

DEPLOY THREE -TIER ARCHITECTURE IN AWS USING TERRAFORM

Prepared by,
T.AJAY

Deploy three -tier architecture in AWS using Terraform

WHAT IS TERRAFORM?

- Terraform is an open-source infrastructure as code (IAC) tool developed by HashiCorp. It allows you to define and manage your infrastructure using declarative configuration files.
- With Terraform, you can create, modify, and manage various cloud resources and infrastructure components across different cloud providers in a consistent and repeatable manner.

Prerequisites:-

- Basic knowledge of AWS&Terraform
- AWS account
- IAM user
- GitHub account
- AWS access key& secret key

List of Steps :-

1. Create a file for the vpc
2. Create a file for the subnet
3. Create a file for the IGW
4. Create a file for the Route table
5. Create a file for the keypair
6. Create a file for the userdata
7. Create a file for the security group for the frontend
8. Create a file for the EC2 instance
9. Create a file for the security group for the database
10. Create a file for the Application load balancer
11. Create a file for the RDS instance
12. Create a file for the outputs
13. Verify the resources

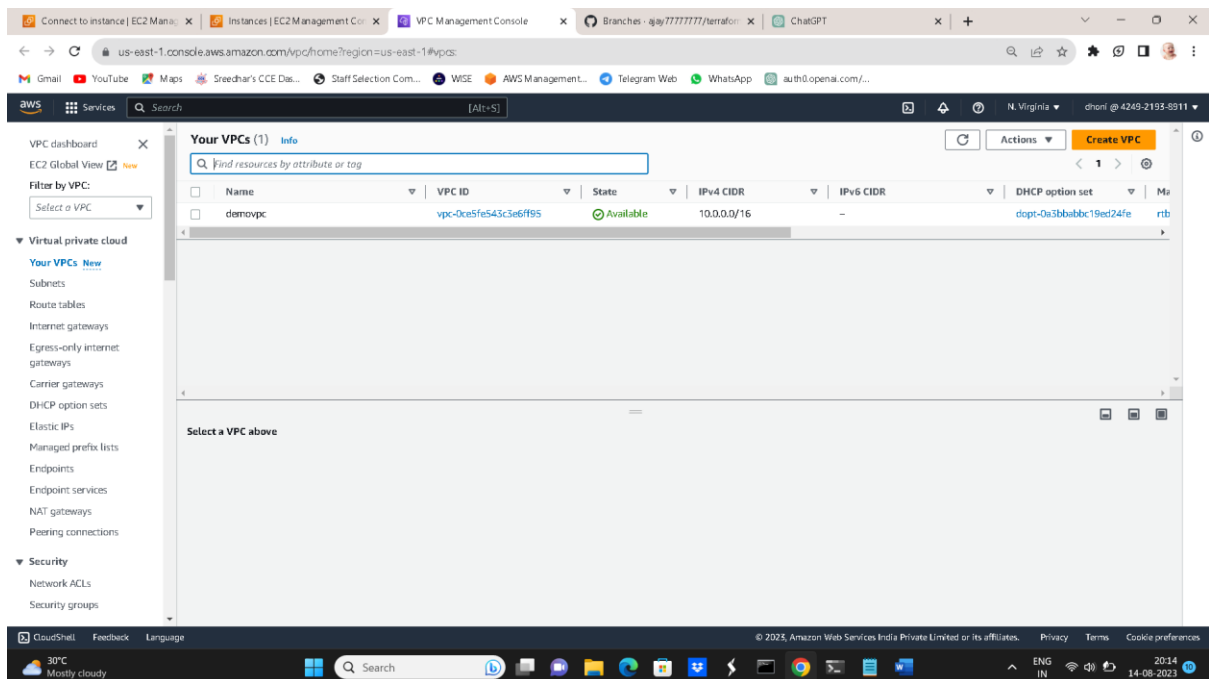
Step-1:- create a file for the vpc

- Create vpc.tf file and add the below code to it.

```
ec2-user@ip-172-31-32-255:~$ cat vpc.tf
# Creating VPC
resource "aws_vpc" "demovpc" {
  cidr_block      = "10.0.0.0/16"
  instance_tenancy = "default"
  tags = {
    Name = "demovpc"
  }
}
```

"vpc.tf" 8L, 150B

- Vpc is created with name demovpc.



