TERRAFORM PROJECT

TERRAFORM:

Terraform is an open-source infrastructure as code (IaC) tool created by HashiCorp that allows users to define and provision infrastructure using a high-level configuration language known as HashiCorp Configuration Language (HCL) or JSON.

KEY FEATURES OF TERRAFORM:

- **Infrastructure as Code:** Write and manage infrastructure in a declarative configuration file, allowing for version control and collaborative development.
- **Provider Ecosystem:** Terraform supports a wide range of cloud providers (like AWS, Azure, GCP) and other services, enabling users to manage both cloud and on-premises resources.
- **Resource Management:** Define resources such as virtual machines, databases, and networking components. Terraform can create, update, and delete resources based on the configuration.
- **Execution Plans**: Terraform generates an execution plan before applying changes, showing what actions it will take. This allows users to review and understand changes before they are executed.

COMMON USE CASES:

- **Provisioning Cloud Resources**: Automate the setup of virtual machines, storage, databases, and networking in cloud environments.
- **Multi-Cloud Deployment**: Manage resources across multiple cloud providers from a single configuration file.
- **Infrastructure Automation:** Streamline the process of infrastructure updates, scaling, and configuration management.
- **Collaboration and Versioning:** Use version control systems (like Git) to track changes in infrastructure configurations.

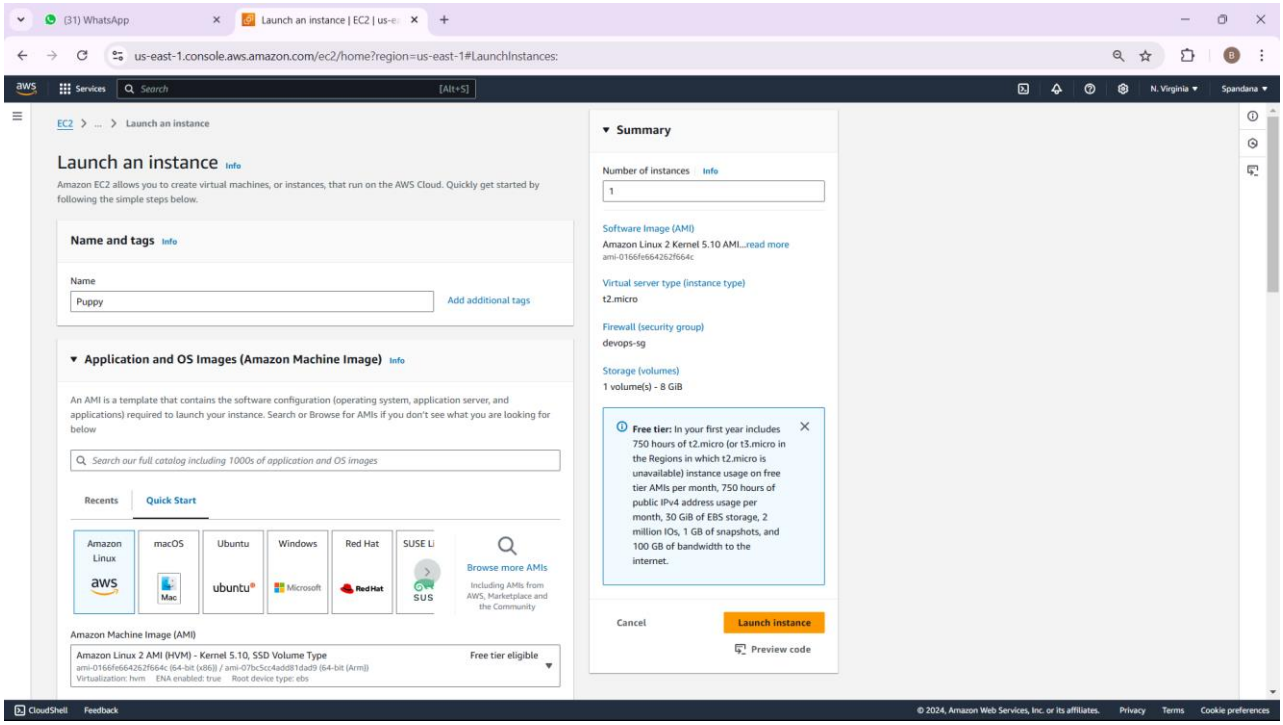
Terraform - infrastructure:

```
├── var.tf
├── vpc.tf
├── igw.tf
├── route_table_public.tf
├── ec2.tf
├── web_sg.tf
├── database_sg.tf
├── alb.tf
├── rds.tf
└── outputs.tf
```

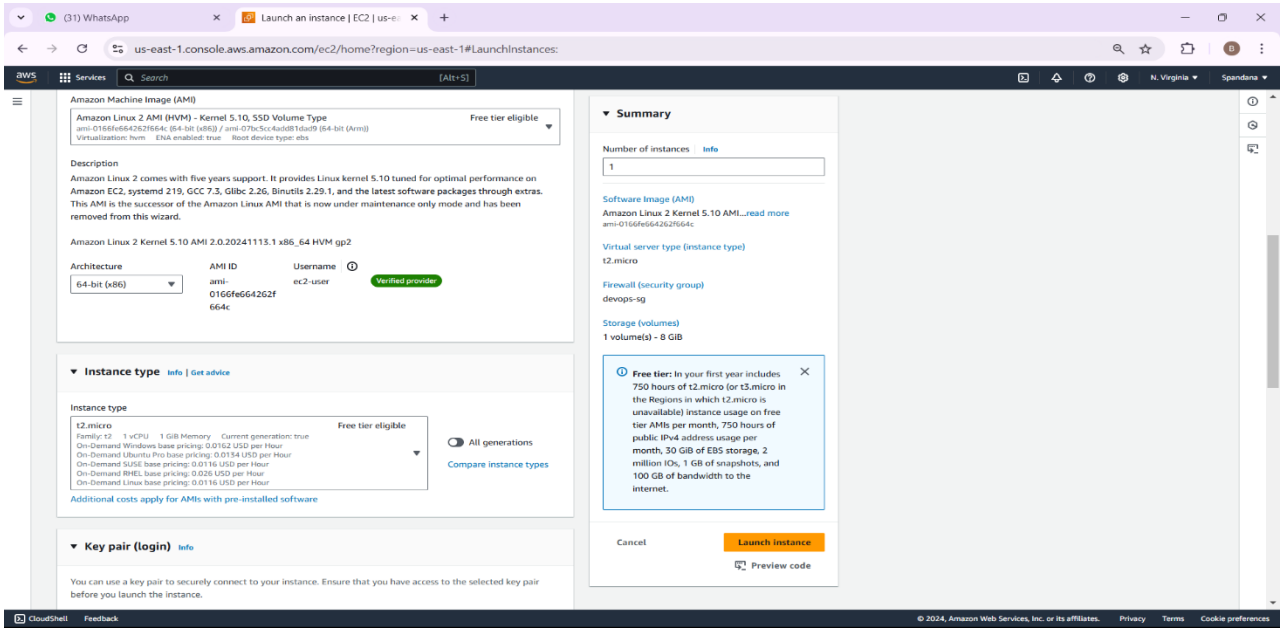
TERRAFORM COMMANDS:

