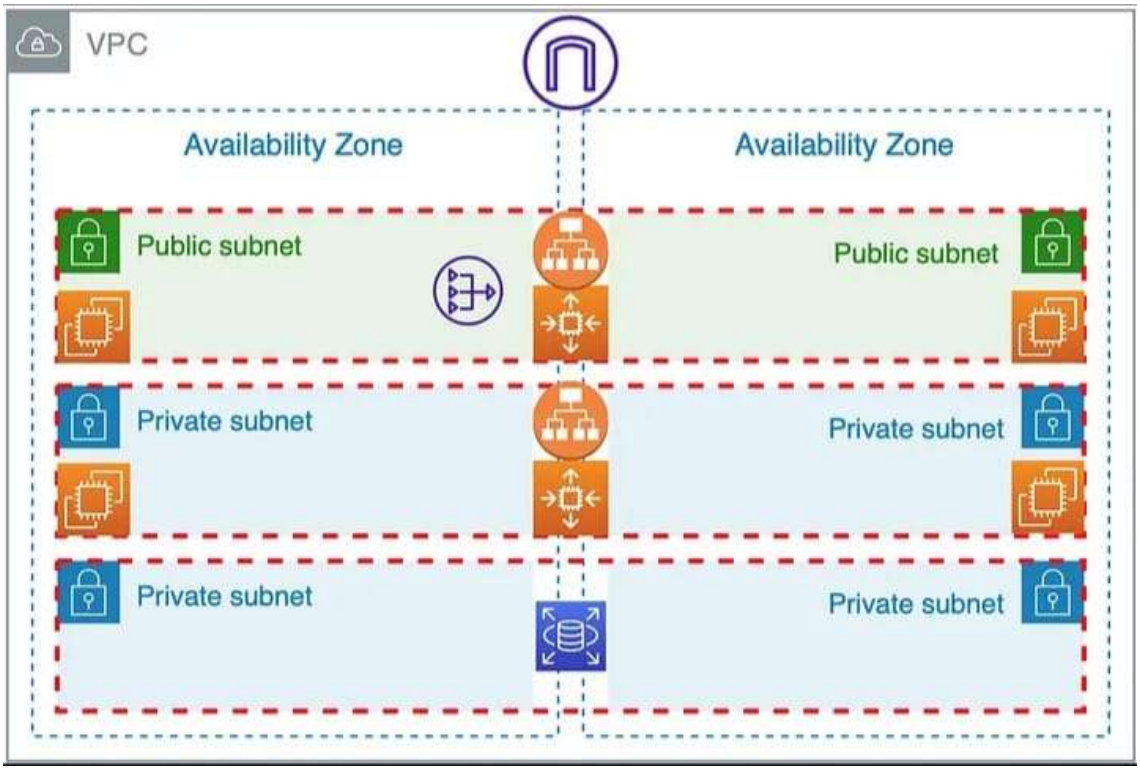
P. CHARAN KUMAR REDDY

Mail Id: charankumarreddyperam@gmail.com

Source code GitHub Link:

https://github.com/charanreddy7958/3-Tier-Architecture-using-Terraform

3-Tier Architecture using Terraform



What is a Three-Tier Architecture?

A **three-tier architecture** separates your application into:

1. **Presentation Layer** (Frontend)
2. **Application Layer** (Backend)
3. **Data Layer** (Database)

This improves scalability, security, and maintainability.

