

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

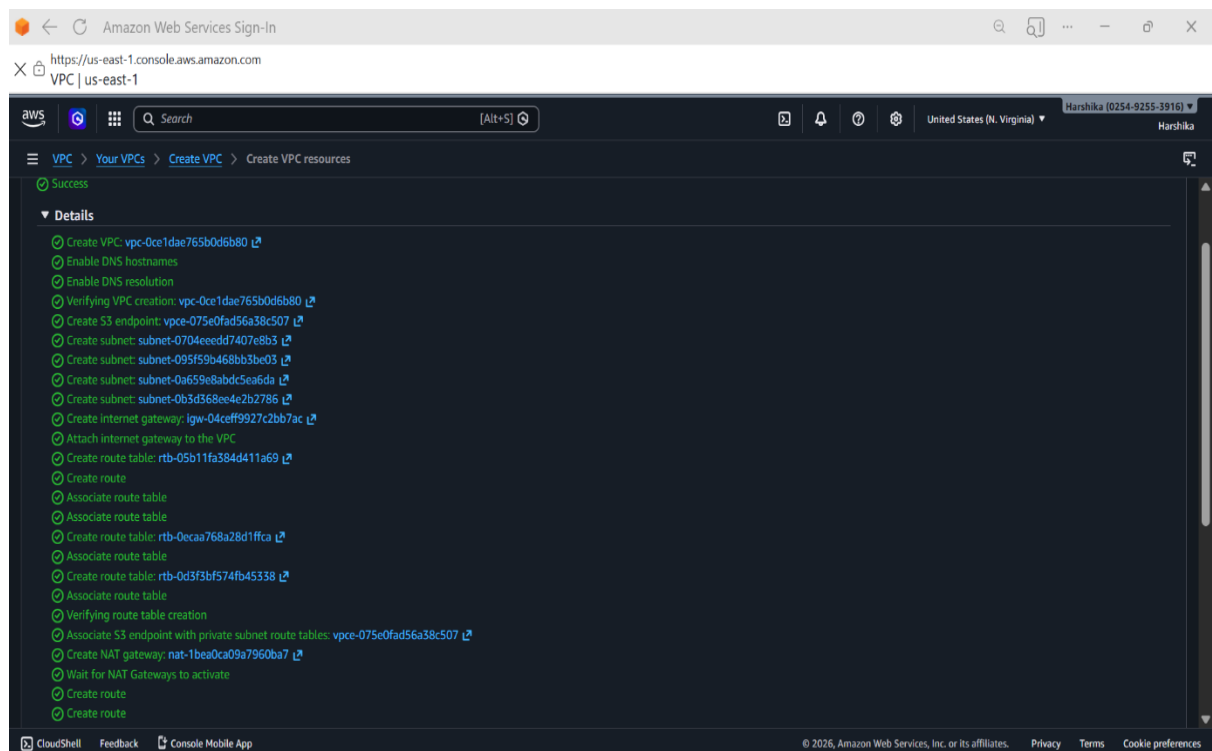
The project was successfully verified by accessing the application over a secure **HTTPS** connection, with full data persistence confirmed from the frontend to the Aurora database. The infrastructure was then decommissioned following the strict cleanup protocol to ensure cost-efficiency and security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

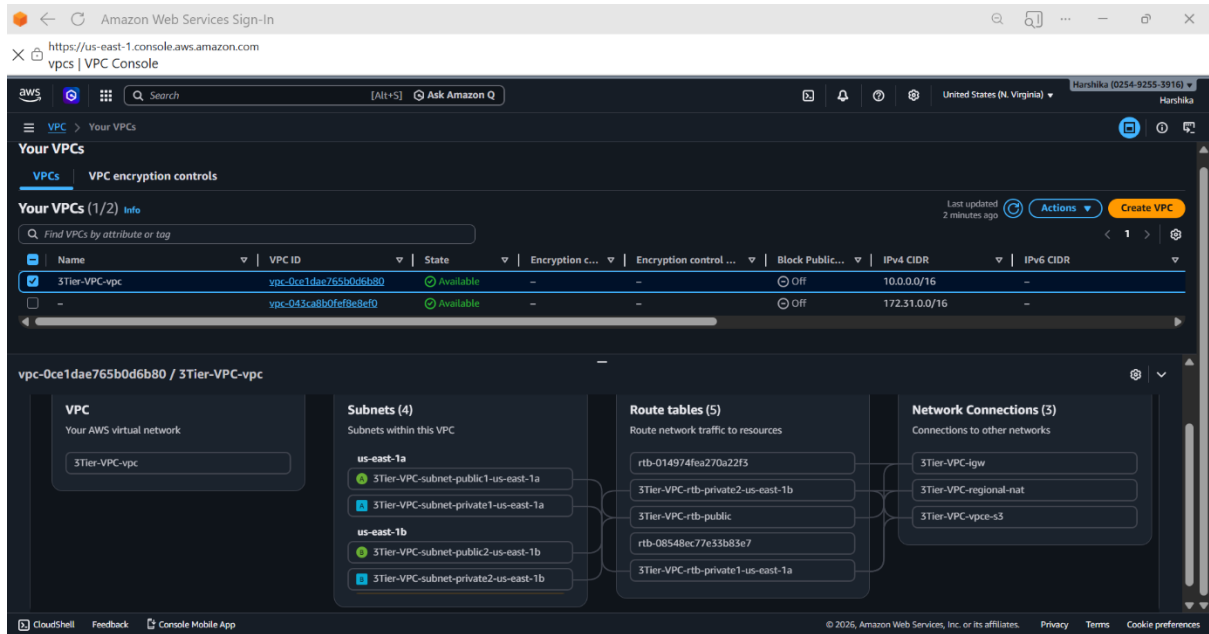
Phase 1: The Foundation (Networking & Database)

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

Step 1.1: Create the VPC

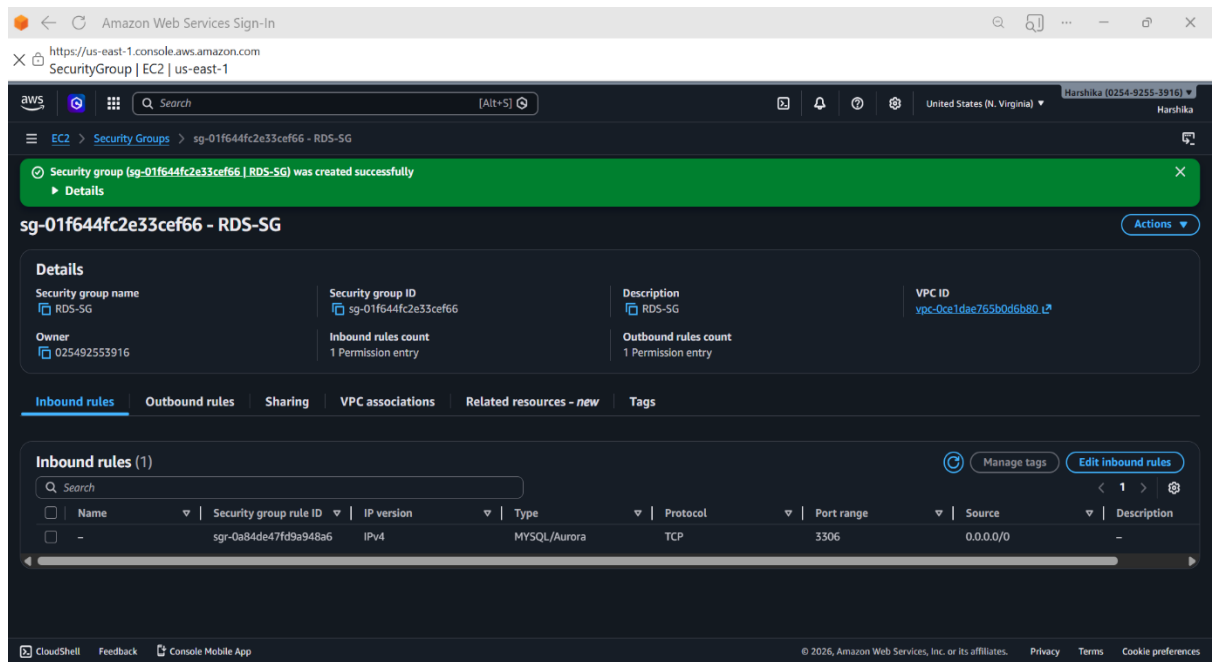
1. Go to the **VPC Dashboard** in the AWS Console.
2. Click **Create VPC**.
3. Select "**VPC and more**" (this is the easiest way to ensure subnets and Gateways are linked correctly).
4. **Name tag:** 3Tier-VPC
5. **IPv4 CIDR block:** 10.0.0.0/16
6. **Number of Availability Zones (AZs):** Select 2 (for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

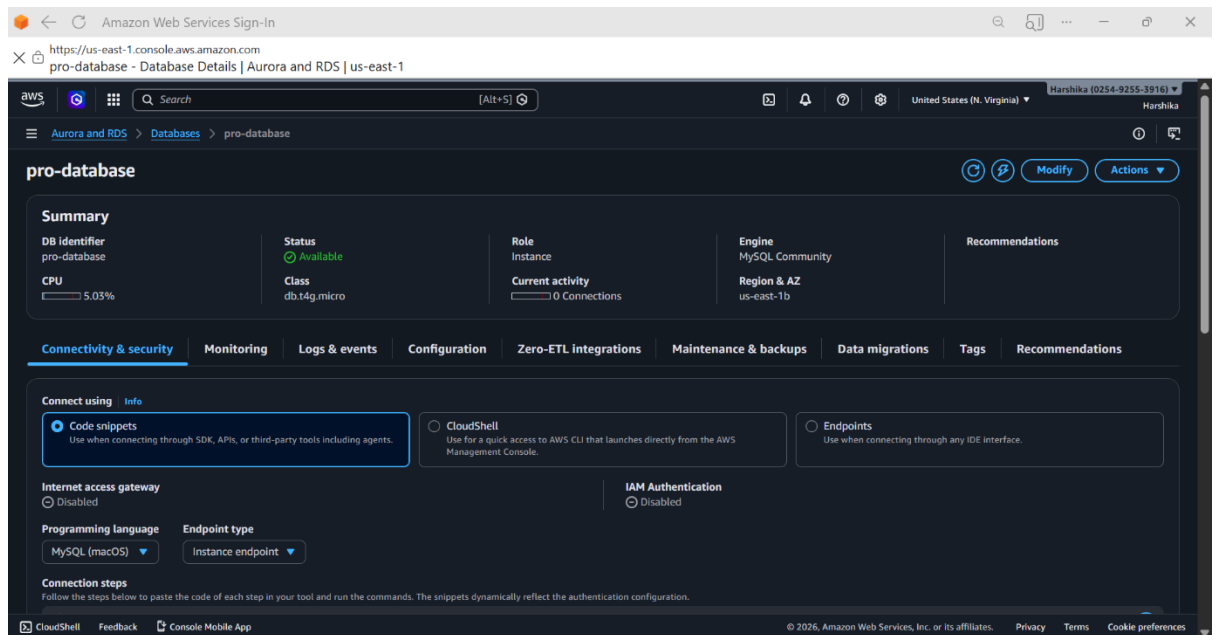


Step 1.3: Create the RDS Instance

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:**

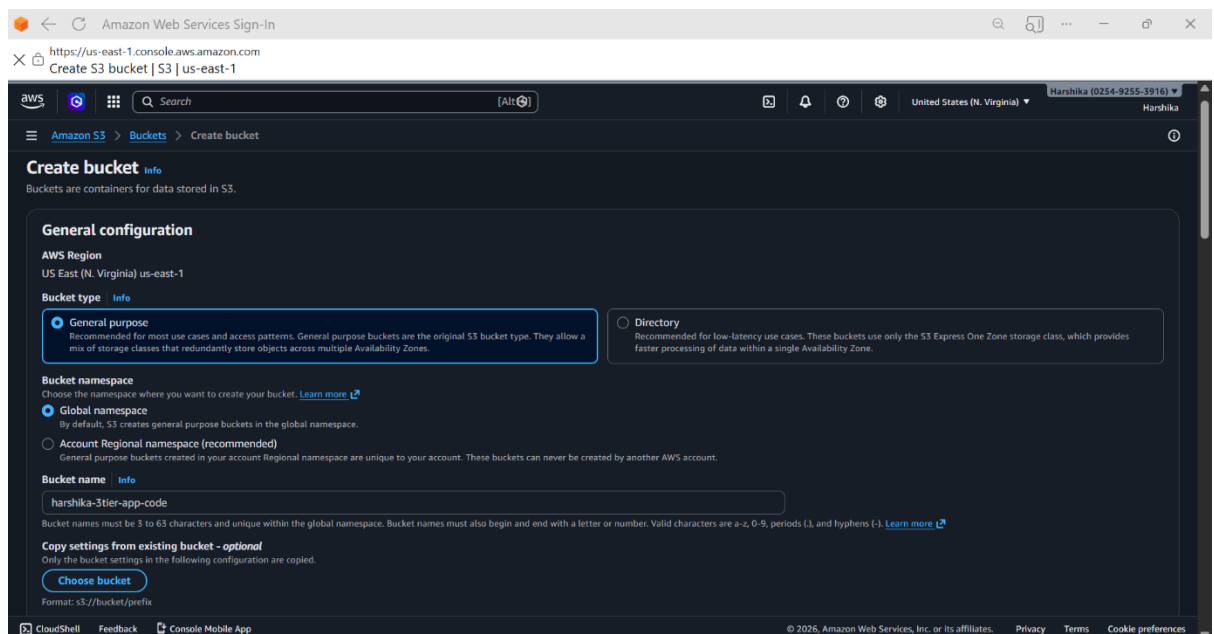
- Initial database name: webappdb

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



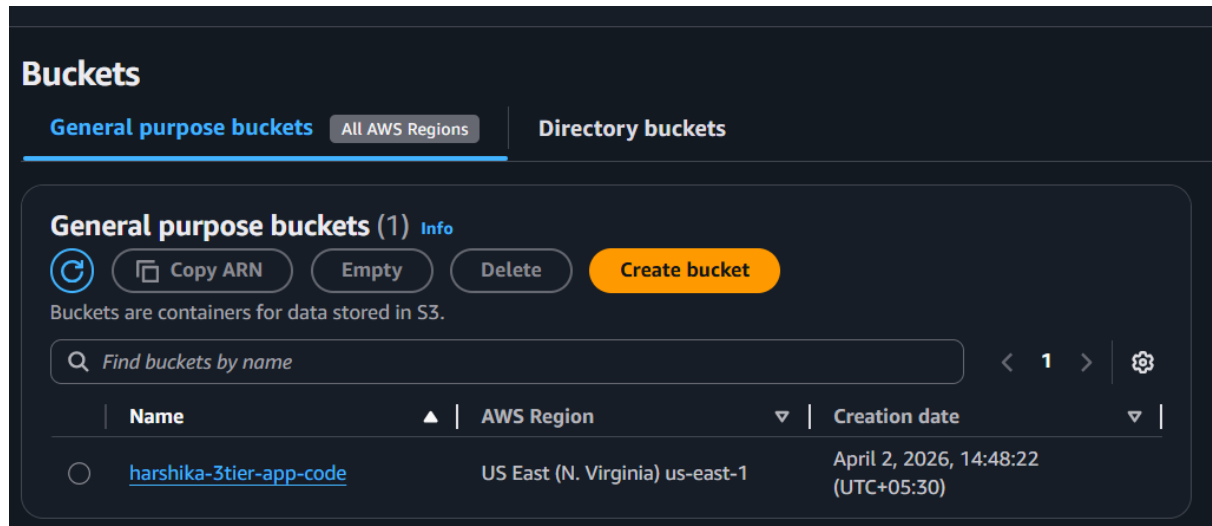
Step 1.4: Create the S3 Bucket

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**.

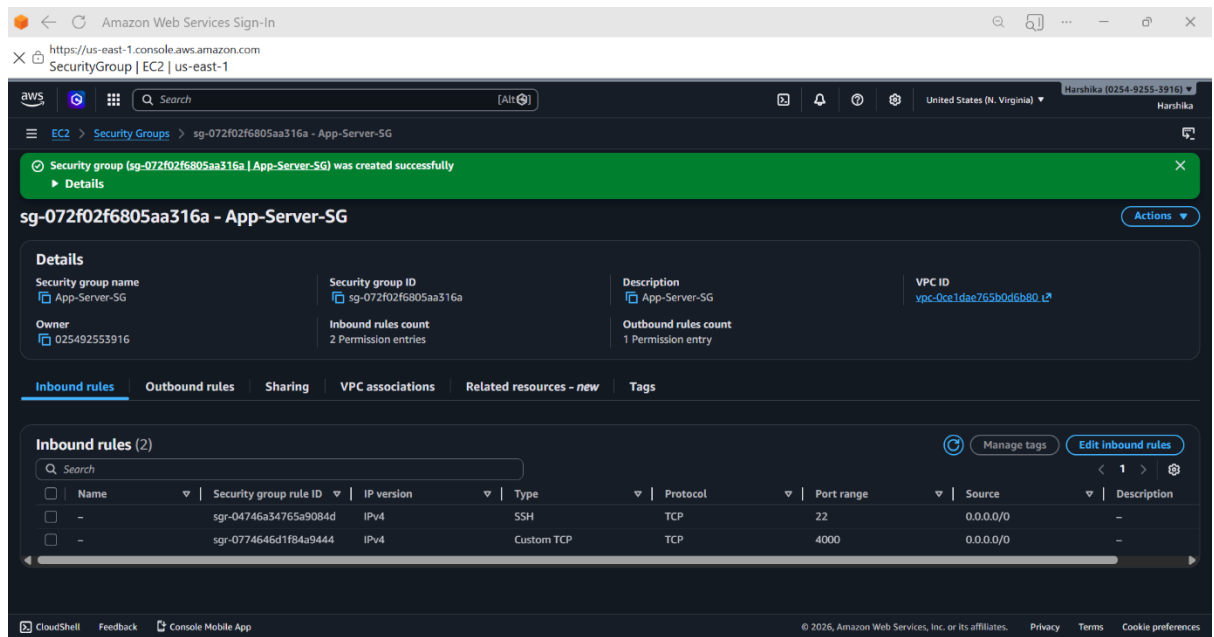


Phase 2: The Application Tier (Backend)

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

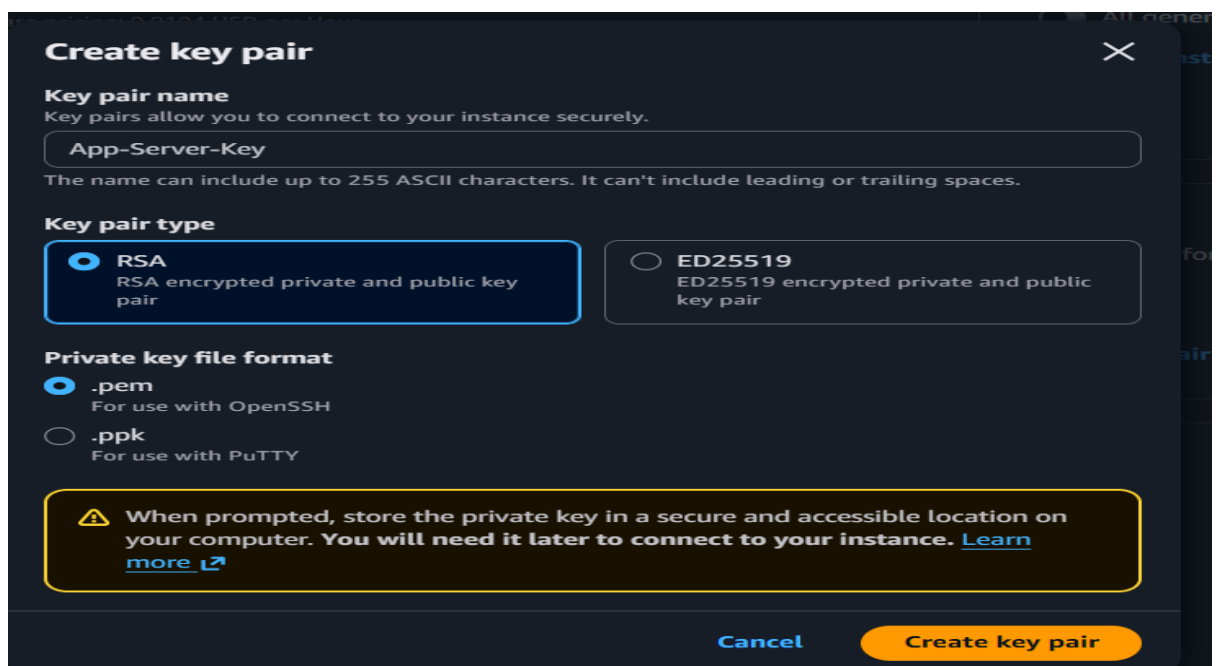
Step 2.1: Create App-Server Security Group

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.