- Terraform init
- Terraform validate
- Terraform plan
- Terraform apply
- Terraform show
- Terraform destroy

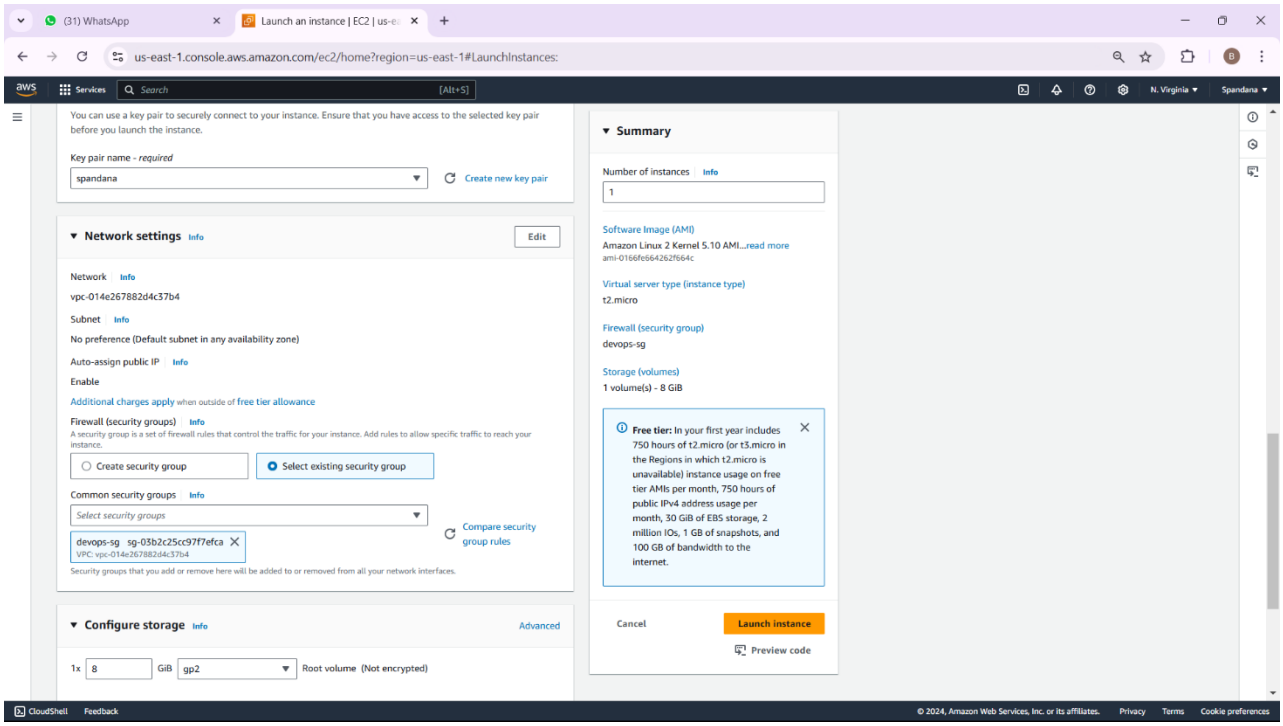
First step we need to create one instance using own configuration, On the EC2 Dashboard, click **Launch Instance**. Enter a name for your instance (e.g., "Puppy") .



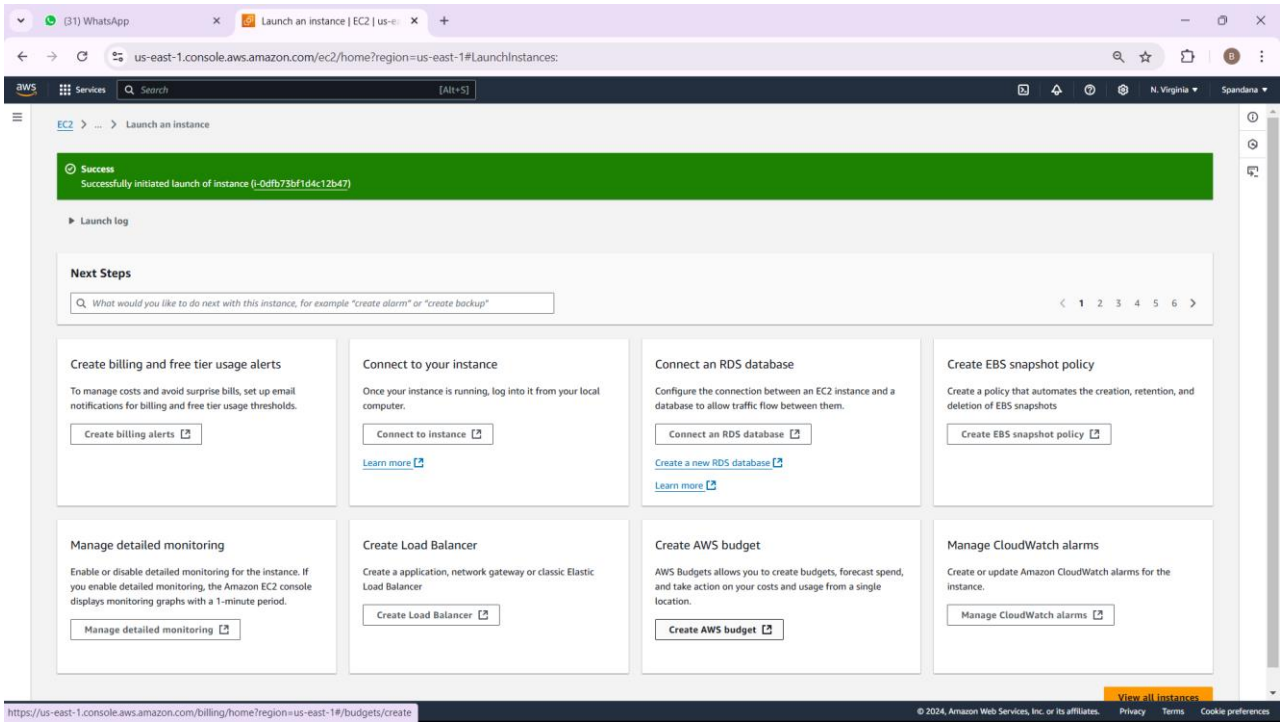
Choose an operating system for your instance, such as Amazon Linux, Ubuntu, or Windows Server based on our preference I have taken an amazon Linux, Select a free-tier eligible. Pick an instance type based on your requirements. For basic usage, choose **t2.micro**, which is free-tier eligible.