Steps to Deploy Three-Tier Architecture Using Terraform

- **Step 1: Define Project Structure**

three-tier-aws/

├── terraform-block.tf

├── provider.tf

├── variables.tf

├── vpc.tf

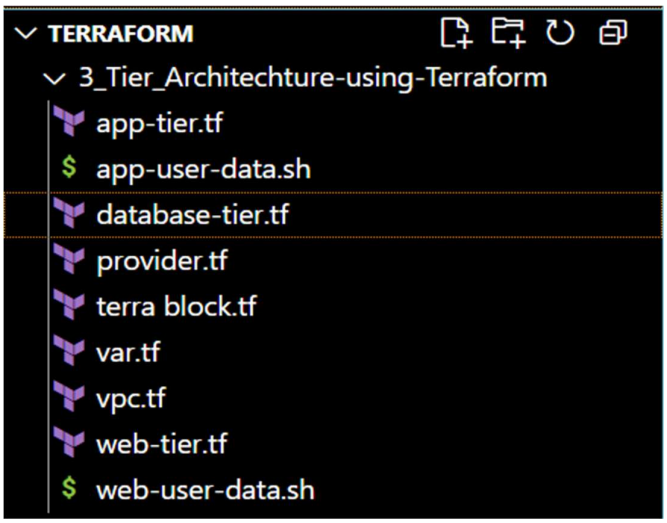
├── web-tier.tf

├── app-tier.tf

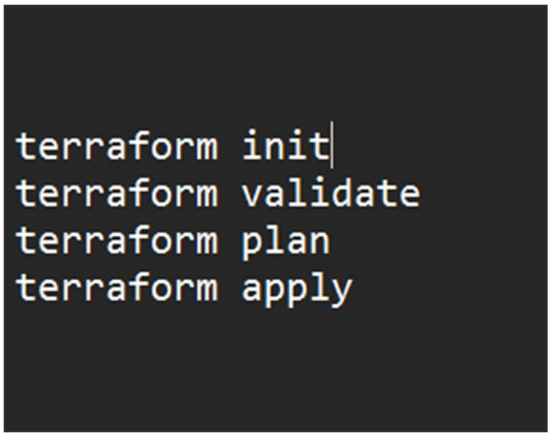
├── database-tier.tf

├── user-data1.tf

├── user-data2.tf



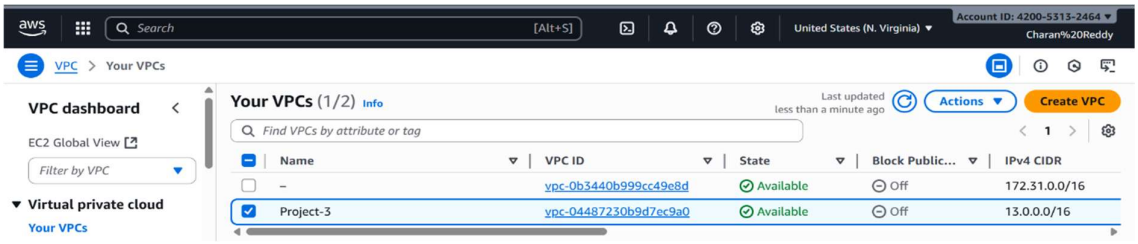
Terraform initialization



Terraform commands

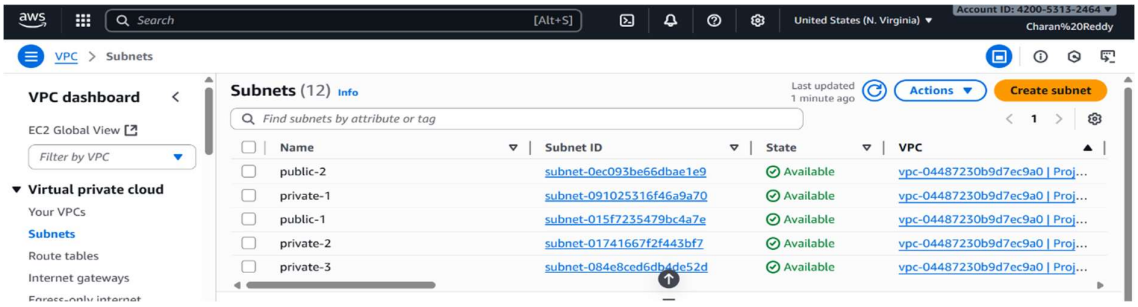
by using the above commands run the terraform

- **Step 2: Create VPC, Subnets and security group (vpc.tf)**
  - Create a custom VPC