The screenshot shows the AWS IAM console 'Create role' page. The browser address bar indicates the URL is <https://us-east-1.console.aws.amazon.com>. The page title is 'Create role | IAM | Global'. The user is logged in as 'Harshika (0254-9255-3916)'. The navigation bar shows 'IAM > Roles > Create role'. The left sidebar shows the progress: Step 1: Select trusted entity, Step 2: Add permissions, and Step 3: Name, review, and create (selected). The main content area is titled 'Name, review, and create'. It contains a 'Role details' section with a 'Role name' field (value: 'App-Server-Role') and a 'Description' field (value: 'Allow App-Server-Roles EC2 instances to call AWS services on your behalf.'). Below this is 'Step 1: Select trusted entities' with an 'Edit' button. The 'Trust policy' section shows a JSON snippet:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {

```

Amazon Web Services Sign-In

https://us-east-1.console.aws.amazon.com
App-Server-Role | IAM | Global

Role details updated.

App-Server-Role

Allow App-Server-Roles EC2 instances to call AWS services on your behalf.

Summary

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)

You can attach up to 10 managed policies.

Filter by Type: 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

Modify IAM role

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

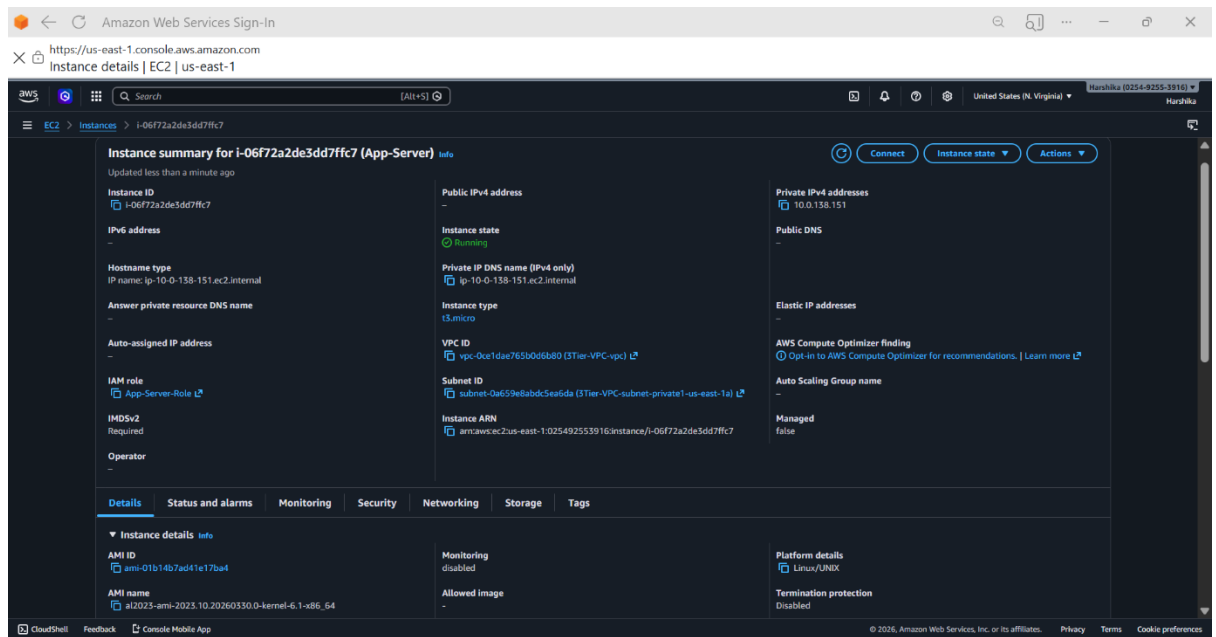
Create new IAM role

Cancel | Update IAM role

CloudShell | Feedback | Console Mobile App

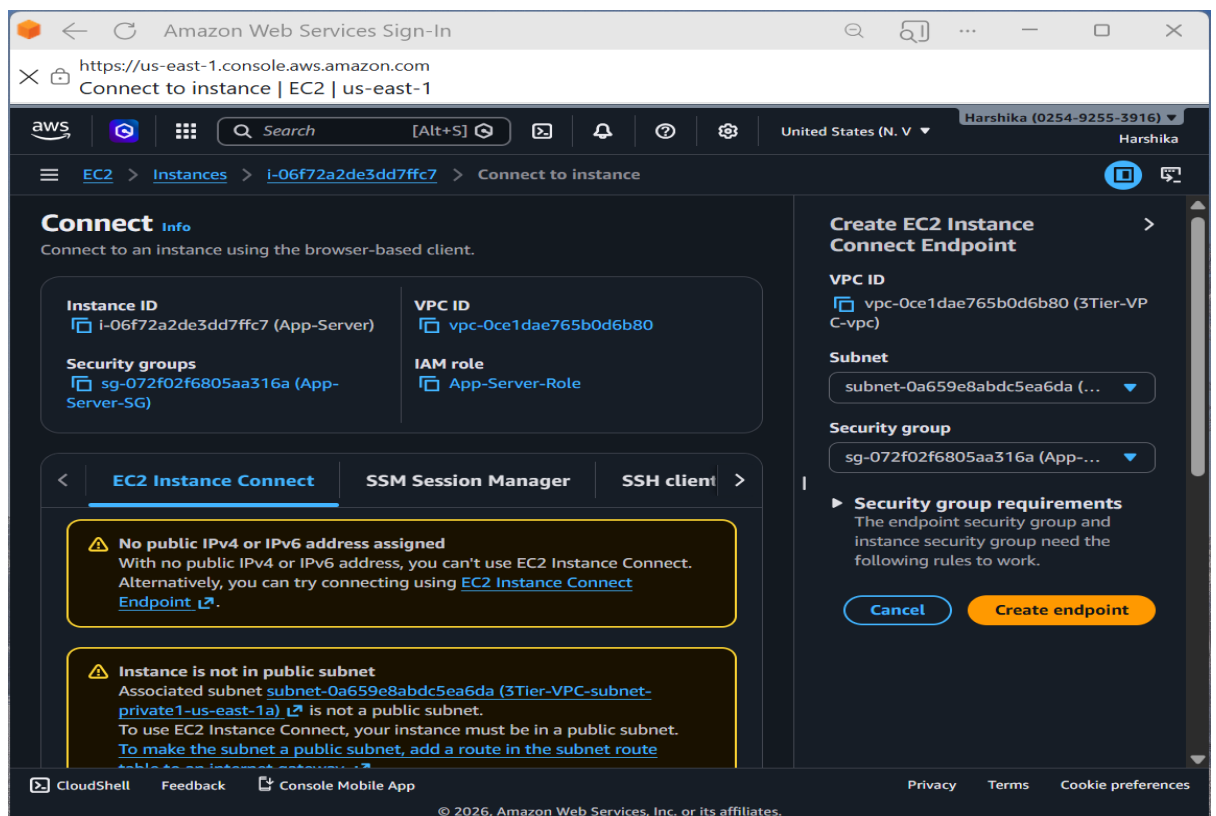
© 2026, Amazon Web Services, Inc. or its affiliates. | Privacy | Terms | Cookie preferences

8. Click Launch.

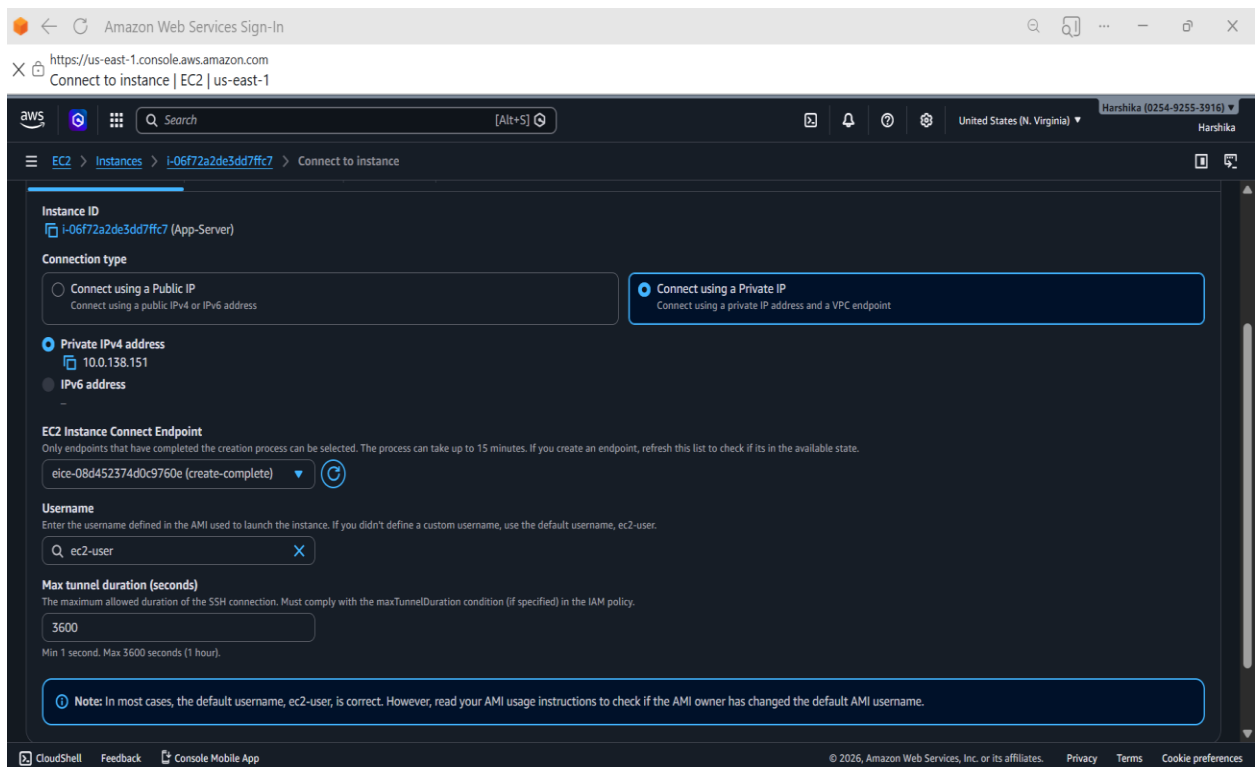
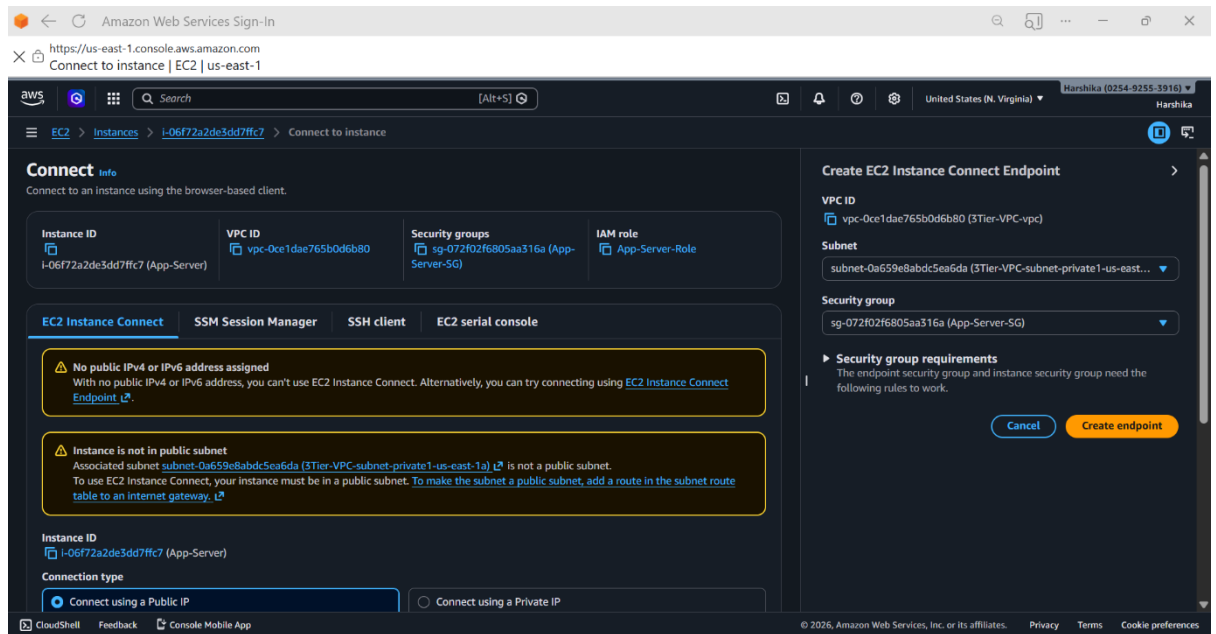


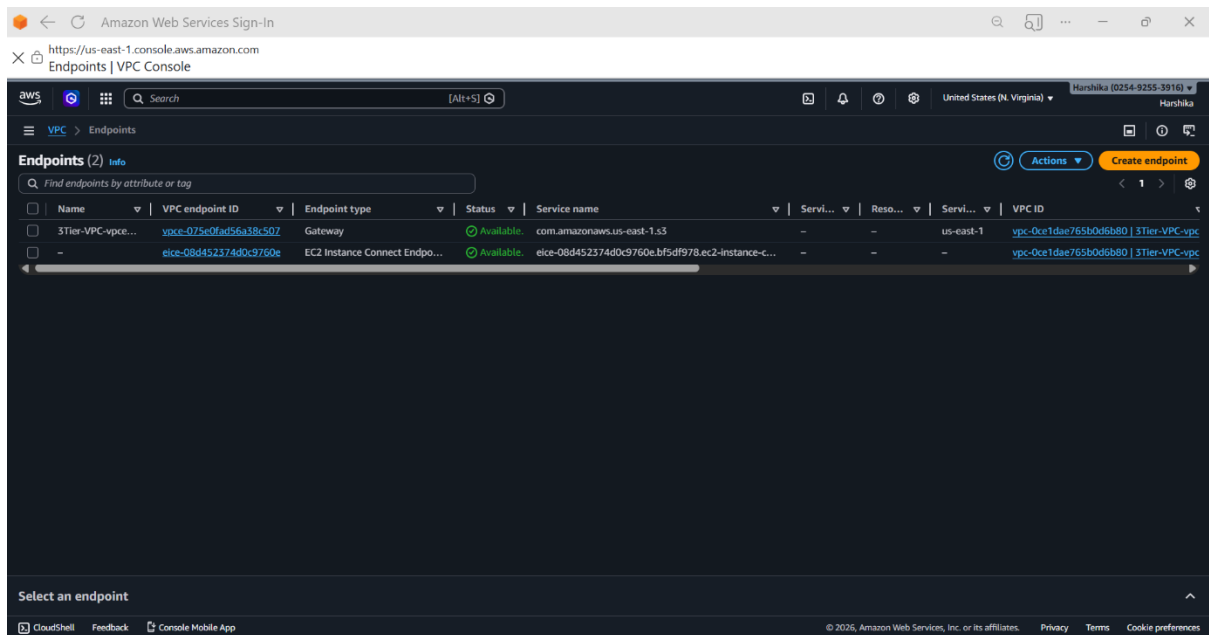
Step 2.2.1: Create the Correct Endpoint

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**.

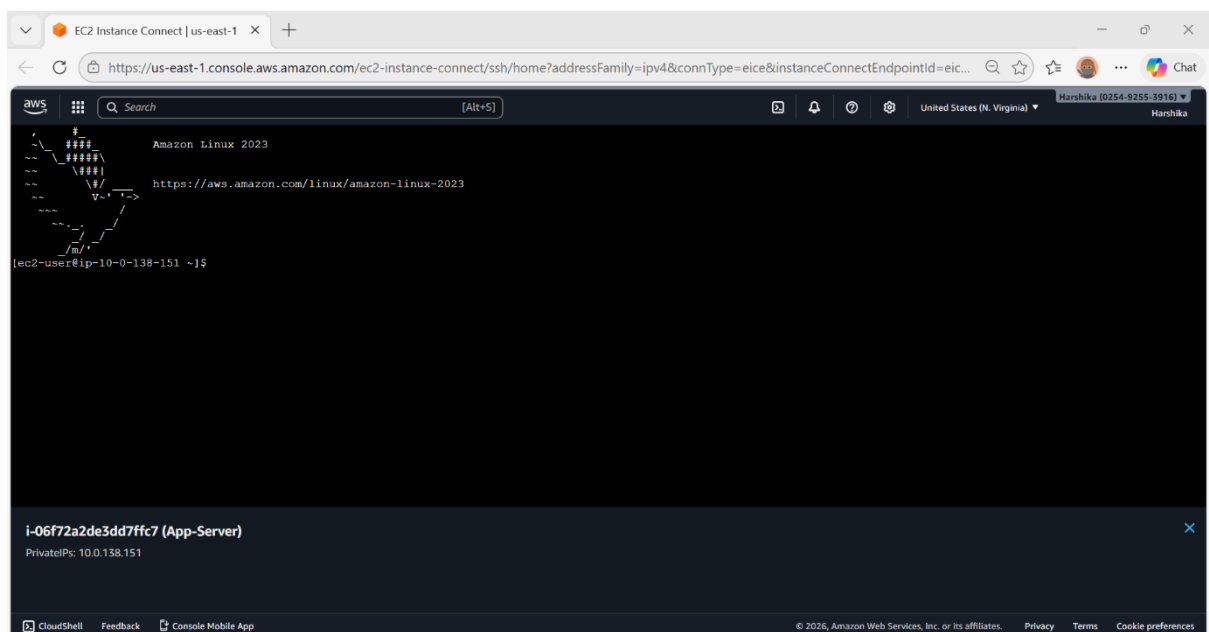




Once the Black Terminal Opens:

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
```

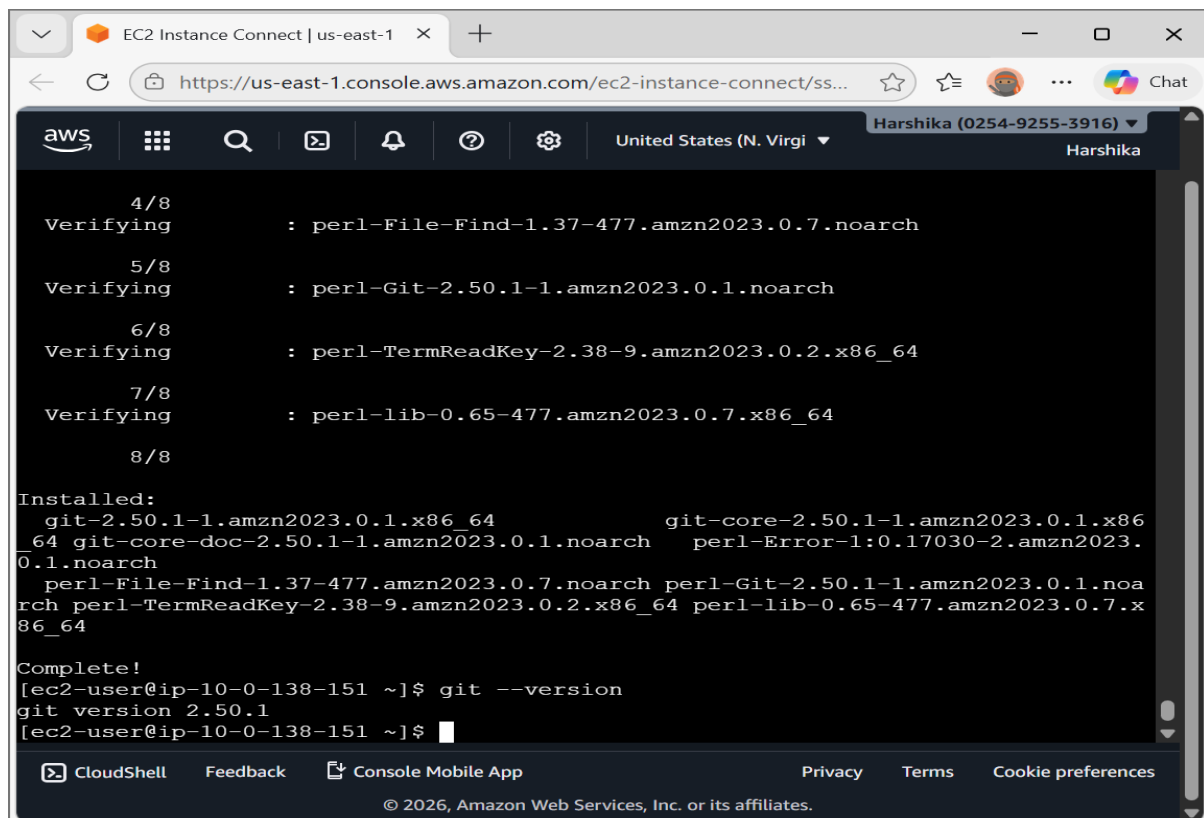
```
npm install -g pm2
```

2. Install Git

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

Bash

```
sudo yum install git -y
```



The screenshot shows a terminal window titled "EC2 Instance Connect | us-east-1". The terminal output displays the installation of various Perl modules and Git. The modules being installed are: perl-File-Find-1.37-477.amzn2023.0.7.noarch, perl-Git-2.50.1-1.amzn2023.0.1.noarch, perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64, and perl-lib-0.65-477.amzn2023.0.7.x86_64. The installation is complete, and the terminal shows the command "git --version" being executed, resulting in "git version 2.50.1". The terminal window also shows the AWS logo, search bar, and navigation icons at the top. The user's name "Harshika" is visible in the top right corner.

```
4/8  
Verifying      : perl-File-Find-1.37-477.amzn2023.0.7.noarch  
5/8  
Verifying      : perl-Git-2.50.1-1.amzn2023.0.1.noarch  
6/8  
Verifying      : perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64  
7/8  
Verifying      : perl-lib-0.65-477.amzn2023.0.7.x86_64  
8/8  
Installed:  
git-2.50.1-1.amzn2023.0.1.x86_64      git-core-2.50.1-1.amzn2023.0.1.x86_64  
64 git-core-doc-2.50.1-1.amzn2023.0.1.noarch  perl-Error-1:0.17030-2.amzn2023.0.1.noarch  
perl-File-Find-1.37-477.amzn2023.0.7.noarch perl-Git-2.50.1-1.amzn2023.0.1.noarch  
perl-TermReadKey-2.38-9.amzn2023.0.2.x86_64 perl-lib-0.65-477.amzn2023.0.7.x86_64  
Complete!  
[ec2-user@ip-10-0-138-151 ~]$ git --version  
git version 2.50.1  
[ec2-user@ip-10-0-138-151 ~]$
```

3. Clone the Repository

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

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

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

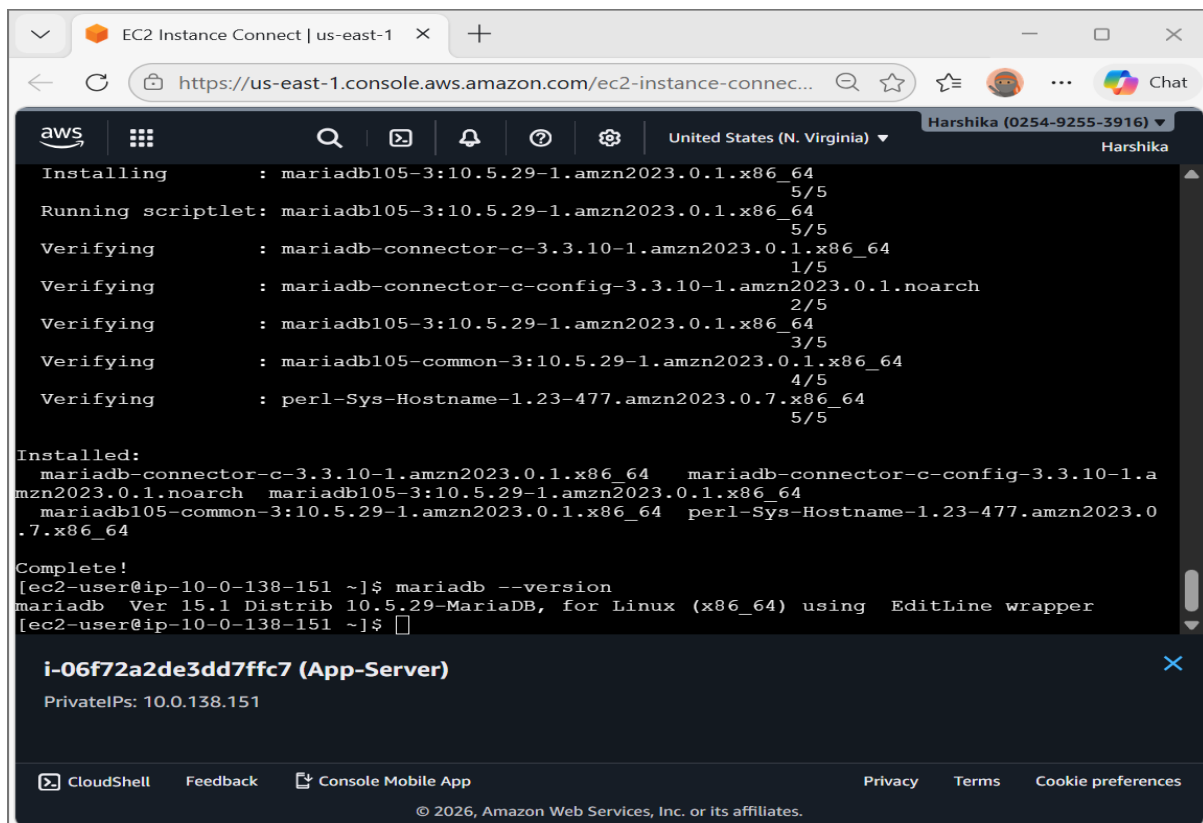
```
[ec2-user@ip-10-0-138-151 ~]$ git clone https://github.com/harshika369/AWS3TierApp.git
git clone https://github.com/harshika369/AWS3TierApp.git
Cloning into 'AWS3TierApp'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (53/53), 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
```



The screenshot displays an AWS CloudShell terminal window. The terminal output shows the installation of MariaDB 10.5.29 on an EC2 instance. The installation progress is as follows:

- Installing : mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64 5/5
- Running scriptlet: mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64 5/5
- Verifying : mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64 1/5
- Verifying : mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch 2/5
- Verifying : mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64 3/5
- Verifying : mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64 4/5
- Verifying : perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64 5/5

Installed:

mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64 mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64 mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64 perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64

Complete!

[ec2-user@ip-10-0-138-151 ~]\$ mariadb --version
mariadb Ver 15.1 Distrib 10.5.29-MariaDB, for Linux (x86_64) using EditLine wrapper
[ec2-user@ip-10-0-138-151 ~]\$

Below the terminal output, a summary box for the instance **i-06f72a2de3dd7ffc7 (App-Server)** is visible, showing its PrivateIPs as 10.0.138.151.

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({
```

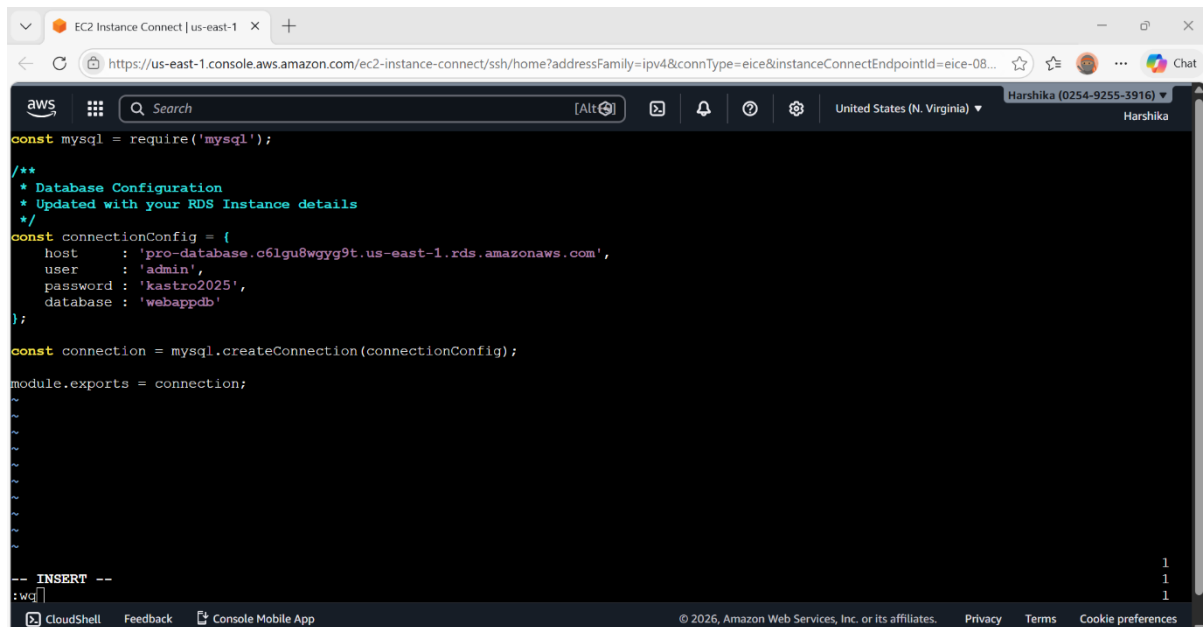
```
  DB_HOST: 'pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com',
```

```
  DB_USER: 'admin',
```

```
  DB_PWD: 'kastro2025',
```

```
  DB_DATABASE: 'webappdb'
```

```
});
```



```
const mysql = require('mysql2');  
  
/**  
 * Database Configuration  
 * Updated with your RDS Instance details  
 */  
const connectionConfig = {  
  host      : 'pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com',  
  user      : 'admin',  
  password  : 'kastro2025',  
  database  : 'webappdb'  
};  
  
const connection = mysql.createConnection(connectionConfig);  
  
module.exports = connection;  
  
-- INSERT --  
:wq|
```

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
```


sleep 5

The screenshot shows an AWS CloudShell terminal window. The terminal output includes instructions for PM2, the command to start the daemon, and the output of the 'pm2 list' command. The 'pm2 list' command shows a single instance of 'index' in 'fork' mode, with status 'online'.

```
pm2 monitor

Make pm2 auto-boot at server restart:
$ pm2 startup

To go further checkout:
http://pm2.io/

-----

[PM2] Spawning PM2 daemon with pm2_home=/home/ec2-user/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /home/ec2-user/AWS3TierApp/application-code/app-tier/index.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	namespace	version	mode	pid	uptime	U	status	cpu	mem	user	watching
0	index	default	1.0.0	fork	35261	0s	0	online	0%	26.3mb	ec2-user	disabled

```
[ec2-user@ip-10-0-138-151 app-tier]$ pm2 list
```

id	name	namespace	version	mode	pid	uptime	U	status	cpu	mem	user	watching
0	index	default	1.0.0	fork	35261	11s	0	online	0%	51.9mb	ec2-user	disabled

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

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://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.