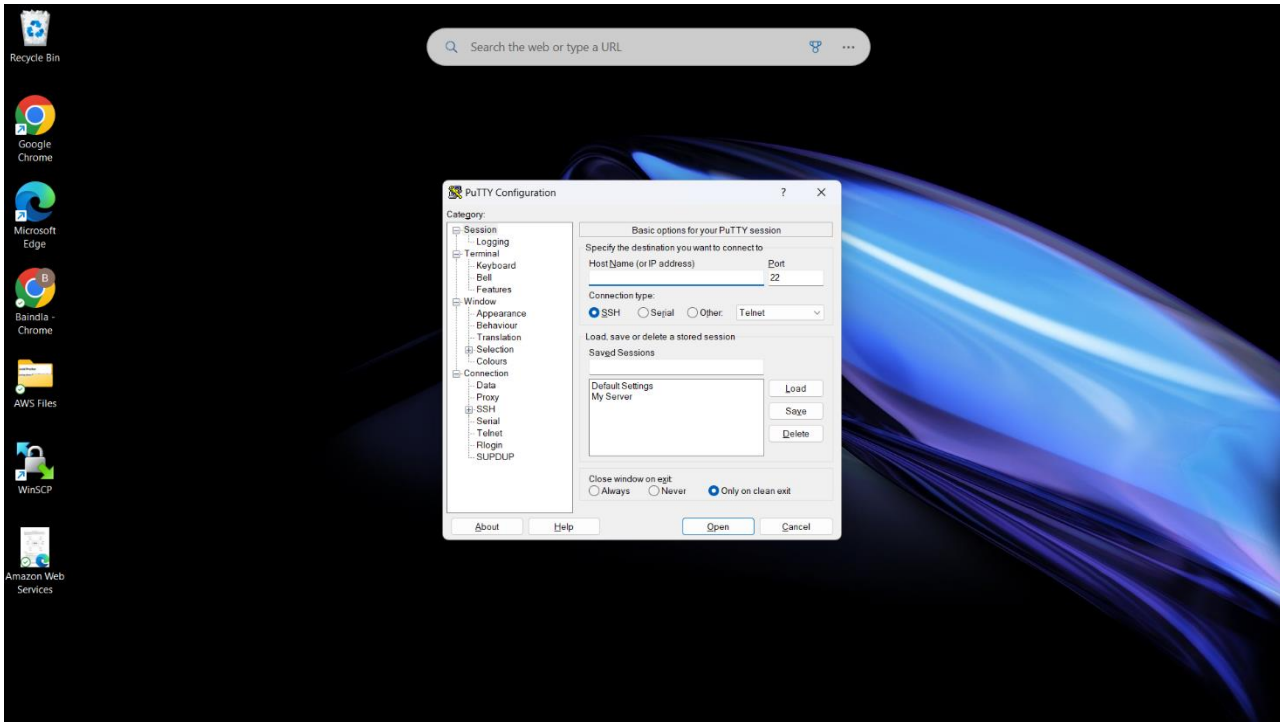
Select a key pair name if u don’t have an key pair we have one option on the right hand side click on that and create a new key pair. I already have one key pair so I selected that key pair.



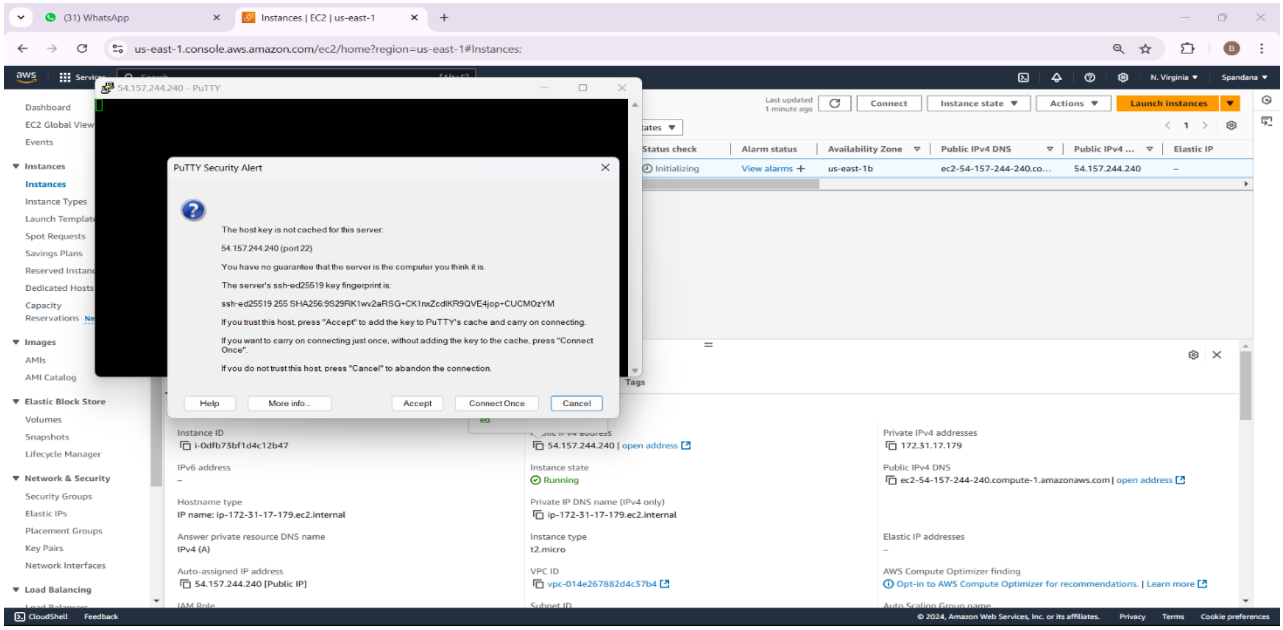
After configuring all the configurations then click on the launch instance.