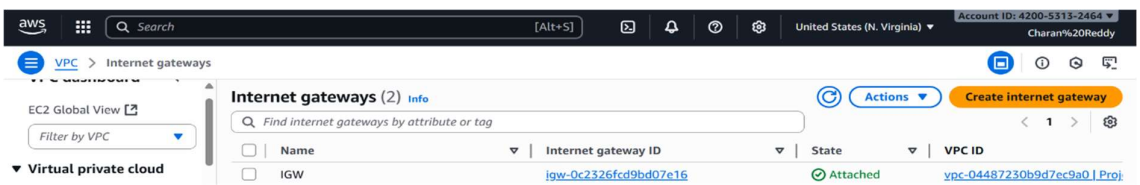
VPC

  - Define public and private subnets

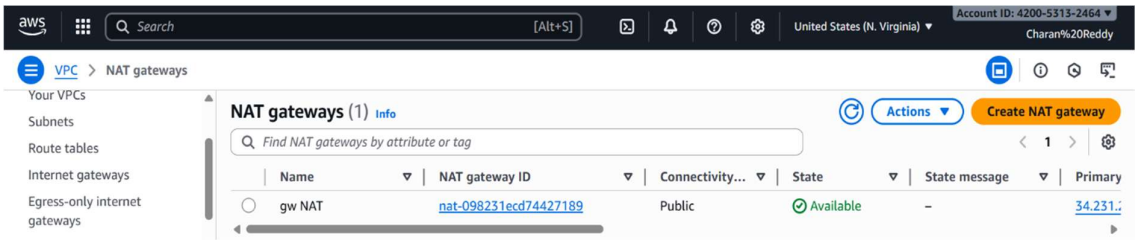


SUBNETS

  - Add Internet Gateway and NAT Gateway



INTERNET GATEWAY

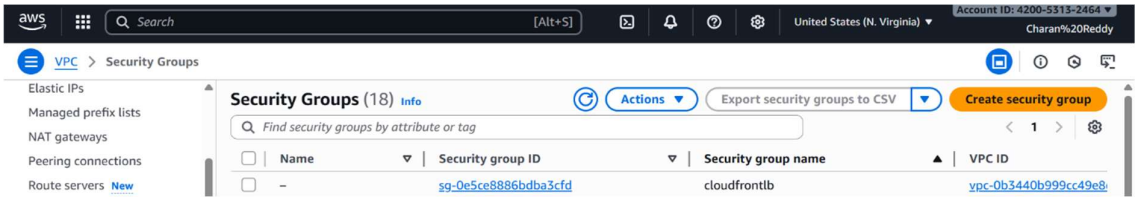


NAT GATEWAY

- Route tables for public/private traffic

ROUTE TABLES

- Create Security Group: Allow their protocols

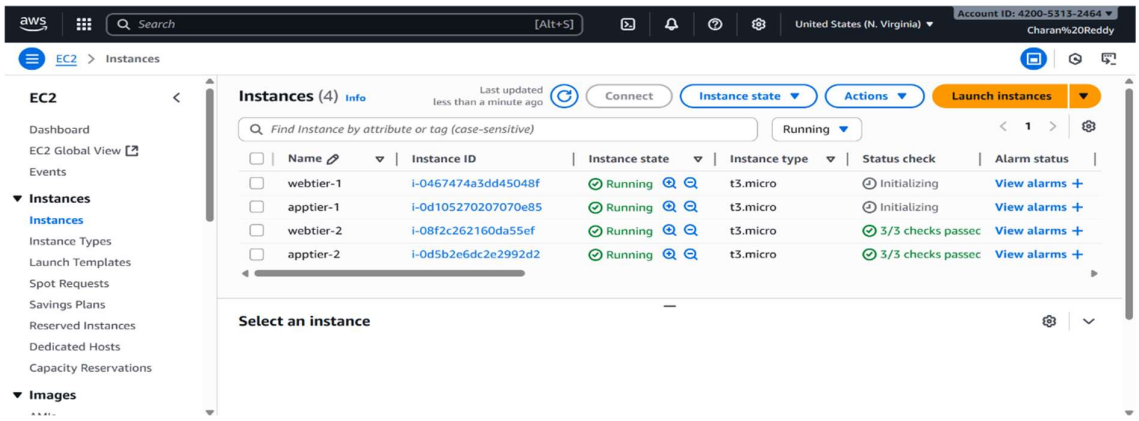


SECURITY GROUPS

```
resource "aws_vpc" "main" { ... }
resource "aws_subnet" "public" { ... }
resource "aws_subnet" "private" { ... }
resource "aws_internet_gateway" "igw" { ... }
resource "aws_nat_gateway" "nat" { ... }
resource "aws_security_group" "frontend_sg" { ... }
resource "aws_security_group" "backend_sg" { ... }
resource "aws_security_group" "db_sg" { ... }
```