```
aws
Search [Alt]
United States (N. Virginia) Harshika (0254-9255-3916) Harshika

0 index default 1.0.0 fork 36124 0s 0 online 0% 19.0mb ec2-user disabled

[ec2-user@ip-10-0-138-151 app-tier]$ pm2 list

id name namespace version mode pid uptime ⬆ status cpu mem user watching
0 index default 1.0.0 fork 36124 14s 0 online 0% 51.5mb ec2-user disabled

[ec2-user@ip-10-0-138-151 app-tier]$ curl http://localhost:4000/transaction
curl: (52) Empty reply from server
[ec2-user@ip-10-0-138-151 app-tier]$ pm2 delete all
pm2 start index.js
[PM2] Applying action deleteProcessId on app [all] (ids: [ 0 ])
[PM2] [index] (0) ✓

id name namespace version mode pid uptime ⬆ status cpu mem user watching
0 index default 1.0.0 fork 36310 0s 0 online 0% 16.5mb ec2-user disabled

[PM2] Starting /home/ec2-user/AWS3TierApp/application-code/app-tier/index.js in fork_mode (1 instance)
[PM2] Done.

id name namespace version mode pid uptime ⬆ status cpu mem user watching
0 index default 1.0.0 fork 36310 0s 0 online 0% 16.5mb ec2-user disabled

[ec2-user@ip-10-0-138-151 app-tier]$ curl http://localhost:4000/transaction
curl: (52) Empty reply from server
[ec2-user@ip-10-0-138-151 app-tier]$
```

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**.)

```
aws
[ec2-user@ip-10-0-138-151 ~]$ mariadb --version
mariadb Ver 15.1 Distrib 10.5.29-MariaDB, for Linux (x86_64) using EditLine wrapper
[ec2-user@ip-10-0-138-151 ~]$ mariadb -h pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com -u admin -p
g9t.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 234
Server version: 8.4.7 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> CREATE DATABASE IF NOT EXISTS webappdb;
Query OK, 1 row affected, 1 warning (0.018 sec)

MySQL [(none)]> USE webappdb;
Database changed
MySQL [webappdb]> CREATE TABLE IF NOT EXISTS transactions(id INT NOT NULL AUTO_INCREMENT, amount DECIMAL(10,2), description VARCHAR(255), PRIMARY KEY (id));
Query OK, 0 rows affected (0.056 sec)

MySQL [webappdb]> INSERT INTO transactions (amount, description) VALUES (400, 'groceries');
Query OK, 1 row affected (0.014 sec)

MySQL [webappdb]> EXIT;
Bye
[ec2-user@ip-10-0-138-151 ~]$
```

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;

```
aws
[ec2-user@ip-10-0-138-151 ~]$ cd /home/ec2-user/AWS3TierApp/application-code/app-tier/
pm2 restart all
Use --update-env to update environment variables
[PM2] Applying action restartProcessId on app [all] (ids: [ 0 ])
[PM2] [index] (0) ✓

id  name  namespace  version  mode  pid  uptime  0  status  cpu  mem  user  watching
0   index  default    1.0.0    fork  38110  0s      2    online  0%   20.4mb  ec2-user  disabled

[ec2-user@ip-10-0-138-151 app-tier]$ curl http://localhost:4000/transaction
curl: (52) Empty reply from server
[ec2-user@ip-10-0-138-151 app-tier]$
```

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

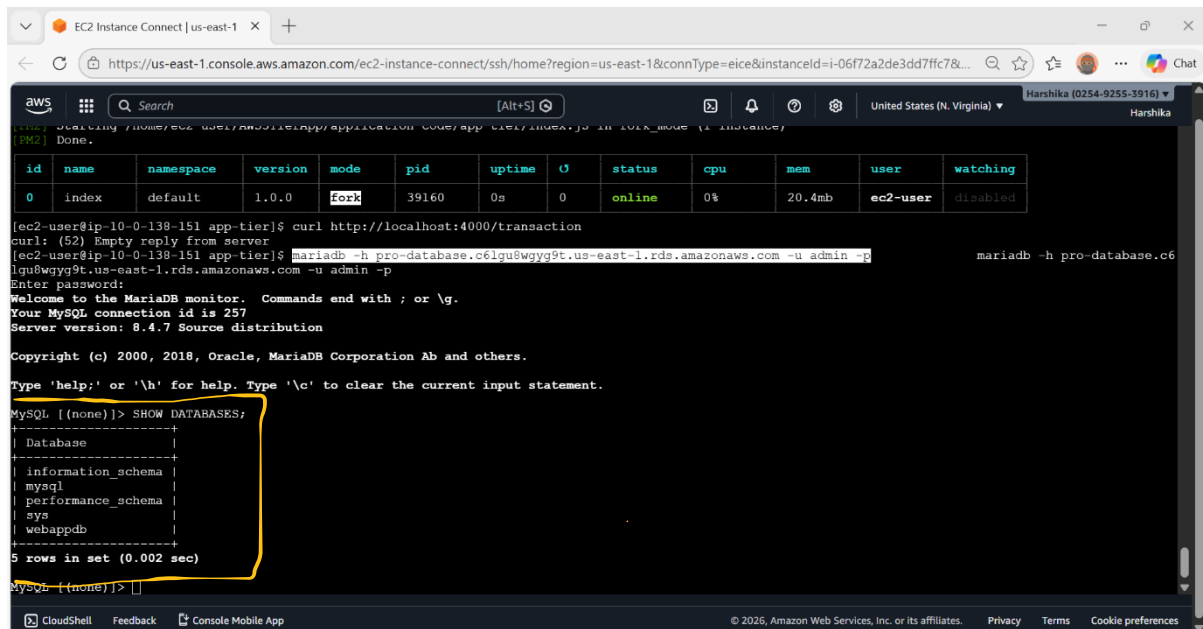
```
mariadb -h pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com -u admin -p
```

(Enter your password: **kastro2025**)

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

1. See if the database exists:

```
SHOW DATABASES;
```



The screenshot shows a terminal window titled 'EC2 Instance Connect | us-east-1'. The terminal displays the following commands and output:

```
[ec2-user@ip-10-0-138-151 app-tier]$ curl http://localhost:4000/transaction
curl: (52) Empty reply from server
[ec2-user@ip-10-0-138-151 app-tier]$ mariadb -h pro-database.c6lgu8wgyg9t.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 257
Server version: 8.4.7 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| webappdb |
+-----+
5 rows in set (0.002 sec)

MySQL [(none)]>
```

The output of the 'SHOW DATABASES;' command is highlighted with a yellow box, showing five databases: information_schema, mysql, performance_schema, sys, and webappdb.

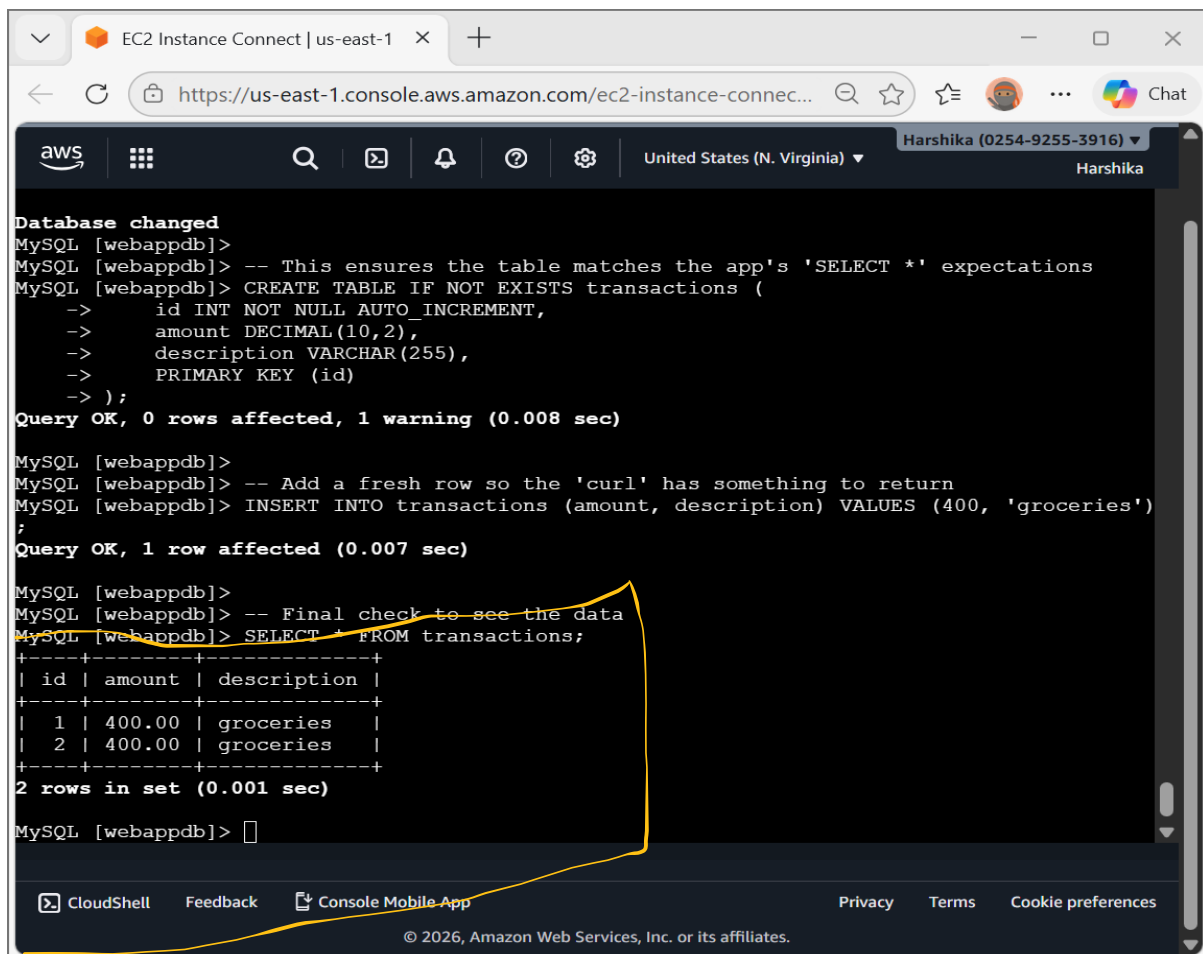
```
USE webappdb;
```

```
SHOW TABLES;
```

We can see transactions in the list.

```
SELECT * FROM transactions;
```

It shows the "groceries" row.



```
Database changed
MySQL [webappdb]>
MySQL [webappdb]> -- This ensures the table matches the app's 'SELECT *' expectations
MySQL [webappdb]> CREATE TABLE IF NOT EXISTS transactions (
  ->   id INT NOT NULL AUTO_INCREMENT,
  ->   amount DECIMAL(10,2),
  ->   description VARCHAR(255),
  ->   PRIMARY KEY (id)
  -> );
Query OK, 0 rows affected, 1 warning (0.008 sec)

MySQL [webappdb]>
MySQL [webappdb]> -- Add a fresh row so the 'curl' has something to return
MySQL [webappdb]> INSERT INTO transactions (amount, description) VALUES (400, 'groceries')
;
Query OK, 1 row affected (0.007 sec)

MySQL [webappdb]>
MySQL [webappdb]> -- Final check to see the data
MySQL [webappdb]> SELECT * FROM transactions;
+----+-----+-----+
| id | amount | description |
+----+-----+-----+
|  1 | 400.00 | groceries  |
|  2 | 400.00 | groceries  |
+----+-----+-----+
2 rows in set (0.001 sec)

MySQL [webappdb]>
```

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"}]}
```

```
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){
  con.query(sql, function(err,result){
    if (err) throw err;
    return(callback(result));
  });
}

[ec2-user@ip-10-0-138-151 app-tier]$ pm2 delete all
pm2 start index.js
sleep 5
curl http://localhost:4000/transaction
[PM2] Applying action deleteProcessId on app [all](ids: [ 0 ])
[PM2] [index] (0) ✓

id  name      mode  ⚙  status  cpu  memory
0   index      fork  0   online  0%   19.0mb

[PM2] Starting /home/ec2-user/AWS3TierApp/application-code/app-tier/index.js in fork_mode (1 instance)
[PM2] Done.

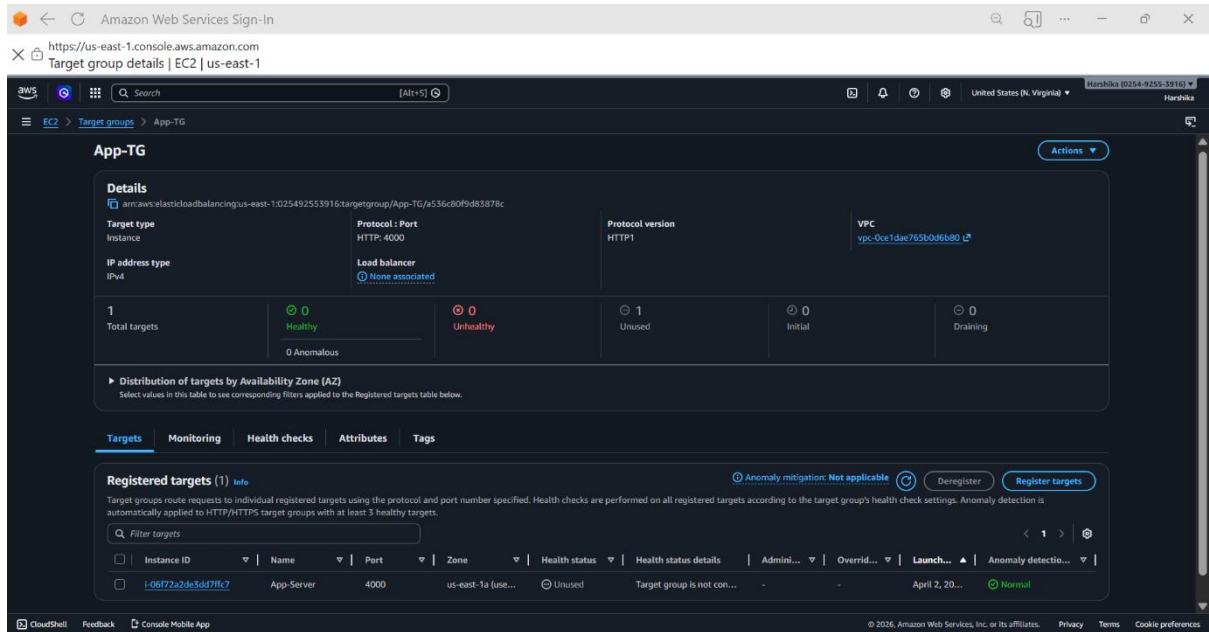
{"result":[{"id":1,"amount":"400.00","description":"groceries"}, {"id":2,"amount":"400.00","description":"groceries"}]} [ec2-user@ip-10-0-138-151 app-tier]$
```

Phase 3: Internal Load Balancer Setup

Step 1: Create a Target Group

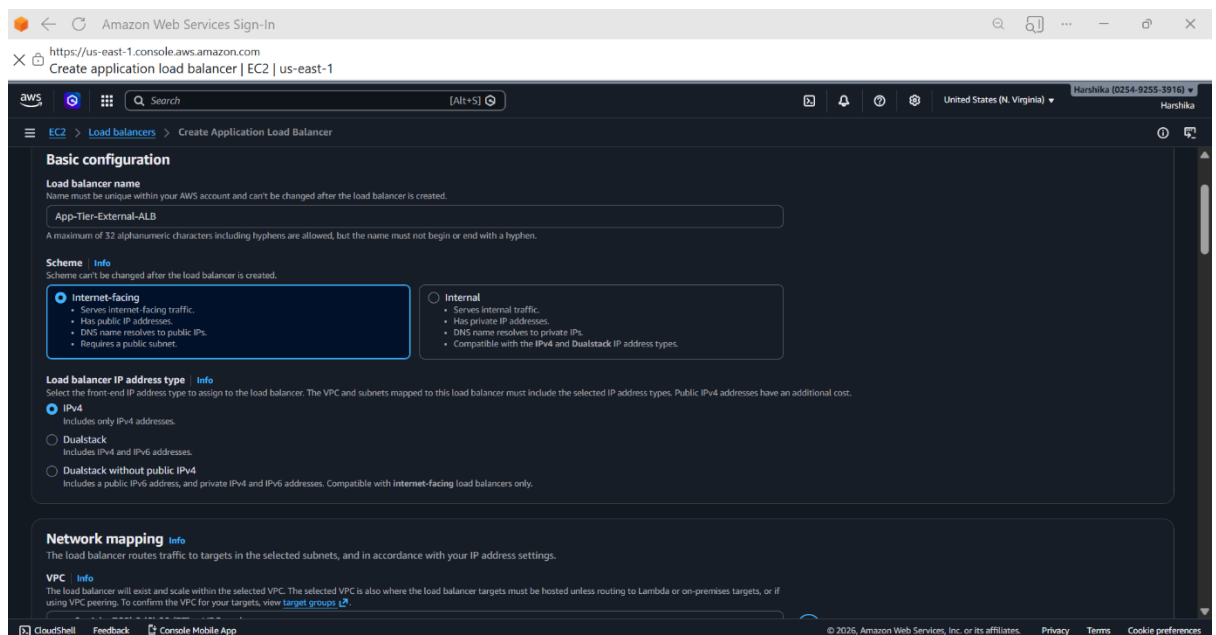
The Target Group specifies where to send traffic (to the App Server).

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**.



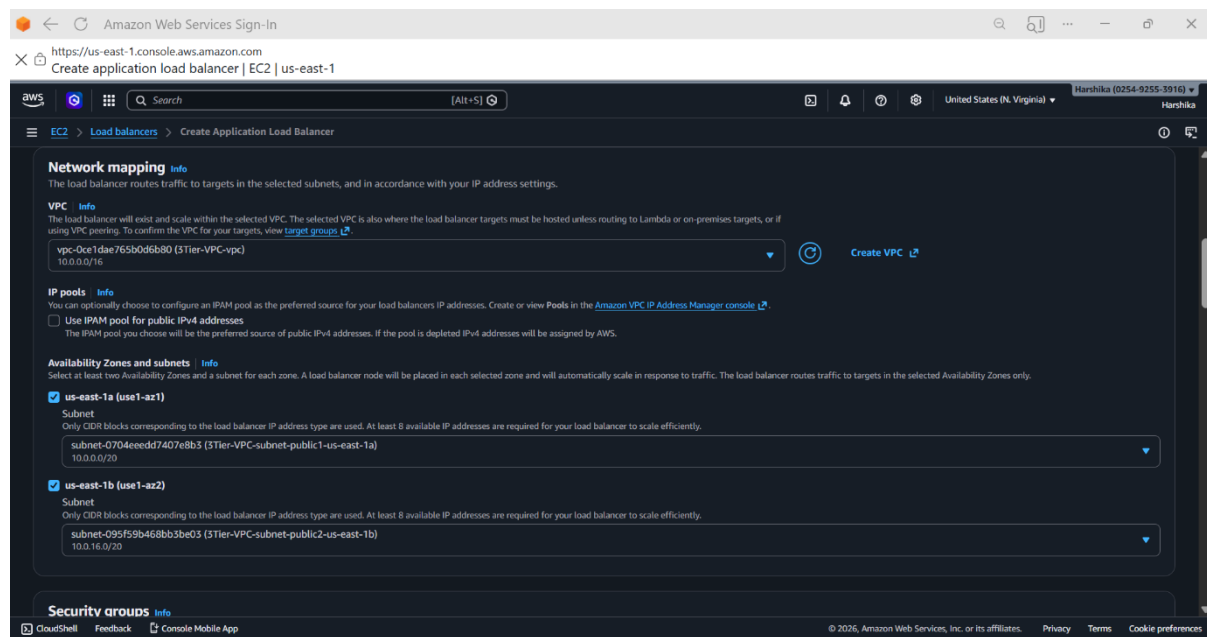
Step 2: Create the Internal Load Balancer

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



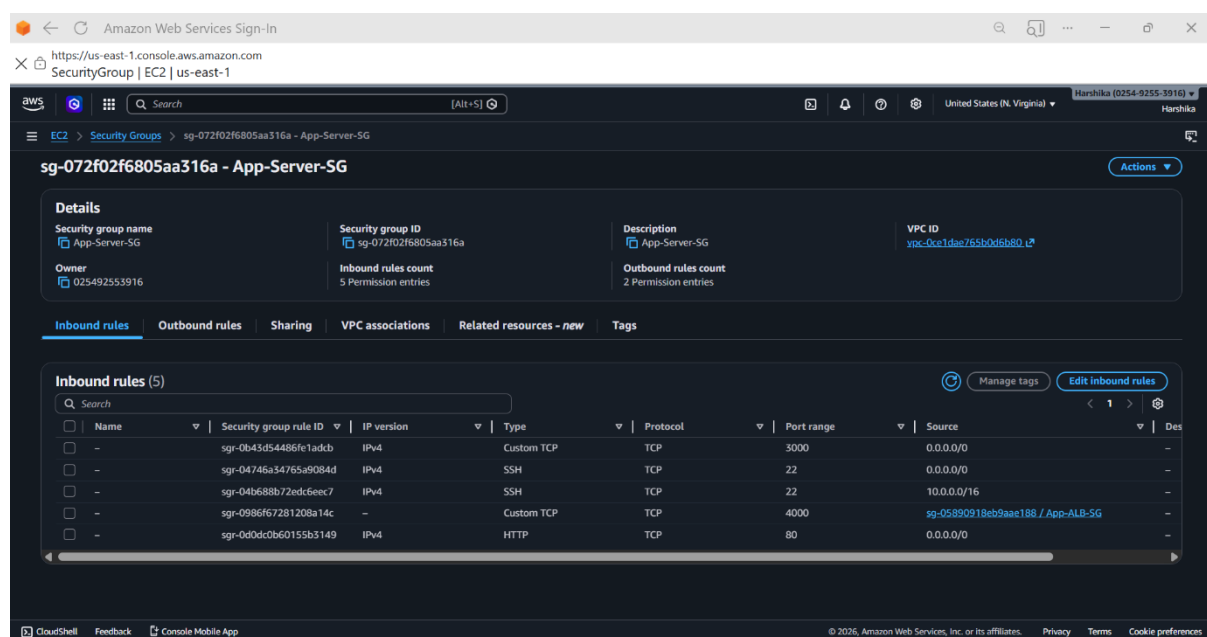
4. Network Mapping:

- **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).



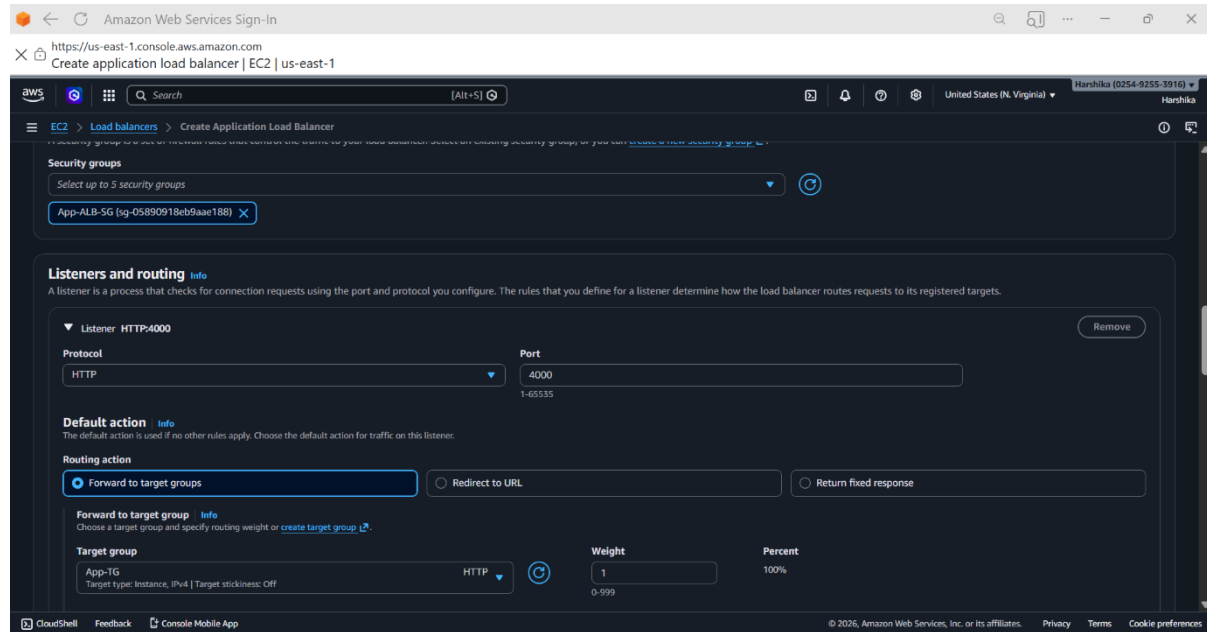
5. Security Groups:

- **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**.

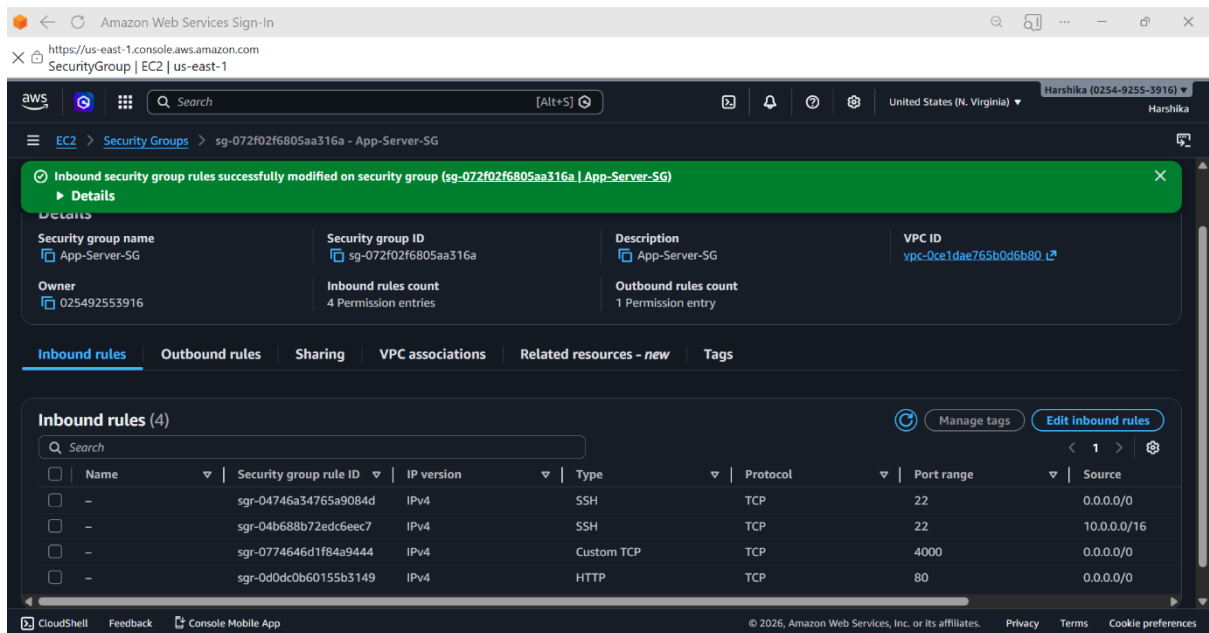


7. **Create:** Click Create and wait for the status to show **Active**.