Step 2:- Create a file for the Subnet

- For this project, I will create total 6 subnets for the front-end and back-end with a mixture of public & private subnet
- Create subnet.tf file and add the below code to it

```
ec2-user@ip-172-31-32-255:~$ cat subnet.tf
# Creating 1st web subnet
resource "aws_subnet" "public-subnet-1" {
  vpc_id            = aws_vpc.demovpc.id
  cidr_block        = "10.0.1.0/24"
  map_public_ip_on_launch = true
  availability_zone  = "us-east-1a"
  tags = {
    Name = "Web Subnet 1"
  }
}

# Creating 2nd web subnet
resource "aws_subnet" "public-subnet-2" {
  vpc_id            = aws_vpc.demovpc.id
  cidr_block        = "10.0.2.0/24"
  map_public_ip_on_launch = true
  availability_zone  = "us-east-1b"
  tags = {
    Name = "Web Subnet 2"
  }
}

# Creating 1st Application subnet
resource "aws_subnet" "application-subnet-1" {
  vpc_id            = aws_vpc.demovpc.id
  cidr_block        = "10.0.3.0/24"
  map_public_ip_on_launch = false
  availability_zone  = "us-east-1a"
  tags = {
    Name = "Application Subnet 1"
  }
}

# Creating 2nd Application subnet
resource "aws_subnet" "application-subnet-2" {
  vpc_id            = aws_vpc.demovpc.id
  cidr_block        = "10.0.4.0/24"
  map_public_ip_on_launch = false
  availability_zone  = "us-east-1b"
  tags = {
    Name = "Application Subnet 2"
  }
}

# Create Database Private Subnet
resource "aws_subnet" "database-subnet-1" {
  vpc_id            = aws_vpc.demovpc.id
  cidr_block        = "10.0.5.0/24"
  availability_zone  = "us-east-1a"
  tags = {
    Name = "Database Subnet 1"
  }
}

# Create Database Private Subnet
resource "aws_subnet" "database-subnet-2" {
  vpc_id            = aws_vpc.demovpc.id
  cidr_block        = "10.0.6.0/24"
  availability_zone  = "us-east-1b"
  tags = {
    Name = "Database Subnet 2"
  }
}
```

- Created 6 subnets.

The screenshot shows the AWS Management Console interface for the 'us-east-1' region. The 'Subnets' page is active, displaying a table with 6 subnets. The table columns are Name, Subnet ID, State, VPC, IPv4 CIDR, and IPv6 CIDR. All subnets are in the 'Available' state. The left sidebar shows the navigation menu with 'Subnets' selected under 'Virtual private cloud'.

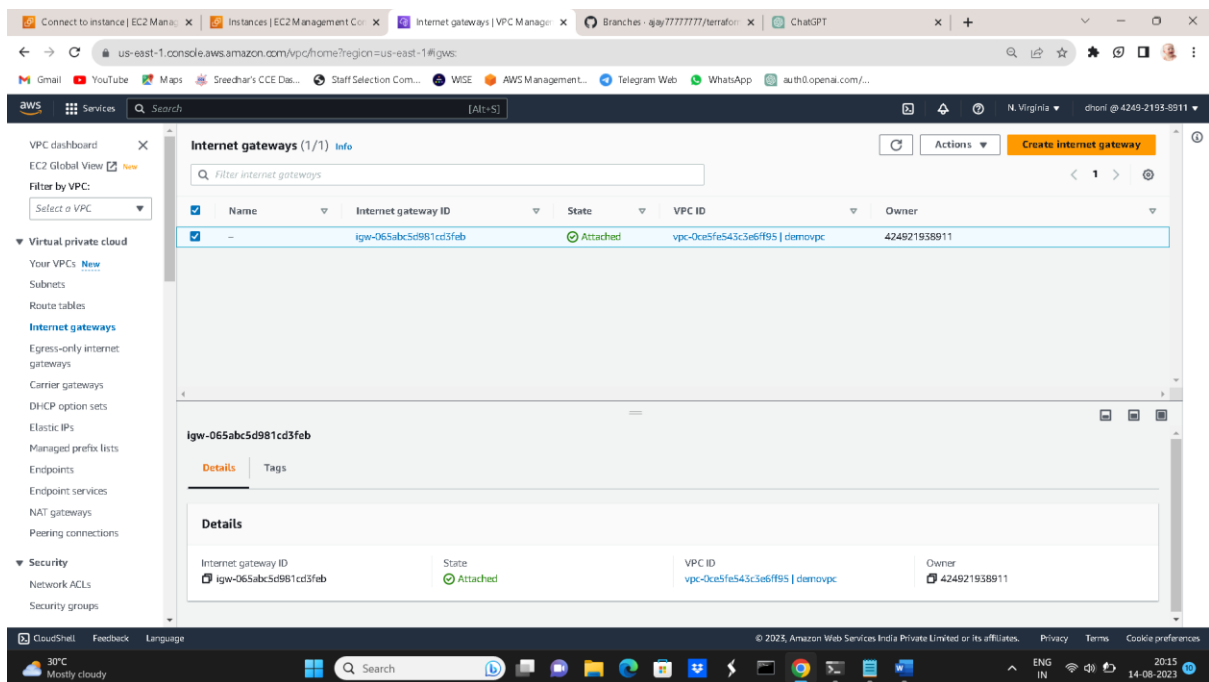
Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR
Database Subnet 2	subnet-0c4839485fce0d2d1	Available	vpc-0ce5fe543c3e6f95 demovpc	10.0.6.0/24	-
Database Subnet 1	subnet-0f72318969952bcb7e	Available	vpc-0ce5fe543c3e6f95 demovpc	10.0.5.0/24	-
Web Subnet 1	subnet-0892c069d38867df	Available	vpc-0ce5fe543c3e6f95 demovpc	10.0.1.0/24	-
Application Subnet 2	subnet-0cc8bceced1038ef2	Available	vpc-0ce5fe543c3e6f95 demovpc	10.0.4.0/24	-
Application Subnet 1	subnet-008b5a696b321a7ac	Available	vpc-0ce5fe543c3e6f95 demovpc	10.0.3.0/24	-
Web Subnet 2	subnet-06814827a62f9454c	Available	vpc-0ce5fe543c3e6f95 demovpc	10.0.2.0/24	-

