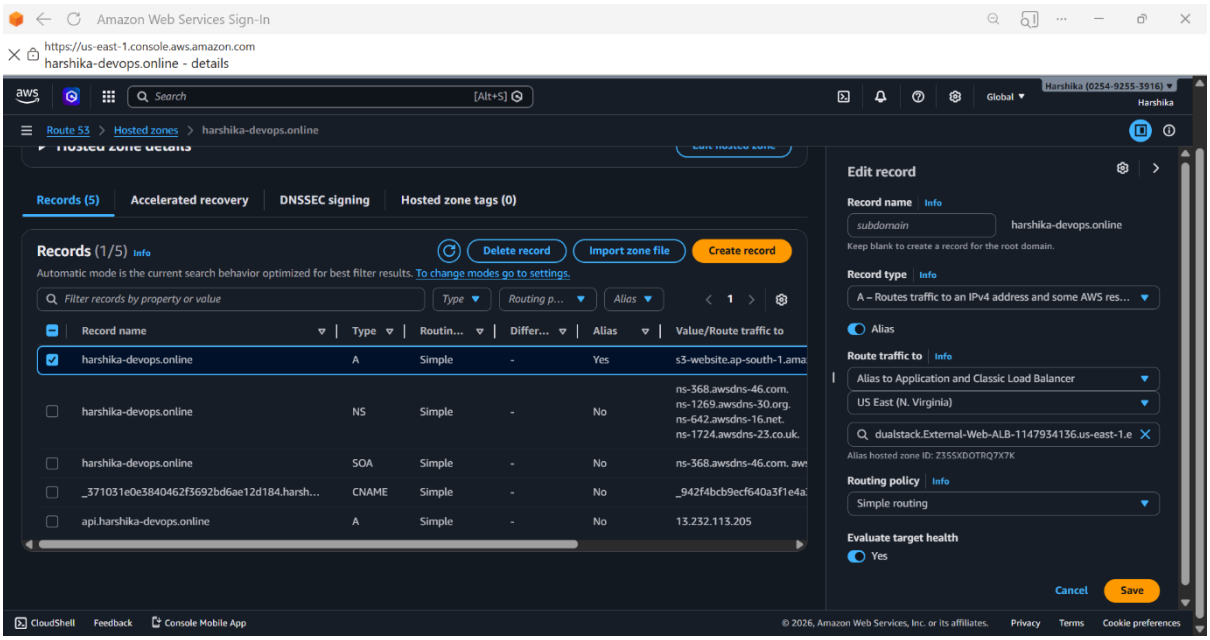
Step 3: Point the Domain to your Web Load Balancer

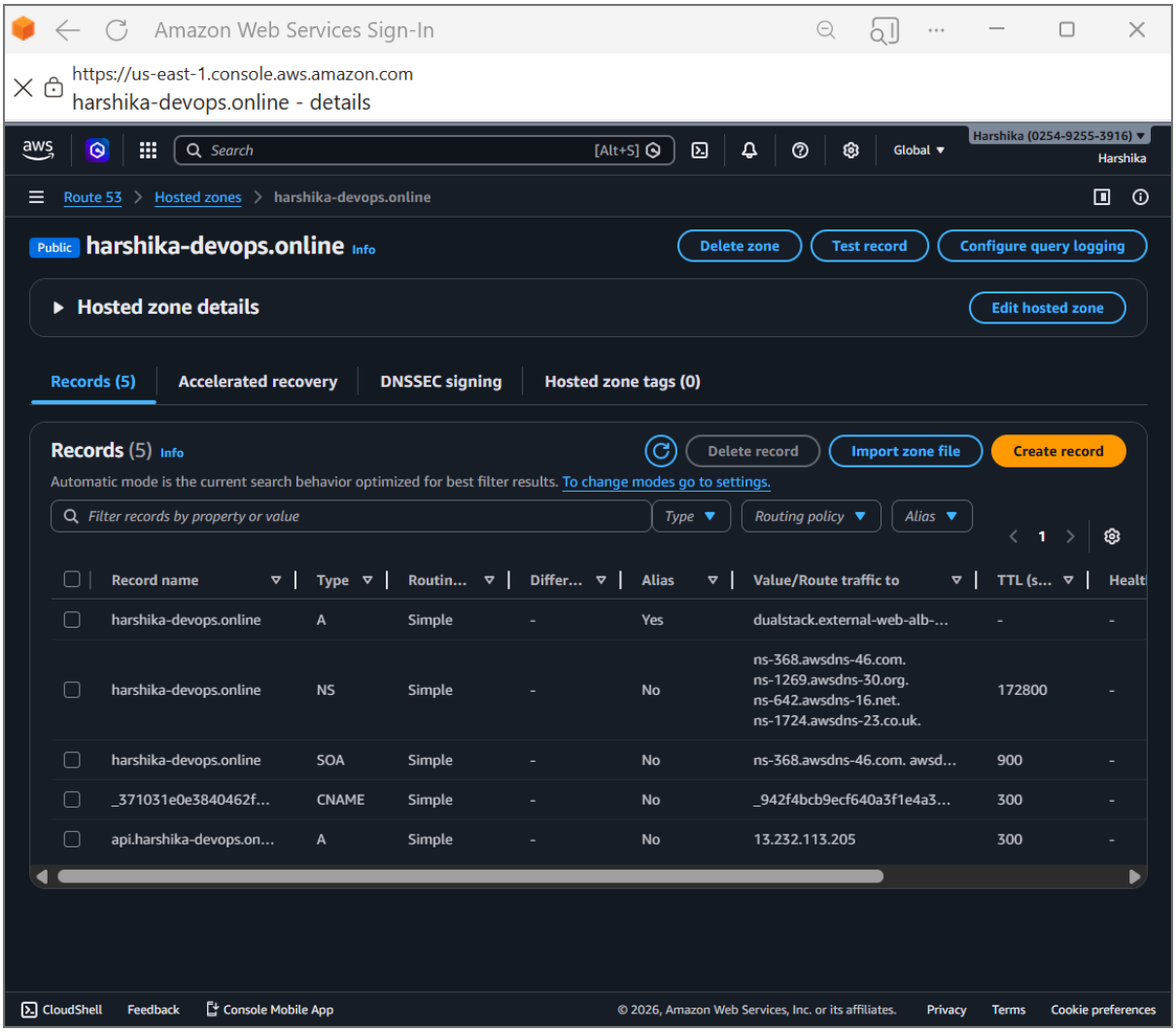
Now we tell Route 53 that harshika-devops.online should lead to the **External-Web-ALB**.

1. Inside your Route 53 Hosted Zone, click **Create record**.
2. **Record name:** Leave it blank (for the root domain) or type www.
3. **Record type:** A - Routes traffic to an IPv4 address and some AWS resources.
4. Toggle the **Alias** switch to **ON**.



5. **Route traffic to:**
    - Choose **Alias to Application and Classic Load Balancer**.
    - Choose your **Region**
    - Select your **external-web-alb** DNS name from the list.
6. Click **Create records**.