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/
```

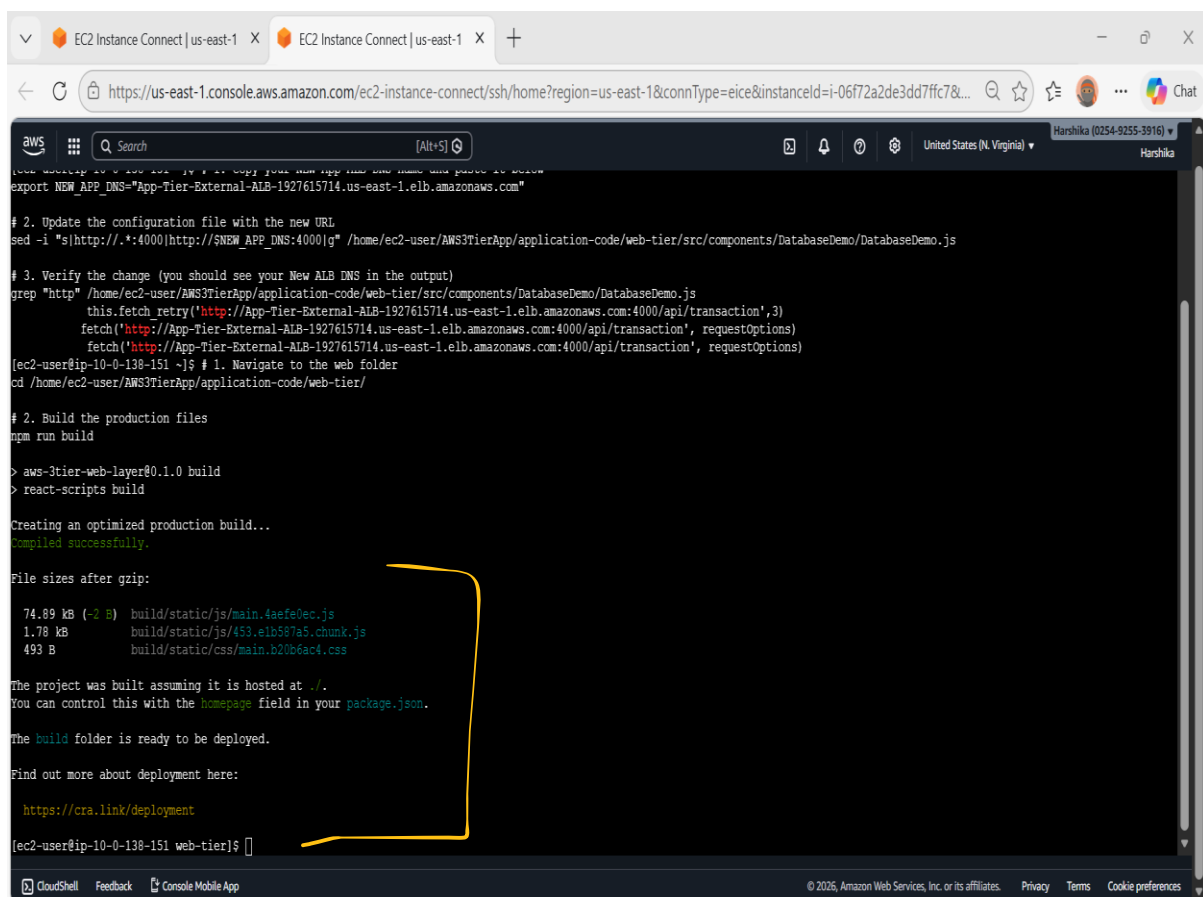
2. Build the production files

```
npm run build
```

Step 4: Start the Web Service

1. Start the web tier using PM2

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



The screenshot shows a terminal window within an AWS EC2 Instance Connect session. The terminal displays the following commands and output:

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

# 2. Update the configuration file with the new 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

# 3. Verify the change (you should see your New ALB DNS in the output)
grep "http" /home/ec2-user/AWS3TierApp/application-code/web-tier/src/components/DatabaseDemo/DatabaseDemo.js
this.fetch_retry('http://App-Tier-External-ALB-1927615714.us-east-1.elb.amazonaws.com:4000/api/transaction', 3)
  fetch('http://App-Tier-External-ALB-1927615714.us-east-1.elb.amazonaws.com:4000/api/transaction', requestOptions)
  fetch('http://App-Tier-External-ALB-1927615714.us-east-1.elb.amazonaws.com:4000/api/transaction', requestOptions)

[ec2-user@ip-10-0-138-151 ~]$ # 1. Navigate to the web folder
cd /home/ec2-user/AWS3TierApp/application-code/web-tier/

# 2. Build the production files
npm run build

> aws-3tier-web-layer@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

74.89 kB (-2 B) build/static/js/main.4aefe0ec.js
1.78 kB build/static/js/453.elb587a5.chunk.js
493 B build/static/css/main.b20b6ac4.css

The project was built assuming it is hosted at ./
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.

Find out more about deployment here:

https://cra.link/deployment

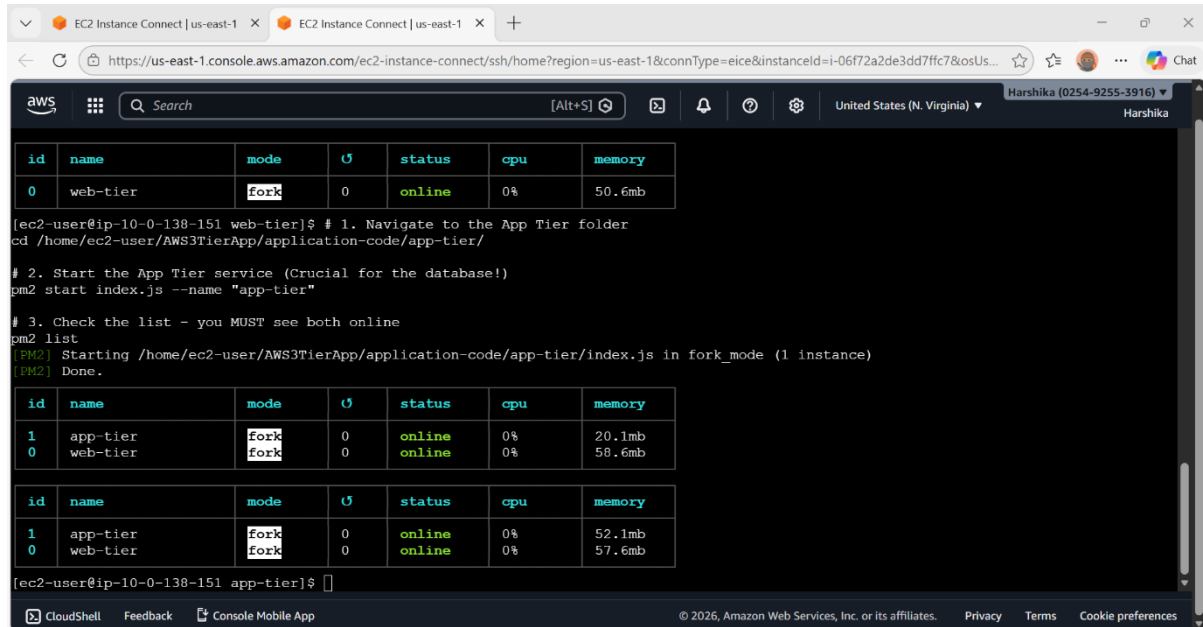
[ec2-user@ip-10-0-138-151 web-tier]$
```

A yellow bracket highlights the output of the `npm run build` command, specifically the file sizes and the message "The build folder is ready to be deployed."

2. Check the status

pm2 list

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



```
[ec2-user@ip-10-0-138-151 web-tier]$ # 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
[PM2] Starting /home/ec2-user/AWS3TierApp/application-code/app-tier/index.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	mode	♿	status	cpu	memory
0	web-tier	fork	0	online	0%	50.6mb

id	name	mode	♿	status	cpu	memory
1	app-tier	fork	0	online	0%	20.1mb
0	web-tier	fork	0	online	0%	58.6mb

id	name	mode	♿	status	cpu	memory
1	app-tier	fork	0	online	0%	52.1mb
0	web-tier	fork	0	online	0%	57.6mb

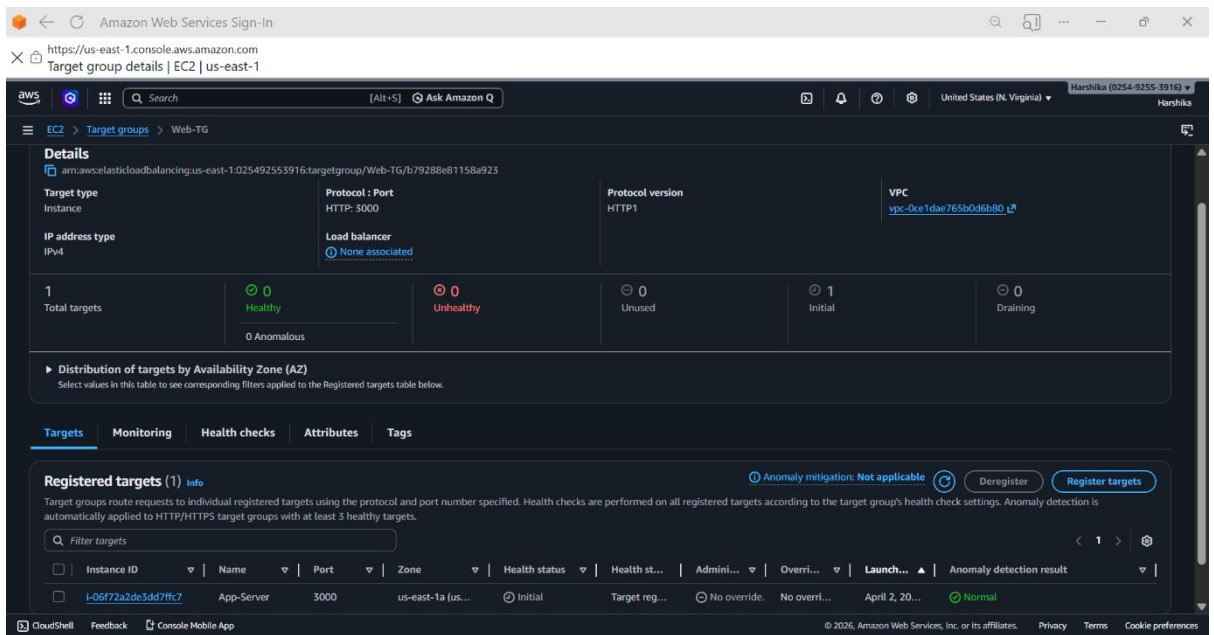
```
[ec2-user@ip-10-0-138-151 app-tier]$
```

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

1. Create a Web Target Group

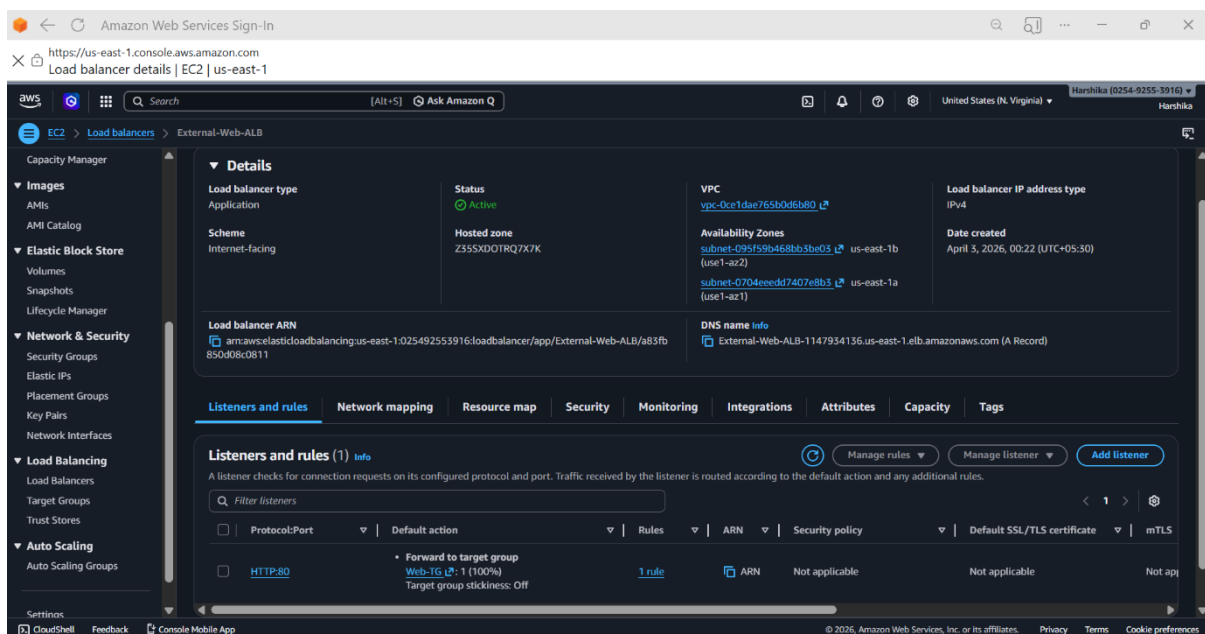
This tells the new Load Balancer to send traffic to your Web Server.

- **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.

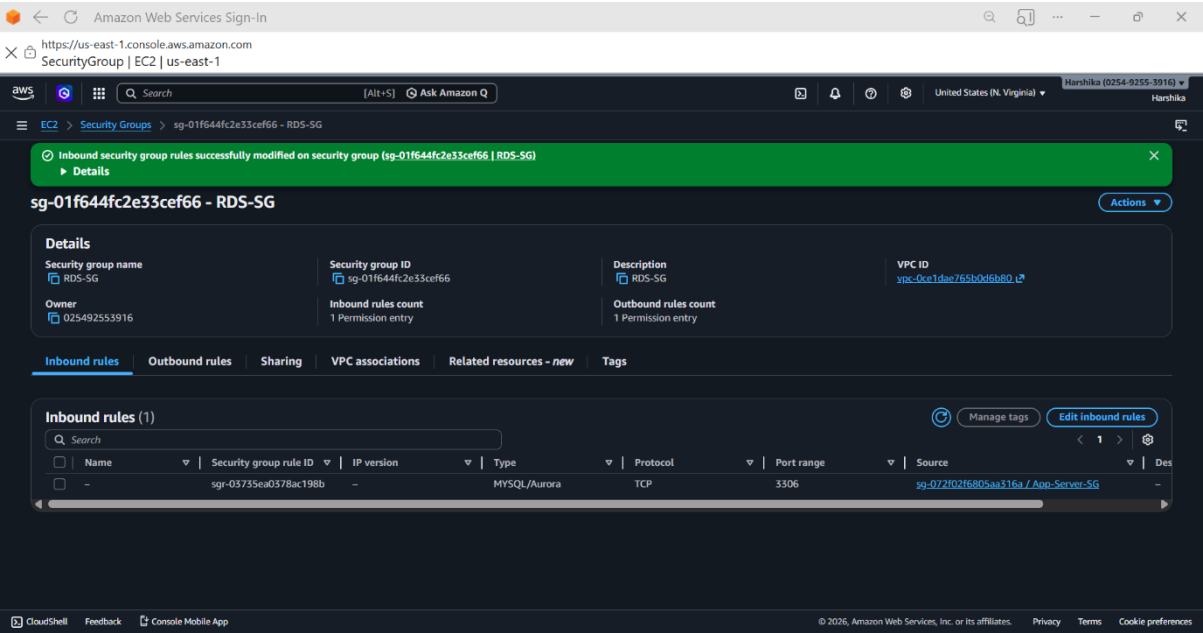


2. The Correct Security Group Setup

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

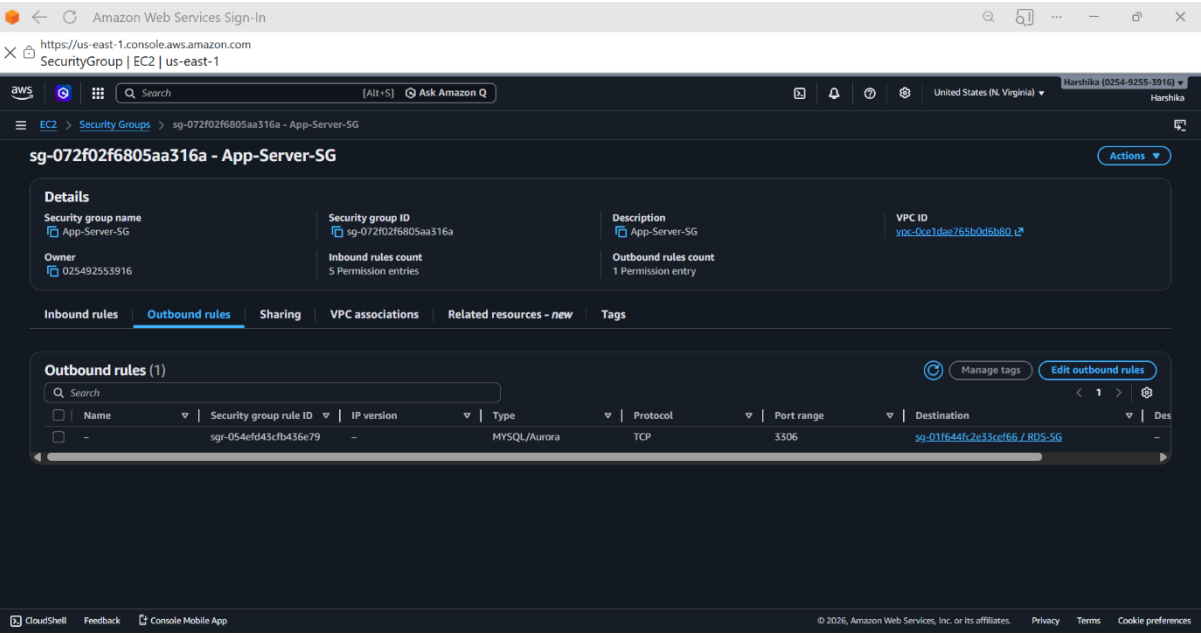
RDS-SG (Inbound Rules)

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



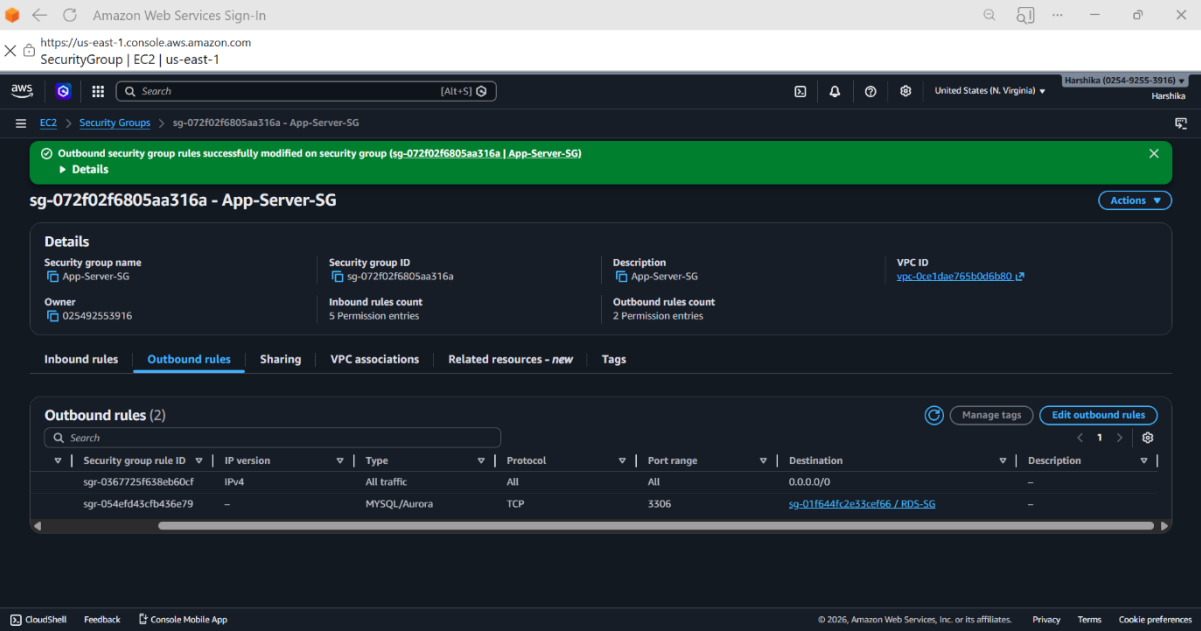
App-Server-SG (Inbound Rules)

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/
```

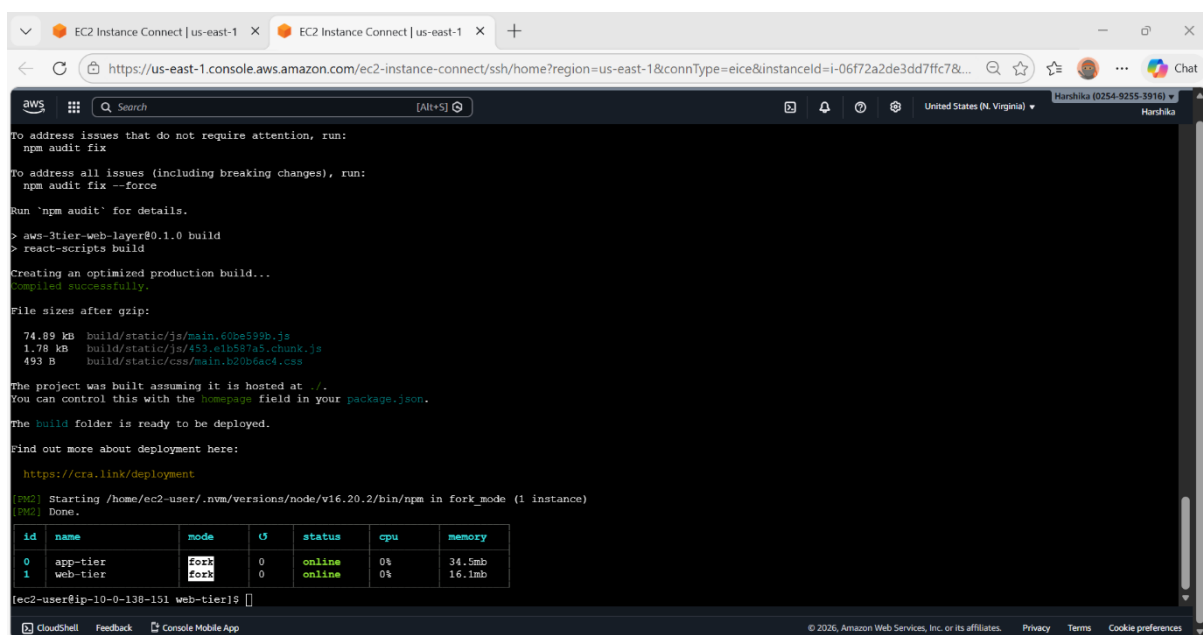
2. Build the production files

```
npm run build
```