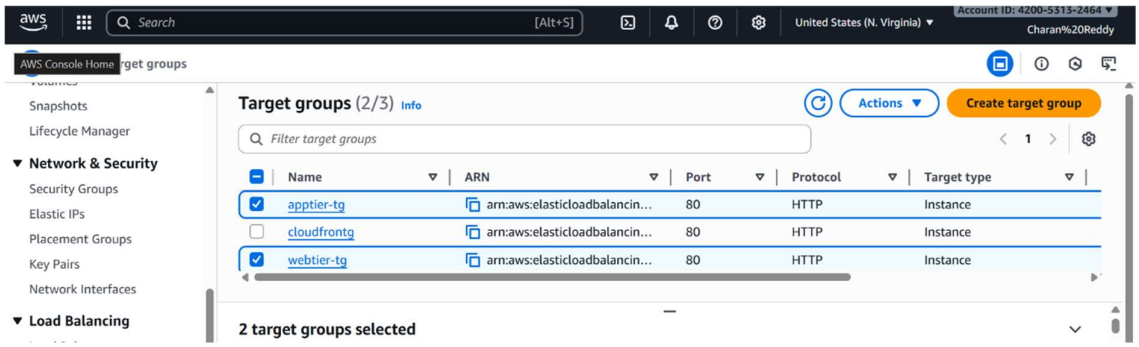
◇ Step 2: Launch EC2 Instances

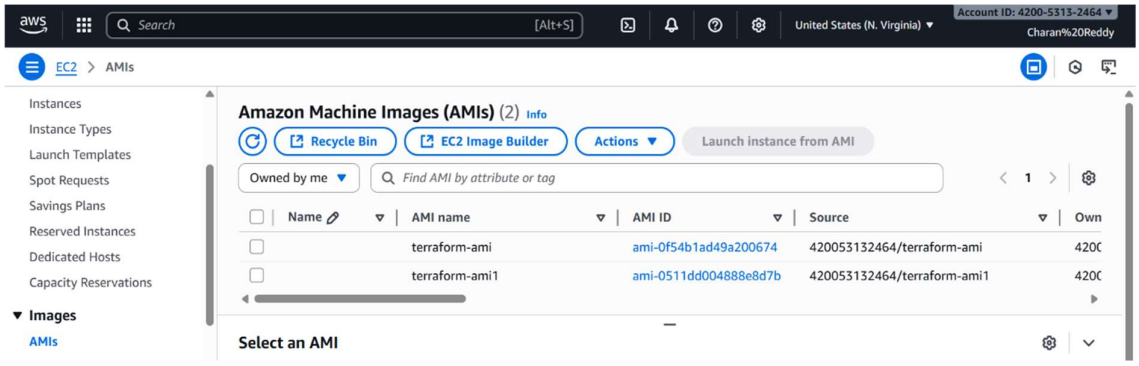
- In public subnet
- Use Apache
- Assign public IP
- In private subnet
- Host Java/Tomcat