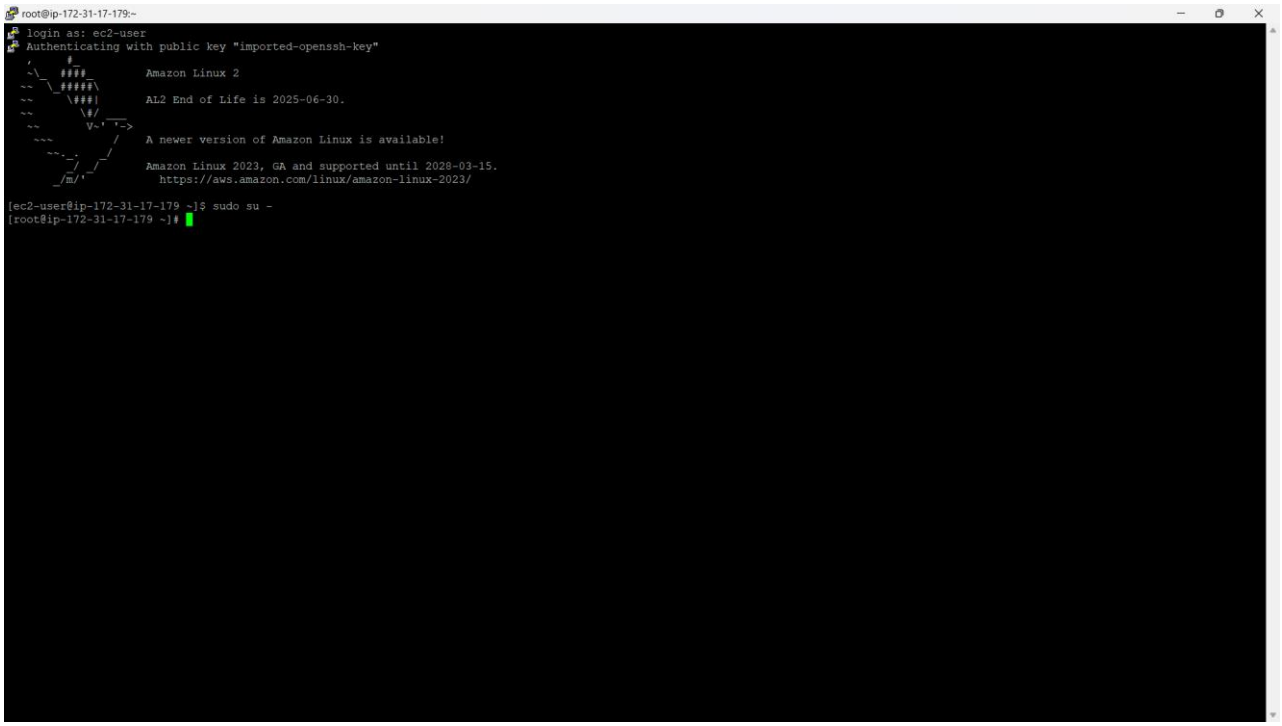
After creating an instance click on that instance in detail section we can able to see the public ip copy that public IP and open PuTTY. We can able to see basic option session of PuTTY in that we need to provide and Public IP or DNS of an instance.



After providing an public IP, we need to click on SSH we can able to see in the left side after that we can able to see Auth clicking on that we can able to see the credentials after clicking on that we need to provide .ppk file click on browse option directly it will navigate to the folder where we have saved .pk file select that file it directly open one terminal.

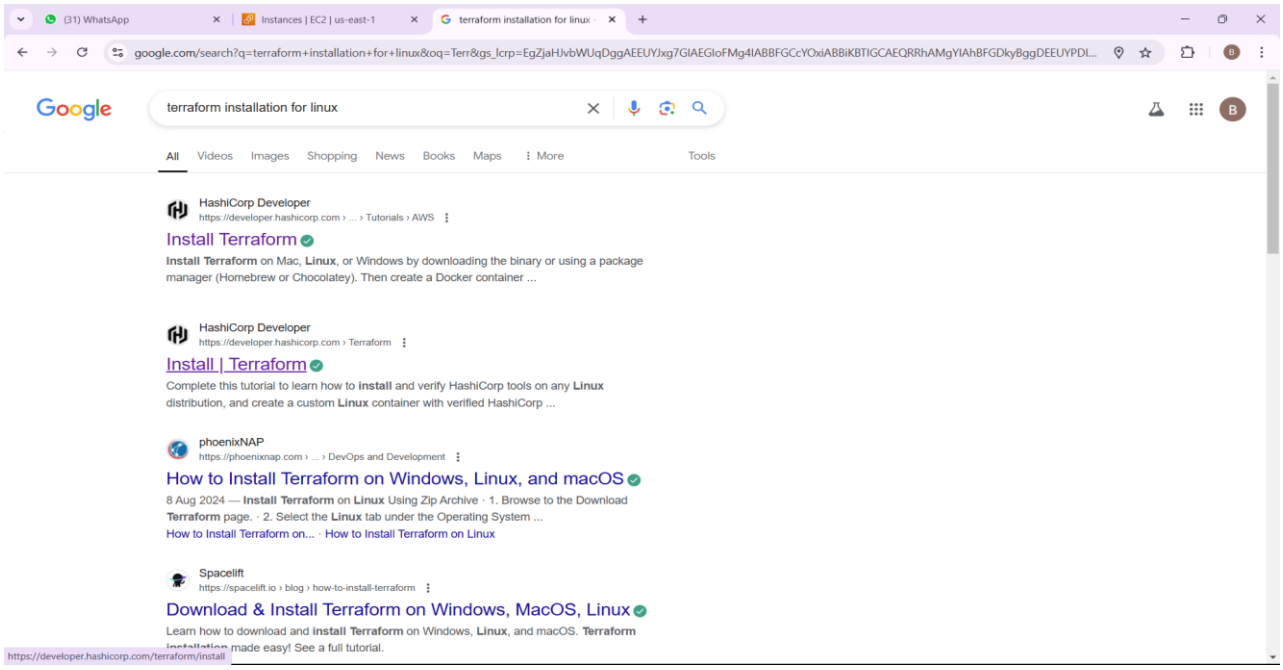


Click on accept button that shown in above page. We need to login as a ec2-user if we select amazon linux2 AMI, We need to login as based on our configuration of AMI.