Step 3:- Create a file for the Internet Gateway

- Create igw.tf file and add the below code to it

```
ec2-user@ip-172-31-32-255:~$ cat igw.tf
# Creating Internet Gateway
resource "aws_internet_gateway" "demogateway" {
  vpc_id = aws_vpc.demovpc.id
}
```

- Created internet gateway



Step 4:- Create a file for the Route table

- Create `route_table_public.tf` file and add the below code to it

```
resource "aws_route_table" "route" {
  vpc_id = aws_vpc.demovpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.demogateway.id
  }
  tags = {
    Name = "Route to internet"
  }
}

# Associating Route Table
resource "aws_route_table_association" "rt1" {
  subnet_id      = aws_subnet.public-subnet-1.id
  route_table_id = aws_route_table.route.id
}

# Associating Route Table
resource "aws_route_table_association" "rt2" {
  subnet_id      = aws_subnet.public-subnet-2.id
  route_table_id = aws_route_table.route.id
}

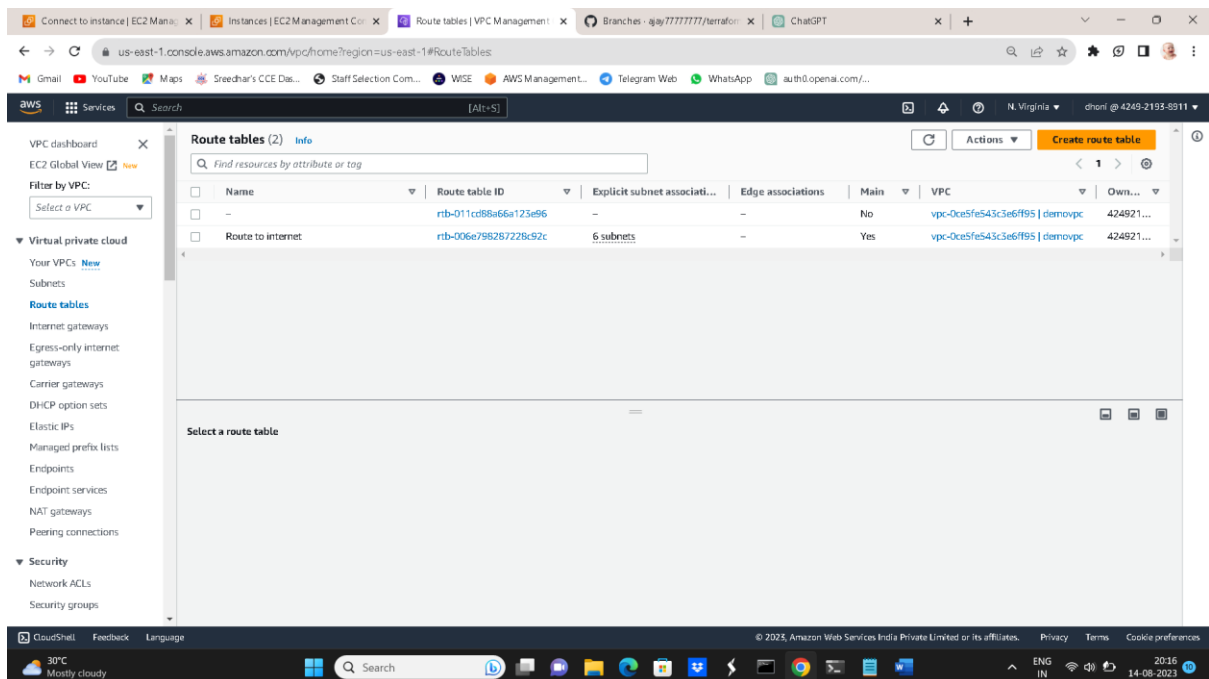
# Associating Route Table
resource "aws_route_table_association" "rt3" {
  subnet_id      = aws_subnet.application-subnet-1.id
  route_table_id = aws_route_table.route.id
}

# Associating Route Table
resource "aws_route_table_association" "rt4" {
  subnet_id      = aws_subnet.application-subnet-2.id
  route_table_id = aws_route_table.route.id
}

# Associating Route Table
resource "aws_route_table_association" "rt5" {
  subnet_id      = aws_subnet.database-subnet-1.id
  route_table_id = aws_route_table.route.id
}

# Associating Route Table
resource "aws_route_table_association" "rt6" {
  subnet_id      = aws_subnet.database-subnet-2.id
  route_table_id = aws_route_table.route.id
}
```