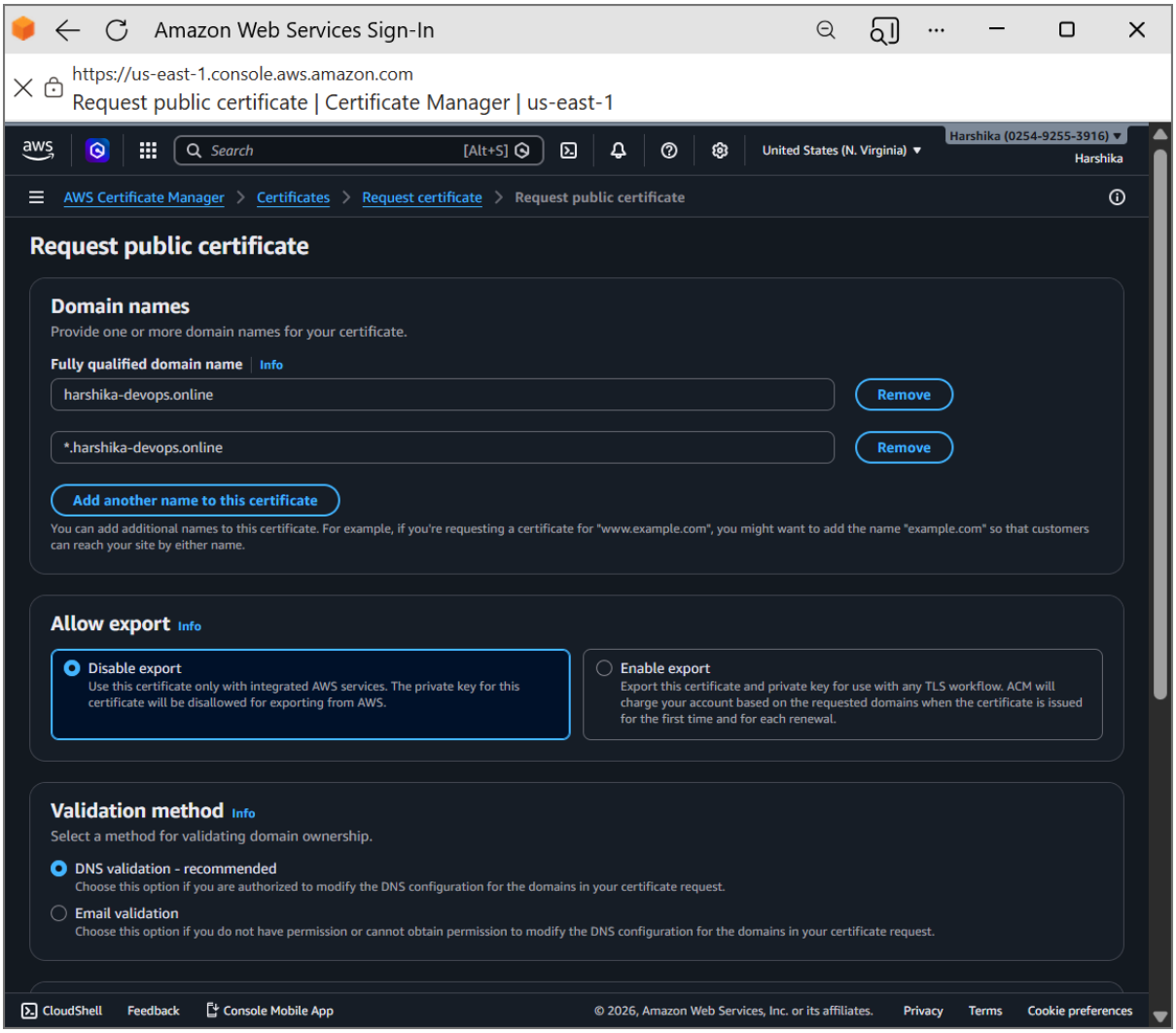




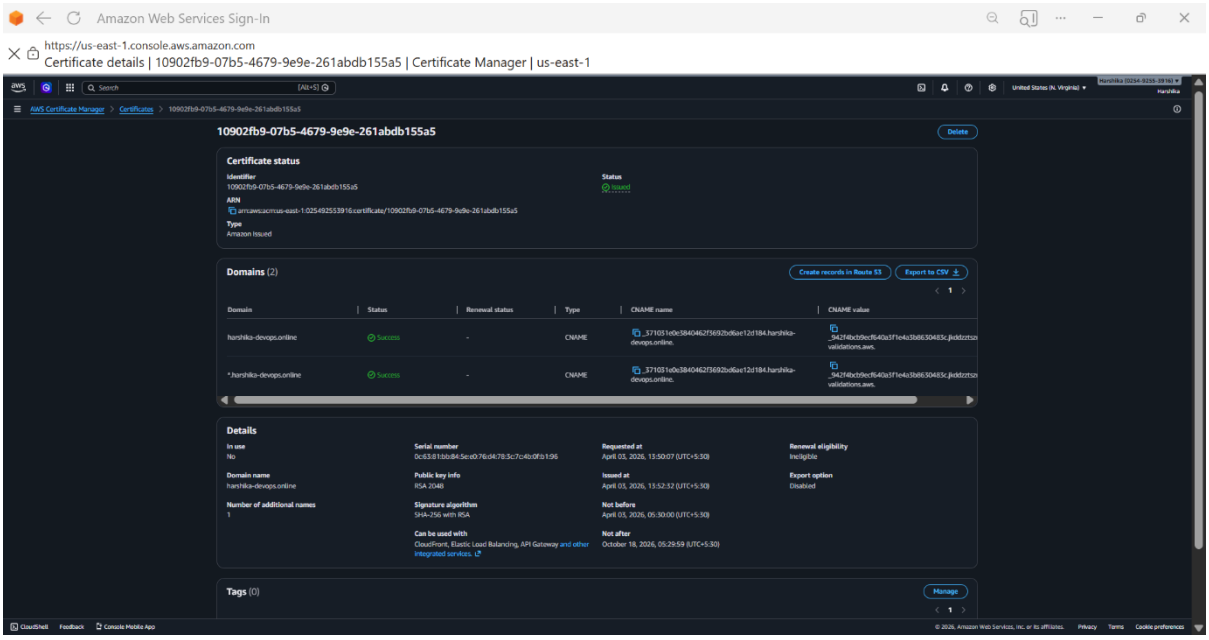
Step 4: Secure your Domain with SSL (ACM)

Providing an https:// connection is a key requirement for modern DevOps projects.

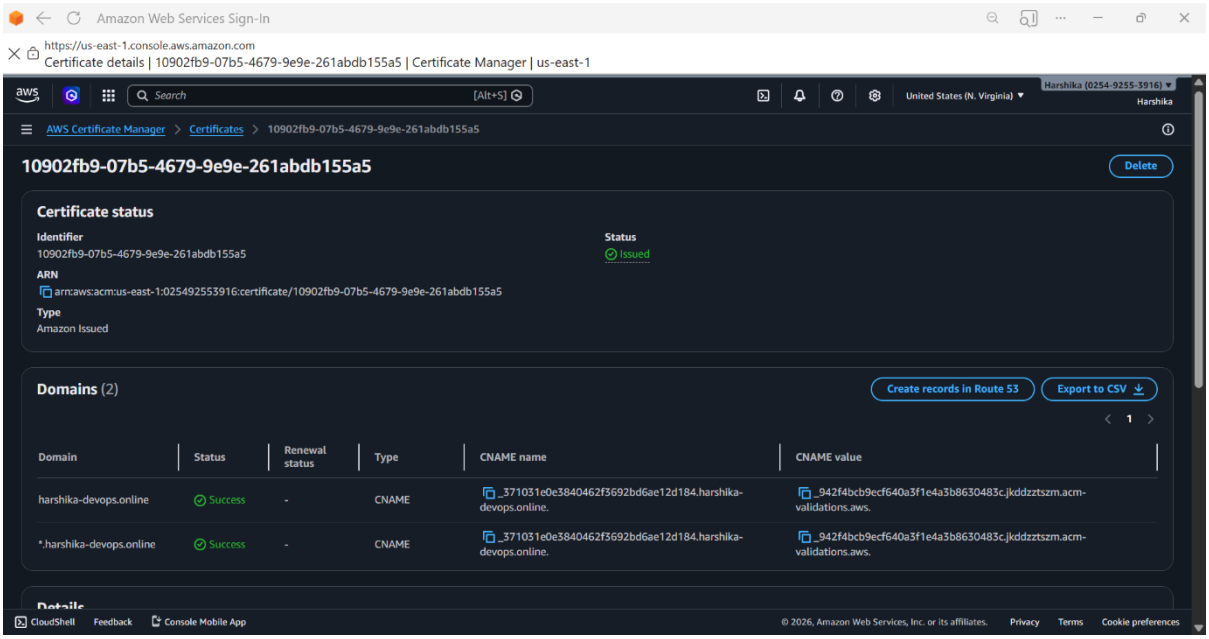
1. Go to **AWS Certificate Manager (ACM)**.
2. Click **Request a certificate** -> **Request a public certificate**.
3. **Fully qualified domain name:** Enter harshika-devops.online and add another for *.harshika-devops.online.



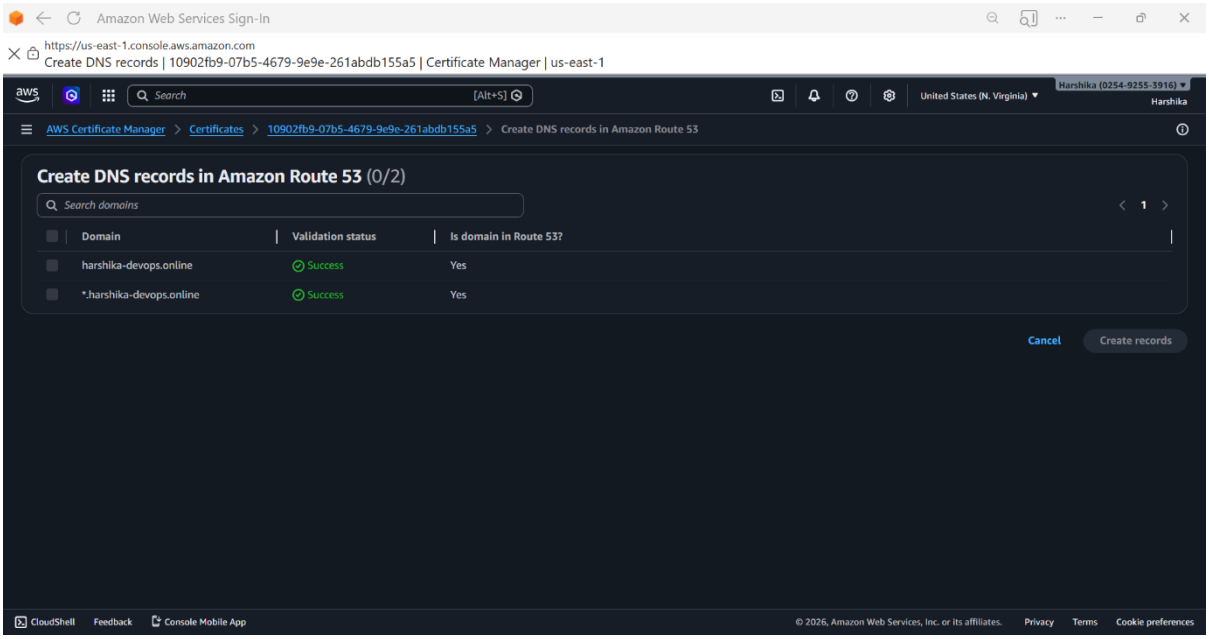
4. Select **DNS validation** and click **Request**.

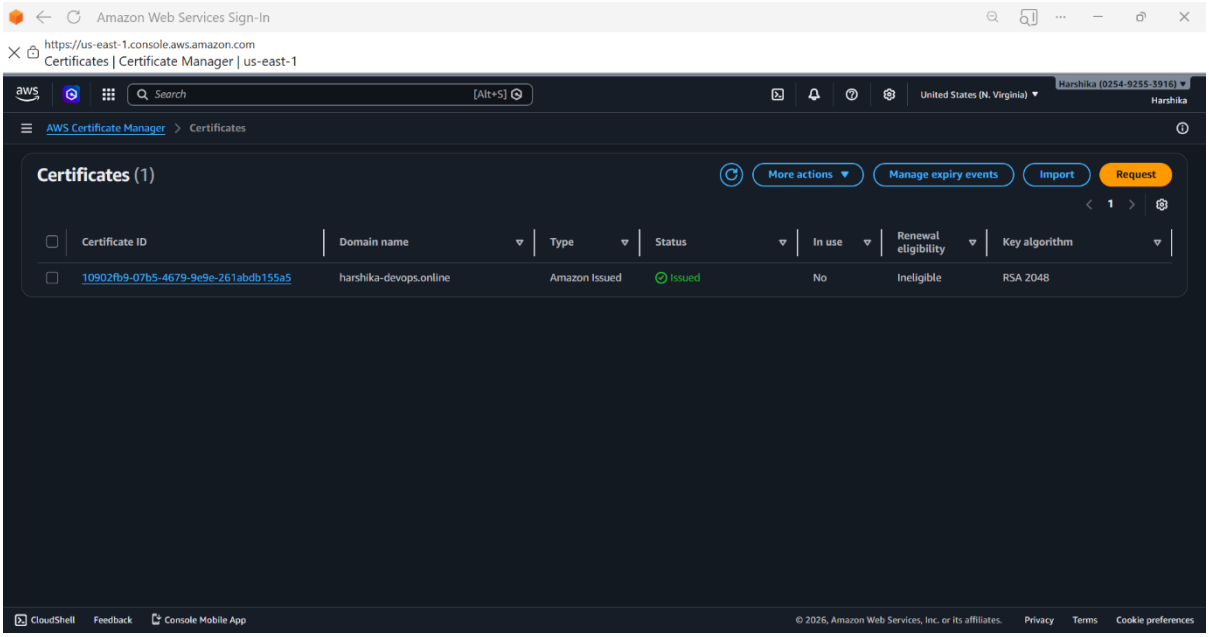


5. On the next screen, click the **Certificate ID**. Under the "Domains" section, click **Create records in Route 53**. This automatically adds the CNAME records needed to prove you own the domain.