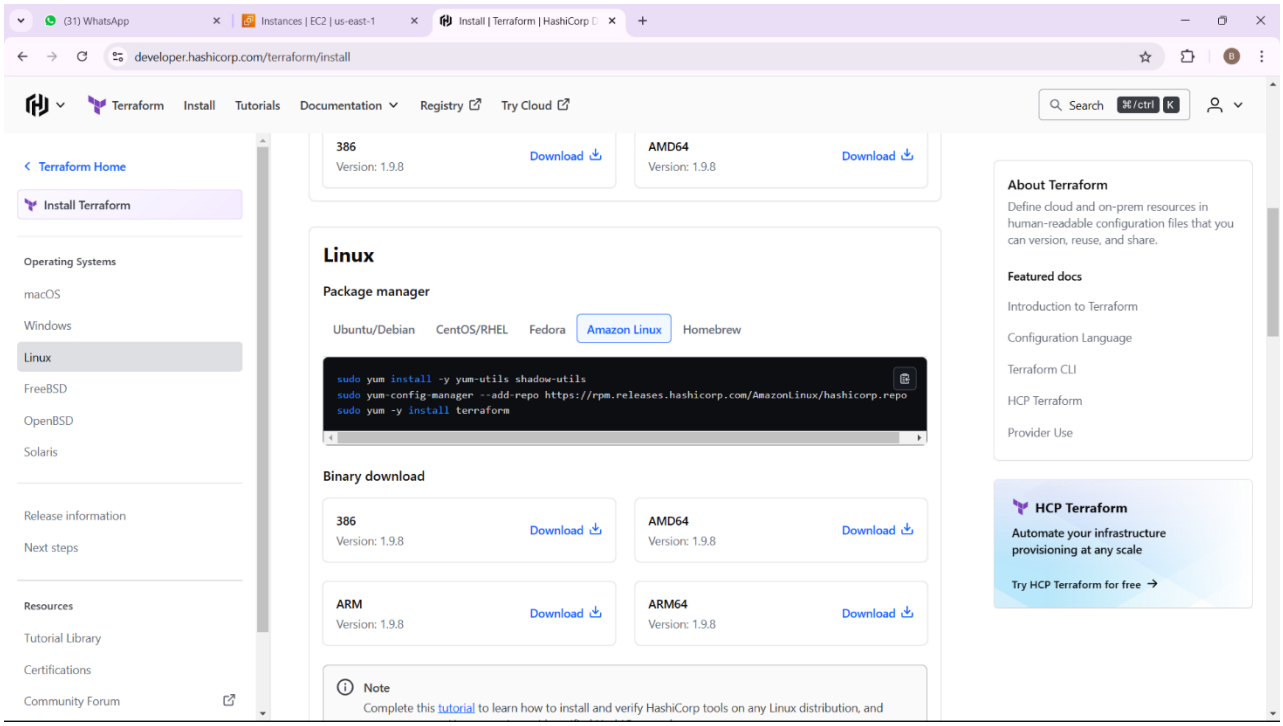


STEP 1: CREATE A VARIABLES

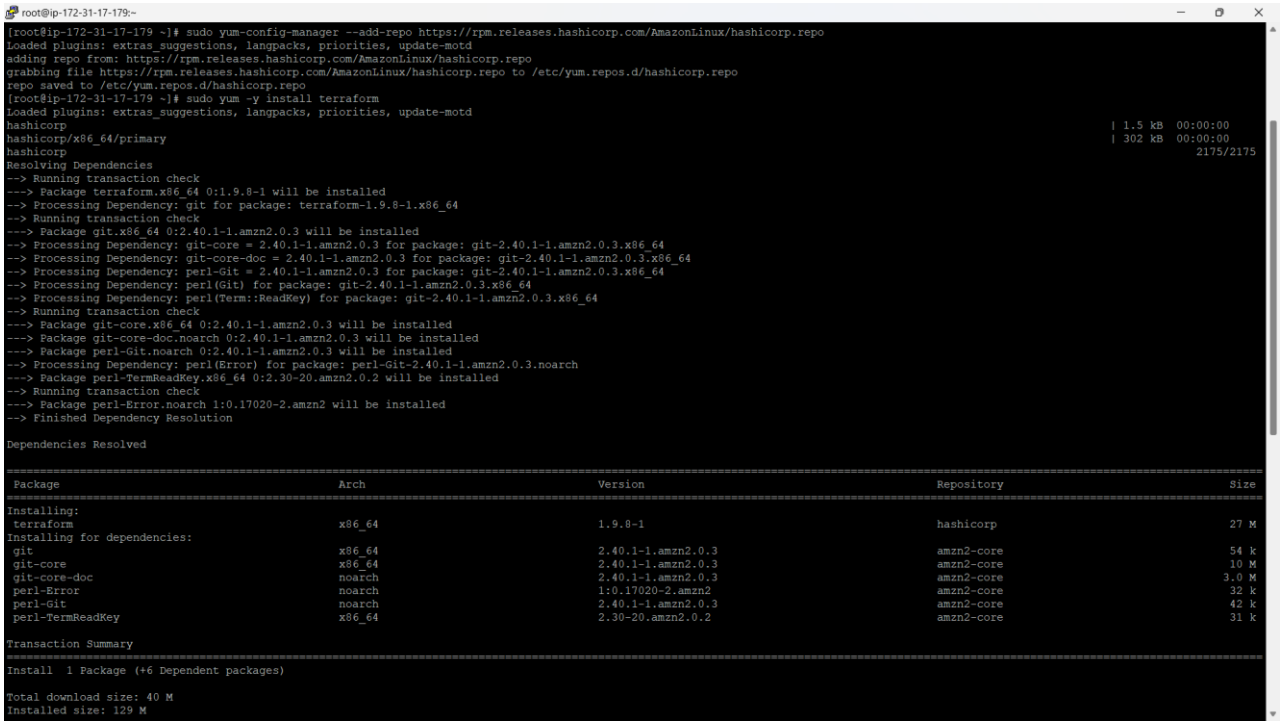
If we want to create an any file first we need to install an terraform in our local machine so for that we need to visit an official website as sown in below.

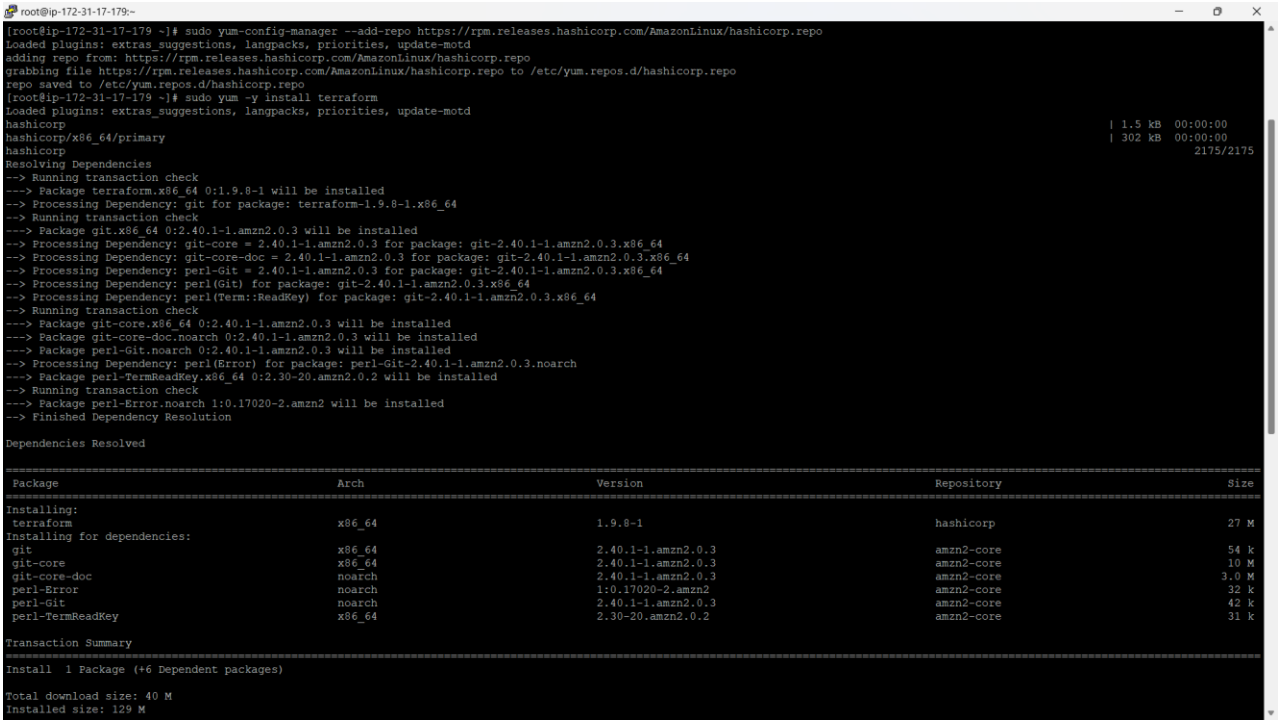


Go to that website and click on amazon Linux as shown in below image click on that we can able to see some commands run that commands in the terminal and terraform has been installed successfully.