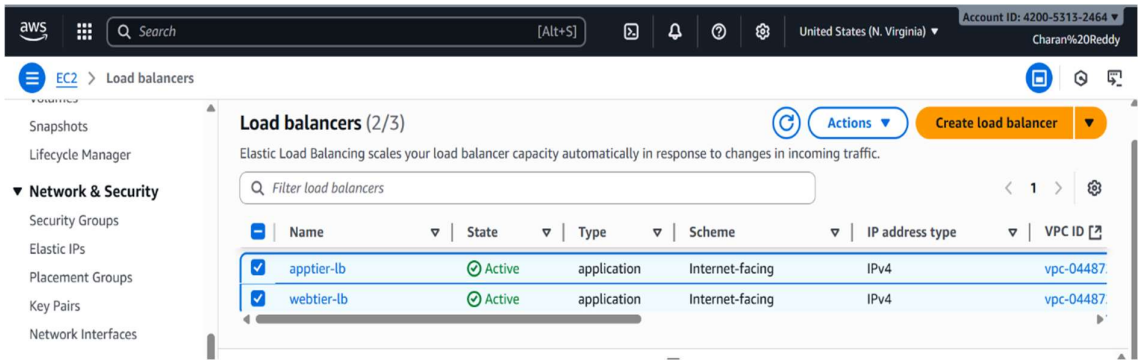
INSTANCES

- Add Auto Scaling Groups and Load Balancers

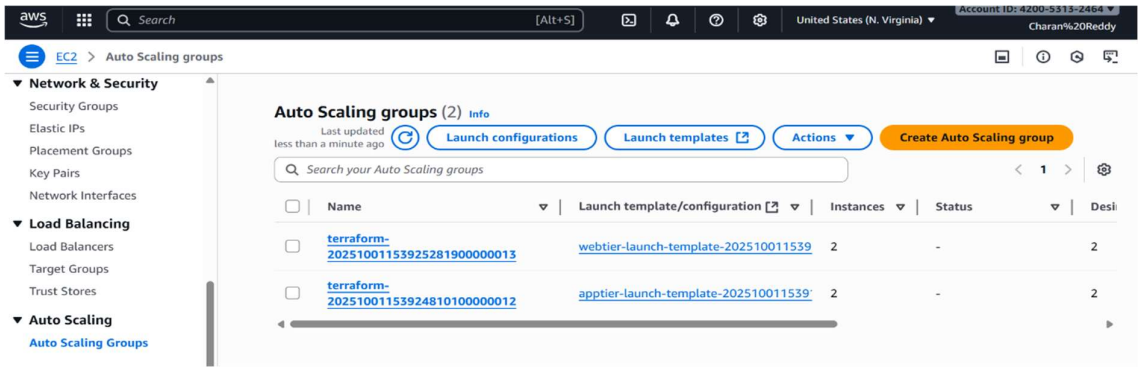




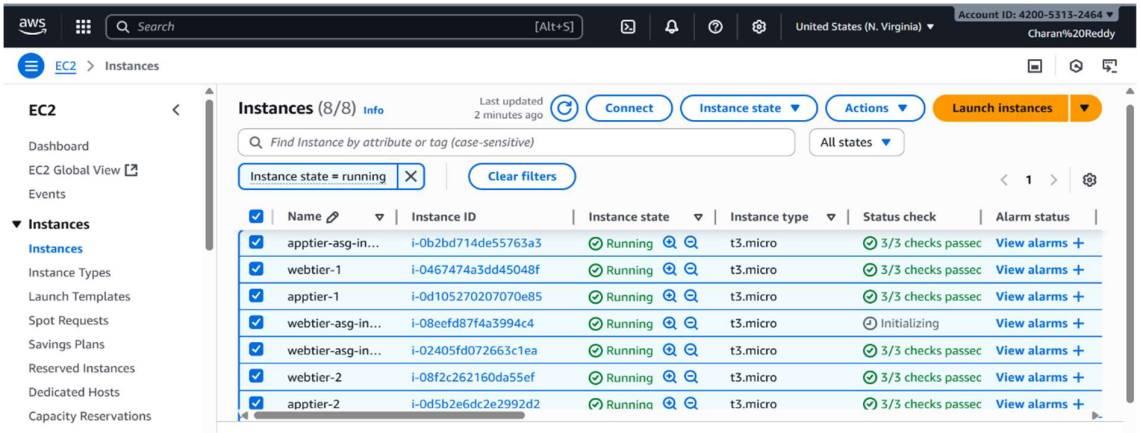
TARGET GROPUS & AMI



LOAD BALANCERS



AUTO SCALING



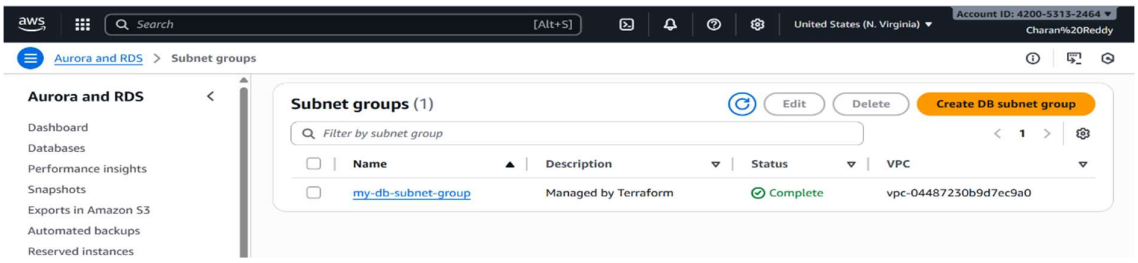
INSTANCES AFTER AUTO-SCALING

```
resource "aws_instance" "web-tier" { ... }

resource "aws_instance" "app-tier" { ... }
```