6. Wait a few minutes until the Status changes from "Pending validation" to **Issued**.

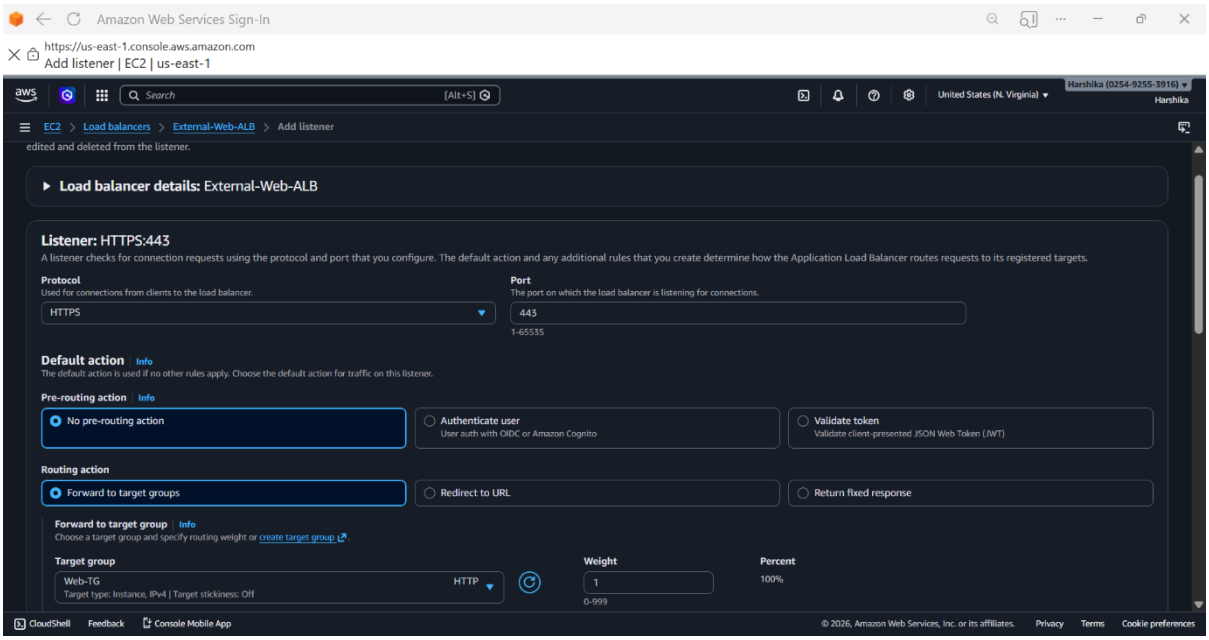




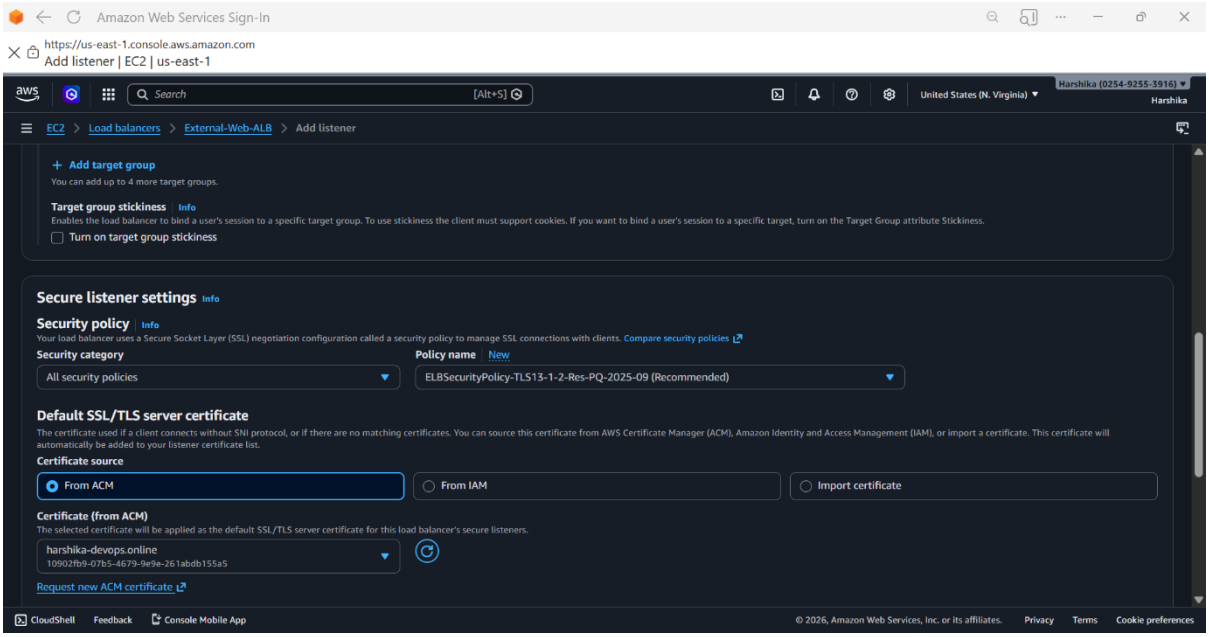
Step 5: Update the Load Balancer Listener

Now, we must tell your Web Load Balancer to use that certificate.

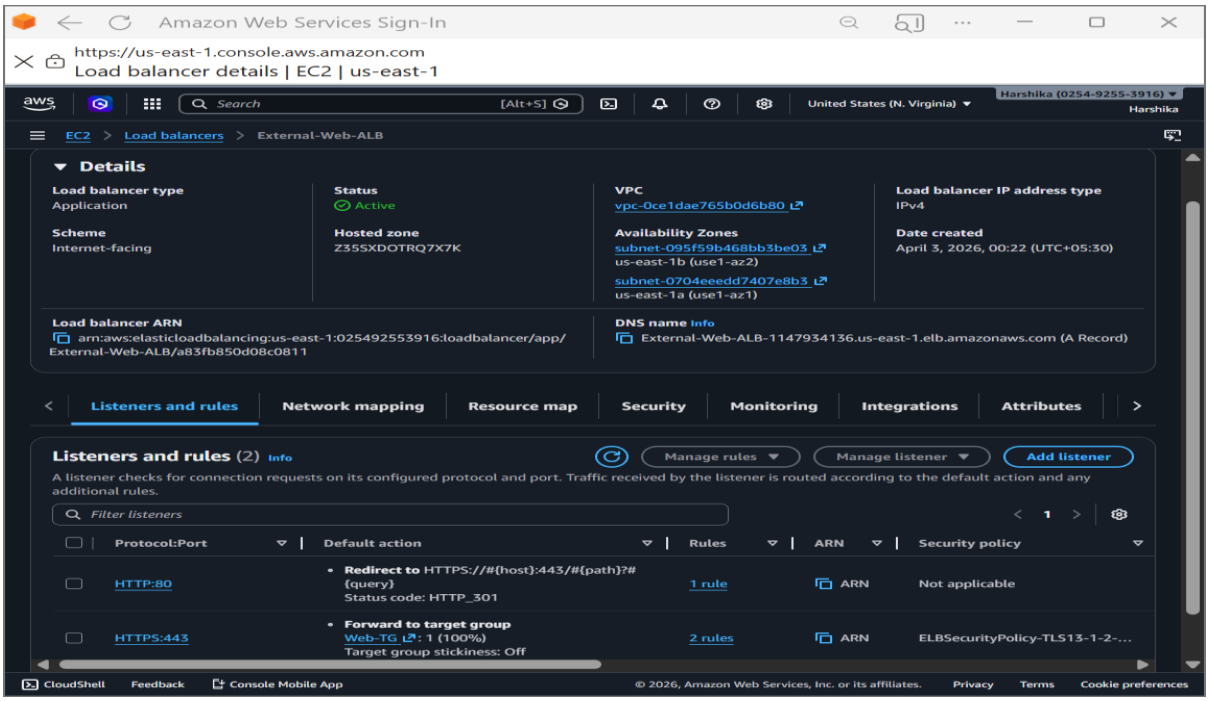
1. Go to **EC2 Console** -> **Load Balancers** -> Select your **External-Web-ALB**.



2. Go to the **Listeners** tab.



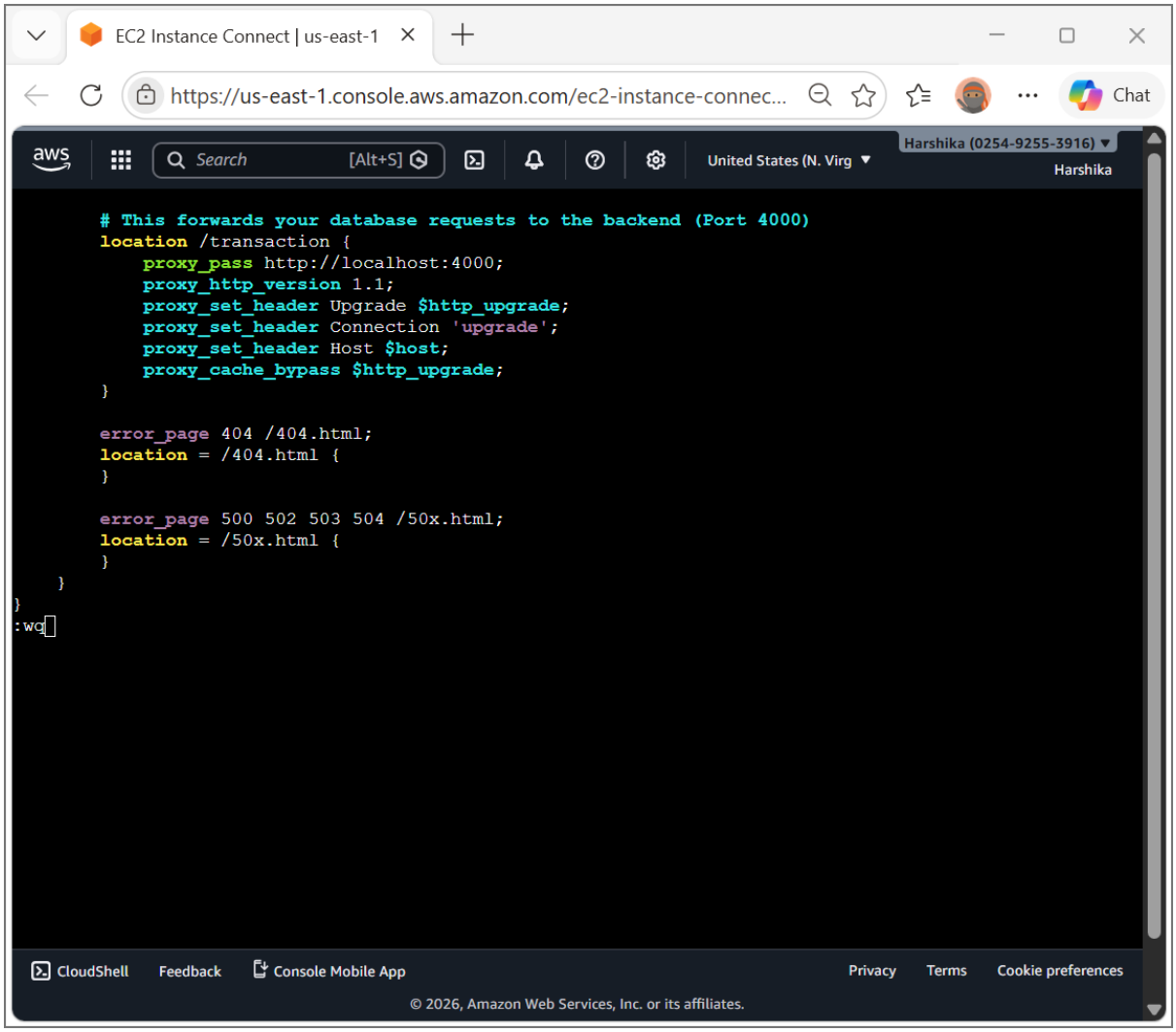
3. Click **Add listener**.
    - **Protocol:** HTTPS
    - **Port:** 443
    - **Action:** Forward to your **Web-Tier-Target-Group**.
    - **Secure listener settings:** Select the certificate you just created in ACM.
4. Edit the HTTP:80 listener to **redirect** to HTTPS:443.