- Created route table with name route to internet.



Step-5:- Create a file for the keypair

```
ec2-user@ip-172-31-32-255:~$ cat terraform.tf
resource "aws_key_pair" "ajay" {
  key_name     = "ajay"
  public_key   = tls_private_key.rsa.public_key_openssh
}
resource "tls_private_key" "rsa" {
  algorithm = "RSA"
  rsa_bits  = 4096
}
resource "local_file" "tf-key" {
  content = tls_private_key.rsa.private_key_pem
  filename = "ajay"
}
```

Step 6:- Create a file for user data

- Create data.sh file and add the below code to it

```
ec2-user@ip-172-31-32-255:~$ cat data.sh
#!/bin/bash
sudo apt update
sudo apt install -y apache2
sudo systemctl start apache2
sudo systemctl enable apache2
echo "hii" > /var/www/html/index.html
```

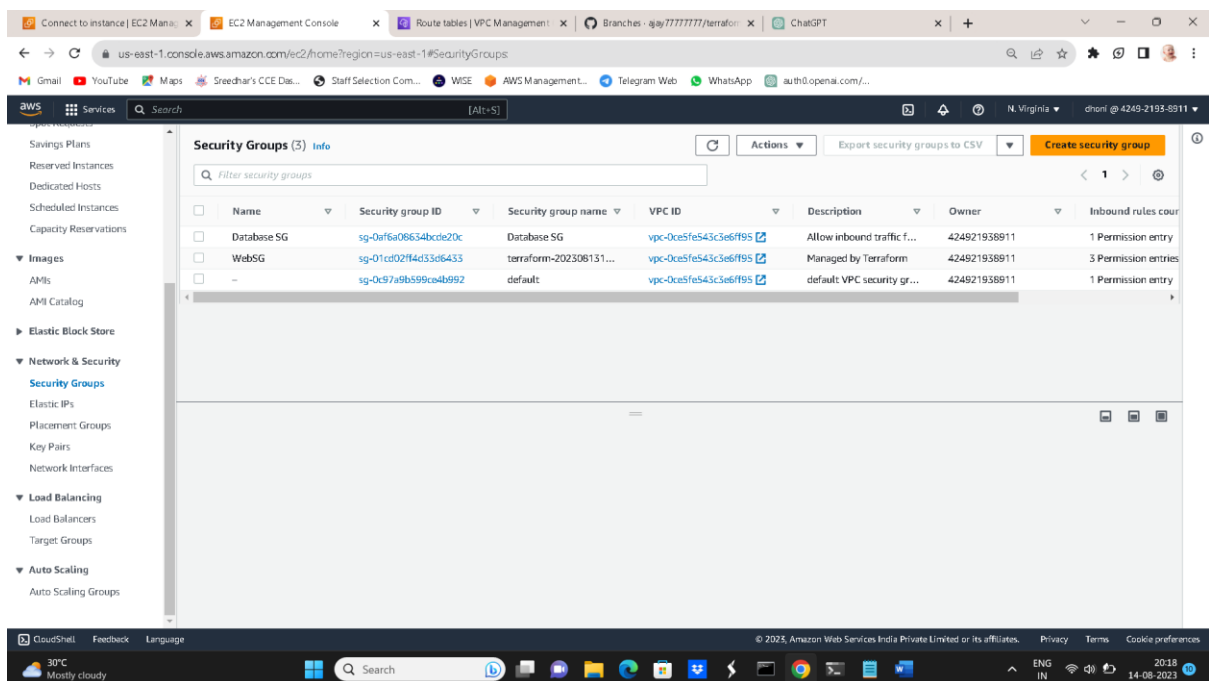
Step 7:- Create a file for Security Group for the FrontEnd tier