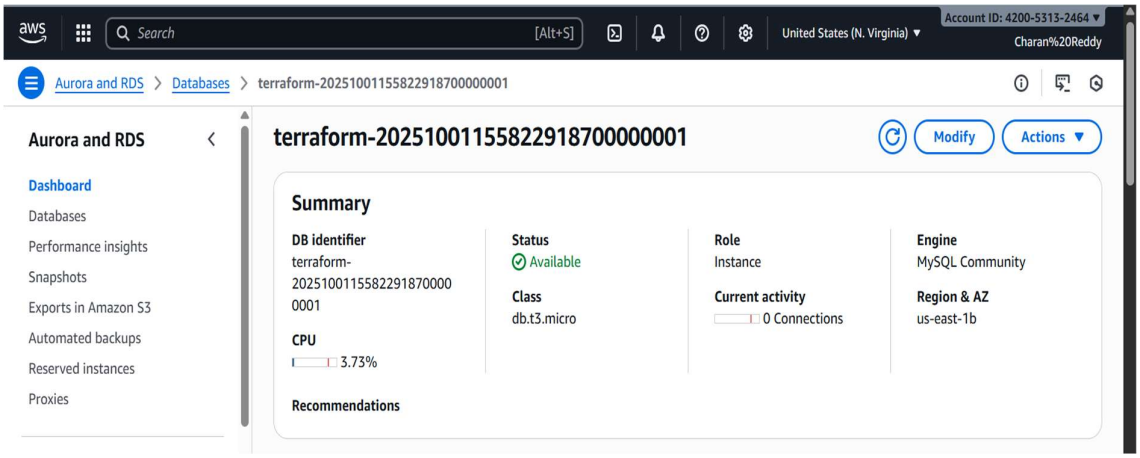
◇ **Step 3: Create RDS Instance (rds.tf)**

- In private subnet
- MySQL engine
- Enable multi-AZ (optional)
- Connect only from backend SG



SUBNET GROUP

- Connect to DB internally



DATABASE

resource "aws_db_instance" "rds" { ... }

◇ **Step 4: Add Variables(var.tf)**

- Use variables.tf for AMIs, instance types, subnet IDs, DB passwords.

  Variable “instance types” {…}

  Variable “password” {…}

- **Step 5: Initialize and Deploy**

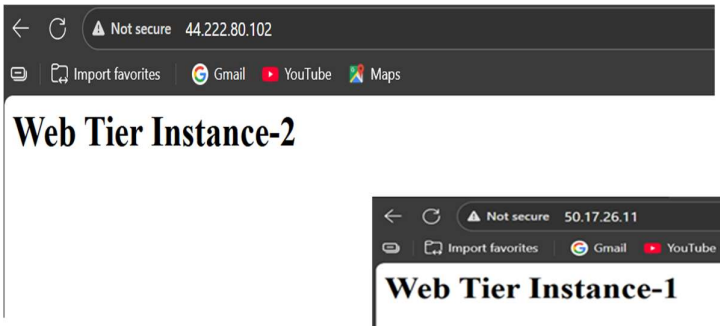
terraform init

terraform valiadte

terraform plan

terraform apply

- **Step 6: Test and Validate**
  - Access frontend via public IP



ACCESS WEB-SERVERS-1 & 2

- Ensure backend connects to DB using APP servers

```
root@ip-110-0-23-71:~# mysql -h database1.cxkeaqaaq645.eu-west-3.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 50
Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

- Create Database in it