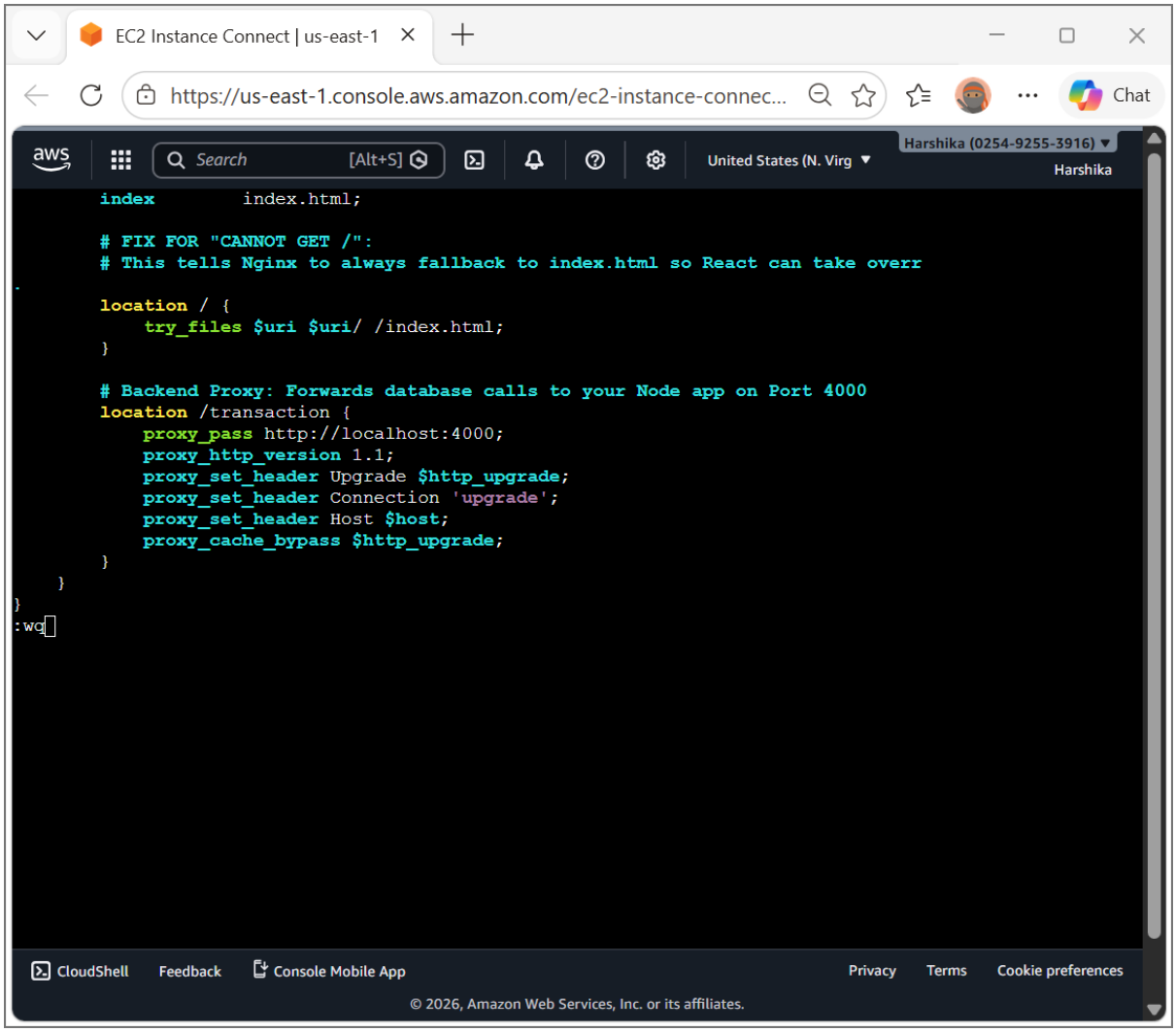


Step 5: Final Frontend Code Update

Since you are now using a domain, your React frontend needs to point to the Load Balancer via the domain name, not the long AWS DNS string.

1. Open your frontend file: vi /home/ec2-user/AWS3TierApp/application-code/web-tier/src/components/DatabaseDemo/DatabaseDemo.js
2. Update the URL to use your domain:
    - **Change from:** http://app-tier-external-alb...:4000/transaction
    - **Change to:** http://harshika-devops.online:4000/transaction

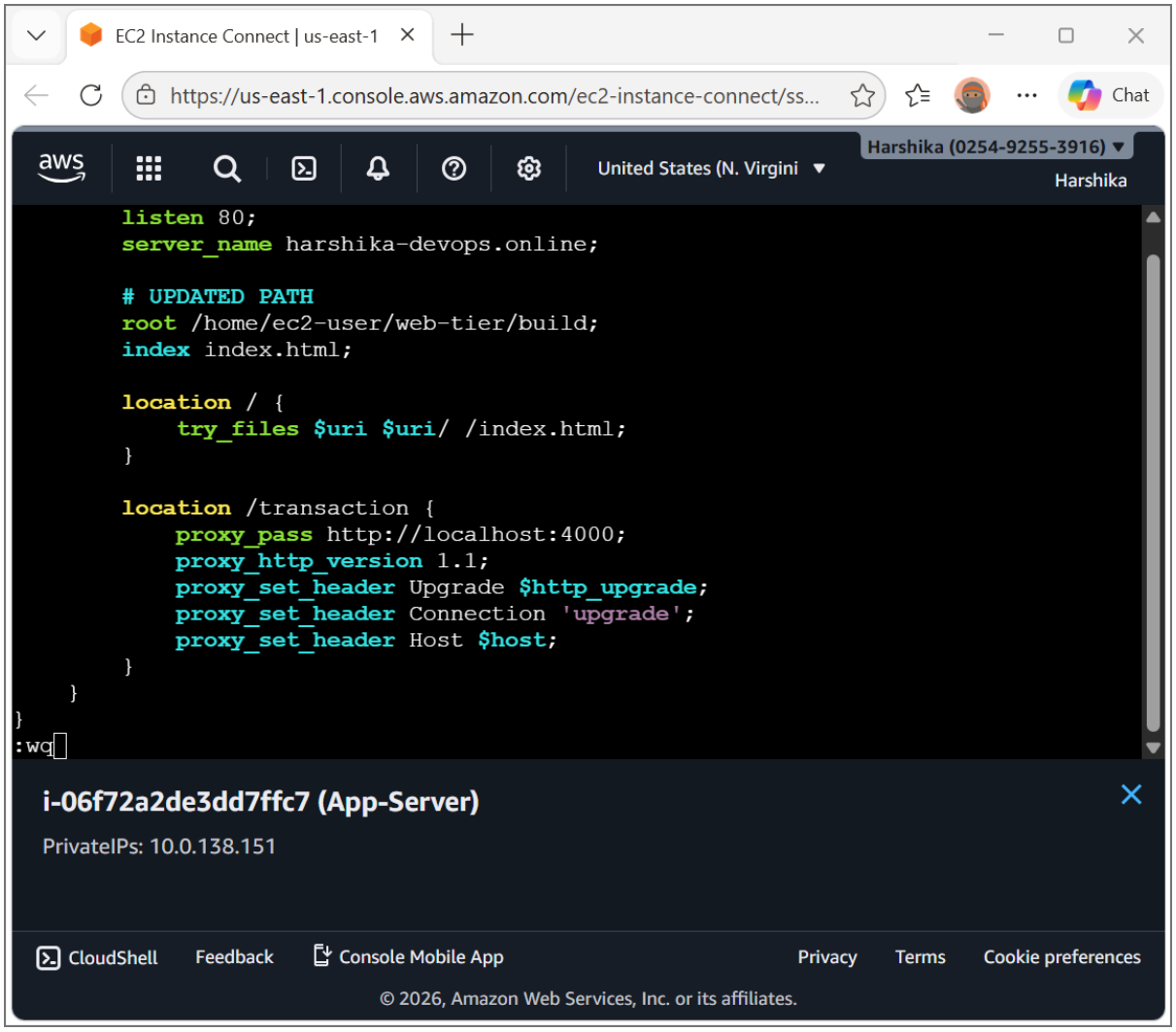




Step 6: The "Ultimate" Nginx Config

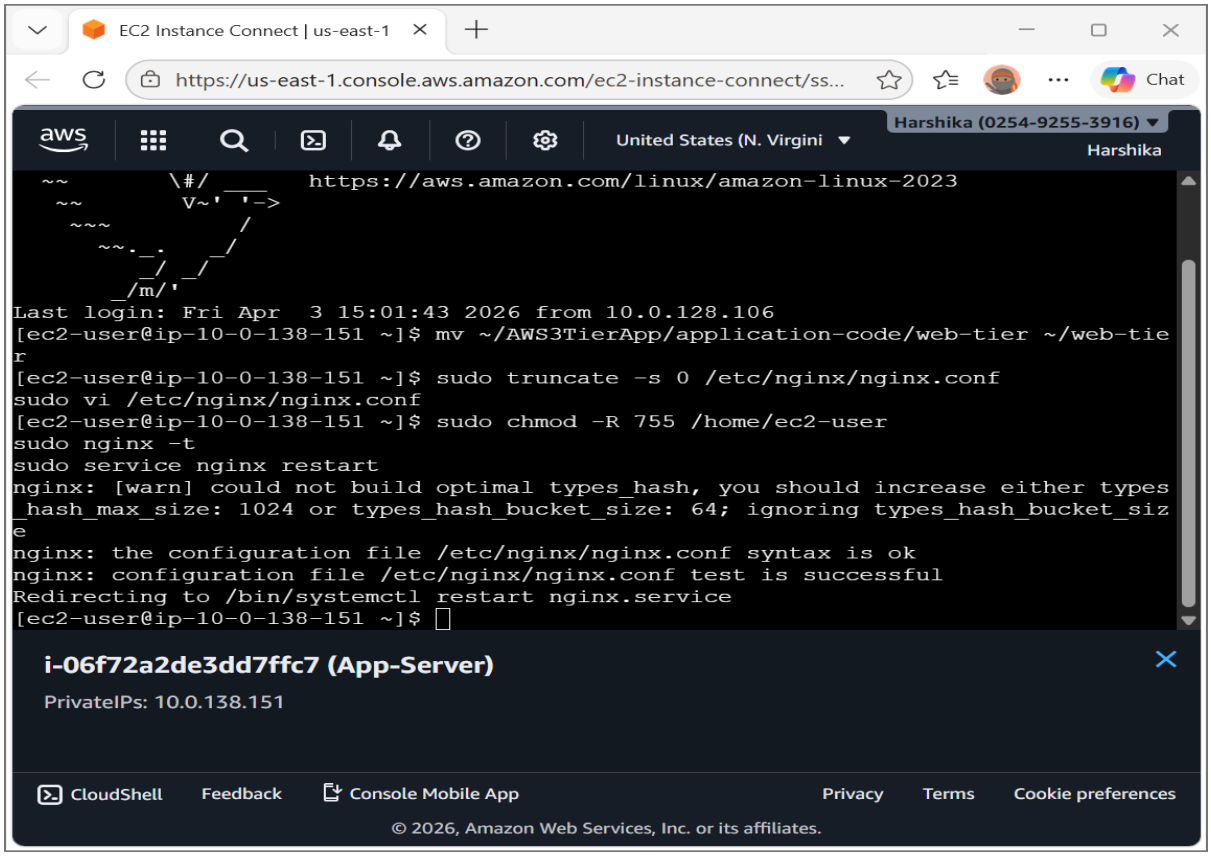
```
Nginx

user nginx;

worker_processes auto;

events { worker_connections 1024; }


http {

  include /etc/nginx/mime.types;

  server {
```

```
    listen 80;
    server_name harshika-devops.online;

    root /home/ec2-user/web-tier/build;
    index index.html;

    # Fix for "Cannot GET"
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Fix for "502 Bad Gateway"
    location /transaction {
        proxy_pass http://127.0.0.1:4000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
  }
}
```