3. Start the web tier using PM2

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

```
pm2 list
```



The screenshot shows the AWS CloudShell terminal interface. The terminal output displays the results of running `npm run build`, including file sizes after gzip and a confirmation that the build is ready for deployment. Below the output, the `pm2` command is used to start the application, and the `pm2 list` command is run to show the status of the processes. The `pm2 list` output shows two processes: `app-tier` and `web-tier`, both in the `fork` mode and `online` status.

```
To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run 'npm audit' for details.

> aws-3tier-web-layer@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

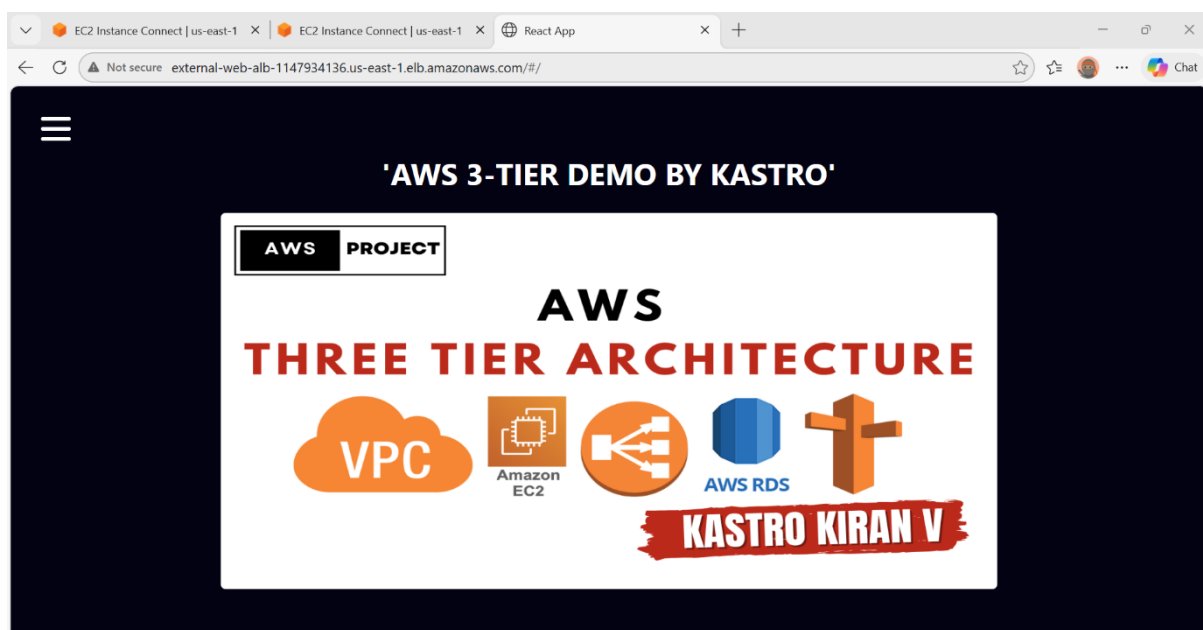
  74.89 kB  build/static/js/main.60be599b.js
  1.78 kB   build/static/js/453.elb587a5.chunk.js
  493 B     build/static/css/main.b20b6ec4.css

The project was built assuming it is hosted at ./.
You can control this with the homepage field in your package.json.
The build folder is ready to be deployed.
Find out more about deployment here:
  https://cra.link/deployment

[PM2] Starting /home/ec2-user/.nvm/versions/node/v16.20.2/bin/npm in fork_mode (1 instance)
[PM2] Done.

id  name    mode  u    status  cpu  memory
0   app-tier  fork  0    online  0%   34.5mb
1   web-tier  fork  0    online  0%   16.1mb

ec2-user@ip-10-0-138-151 web-tier$
```



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

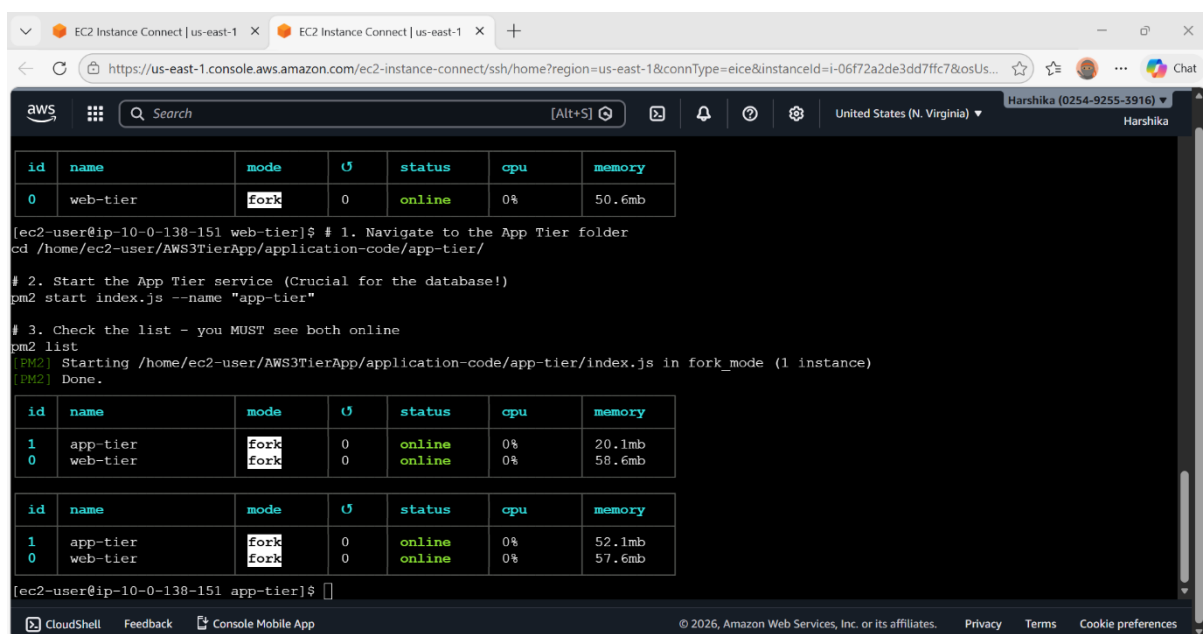
```
cd /home/ec2-user/AWS3TierApp/application-code/app-tier/
```

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
```



The screenshot shows a terminal window with the following content:

```
[ec2-user@ip-10-0-138-151 web-tier]$ # 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
[PM2] Starting /home/ec2-user/AWS3TierApp/application-code/app-tier/index.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	mode	U	status	cpu	memory
0	web-tier	fork	0	online	0%	50.6mb

id	name	mode	U	status	cpu	memory
1	app-tier	fork	0	online	0%	20.1mb
0	web-tier	fork	0	online	0%	58.6mb

id	name	mode	U	status	cpu	memory
1	app-tier	fork	0	online	0%	52.1mb
0	web-tier	fork	0	online	0%	57.6mb

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

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

aws

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.c6lgu8wgwg9t.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
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

aws

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.c6lgu8wgwg9t.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
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>

```
    }

    handleButtonClick() {
      console.log(this.state.text_amt);
      console.log(this.state.text_desc);
      const requestOptions = {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({"amount": this.state.text_amt, "desc" : this
.state.text_desc})
      }
      fetch('http://App-Tier-External-ALB-1927615714.us-east-1.elb.amazonaws
.com:4000/api/transaction', requestOptions)
        .then(response => response.json())
        .then(data => this.populateData())

      this.setState({text_amt : "", text_desc:""});
    }
  }
}
```

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

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

The screenshot shows an AWS EC2 Instance Connect session with the following terminal output:

```

Last login: Fri Apr 3 06:18:26 2026 from 10.0.128.106
[ec2-user@ip-10-0-138-151 ~]$ vi /home/ec2-user/AWS3TierApp/application-code/web-tier/src/components/DatabseDemo/DatabseDemo.js
[ec2-user@ip-10-0-138-151 ~]$ vi /home/ec2-user/AWS3TierApp/application-code/web-tier/src/components/DatabseDemo/DatabseDemo.js
[ec2-user@ip-10-0-138-151 ~]$ cd /home/ec2-user/AWS3TierApp/application-code/web-tier/
$ npm run build
$ npm restart web-tier

$ aws-tier-web-layer@0.1.0 build
$ react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

74.89 kB (+2 B)   build/static/js/main.dcc9fde.js
1.78 kb          build/static/js/453.elb587a5.chunk.js
493 B            build/static/css/main.b20bae4.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.

Find out more about deployment here:

https://cre.link/deployment

$ npm --update-env to update environment variables
$ npm Applying action restartProcessId on app [web-tier] (ids: [ 1 ])
PM2 [web-tier]:() V

id name mode status cpu memory
0 app-tier fork 0 online 0% 37.3mb
1 web-tier fork 0 online 0% 18.3mb
[ec2-user@ip-10-0-138-151 web-tier]$ 

```

- The screenshot shows the AWS Management Console interface for an EC2 Instance Connect session. The terminal window displays the following content:

```

AWS
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Fri Apr 3 07:40:31 2026 from 10.0.128.106
[ec2-user@ip-10-0-138-151 ~]$ pm2 list

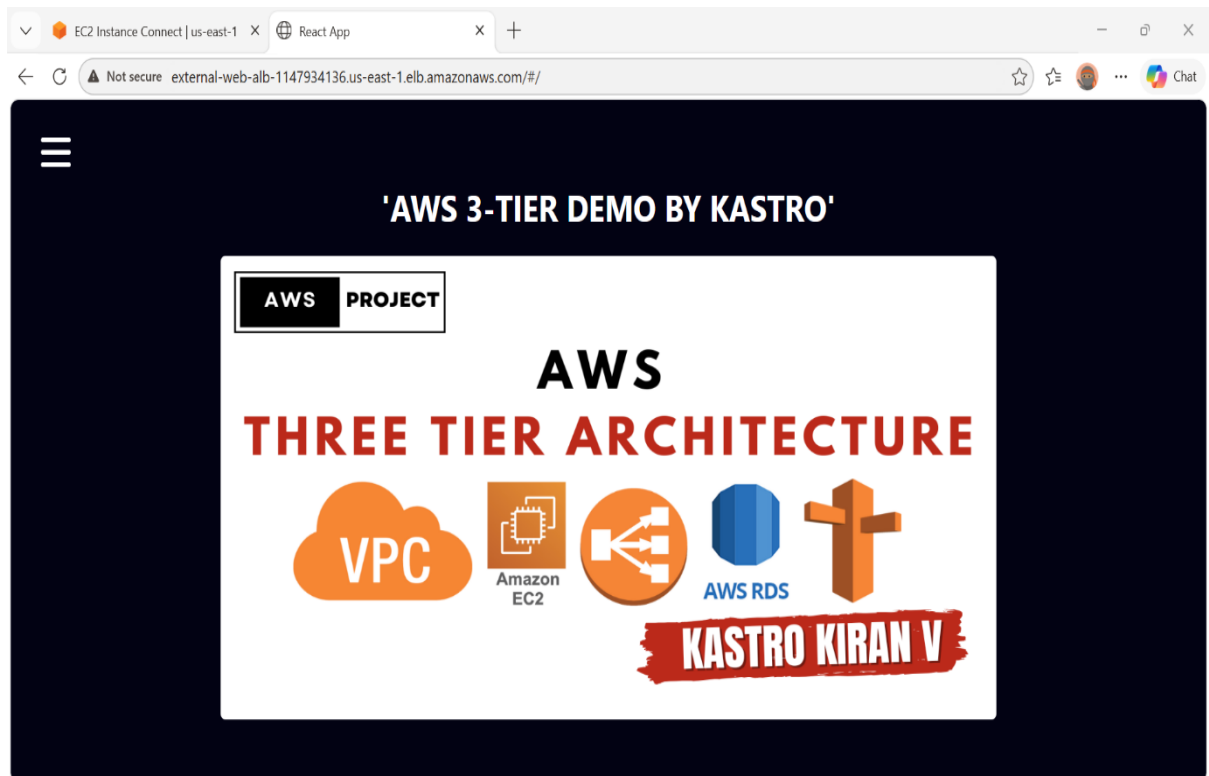
```

id	name	namespace	version	mode	pid	uptime	Q	status	cpu	mem	user	watching
0	app-tier	default	1.0.0	fork	77286	66m	0	online	0%	45.2mb	ec2-user	disabled
1	web-tier	default	N/A	fork	79475	8m	1	online	0%	58.1mb	ec2-user	disabled

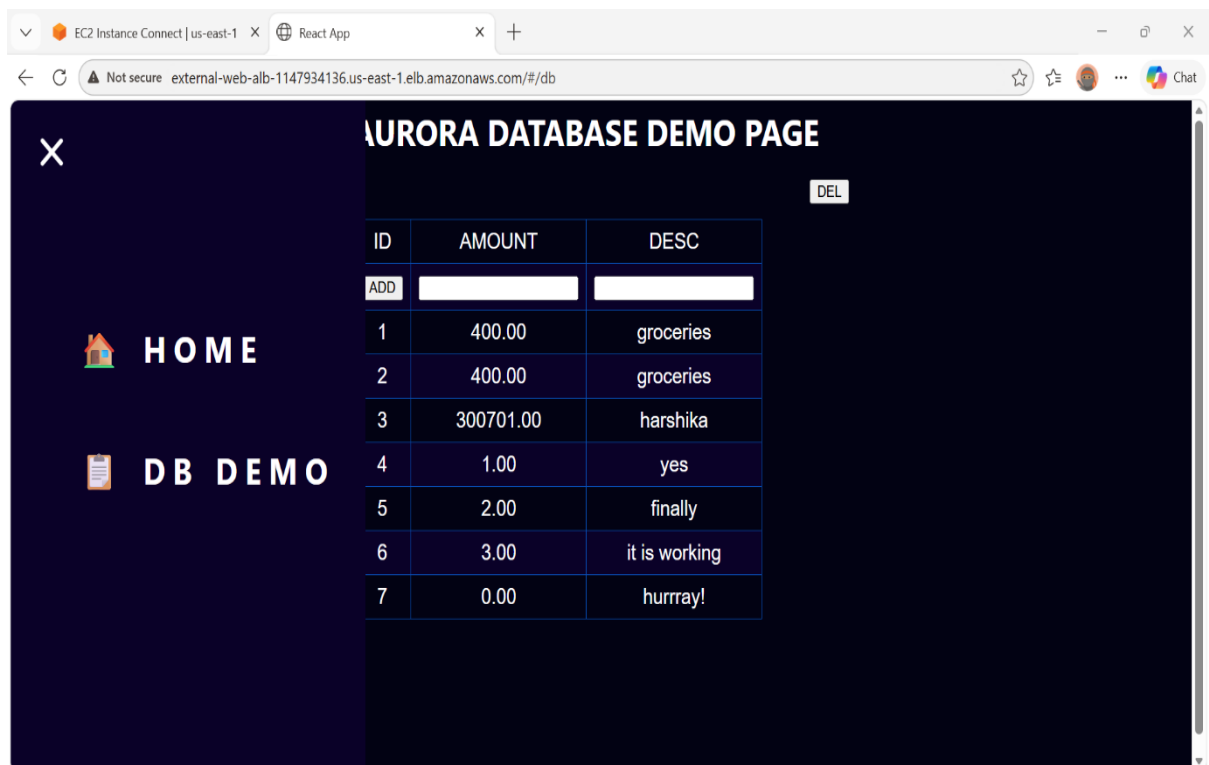
Below the terminal output, a summary card for the instance **i-06f72a2de3dd7ffc7 (App-Server)** is visible, showing the Private IP as 10.0.138.151.

- Copy the URL and paste it into any browser.

Successful!



Saving the entry in the table.



Saved entries!

EC2 Instance Connect | us-east-1

React App

external-web-alb-1147934136.us-east-1.elb.amazonaws.com/#/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	hurray!

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

Domain name search – check and | X

https://www.hostinger.com/in/domain-name-results?from=...

Don't miss the limited-time deals!

HOSTINGER

Domain search AI domain generator

harshika-devops.online

EXACT MATCH

harshika-devops.online **SAVE 97%**

For first year

₹89.00/1st yr ~~₹3,139.00~~

Make it yours **Ask Kodee**

Cart | Hostinger

https://cart.hostinger.com/pay/1563aec8-6956-46fd-88ce-53965288...

HOSTINGER

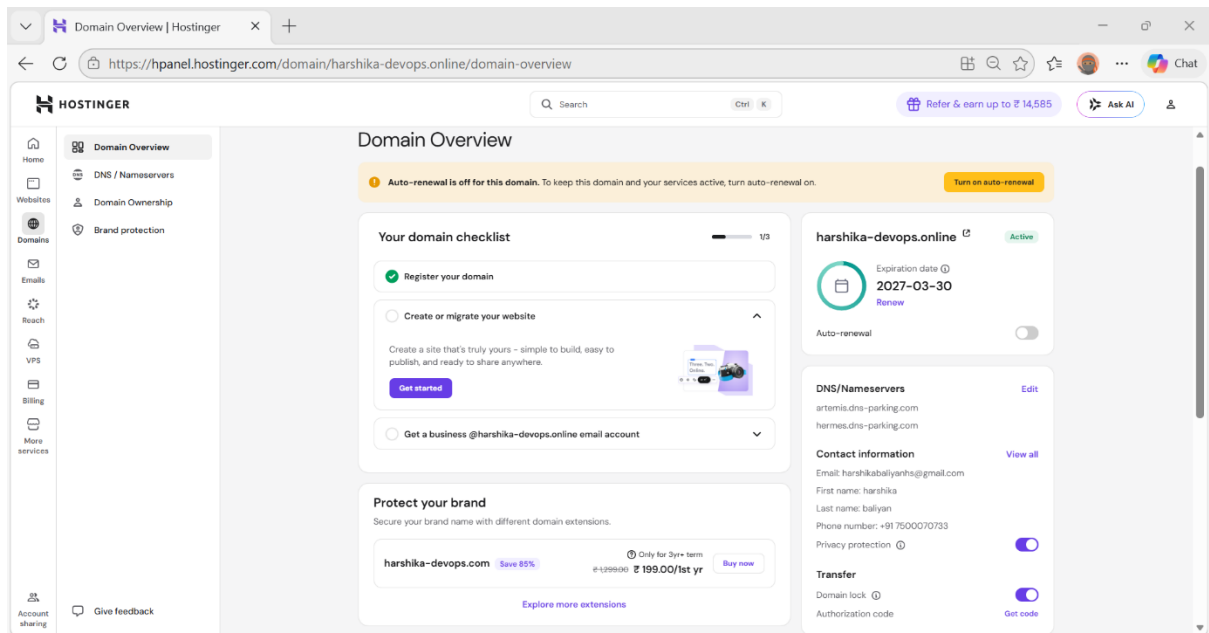
Your cart

Order summary X

harshika-devops.online	
Domain registration – 1 year	₹3,139.00 ₹80.10
ICANN fee	₹17.44
Domain privacy protection ?	₹0.00
Taxes ? ₹17.56	
WPIODOMAIN -10%	-₹8.90
Total	₹3,174.00 ₹115.10

Coupon applied: **WPIODOMAIN** X

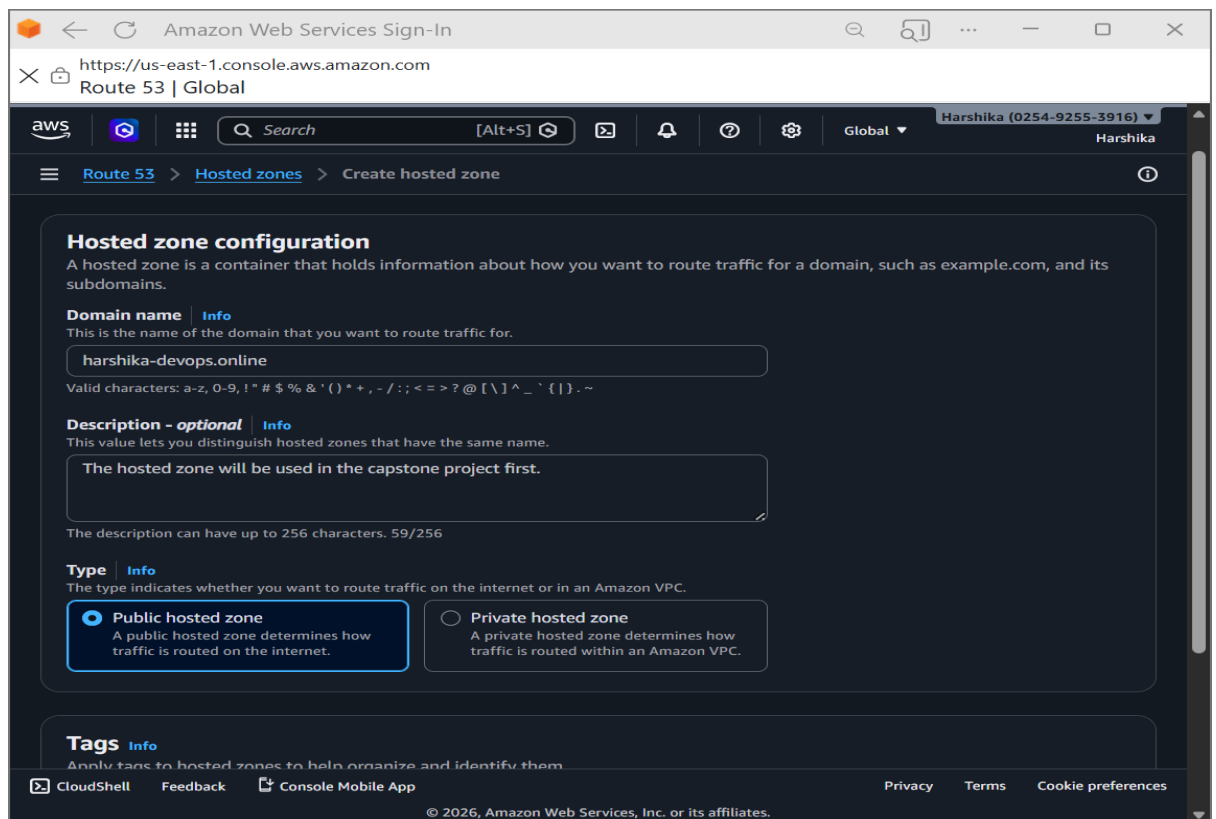
Continue



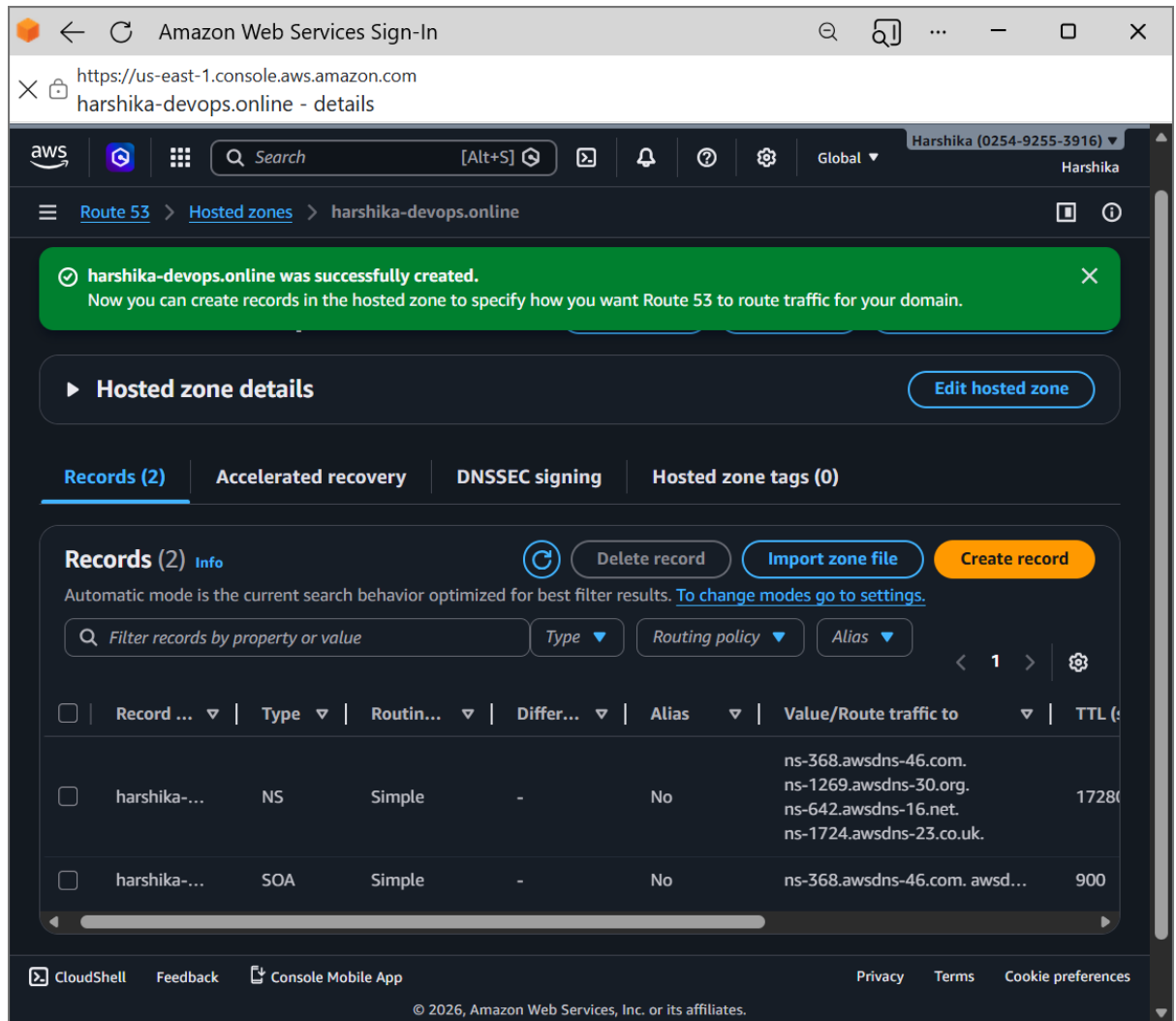
Step 1: Create a Hosted Zone

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...

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...

Refer & earn up to ₹ 14,585

Ask AI

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.

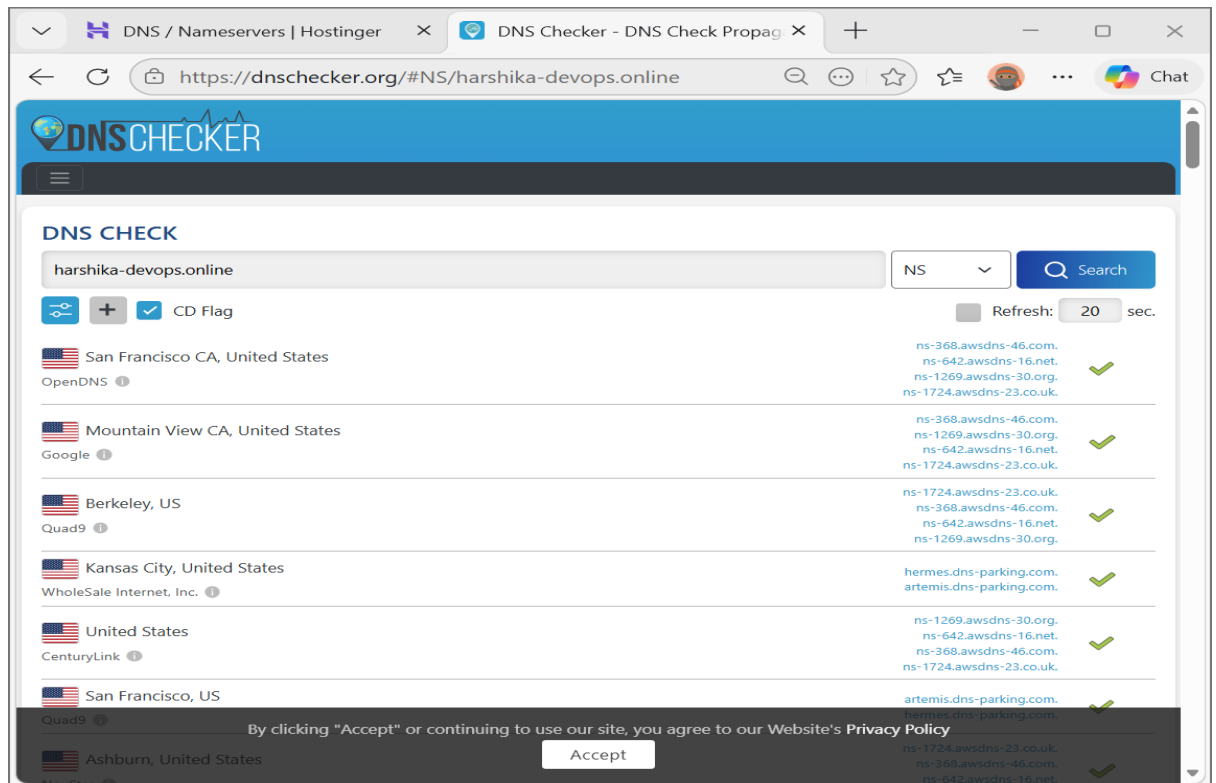
Close

Change Nameservers

Manage DNS

These records determine how your domain is configured for email, web servers or other services.