```
mysql> CREATE DATABASE charan;
Query OK, 1 row affected (0.02 sec)
```

- By using above database create table and insert some values

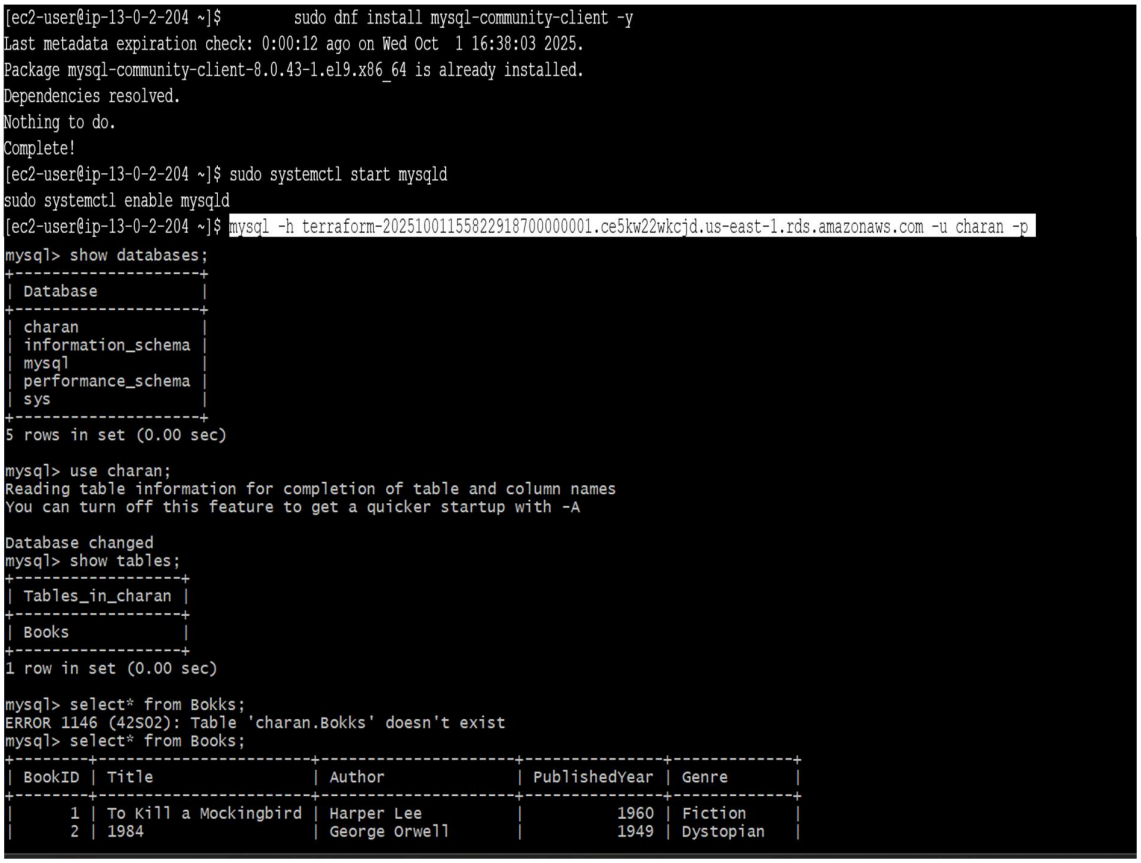
```
mysql> use charan;
Database changed
mysql> CREATE TABLE Books (
    ->     BookID INT PRIMARY KEY,
    ->     Title VARCHAR(100),
    ->     Author VARCHAR(100),
    ->     PublishedYear INT,
    ->     Genre VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO Books (BookID, Title, Author, PublishedYear, Genre)
    -> VALUES
    -> (1, 'To Kill a Mockingbird', 'Harper Lee', 1960, 'Fiction'),
    -> (2, '1984', 'George Orwell', 1949, 'Dystopian'),
    -> (3, 'The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Classic'),
    -> (4, 'Sapiens', 'Yuval Noah Harari', 2011, 'Non-fiction');
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> show tables;
+------------------+
| Tables_in_charan |
+------------------+
| Books            |
+------------------+
1 row in set (0.00 sec)
```

- Connect other server, install MySQL server, and check the table is located or not which we created in app-server-1

```
A newer release of "Amazon Linux" is available.
  Version 2023.9.20250929:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-13-0-2-204 ~]$
```



- Successfully showed the table in app-server-2 which we created in the app-server-1.

Source code: **https://github.com/charanreddy7958/3-Tier-Architecture-using-Terraform**

CONCLUSION

The implementation of a **Three-Tier Architecture in AWS using Terraform** successfully demonstrates the power of Infrastructure as Code (IaC) in building scalable, secure, and modular cloud environments. By separating the application into **Web**, **Application**, and **Database** tiers, we achieved:

- **Improved fault isolation** and resource control
- **Automated provisioning** of EC2, VPC, and RDS components
- **Reusable Terraform modules** for consistent deployments
- **High availability** through Multi-AZ configurations and security best practices

This project highlights how DevOps and cloud engineering principles can be applied to design production-ready architectures with minimal manual intervention. It also reinforces the importance of modular design, version control, and declarative infrastructure in modern cloud workflows.