Bash

```
# Allow Nginx to connect to the backend port
sudo setsebool -P httpd_can_network_connect 1

# Allow Nginx to read files in the home directory
sudo setsebool -P httpd_enable_homedirs 1
sudo chcon -Rt httpd_sys_content_t /home/ec2-user/web-tier/build

# Restart to apply
sudo service nginx restart
```



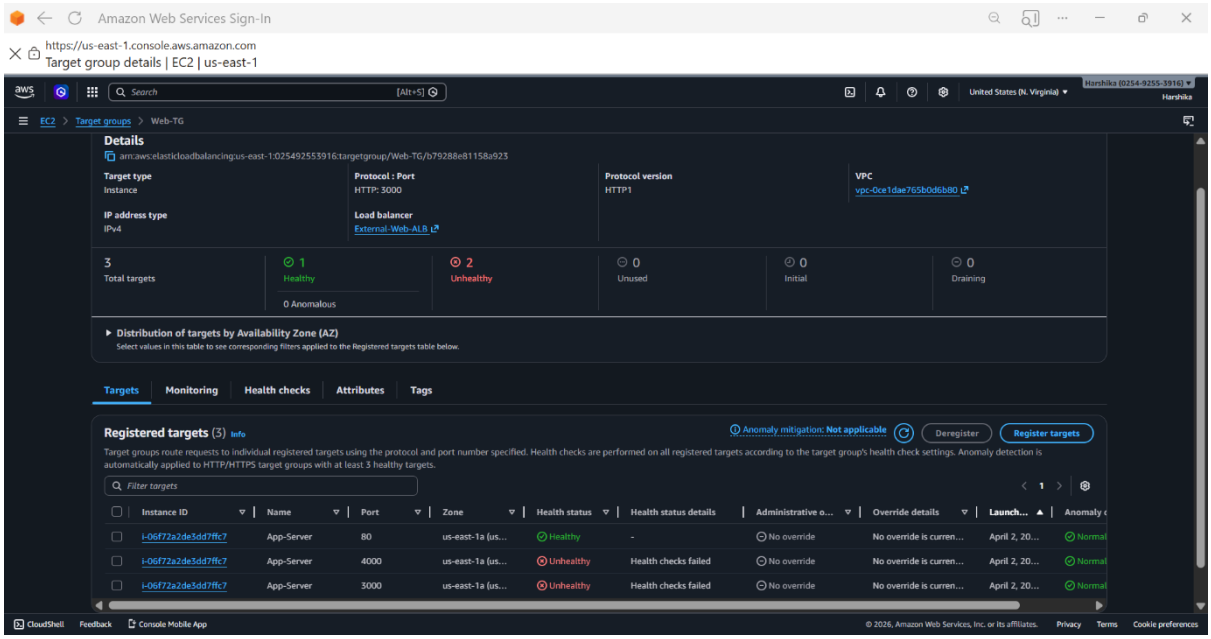
Final Check

1. Close all your browser tabs.
2. Open a **New Incognito Window/ new tab**.
3. Go to https://harshika-devops.online

Troubleshooting:

The problem is that the **Target Group (Web-TG)** is looking for the app on **Port 3000**, but the Nginx is listening on **Port 80**,the Node.js backend is on **Port 4000**.

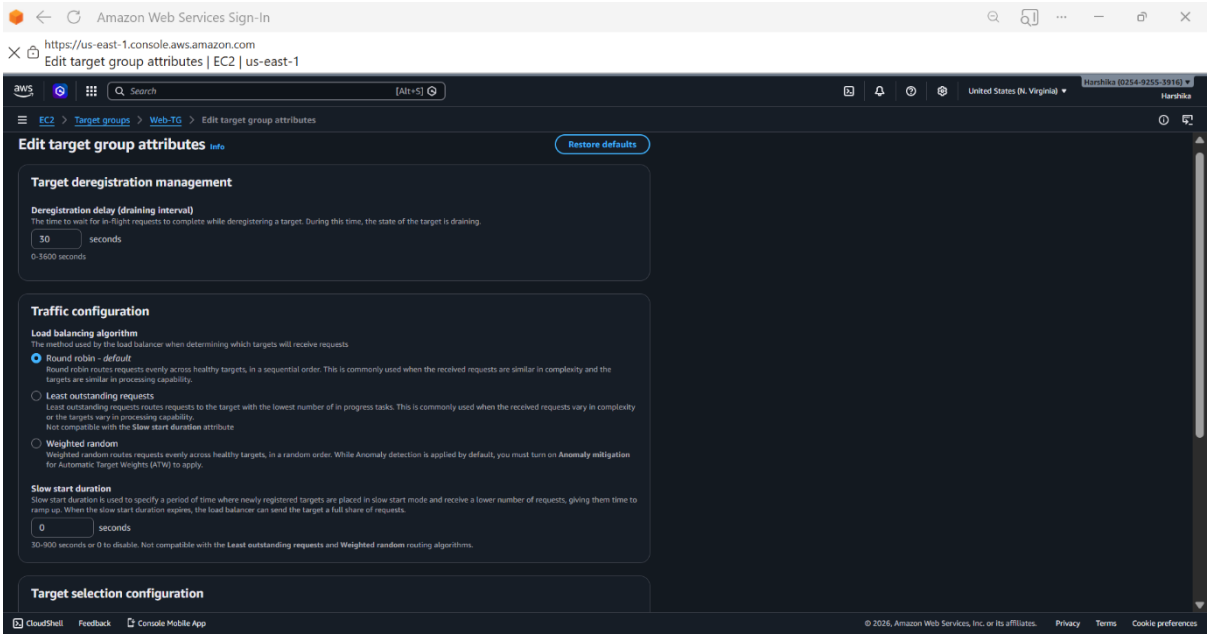
fix the "Unhealthy" status:



1. Fix the Target Group Port

The Load Balancer is trying to talk to Port 3000, but nothing is answering there, which is why it shows **"Unhealthy"**.

- Go to **Target Groups** in the EC2 Console.

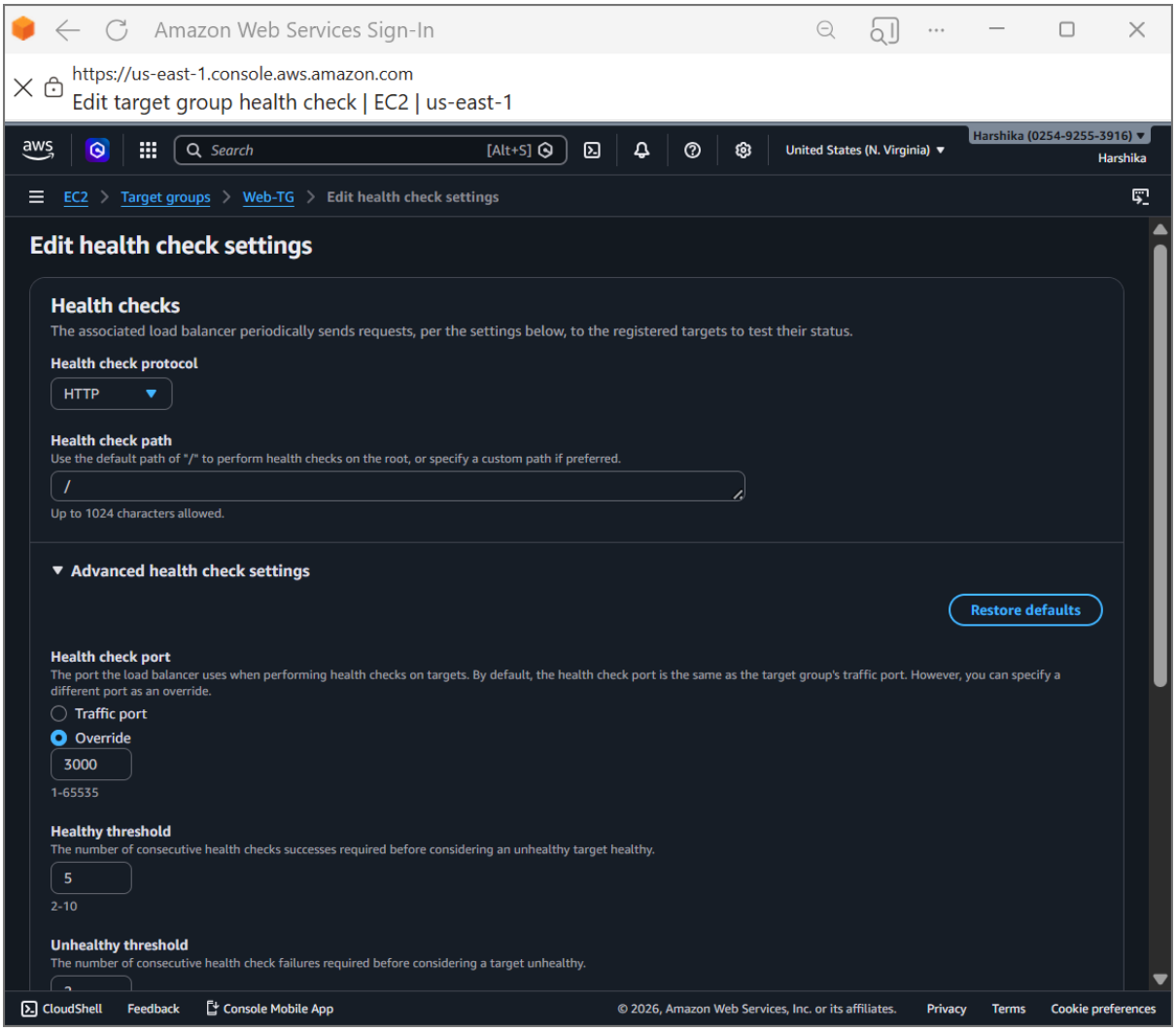


- Select **Web-TG**.
- Click the **Attributes** tab and then **Edit**.
- Change the **Port** from 3000 to **80**.
- **Save changes.**

2. Update the Health Check

While still in **Web-TG**:

- Click the **Health checks** tab.
- Click **Edit**.



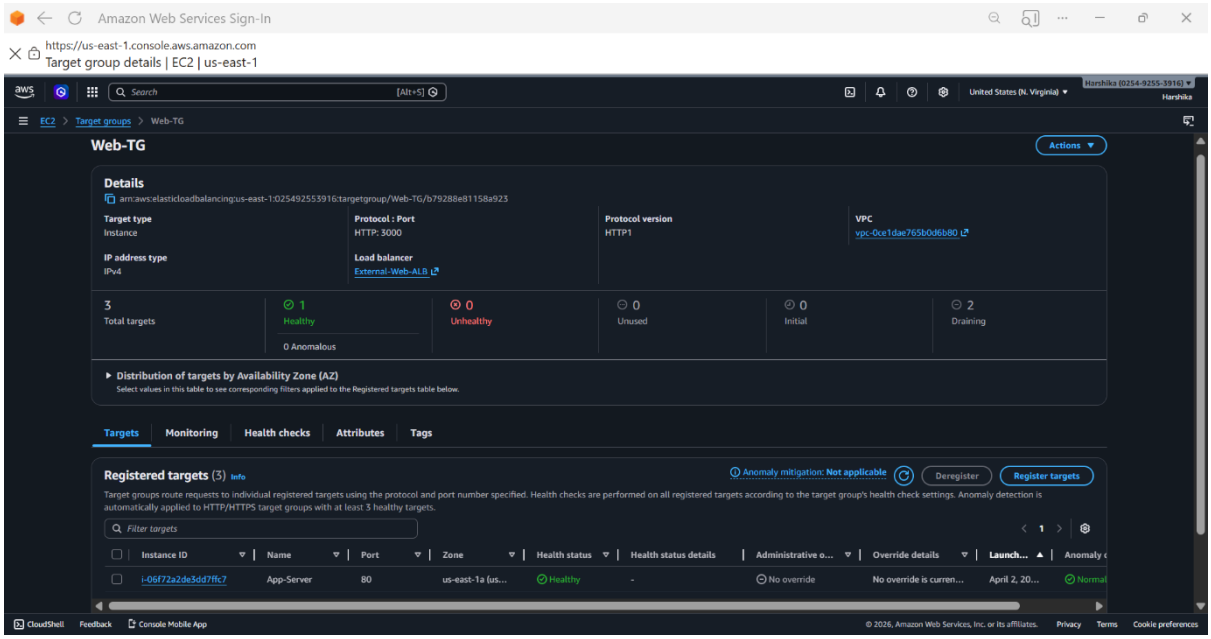
- Ensure the **Health check port** is set to the **Traffic port**.
- **Save changes.**

- *Wait 2–3 minutes.* The status should change from "Unhealthy" to "**Healthy**".