Your domain is currently using Hostinger's default nameservers. To switch your domain to custom nameservers, click the button below.



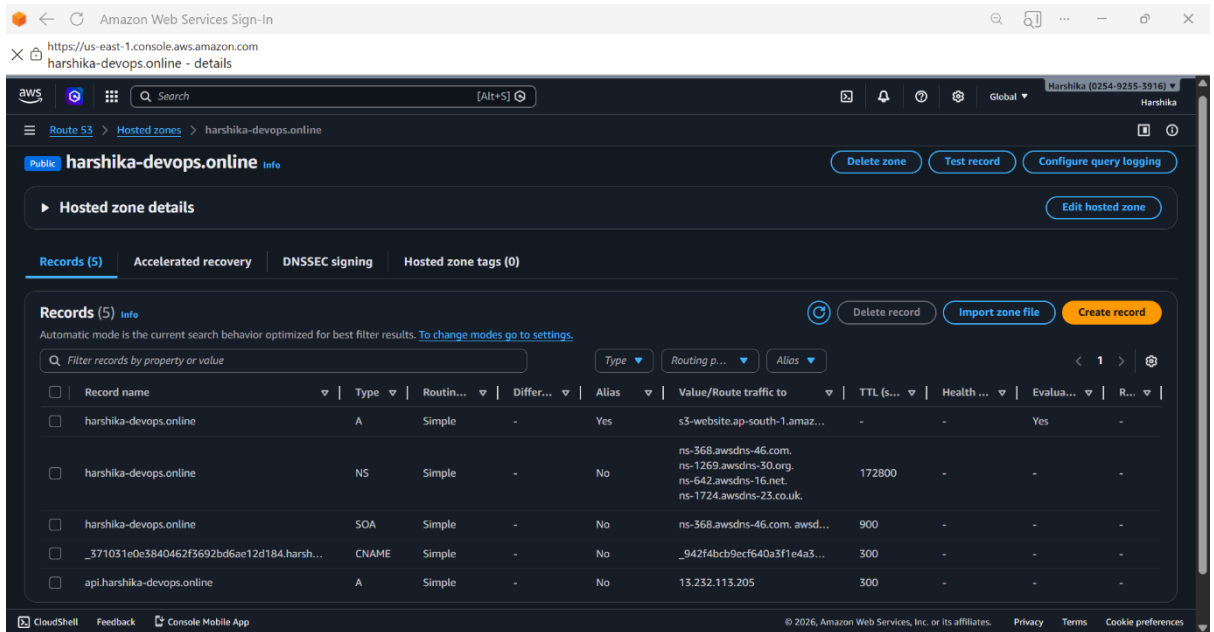
4. Change the "Default" name servers to the **Custom** AWS ones you just copied.

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**.

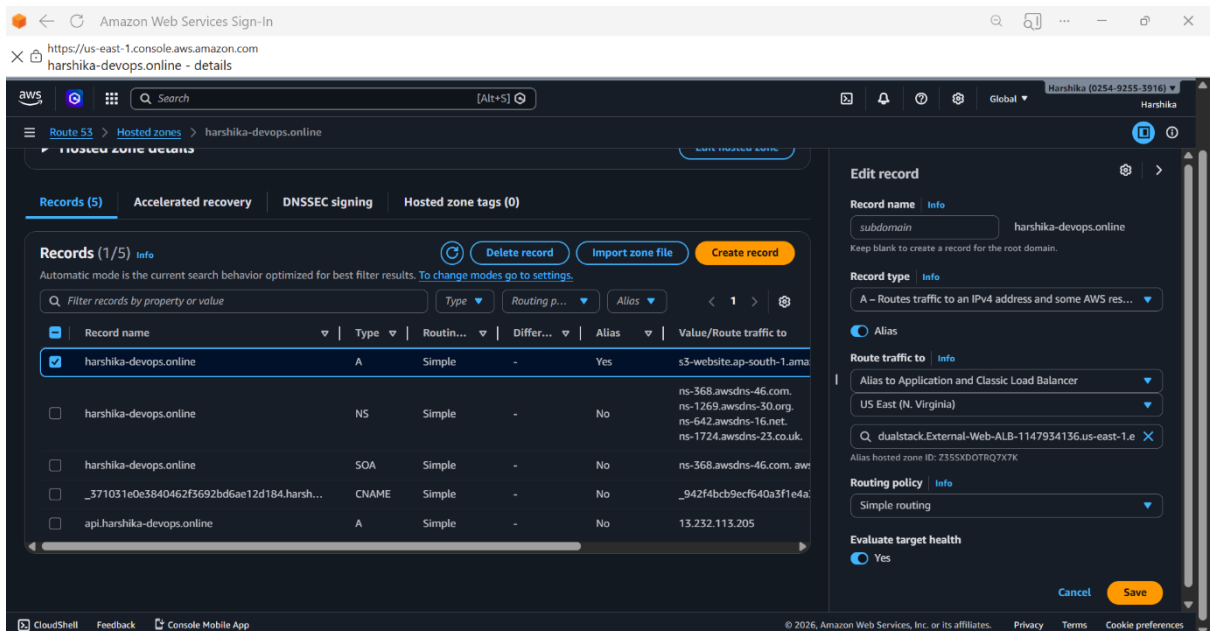
1. Inside your Route 53 Hosted Zone, click **Create record**.
2. **Record name:** Leave it blank (for the root domain) or type `www`.
3. **Record type:** A - Routes traffic to an IPv4 address and some AWS resources.
4. Toggle the **Alias** switch to **ON**.

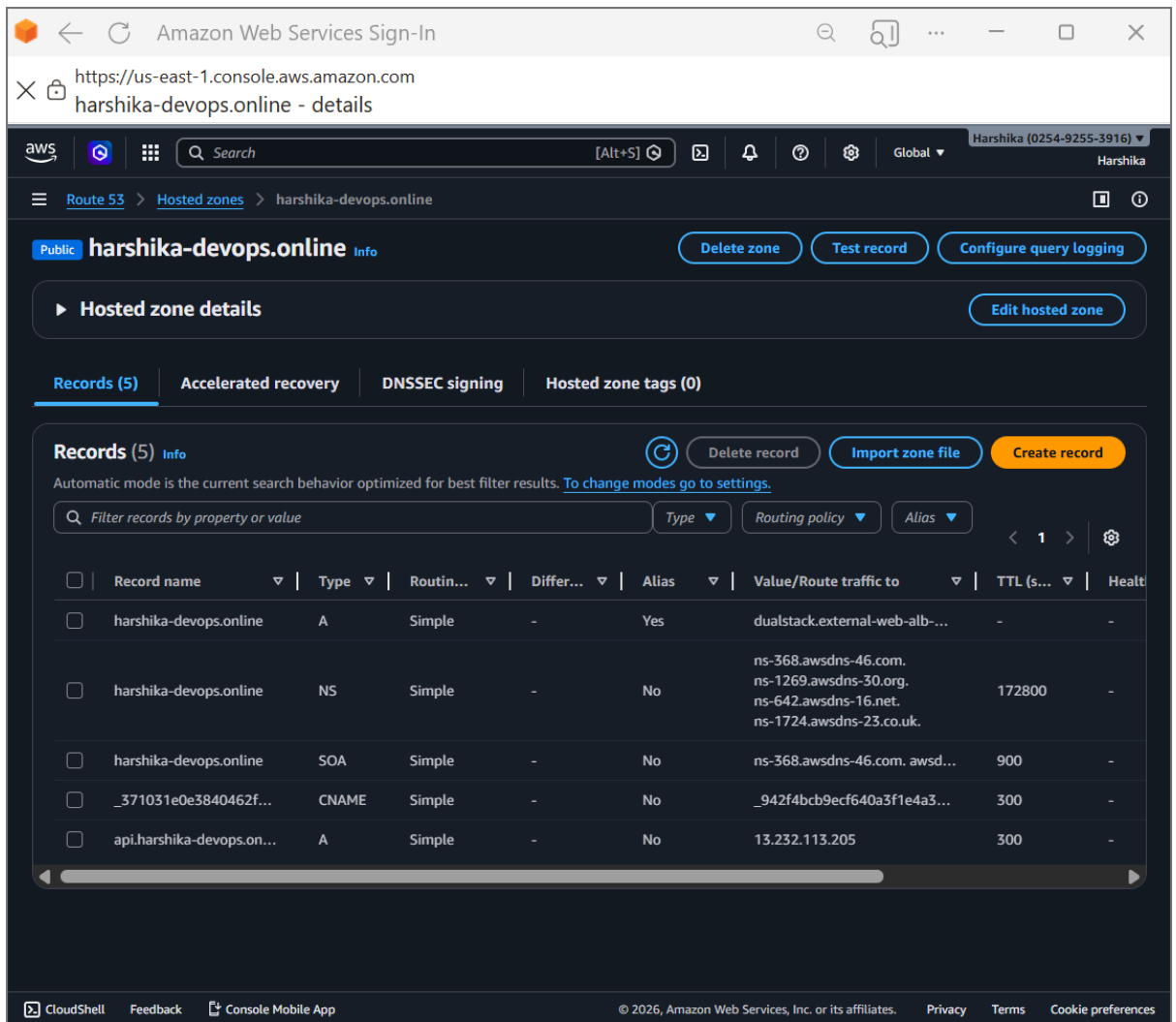


5. Route traffic to:

- 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**.



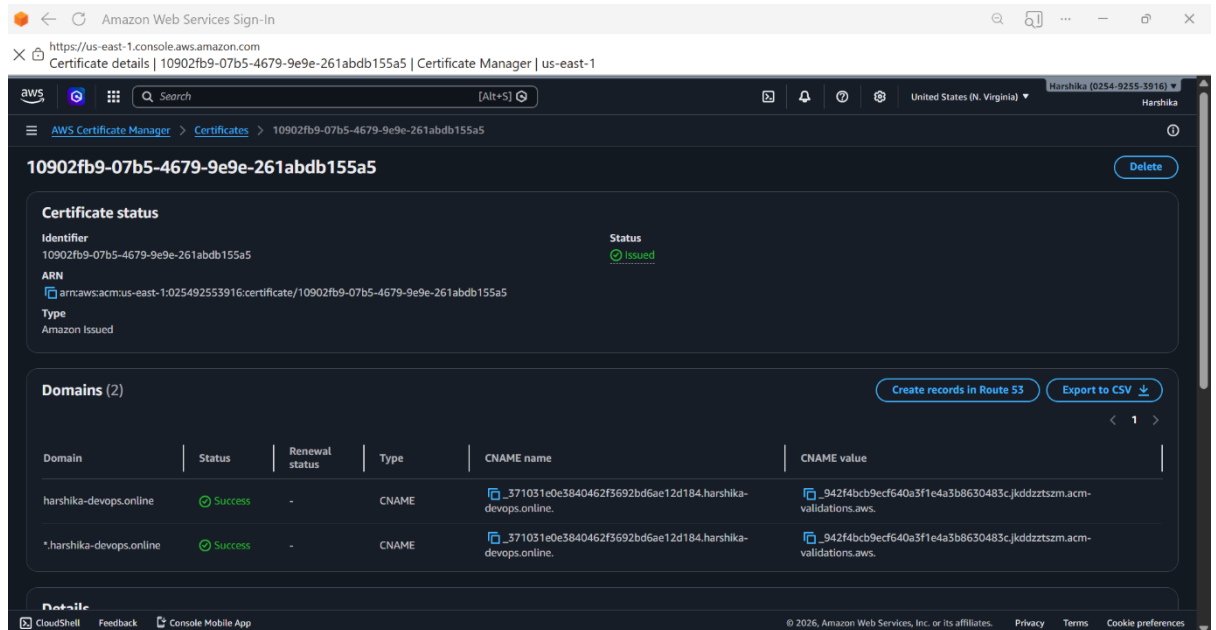


Step 4: Secure your Domain with SSL (ACM)

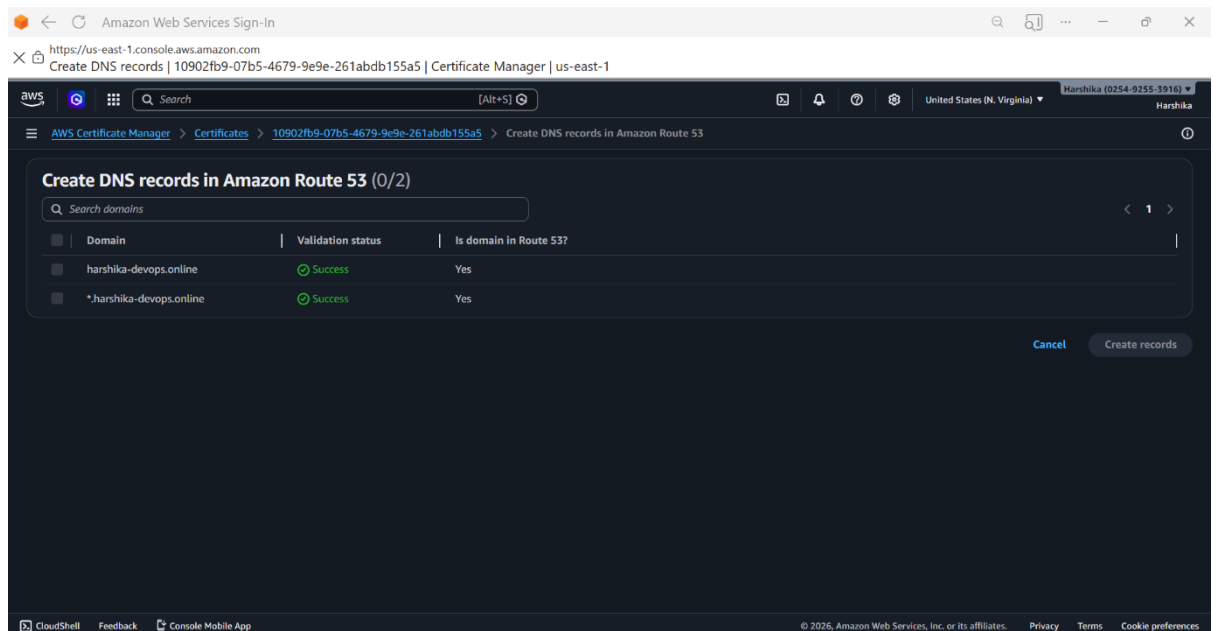
Providing an `https://` connection is a key requirement for modern DevOps projects.

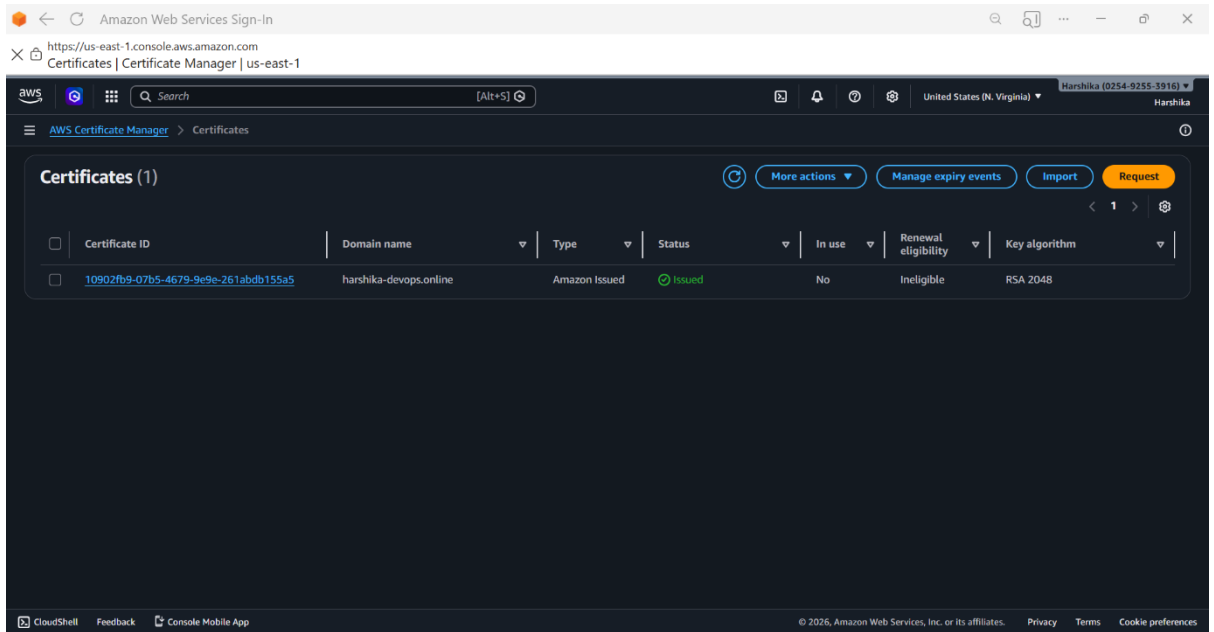
1. Go to **AWS Certificate Manager (ACM)**.
2. Click **Request a certificate** -> **Request a public certificate**.
3. **Fully qualified domain name:** Enter `harshika-devops.online` and add another for `*.harshika-devops.online`.

- 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.



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

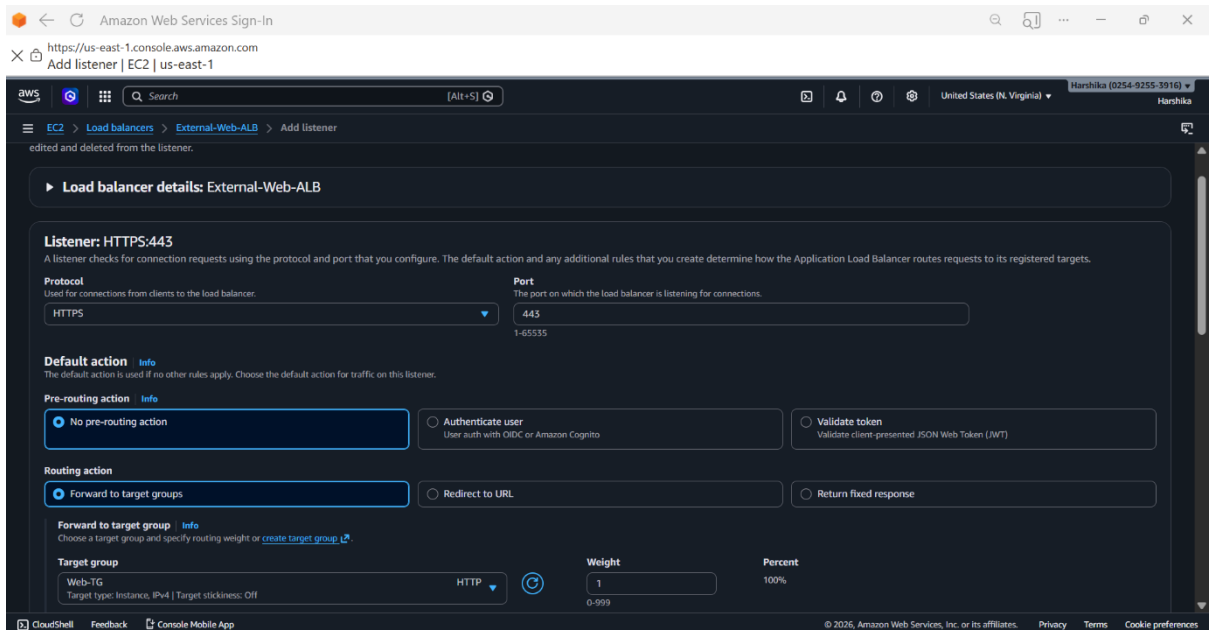




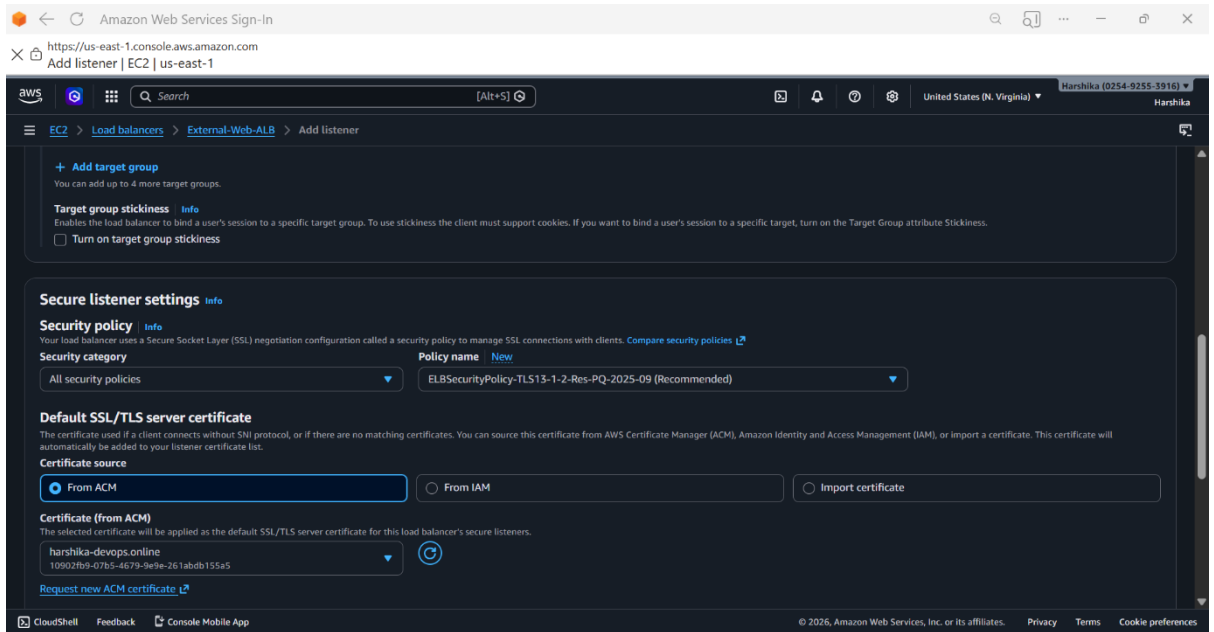
Step 5: Update the Load Balancer Listener

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

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



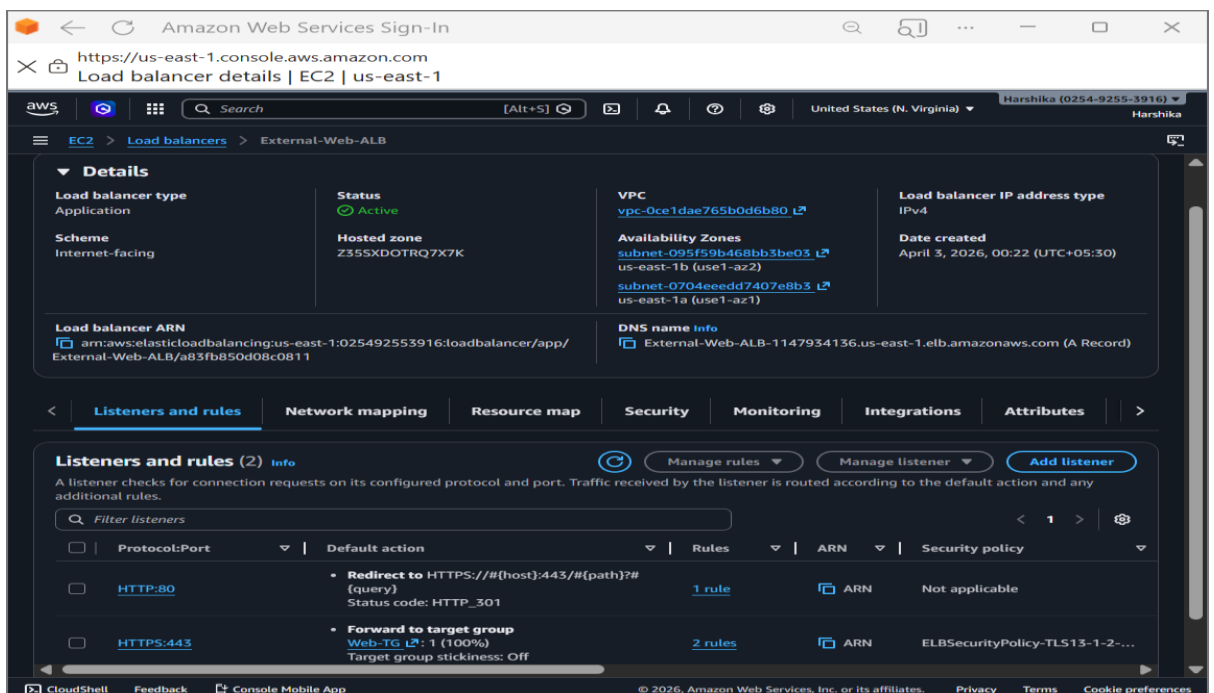
2. Go to the **Listeners** tab.