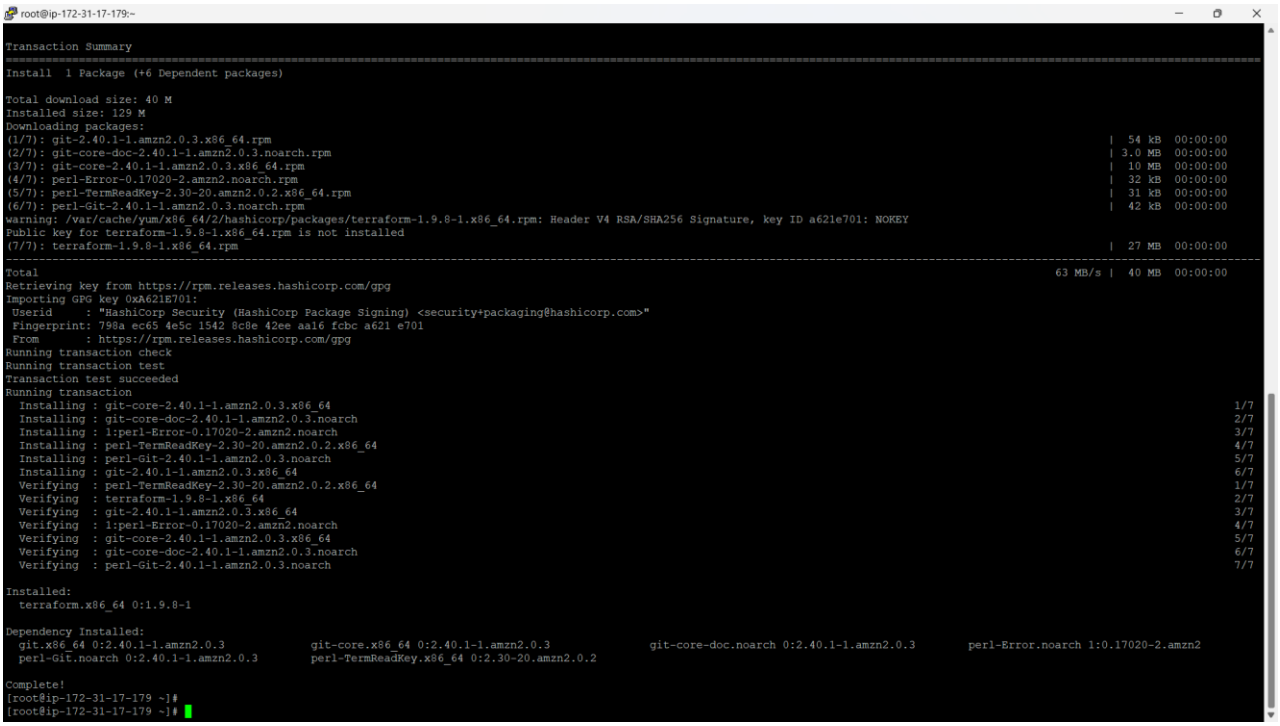


In the below image we can able to see that the downloading process of the terraform.





In the below image we can able to see that terraform installation completed successfully.



Keeps your configuration clean and organized. We need to create a variables file **vars.tf** which allows us to update values in one place without changing multiple files. Makes it easier to reuse configurations for different environments This setup makes our Terraform setup flexible and manageable, especially for large projects. After that we need to initiate the terraform then we need to run the **vars.tf** using **terraform plan**. After that we need to apply it using **terraform apply** command.

STEP 2: CREATING A FILE FOR VPC

To interact with AWS using Terraform, we need two key files: provider.tf and vpc.tf. These files ensure Terraform can authenticate to your AWS account and create the required resources, like a Virtual Private Cloud (VPC). The provider.tf file defines the provider (in this case, AWS) and sets up access credentials. It acts as a bridge between Terraform and AWS.

Note: Storing keys directly in the file is not secure. For production, use environment variables or AWS CLI credentials instead.

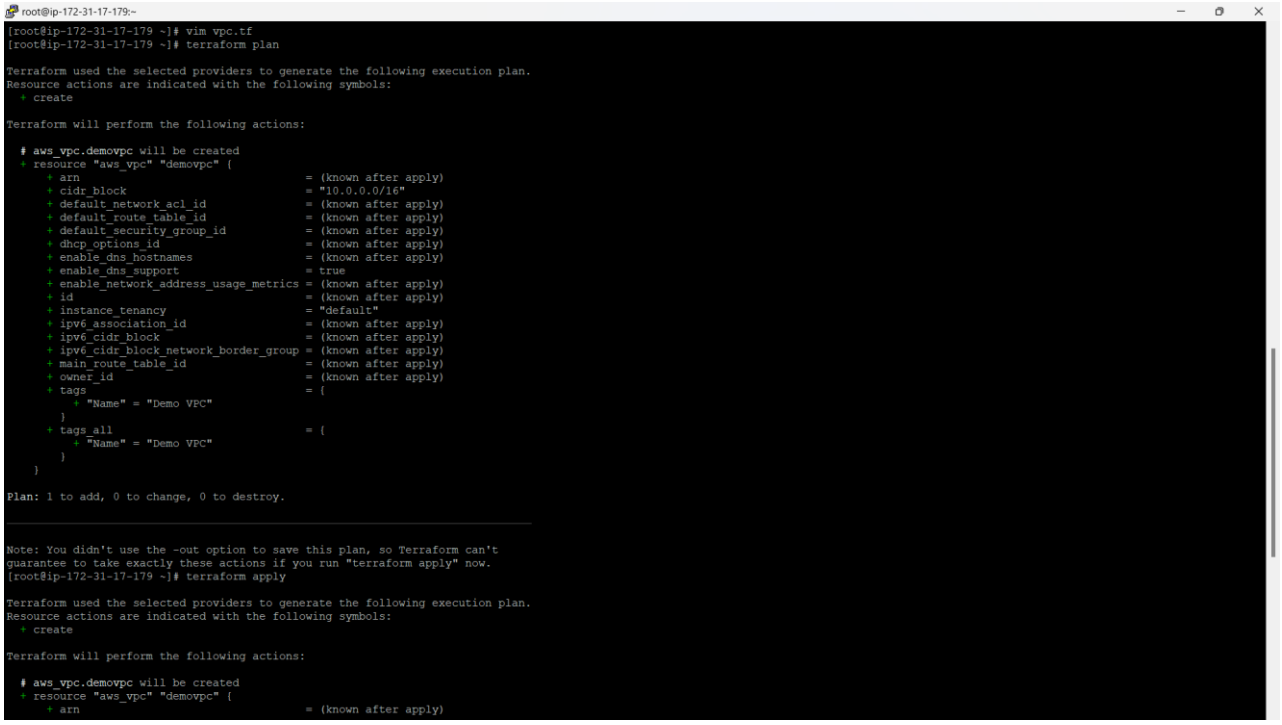
After providing credentials in CLI we can able to access the AWS account for creating an services using terraform.