3. Verify the Listener

The Load Balancer is currently configured to forward **HTTPS:443** traffic to **Web-TG**.

- Because we changed the Target Group to Port 80 in Step 1, the Load Balancer will now correctly take the encrypted traffic and pass it to Nginx on Port 80.

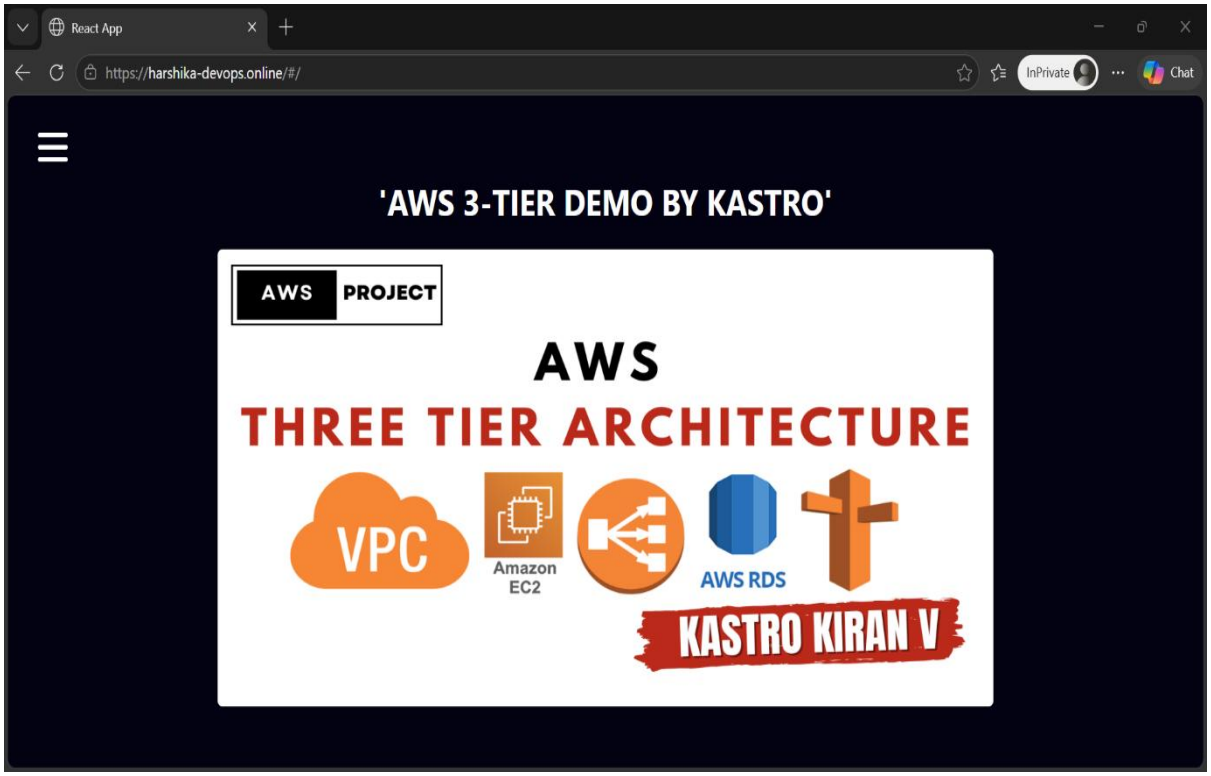


The Final Test

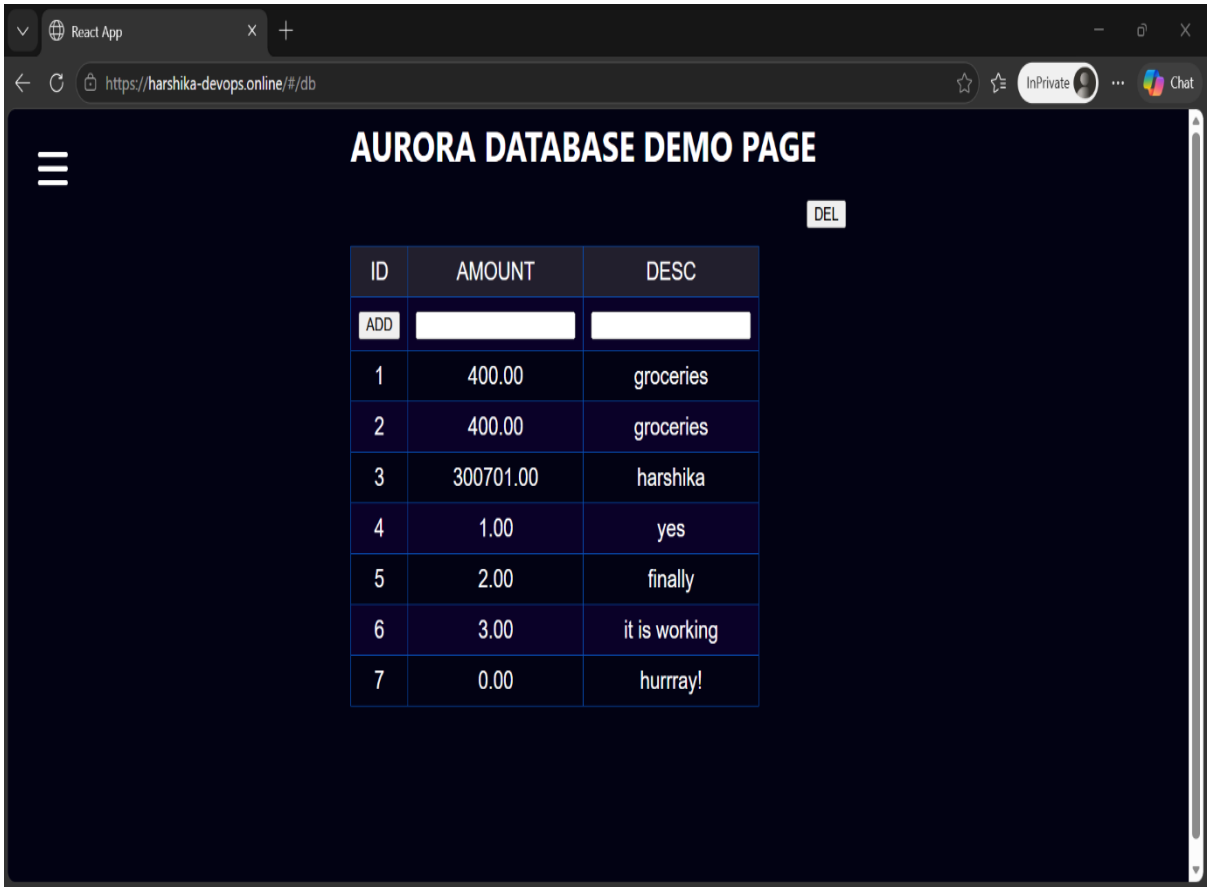
Once the Target Group shows **Healthy** in green:

1. Open your browser.
2. Go to https://harshika-devops.online.
3. Your Route 53 is already correctly set up as an **Alias** to the Load Balancer, so it will now flow through!

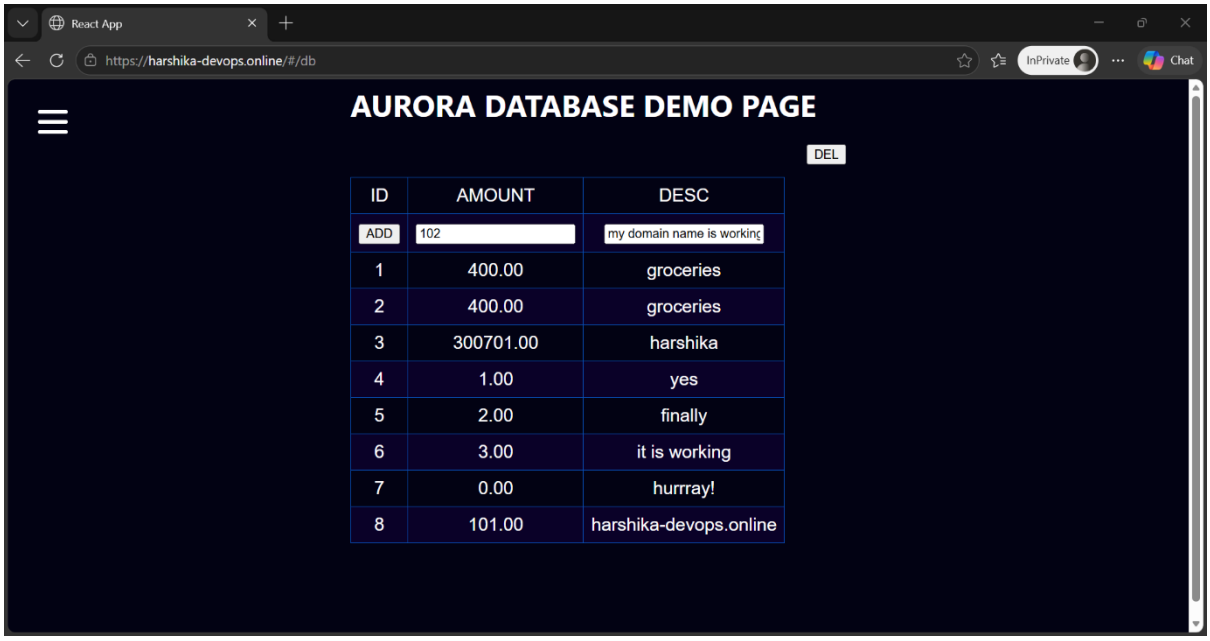
Successful!