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.

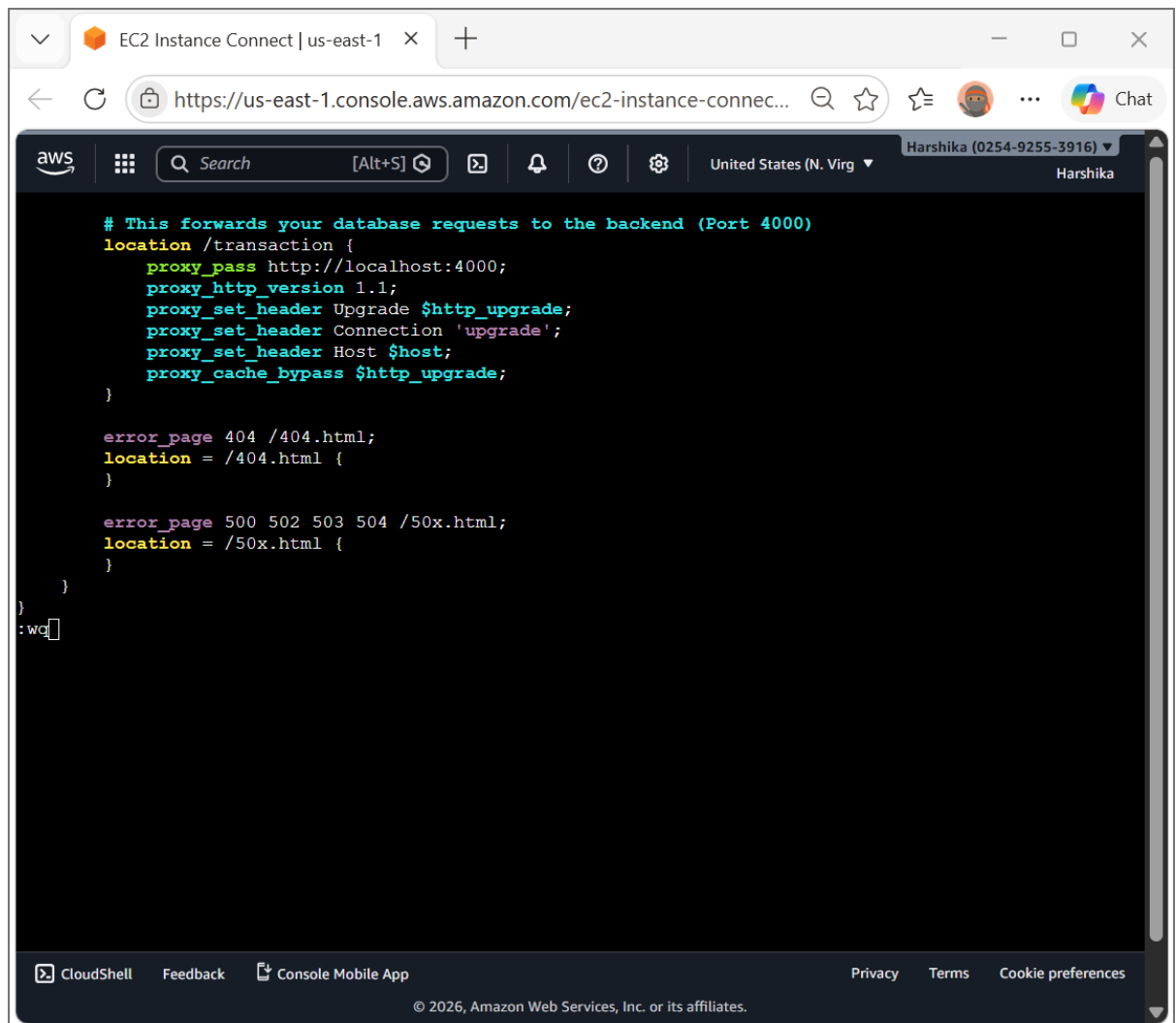
4. Edit the **HTTP:80** listener to **redirect to HTTPS:443**.



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`



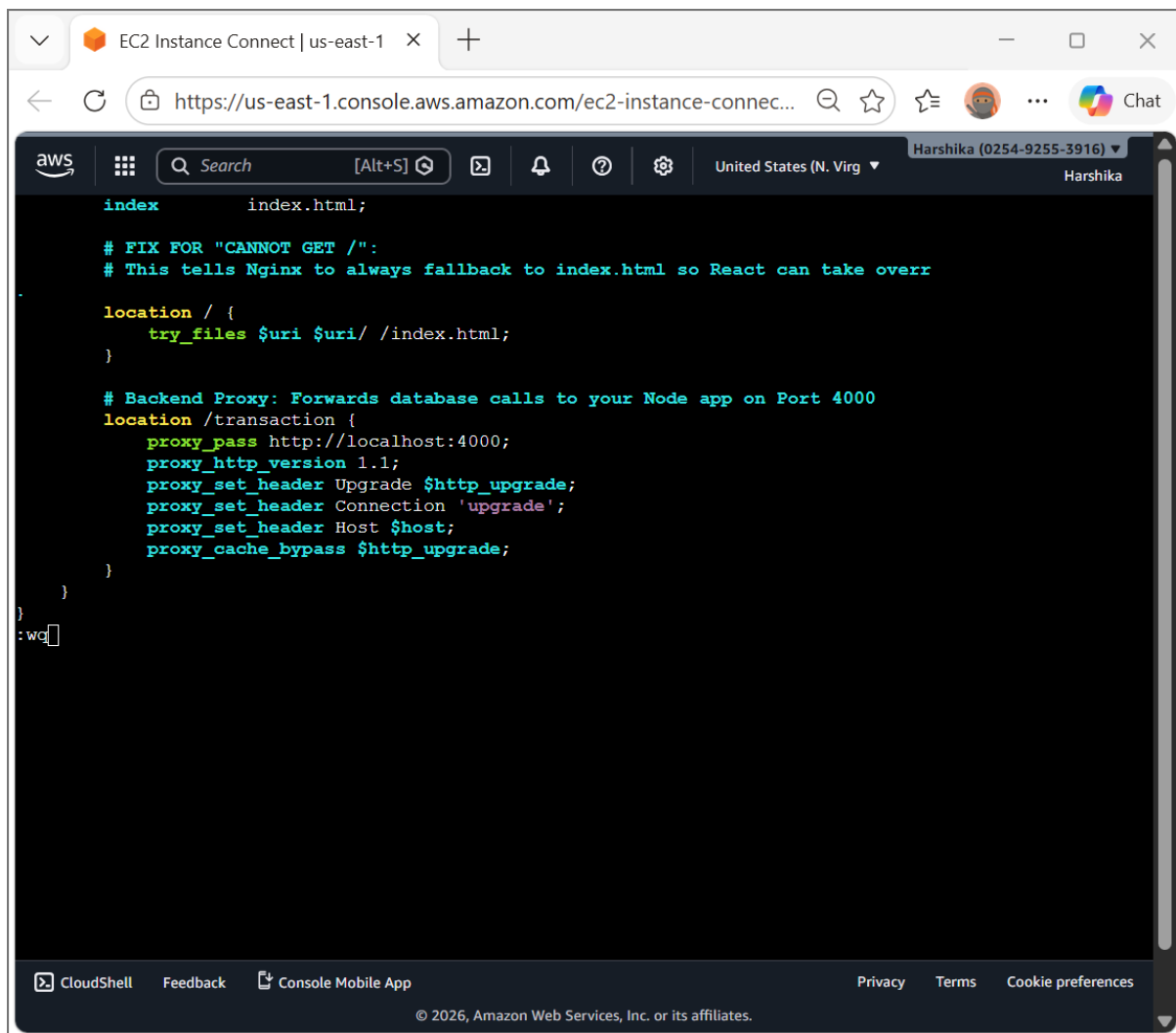
The screenshot shows the AWS CloudShell interface within a web browser. The browser's address bar displays the URL `https://us-east-1.console.aws.amazon.com/ec2-instance-connec...`. The CloudShell terminal window has a dark background and shows the following code being edited:

```
# This forwards your database requests to the backend (Port 4000)
location /transaction {
    proxy_pass http://localhost:4000;
    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;
}

error_page 404 /404.html;
location = /404.html {
}

error_page 500 502 503 504 /50x.html;
location = /50x.html {
}
}
```

The cursor is positioned at the end of the last closing brace. The CloudShell interface includes a search bar, navigation icons, and a footer with links for CloudShell, Feedback, Console Mobile App, Privacy, Terms, and Cookie preferences. The copyright notice at the bottom reads "© 2026, Amazon Web Services, Inc. or its affiliates."



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;
```

```
}
```

```
}
```

```
}
```

The screenshot shows the AWS Management Console interface for an EC2 Instance Connect session. The browser address bar shows the URL: `https://us-east-1.console.aws.amazon.com/ec2-instance-connect/ss...`. The console header includes the AWS logo, navigation icons, and the region "United States (N. Virginia)". The user's name "Harshika" and a phone number "(0254-9255-3916)" are displayed in the top right.

The main area is a CloudShell terminal window with a dark background. It contains the following Nginx configuration code:

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

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

location / {
    try_files $uri $uri/ /index.html;
}

location /transaction {
    proxy_pass http://localhost:4000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
}
}
```

Below the terminal, a notification for instance **i-06f72a2de3dd7ffc7 (App-Server)** is shown, indicating private IP addresses: `PrivateIPs: 10.0.138.151`.

The footer of the console includes links for "CloudShell", "Feedback", "Console Mobile App", "Privacy", "Terms", and "Cookie preferences", along with the copyright notice: "© 2026, Amazon Web Services, Inc. or its affiliates."

Bash

```
# Allow Nginx to connect to the backend port
```

```
sudo setsebool -P httpd_can_network_connect 1
```

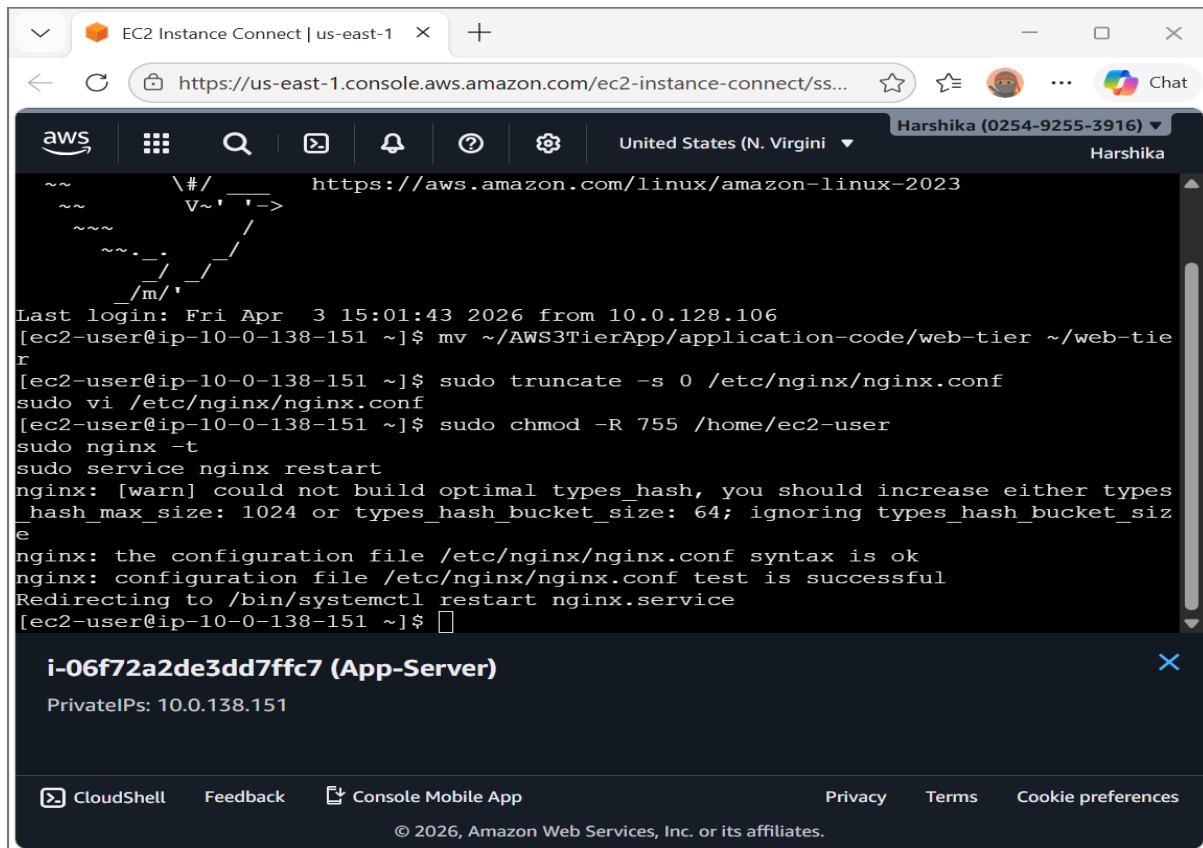
```
# Allow Nginx to read files in the home directory
```

```
sudo setsebool -P httpd_enable_homedirs 1
```

```
sudo chcon -Rt httpd_sys_content_t /home/ec2-user/web-tier/build
```

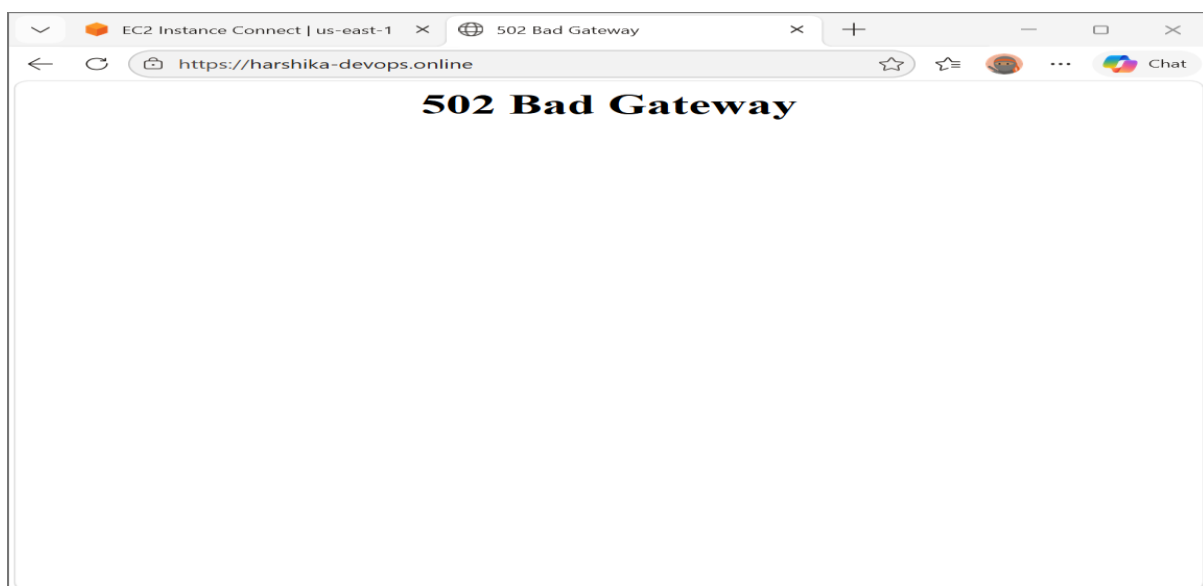
```
# Restart to apply
```

```
sudo service nginx restart
```



Final Check

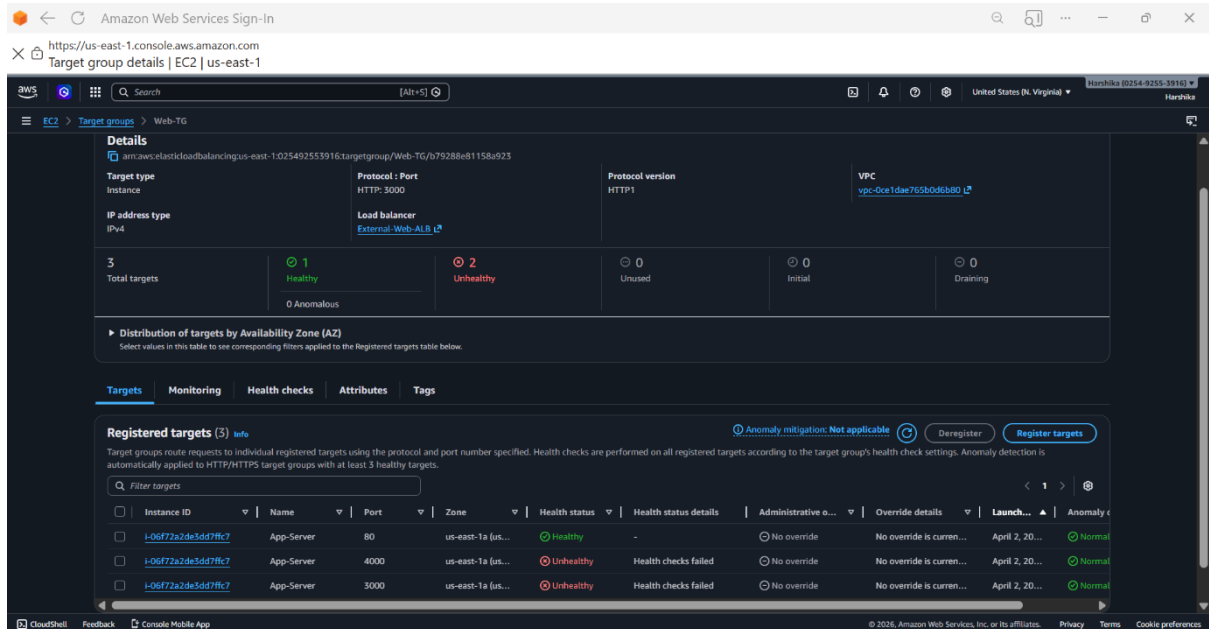
1. Close all your browser tabs.
2. Open a **New Incognito Window/ new tab**.
3. Go to <https://harshika-devops.online>



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:



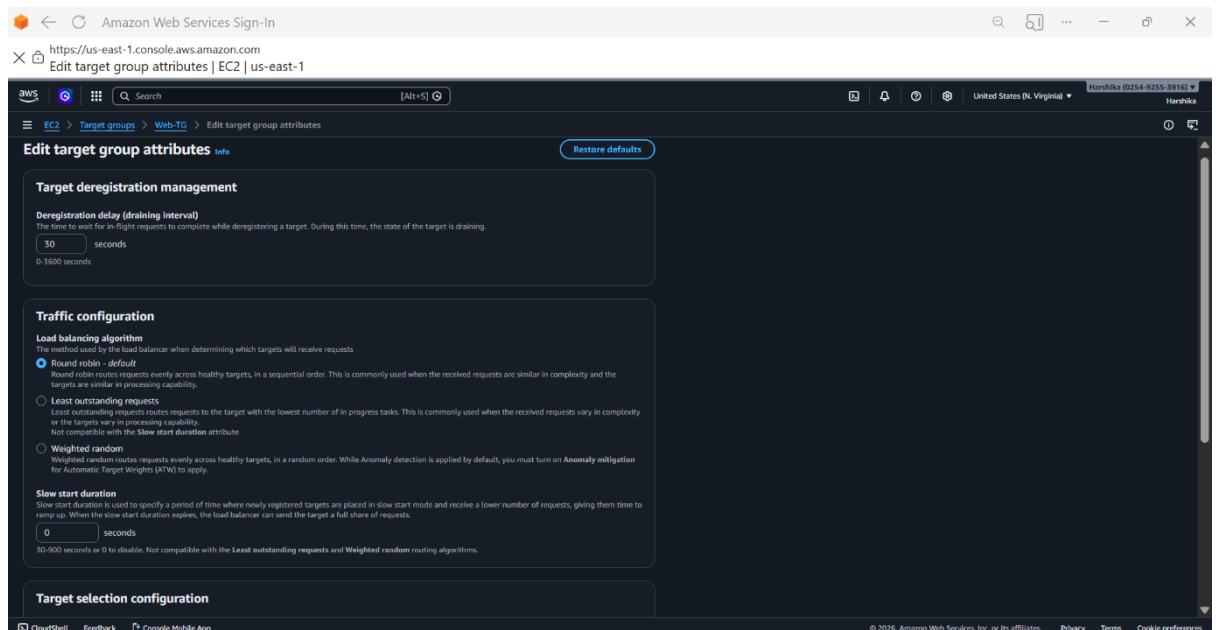
The screenshot shows the AWS Management Console for the 'Web-TG' target group. The 'Details' tab is selected, showing the target group's configuration. The 'Health status' section indicates that the target group is 'Unhealthy' with 2 unhealthy targets and 1 healthy target. The 'Distribution of targets by Availability Zone (AZ)' section shows that the targets are distributed across three AZs. The 'Targets' tab is also visible, showing a table of registered targets. The table has columns for Instance ID, Name, Port, Zone, Health status, Health status details, Administrative actions, Override details, Launch time, and Anomaly status. The targets on ports 4000 and 3000 are marked as 'Unhealthy' with the reason 'Health checks failed'.

Instance ID	Name	Port	Zone	Health status	Health status details	Administrative actions	Override details	Launch time	Anomaly status
i-06f72a2de3dd7ffc7	App-Server	80	us-east-1a (us-east-1)	Healthy	-	No override	No override is currently in effect	April 2, 2024	Normal
i-06f72a2de3dd7ffc7	App-Server	4000	us-east-1a (us-east-1)	Unhealthy	Health checks failed	No override	No override is currently in effect	April 2, 2024	Normal
i-06f72a2de3dd7ffc7	App-Server	3000	us-east-1a (us-east-1)	Unhealthy	Health checks failed	No override	No override is currently in effect	April 2, 2024	Normal

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.



The screenshot shows the 'Edit target group attributes' page for the 'Web-TG' target group. The page has a 'Restore defaults' button and three main sections: 'Target deregistration management', 'Traffic configuration', and 'Target selection configuration'. The 'Target deregistration management' section shows the 'Deregistration delay (draining interval)' set to 30 seconds. The 'Traffic configuration' section shows the 'Load balancing algorithm' set to 'Round robin - default'. The 'Target selection configuration' section is partially visible at the bottom.

- 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**.

The screenshot shows the AWS Management Console interface for editing health check settings. The breadcrumb trail is: EC2 > Target groups > Web-TG > Edit health check settings. The page title is 'Edit health check settings'.

Health checks
The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.

Health check protocol
HTTP

Health check path
Use the default path of "/" to perform health checks on the root, or specify a custom path if preferred.
/
Up to 1024 characters allowed.

Advanced health check settings

Health check port
The port the load balancer uses when performing health checks on targets. By default, the health check port is the same as the target group's traffic port. However, you can specify a different port as an override.
☐ Traffic port
☒ Override
 3000
 1-65535

Healthy threshold
The number of consecutive health checks successes required before considering an unhealthy target healthy.
 5
 2-10

Unhealthy threshold
The number of consecutive health check failures required before considering a target unhealthy.
 3
 1-10

At the bottom right of the settings section is a 'Restore defaults' button.

The footer of the console includes: CloudShell, Feedback, Console Mobile App, © 2026, Amazon Web Services, Inc. or its affiliates, Privacy, Terms, and Cookie preferences.

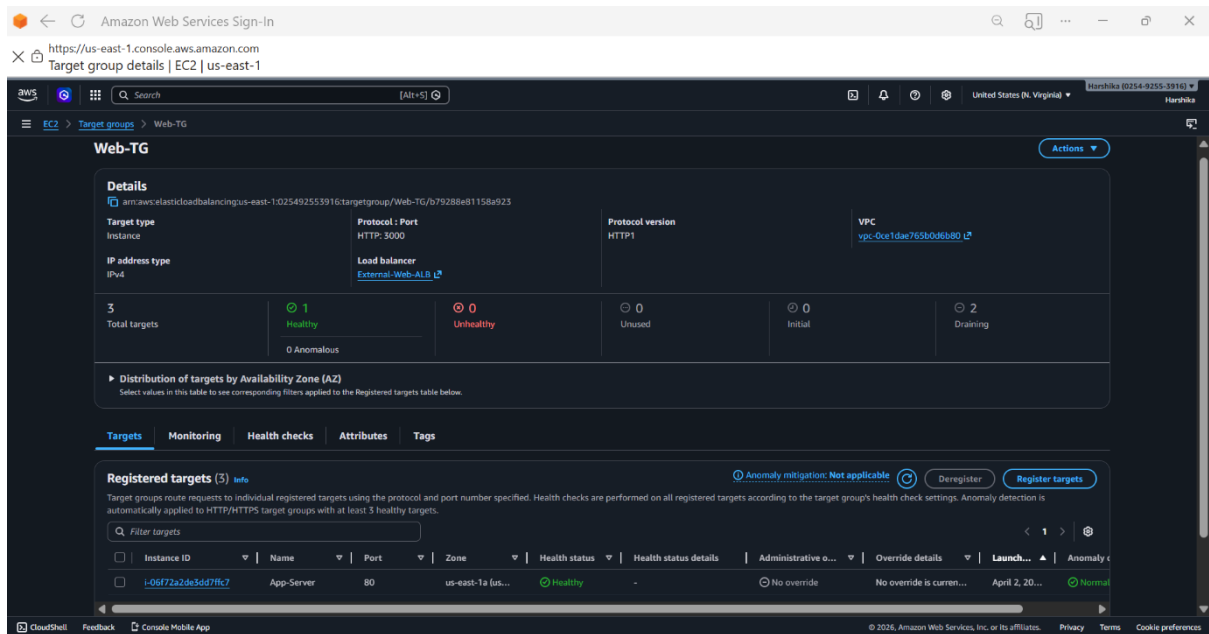
- Ensure the **Health check port** is set to the **Traffic port**.
- **Save changes**.