If we want to create an VPC in the AWS account first we need to create an vpc.tf file using **vim vpc.tf** and then we need to write a script for the creation of the VPC.

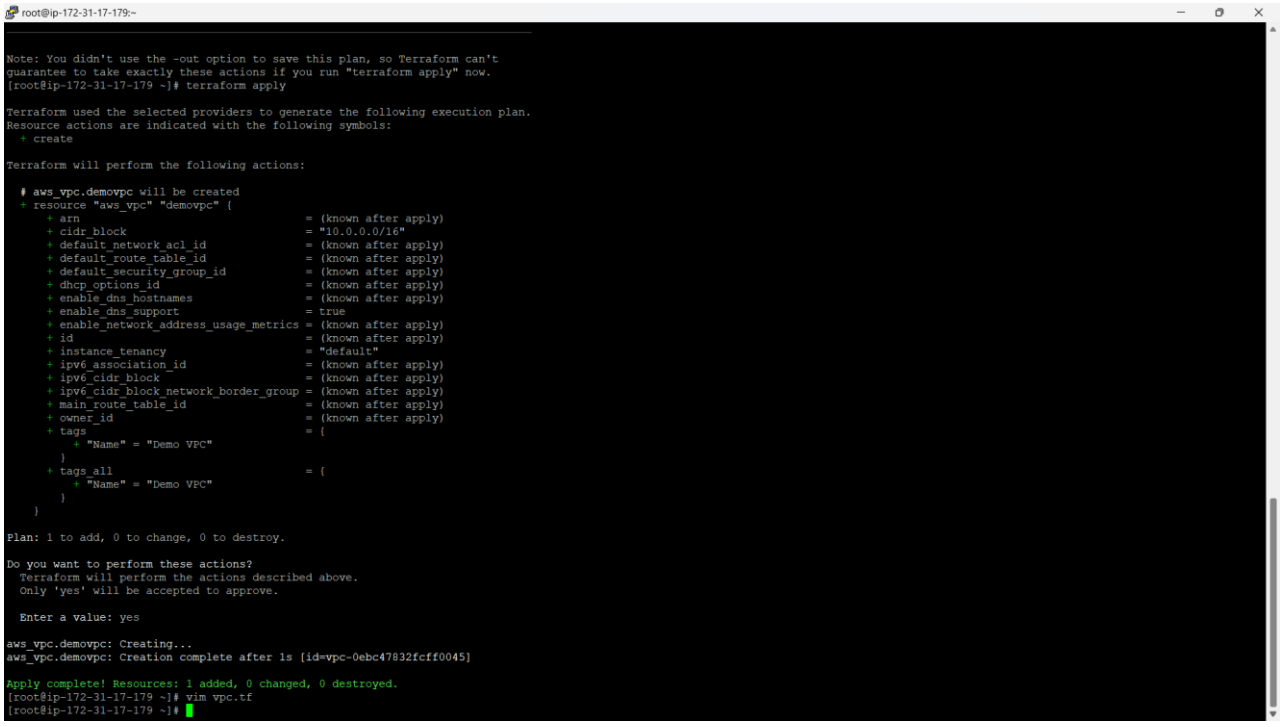
Specifies the resource type (aws_vpc) and a logical name (demovpc). cidr_block defines the range of IP addresses for the VPC (e.g., "10.0.0.0/16"). The value is dynamic and passed using a variable (var.vpc_cidr) to allow flexibility. instance_tenancy is set to "default", meaning EC2 instances launched in the VPC will share hardware with other AWS customers unless specified otherwise. The tags section assigns a name (Demo vpc) to the VPC for easy identification in the AWS Management Console.

After creating an vpc.tf file then we need to run the **terraform plan** command because the terraform plan command is used to preview the changes in Terraform will make to your infrastructure. It shows you what will be added, modified, or deleted based on your Terraform configuration files, without actually making any changes where Terraform tells you what it's about to do, and you can check if everything looks good before applying the changes within the file before applying changes.

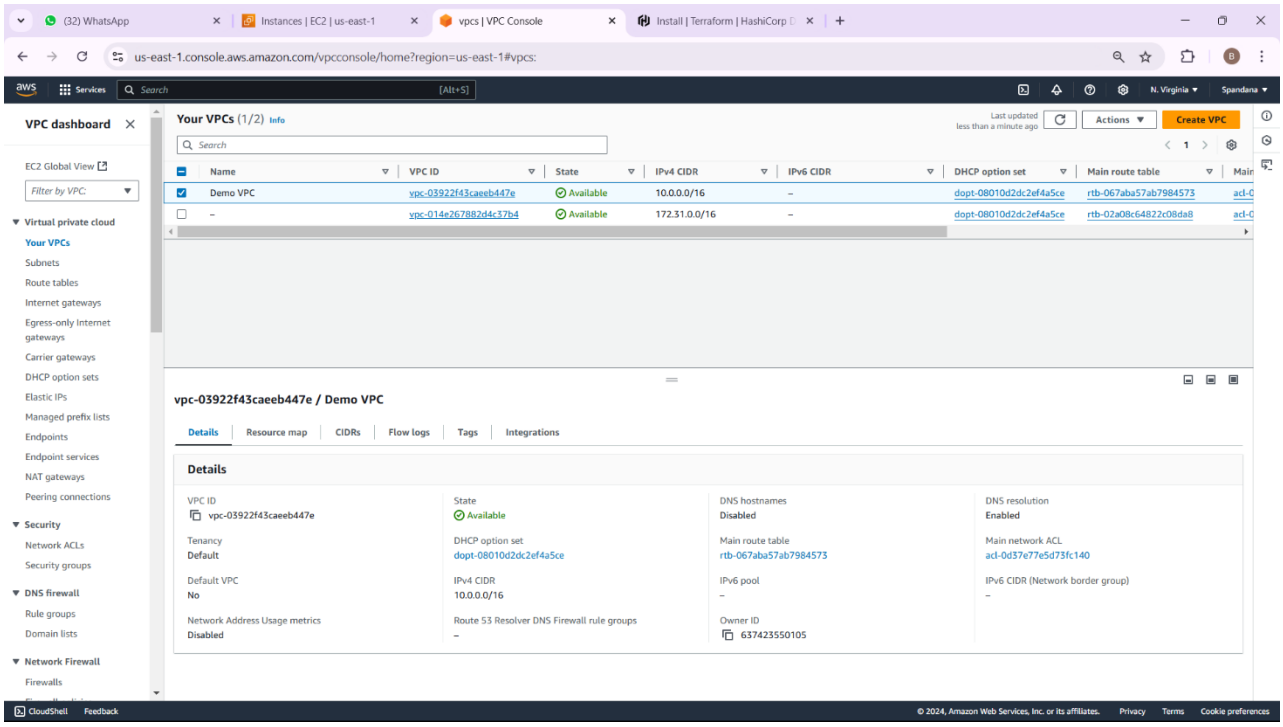
After running an **terraform plan** the below process will be run**.** Then we need to run **terraform apply** then that can be applied.