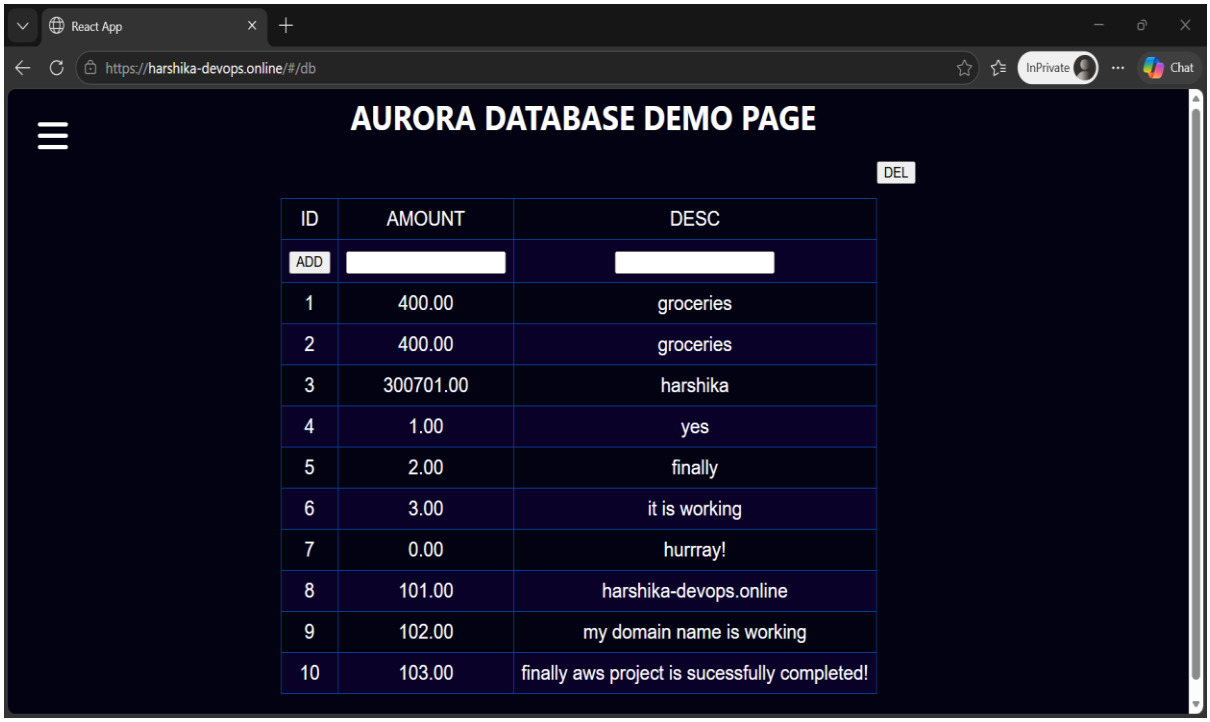
Previously saved entries.



Enter the new entries.

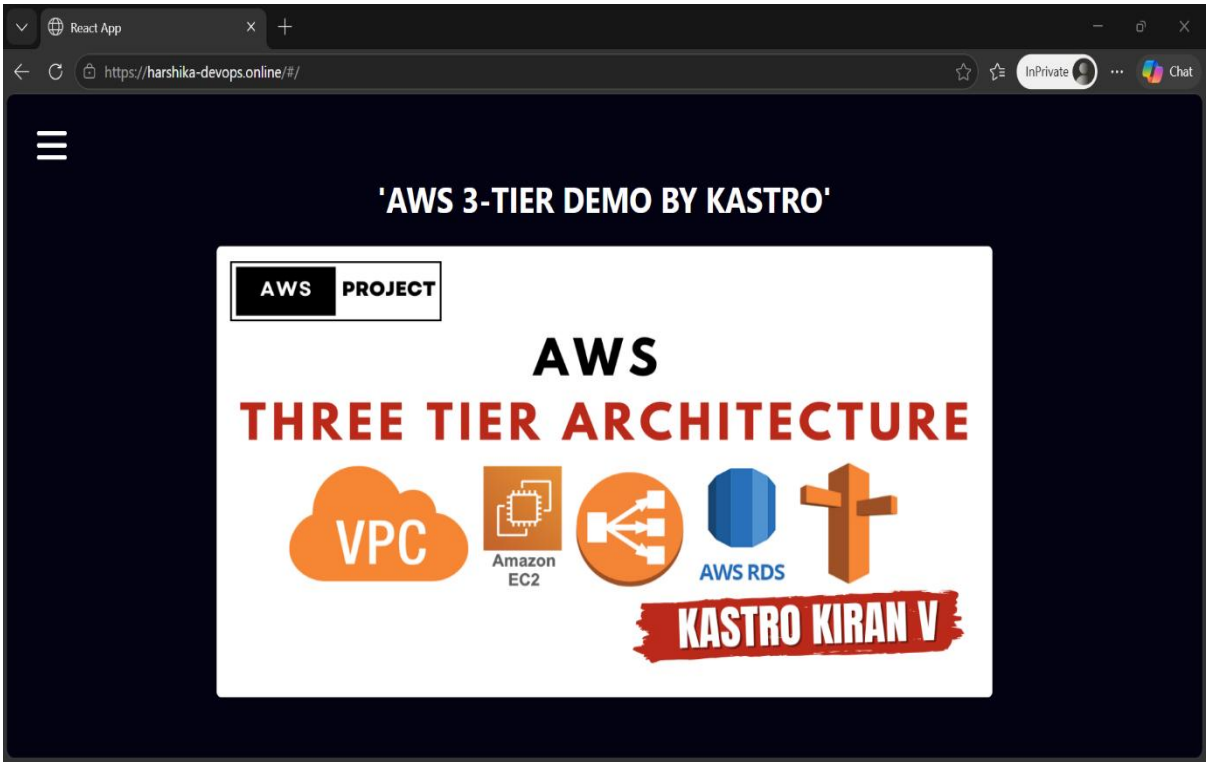


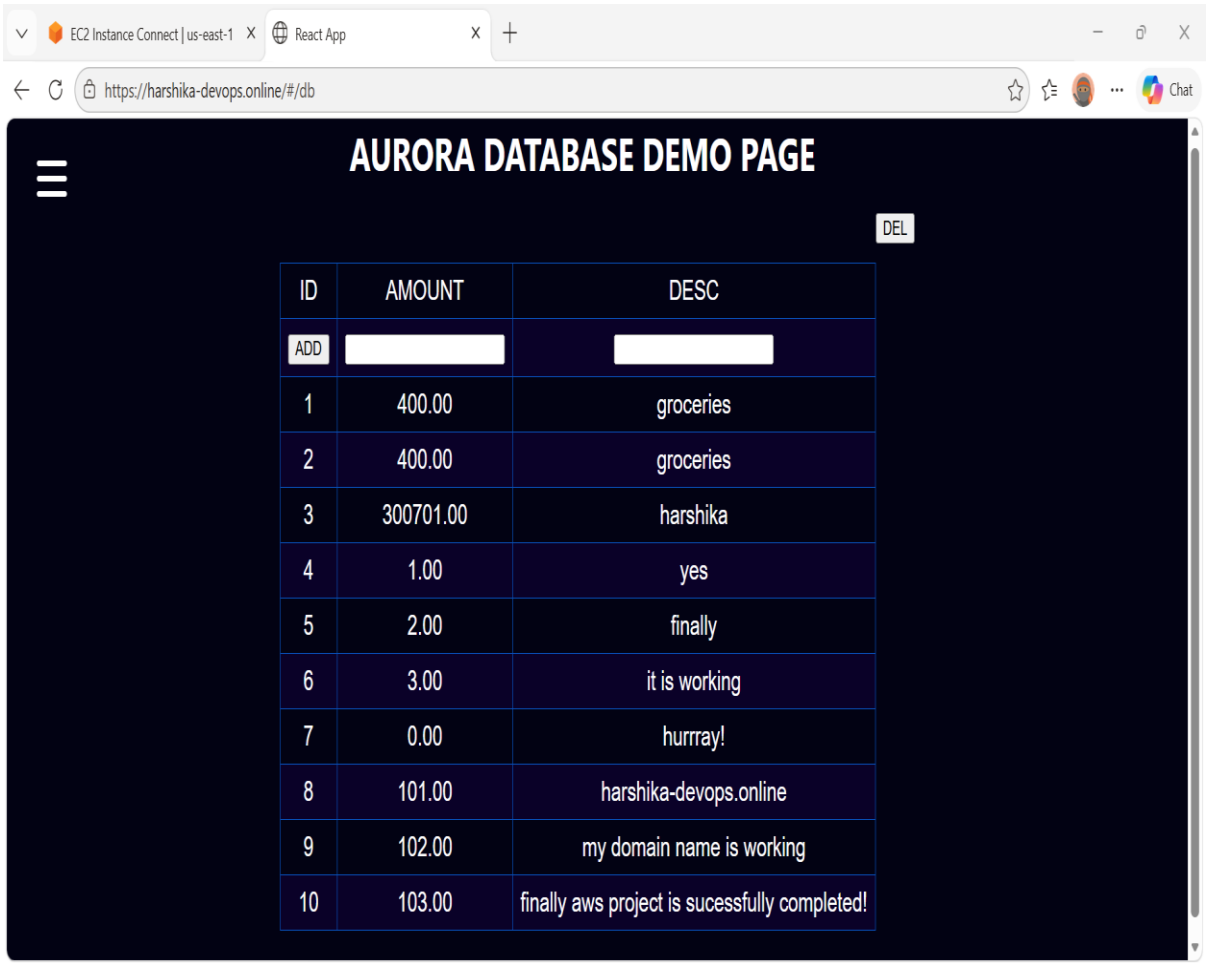
Final entries.



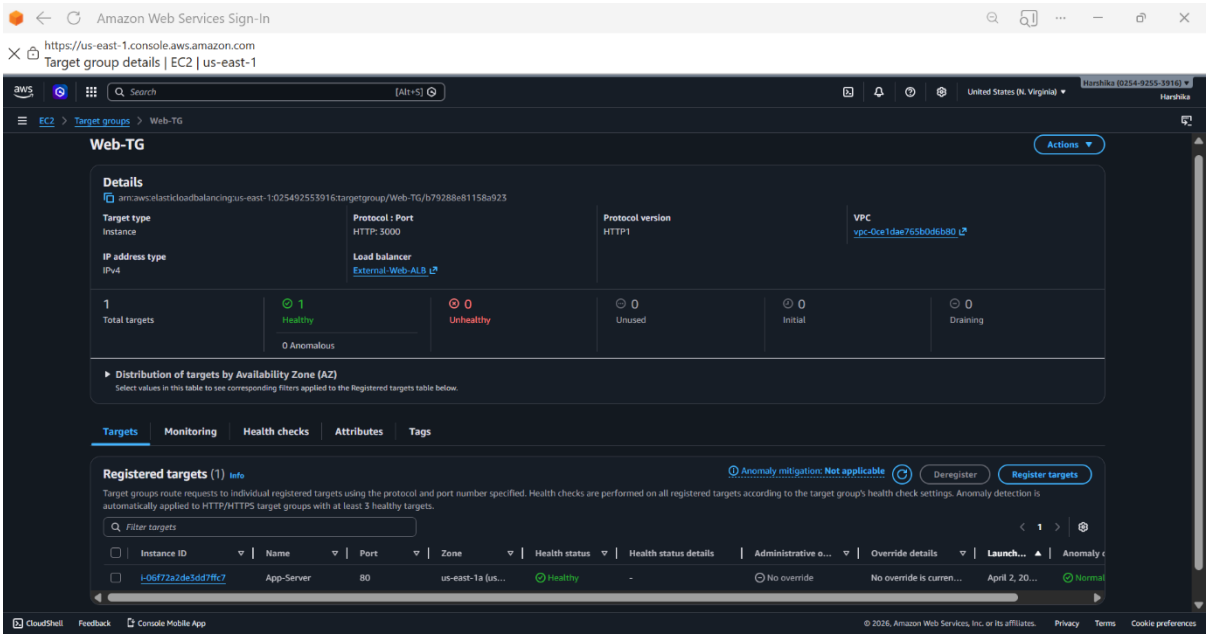
To prove the 3-Tier architecture is functional, putting these four specific screenshots:

1. **The Live Website:** This proves the Frontend, Backend, and Database are all talking to each other.