- 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.

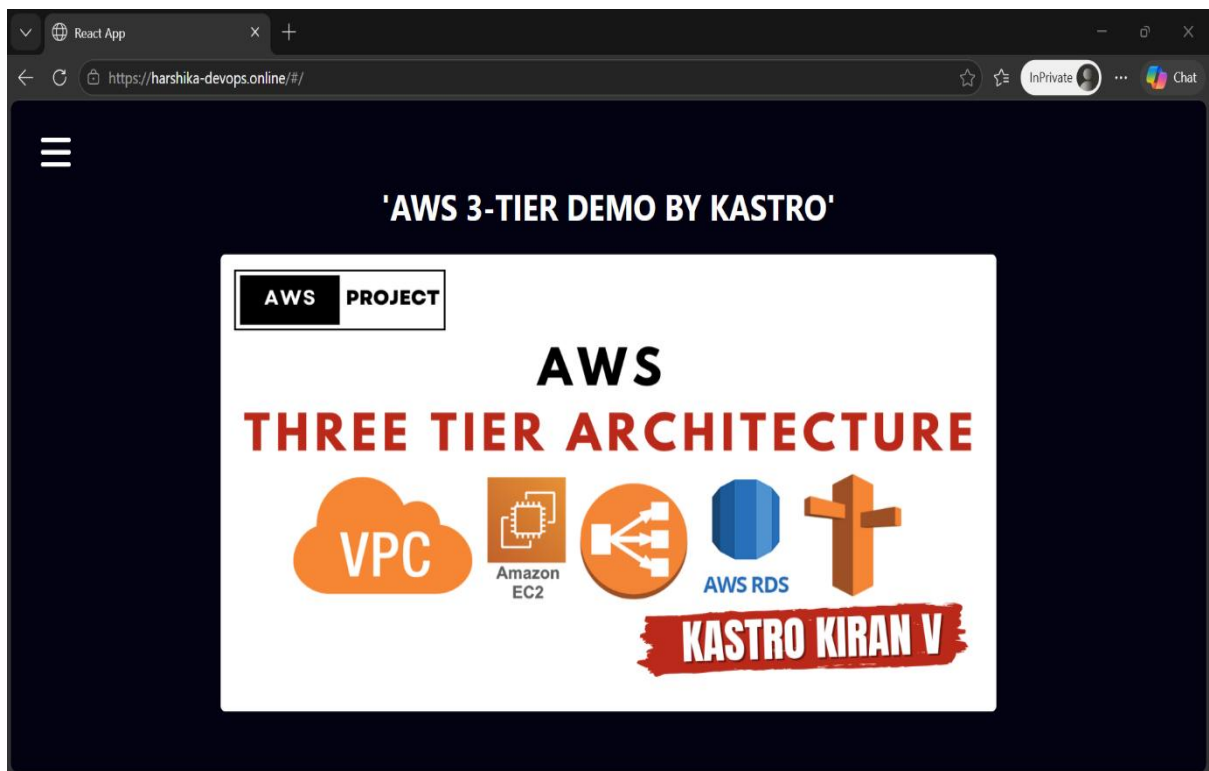


The Final Test

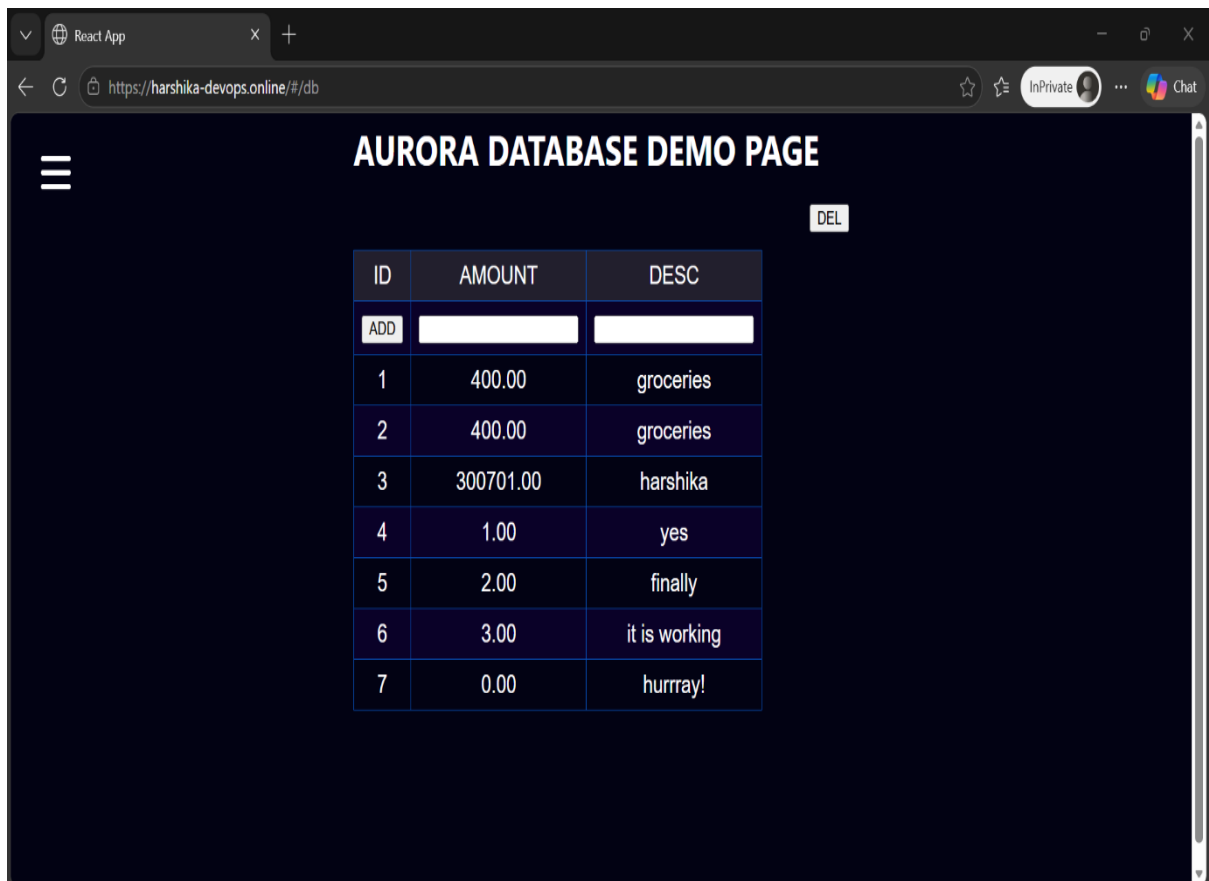
Once the Target Group shows **Healthy** in green:

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!

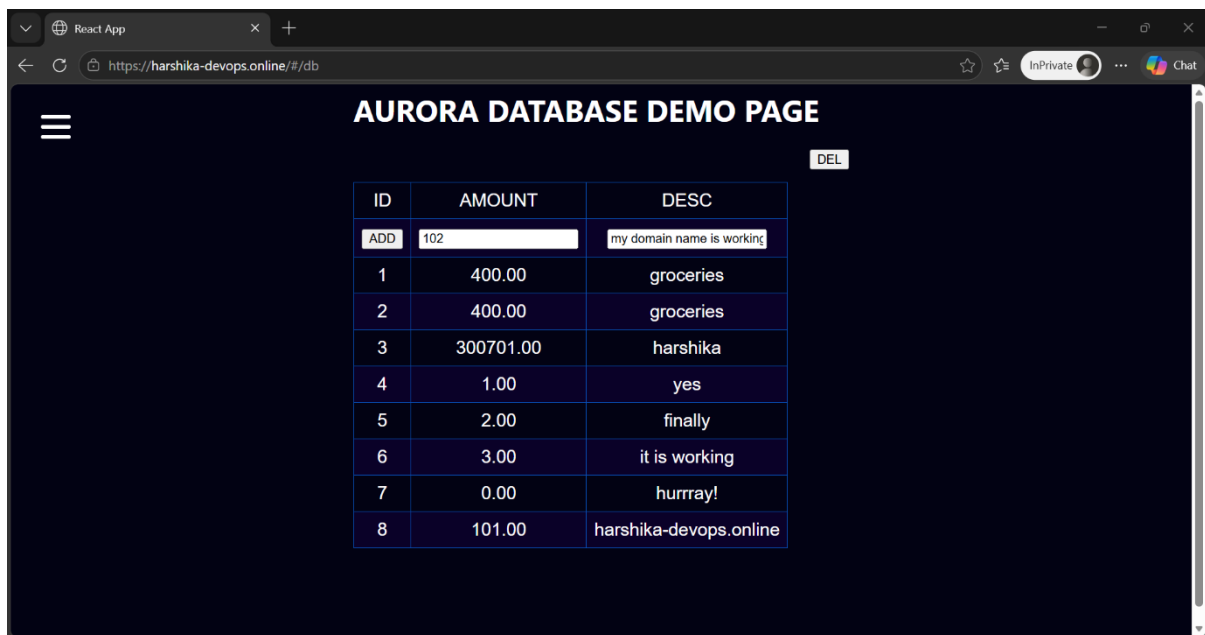
Successful!



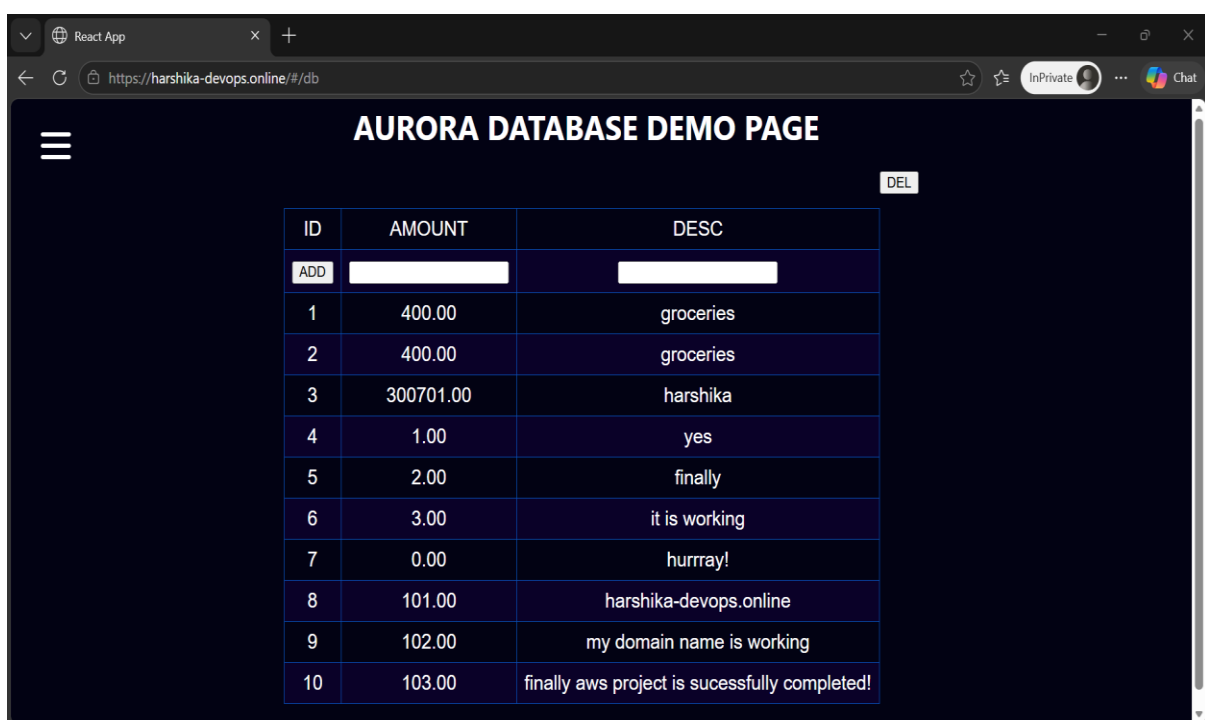
Previously saved entries.



Enter the new entries.

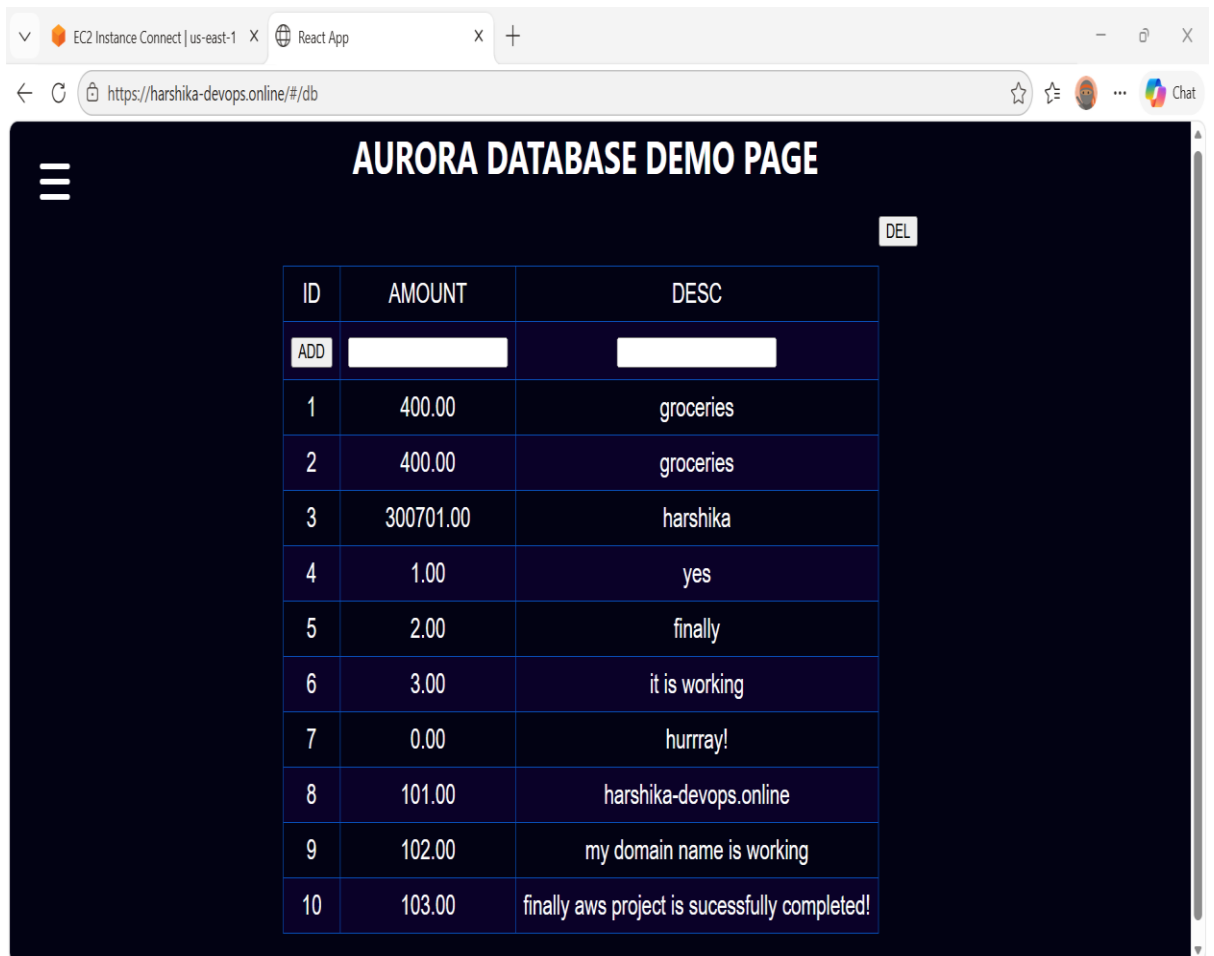
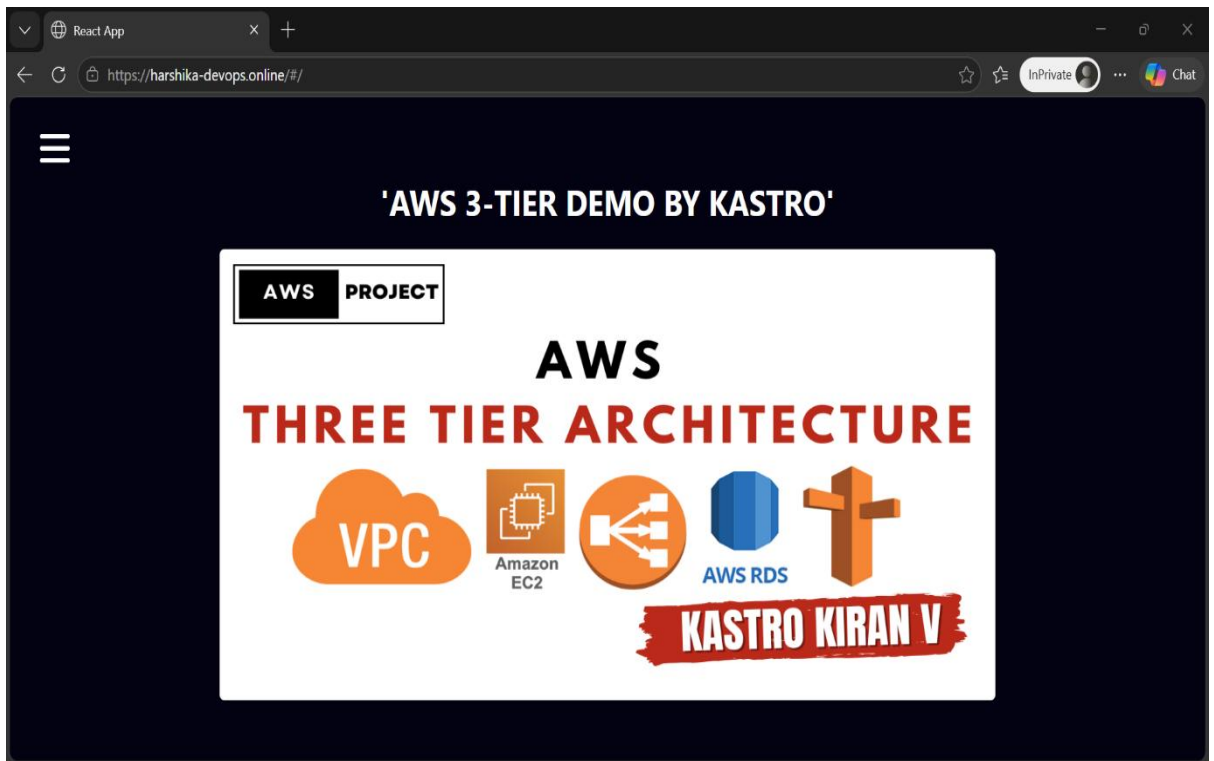


Final entries.

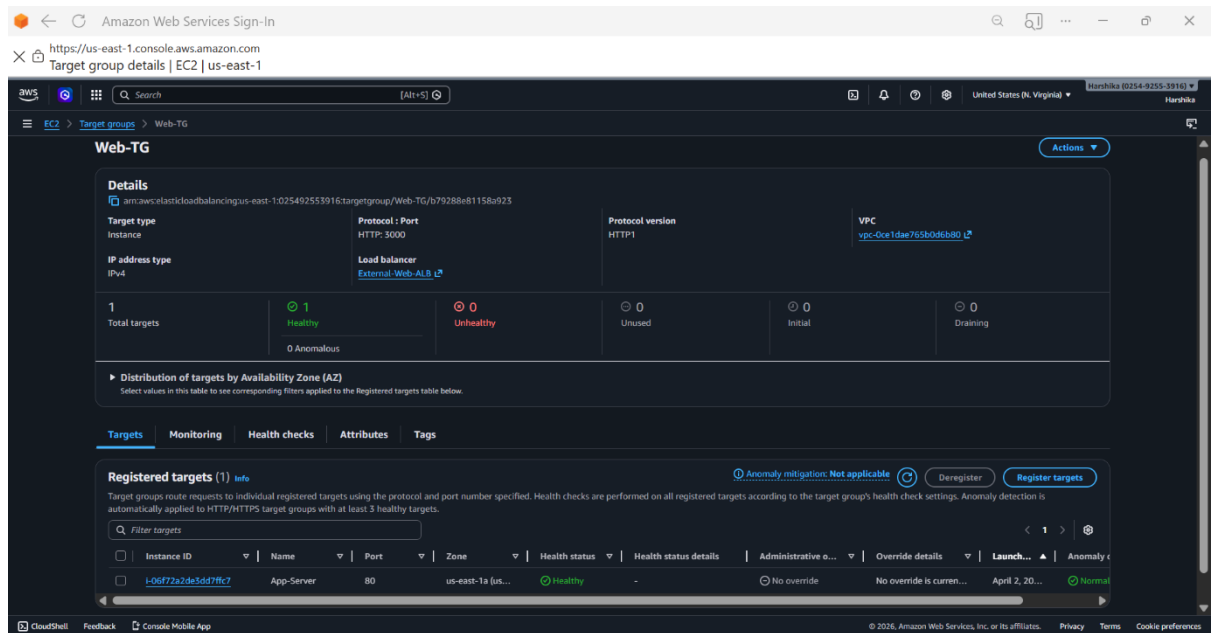


To prove the 3-Tier architecture is functional, putting these four specific screenshots:

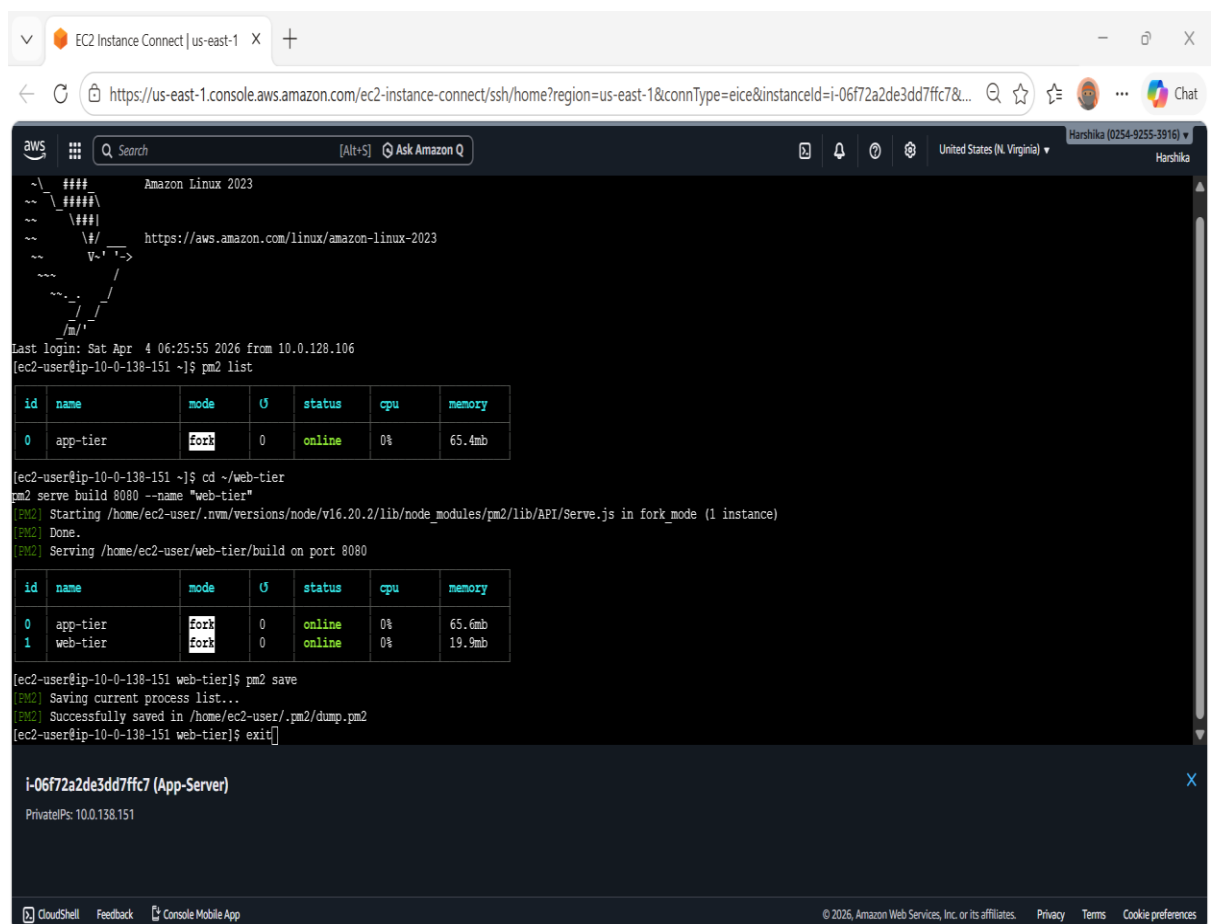
1. **The Live Website:** This proves the Frontend, Backend, and Database are all talking to each other.



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 screenshot shows the AWS Route 53 console for the hosted zone 'harshika-devops.online'. The 'Records' tab is selected, displaying a table of five DNS records. The first record is an Alias record pointing to 'dualstack.external-web-alb-1147954136.us-east-1.elb.amazonaws.com'. The other records are NS, SOA, CNAME, and A records.

Record name	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...	Health ...	Evalu...	Record ID
harshika-devops.online	A	Simple	-	Yes	dualstack.external-web-alb-1147954136.us-east-1.elb.amazonaws.com.	-	-	Yes	-
harshika-devops.online	NS	Simple	-	No	ns-368.awsdns-46.com. ns-1269.awsdns-30.org. ns-642.awsdns-16.net. ns-1724.awsdns-23.co.uk.	172800	-	-	-
harshika-devops.online	SOA	Simple	-	No	ns-368.awsdns-46.com. awsdns-hostmaster.amazon.com. 1 7200 900 1209600 86400	900	-	-	-
_371031e0e3840462f3692...	CNAME	Simple	-	No	_942f4bcb9ecf640a3f1e4a3b6830483c.jkddztzsm.acm-validations.aws.	300	-	-	-
api.harshika-devops.online	A	Simple	-	No	13.232.113.205	300	-	-	-

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! 