- Create web_sg.tf file and add the below code to it

```
ec2-user@ip-172-31-32-255:~$ cat web_sg.tf
# Creating Security Group
resource "aws_security_group" "demosg" {
  vpc_id = aws_vpc.demovpc.id
  # Inbound Rules
  # HTTP access from anywhere
  ingress {
    from_port = 80
    to_port   = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # HTTPS access from anywhere
  ingress {
    from_port = 443
    to_port   = 443
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # SSH access from anywhere
  ingress {
    from_port = 22
    to_port   = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # Outbound Rules
  # Internet access to anywhere
  egress {
    from_port = 0
    to_port   = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "WebSG"
  }
}

"web_sg.tf" 37L, 747B
```

- Created security group with name websg.



Step 8:- Create a file for EC2 instances

- Create ec2.tf file and add the below code to it

```
ec2-user@ip-172-31-32-255:~$ cat ec2.tf
# Creating 1st EC2 instance in Public Subnet
resource "aws_instance" "demoinstance1" {
  ami           = "ami-053b0d53c279acc90"
  instance_type = "t2.micro"
  key_name      = "ajay"
  vpc_security_group_ids = ["${aws_security_group.demosg.id}"]
  subnet_id    = aws_subnet.public-subnet-1.id
  associate_public_ip_address = true
  user_data    = "${file("data.sh")}"
  tags = {
    Name = "My Public Instance1"
  }
}

# Creating 2nd EC2 instance in Public Subnet
resource "aws_instance" "demoinstance2" {
  ami           = "ami-053b0d53c279acc90"
  instance_type = "t2.micro"
  key_name      = "ajay"
  vpc_security_group_ids = ["${aws_security_group.demosg.id}"]
  subnet_id    = aws_subnet.public-subnet-2.id
  associate_public_ip_address = true
  user_data    = "${file("data.sh")}"
  tags = {
    Name = "My Public Instance 2"
  }
}

}
```

"ec2.tf" 26L, 991B

- Created 2 instances successfully.

The screenshot shows the AWS Management Console interface. The 'Instances' page is active, displaying a table with two EC2 instances. The first instance, 'My Public Inst...', is in a 'Running' state with a public IP of 107.22.150. The second instance, 'My Public Inst...', is in a 'Terminated' state. The console also shows a sidebar with navigation options like 'EC2 Dashboard', 'Events', 'Instances', 'Images', 'Elastic Block Store', and 'Network & Security'. The bottom of the screen shows the Windows taskbar with the date and time as 20:17 on 14-08-2023.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4
My Public Inst...	i-0f43f96bcb784da9	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	-	107.22.150.
My Public Inst...	i-086a2347b3b4ba1fd	Terminated	t2.micro	-	No alarms	us-east-1a	-	-

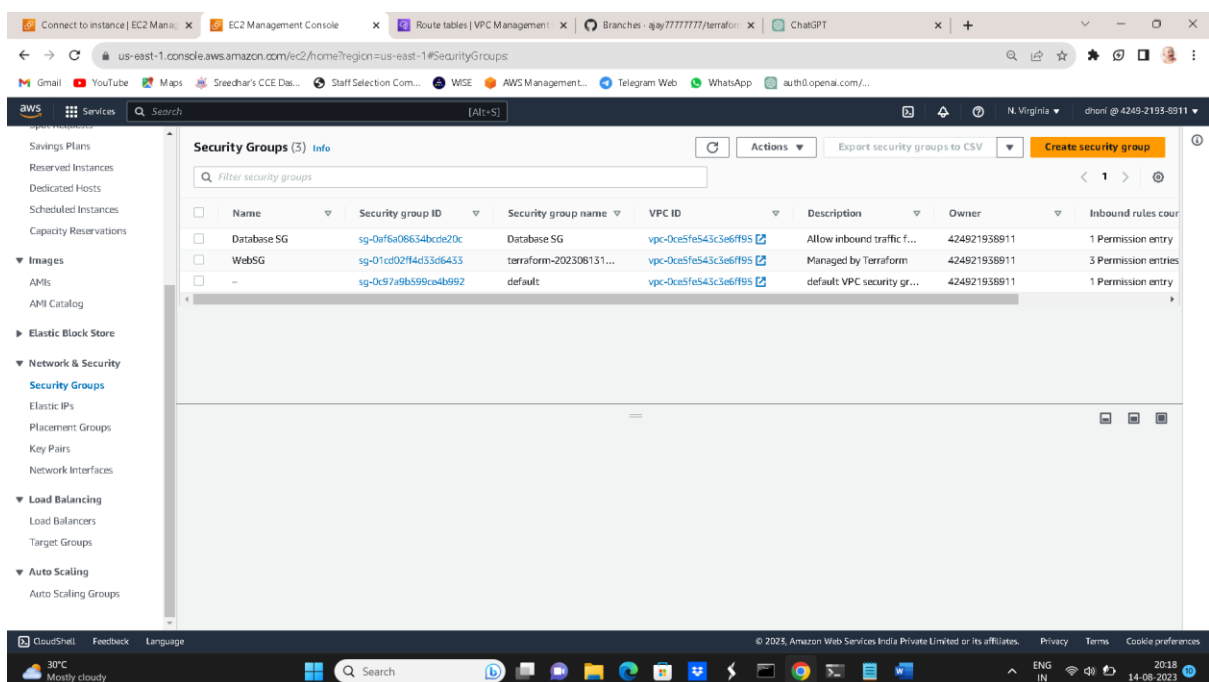
Step 9:- Create a file for Security Group for the Database tier