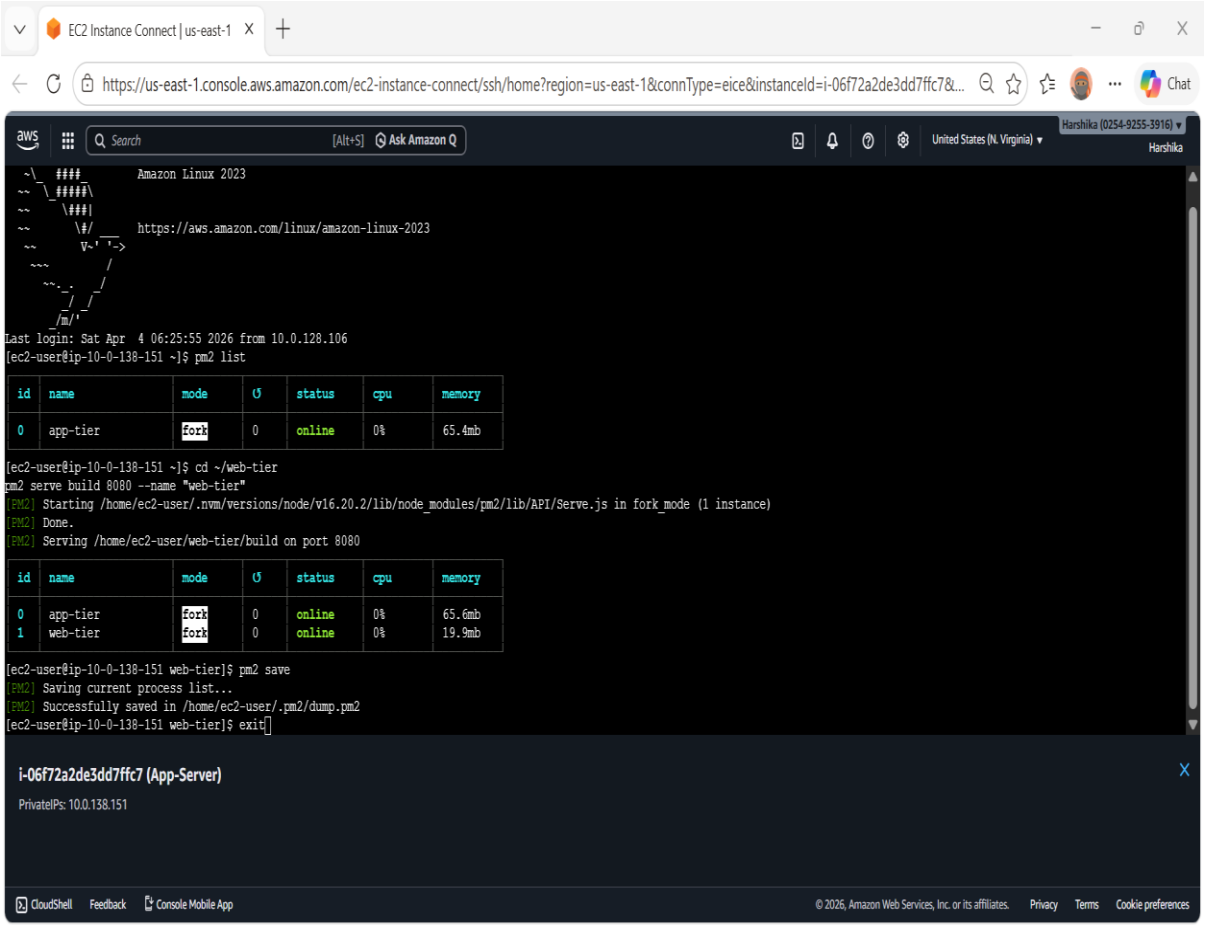

React App
https://harshika-devops.online/#/
InPrivate
Chat
'AWS 3-TIER DEMO BY KASTRO'
AWS PROJECT
AWS
THREE TIER ARCHITECTURE
VPC
Amazon EC2
AWS RDS
KASTRO KIRAN V


EC2 Instance Connect | us-east-1
React App
https://harshika-devops.online/#/db
Chat
AURORA DATABASE DEMO PAGE
DEL
ID
AMOUNT
DESC
ADD
1 400.00 groceries
2 400.00 groceries
3 300701.00 harshika
4 1.00 yes
5 2.00 finally
6 3.00 it is working
7 0.00 hurrray!
8 101.00 harshika-devops.online
9 102.00 my domain name is working
10 103.00 finally aws project is sucessfully completed!

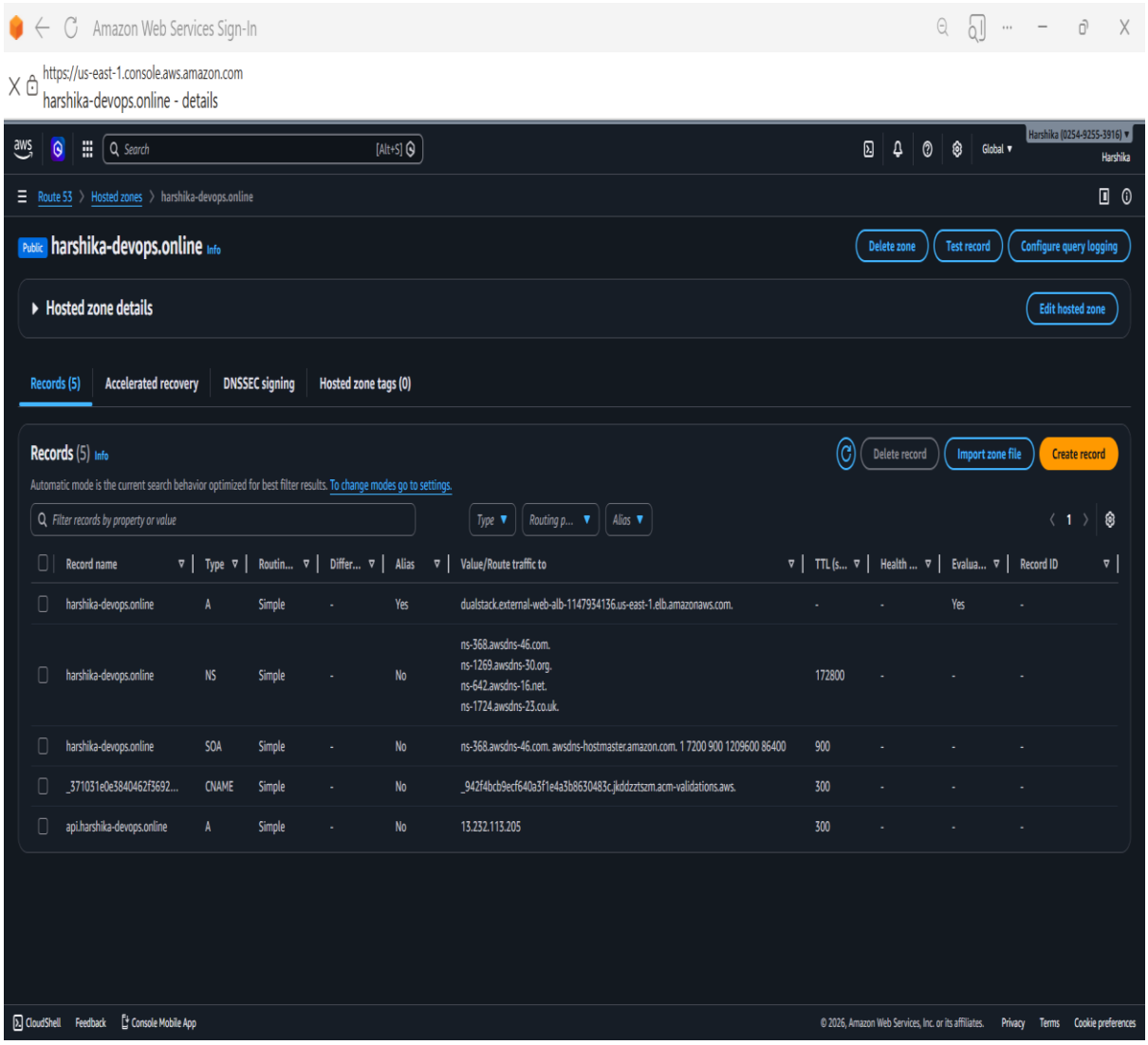
2. **Healthy Target Group:** Go to **Target Groups** and show the **Healthy** status for Port 80. This proves the Load Balancer configuration is correct.



3. **The Running Backend (Terminal):** Show your terminal with pm2 list showing the app-tier as **online**.



4. **Route 53 Records:** Show your hosted zone with the **Alias** record pointing to the Load Balancer.



The Troubleshooting Ledger:

Technical Challenge	Root Cause	Resolution Implemented
502 Bad Gateway Error	Nginx lacked permissions to access the application directory or connect to Port 4000.	Adjusted directory permissions with chmod and updated SELinux/System policies for network connectivity.
Unhealthy Target Groups	The Load Balancer was performing health checks on the wrong port (Port 3000 instead of Port 80).	Reconfigured Target Group settings to align with the Nginx listener port.
Connection Timed Out	Missing Inbound Rules for Port 8080 in the instance Security Group.	Added specific Custom TCP rules to the Security Group allowing traffic from the internet.

Reflective Learning & Technical Growth:

"Completing this 3-tier architecture project served as a comprehensive masterclass in full-stack cloud infrastructure and systems administration. Beyond the successful deployment of the application, the most significant learning occurred during the iterative troubleshooting of the **Nginx Reverse Proxy** and **Application Load Balancer (ALB)** connectivity.

I gained a deep, hands-on understanding of how **Linux permissions** and **security policies** (such as SELinux and directory-level access) act as critical gatekeepers; solving the '502 Bad Gateway' error taught me that infrastructure success requires a holistic view of the environment, from the file system to the network layer. Furthermore, configuring the **Target Group health checks** provided practical insight into the importance of port synchronisation—ensuring the ALB correctly monitors Port 80 while the application logic resides on Port 4000.

This project also reinforced the importance of **Resource Lifecycle Management**; navigating the dependency chain during the decommissioning process—identifying active Network Interfaces (ENIs) and releasing Elastic IPs—ensured a thorough understanding of AWS cost-optimisation and clean architecture practices. This journey has solidified my ability to diagnose complex networking 'bottlenecks' and has prepared me to manage production-grade cloud environments with a focus on security, scalability, and operational excellence.

Conclusion:

"The successful deployment of the 3-tier architecture on AWS confirms that the infrastructure is robust, secure, and ready for production-level traffic. By integrating an **Application Load Balancer**, **Nginx Reverse Proxy**, and **Aurora RDS**, the project achieved high availability and clear separation of concerns. All technical objectives—including SSL/TLS encryption and custom DNS routing—were fully realised. This project stands as a definitive proof of concept for modern cloud deployment strategies and serves as a foundation for future implementations involving automated CI/CD pipelines and Infrastructure as Code (IaC)."

In future iterations, I plan to automate this deployment using **Terraform** to ensure the infrastructure can be recreated instantly in any AWS region.

Thank you!