- Create database_sg.tf file and add the below code to it

```
ec2-user@ip-172-31-32-255:~$ cat database_sg.tf
# Create Database Security Group
resource "aws_security_group" "database-sg" {
  name        = "Database SG"
  description = "Allow inbound traffic from application layer"
  vpc_id      = aws_vpc.demovpc.id
  ingress {
    description = "Allow traffic from application layer"
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    security_groups = [aws_security_group.demosg.id]
  }
  egress {
    from_port   = 32768
    to_port     = 65535
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = {
    Name = "Database SG"
  }
}
```

databasesg.tf 22L, 580B

- Created one database security group with name databasesg and allowed the port number 3306.



Step 10:- Create a file Application Load Balancer

- Create alb.tf file and add the below code to it

```
ec2-user@ip-172-31-32-255:~$ cat alb.tf
# Creating External LoadBalancer
resource "aws_lb" "external-alb" {
  name            = "External-LB"
  internal        = false
  load_balancer_type = "application"
  security_groups = [aws_security_group.demosg.id]
  subnets        = [aws_subnet.public-subnet-1.id, aws_subnet.public-subnet-2.id]
}

resource "aws_lb_target_group" "target-elb" {
  name        = "ALB-TG"
  port        = 80
  protocol    = "HTTP"
  vpc_id      = aws_vpc.demovpc.id
}

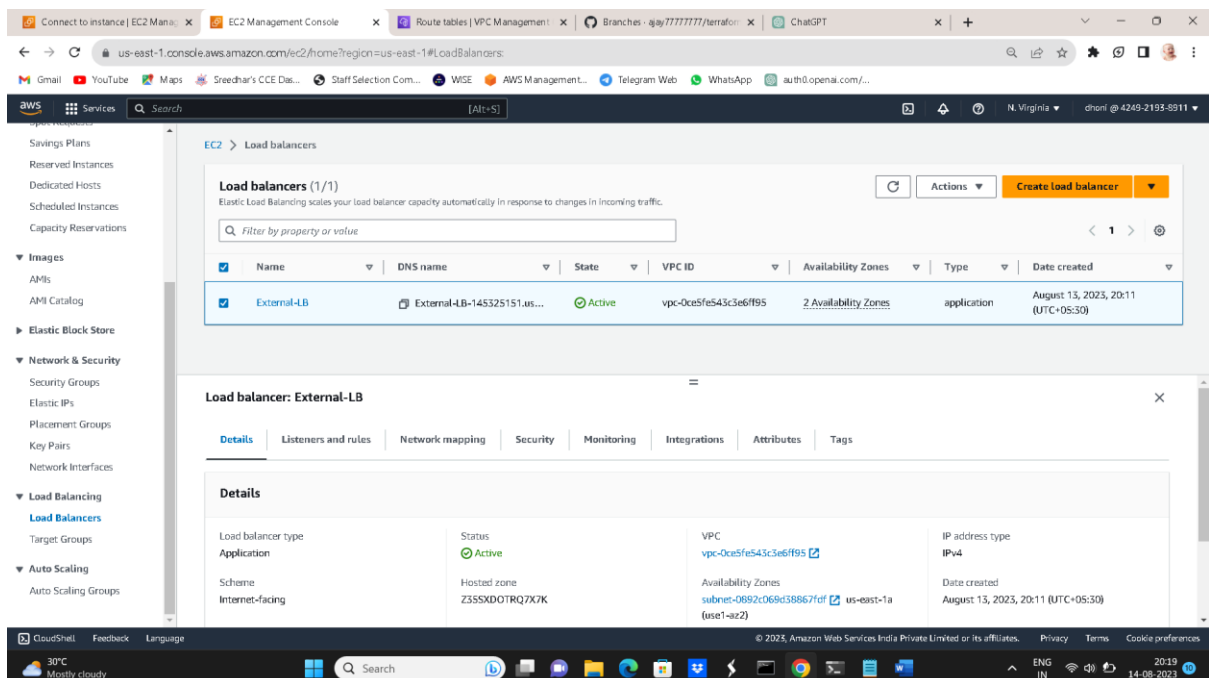
resource "aws_lb_target_group_attachment" "attachment1" {
  target_group_arn = aws_lb_target_group.target-elb.arn
  target_id        = aws_instance.demoinstance1.id
  port             = 80
  depends_on       = [aws_instance.demoinstance1]
}

resource "aws_lb_target_group_attachment" "attachment2" {
  target_group_arn = aws_lb_target_group.target-elb.arn
  target_id        = aws_instance.demoinstance2.id
  port             = 80
  depends_on       = [aws_instance.demoinstance2]
}

resource "aws_lb_listener" "external-lb" {
  load_balancer_arn = aws_lb.external-alb.arn
  port              = "80"
  protocol          = "HTTP"
  default_action {
    type = "forward"
    target_group_arn = aws_lb_target_group.target-elb.arn
  }
}

"alb.tf" 35L, 1194B
```

- load balancer created successfully with name external-lb and attached the target group target-elb and the http listener is port 80.



Step 11:- Create a file for the RDS instance

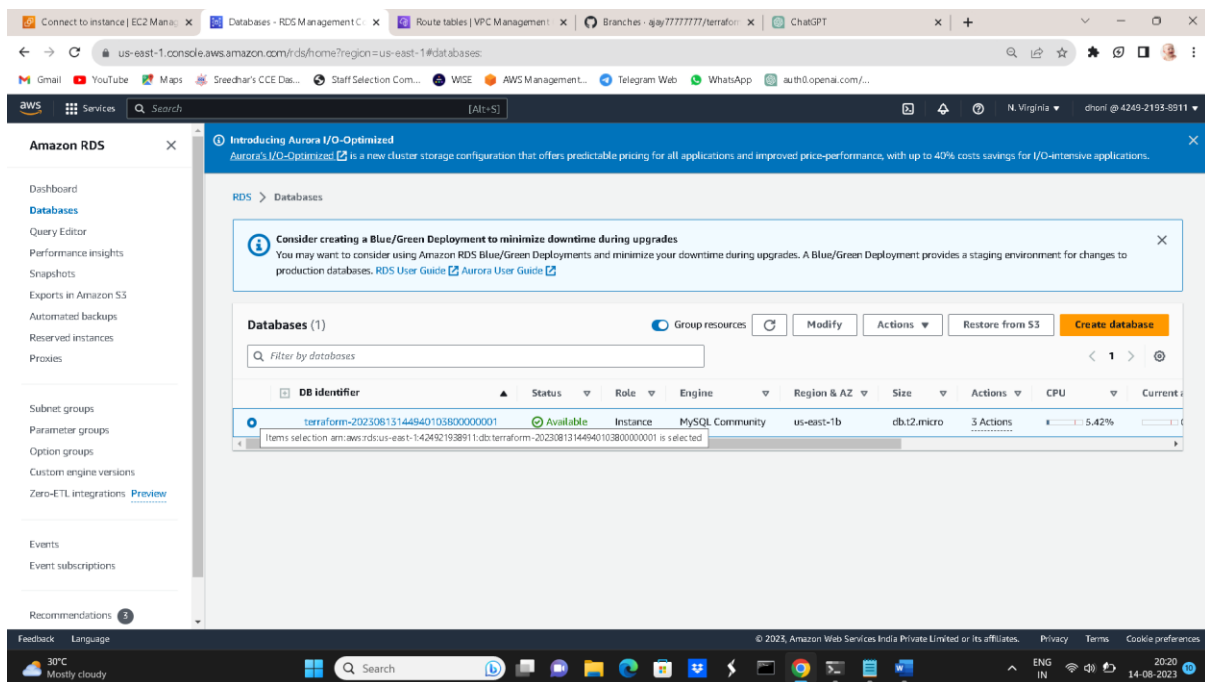
- Create a rds.tf file and add the below code to it

```
ec2-user@ip-172-31-32-255:~$ cat rds.tf
# Creating RDS Instance
resource "aws_db_subnet_group" "default" {
  name       = "main"
  subnet_ids = [aws_subnet.database-subnet-1.id, aws_subnet.database-subnet-2.id]
  tags = {
    Name = "My DB subnet group"
  }
}

resource "aws_db_instance" "demodb" {
  allocated_storage      = 10
  db_subnet_group_name   = aws_db_subnet_group.default.id
  engine                 = "MySQL"
  engine_version         = "8.0.33"
  instance_class         = "db.t2.micro"
  multi_az               = true
  username               = "admin"
  password               = "admin123"
  skip_final_snapshot    = true
  vpc_security_group_ids = [aws_security_group.database-sg.id]
}
```

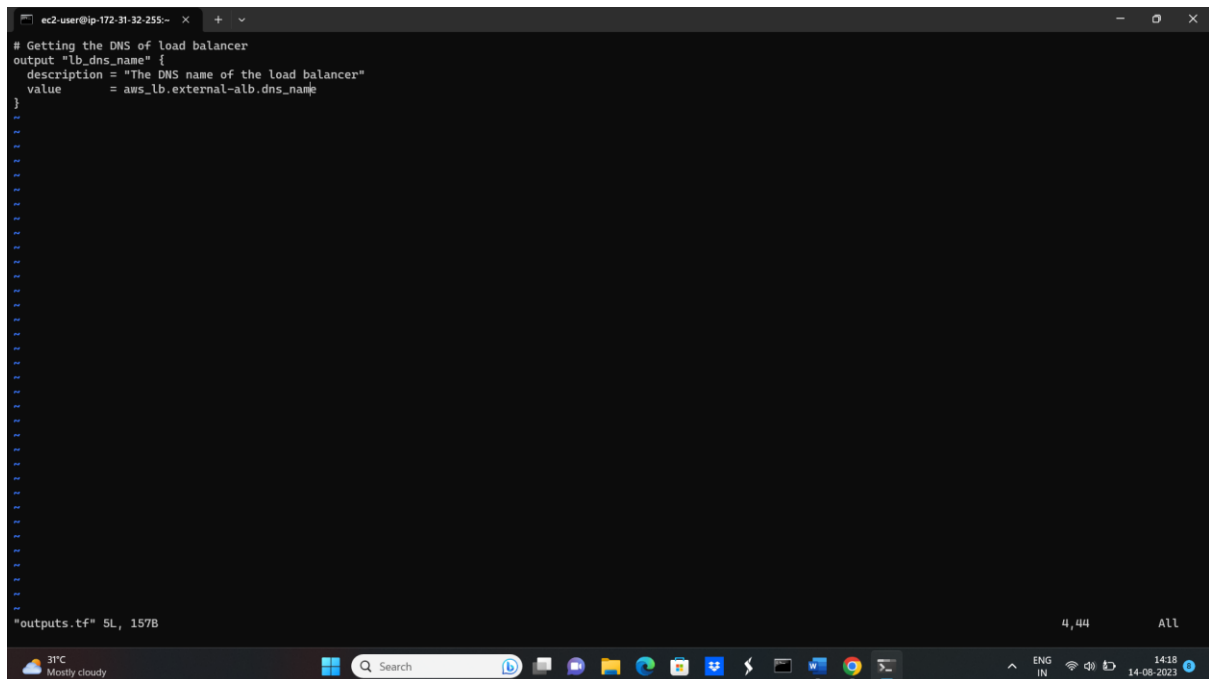
"rds.tf" 20L, 660B

- Mysql Database created successfully.



Step 12:- Create a file for outputs

- Create outputs.tf file and add the below code to it
- In this step the load balancer DNS name is given as output.by using this dns you can access DNS through any browser.



```
ec2-user@ip-172-31-32-255:~$ cat outputs.tf
# Getting the DNS of load balancer
output "lb_dns_name" {
  description = "The DNS name of the load balancer"
  value       = aws_lb.external-alb.dns_name
}
```

The screenshot shows a terminal window with the command prompt 'ec2-user@ip-172-31-32-255:~\$'. The user has created a file named 'outputs.tf' containing Terraform output configuration for a load balancer DNS name. The configuration includes a comment, an output block named 'lb_dns_name' with a description, and a value reference to 'aws_lb.external-alb.dns_name'. The terminal status bar at the bottom shows the file size as 5L, 157B. The Windows taskbar is visible at the bottom of the screen.

So, now our entire code is ready. We need to run the below steps to create infrastructure.

1. terraform init is to initialize the working directory and downloading plugins of the provider
2. terraform validate is to validate the code whether the code contains error or not.
- 1) terraform plan is to create the execution plan for our code
- 2) terraform apply is to create the actual infrastructure.

Step 13:- Verify the resources

- Terraform will create below resources
- 1. VPC
- 2. Application Load Balancer
- 3. Public & Private Subnets
- 4. EC2 instances
- 5. RDS instance
- 6. Route Table
- 7. Internet Gateway
- 8. Security Groups for Web & RDS instances
- 9. Route Table

Once the resource creation finishes you can get the DNS of a load balancer and paste it into the browser and you can see load balancer will send the request to two instances.