In the below image we can able to see the terraform apply has been completed successfully then go to the AWS account and check we can able to see VPC has been created successfully in our AWS account.



In the below image we can able to see that newly created VPC has been created.

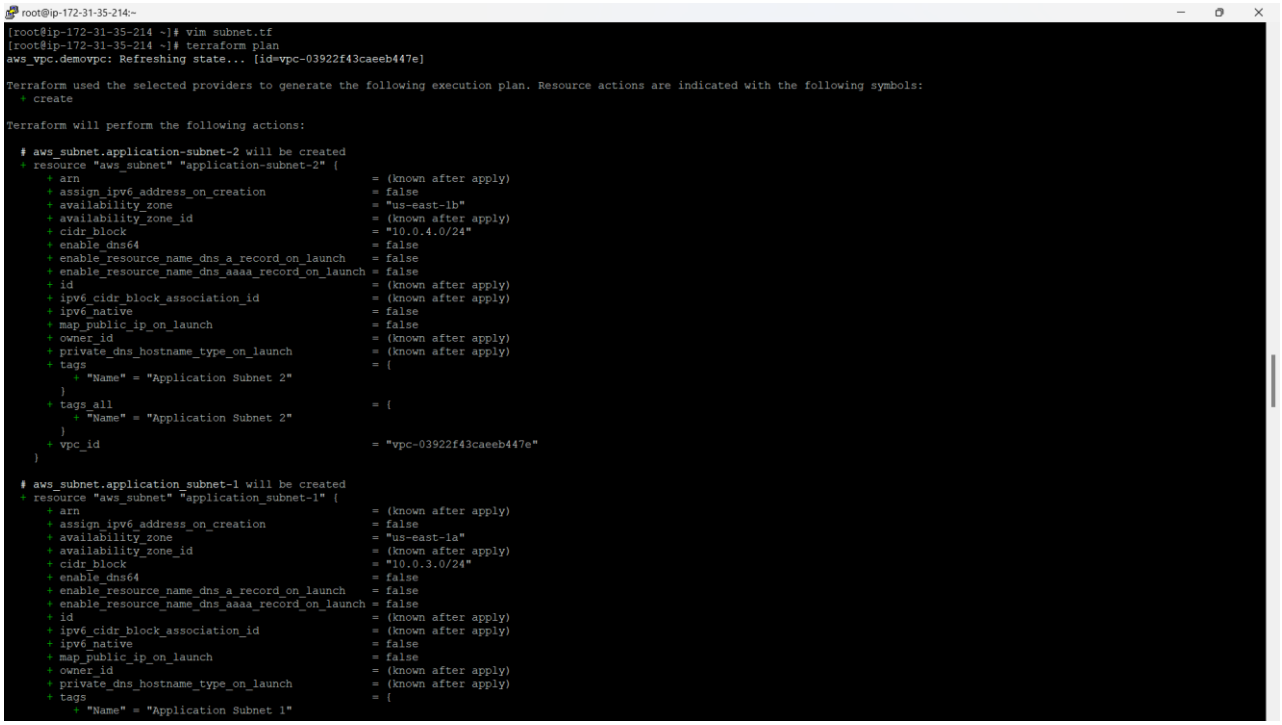


STEP 3: CREATING A FILE FOR SUBNETS

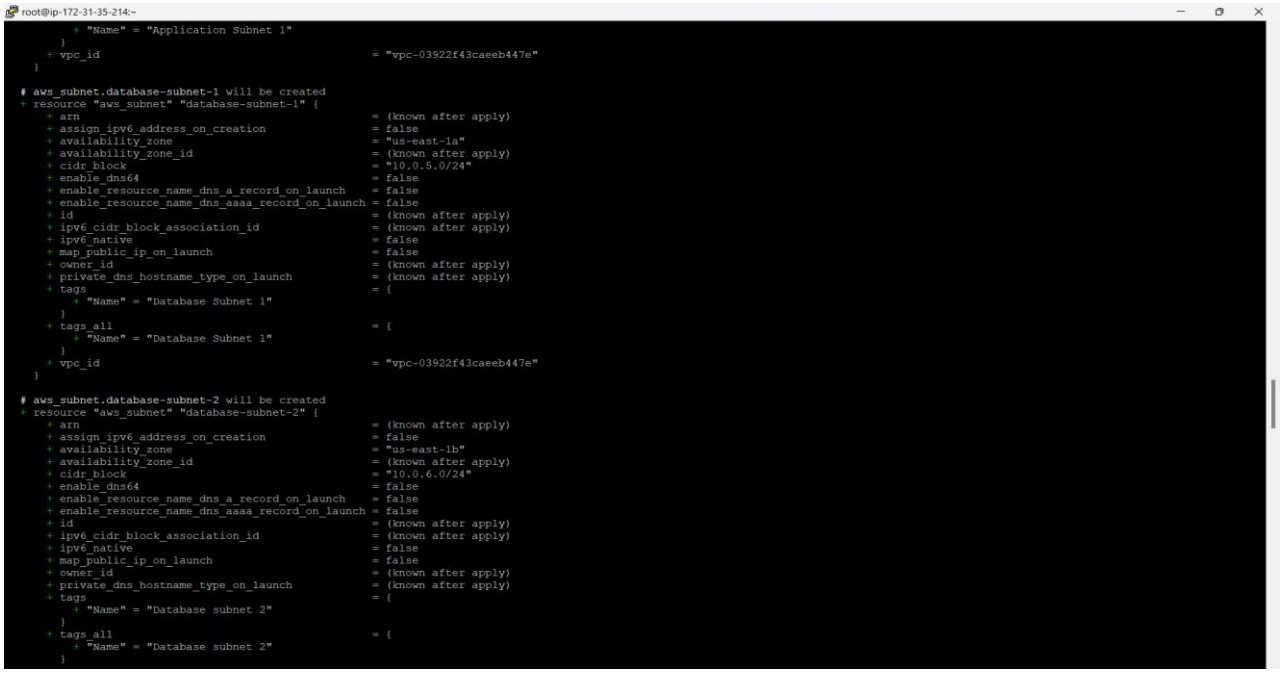
After creating the VPC, the next step is to create subnets within that VPC. I have created **subnet.tf** file for Subnets allow you to organize your network into smaller sections, each with its own CIDR block. I have created 6 subnets for an existing VPC. The subnet is linked to an existing VPC by specifying the vpc_id.

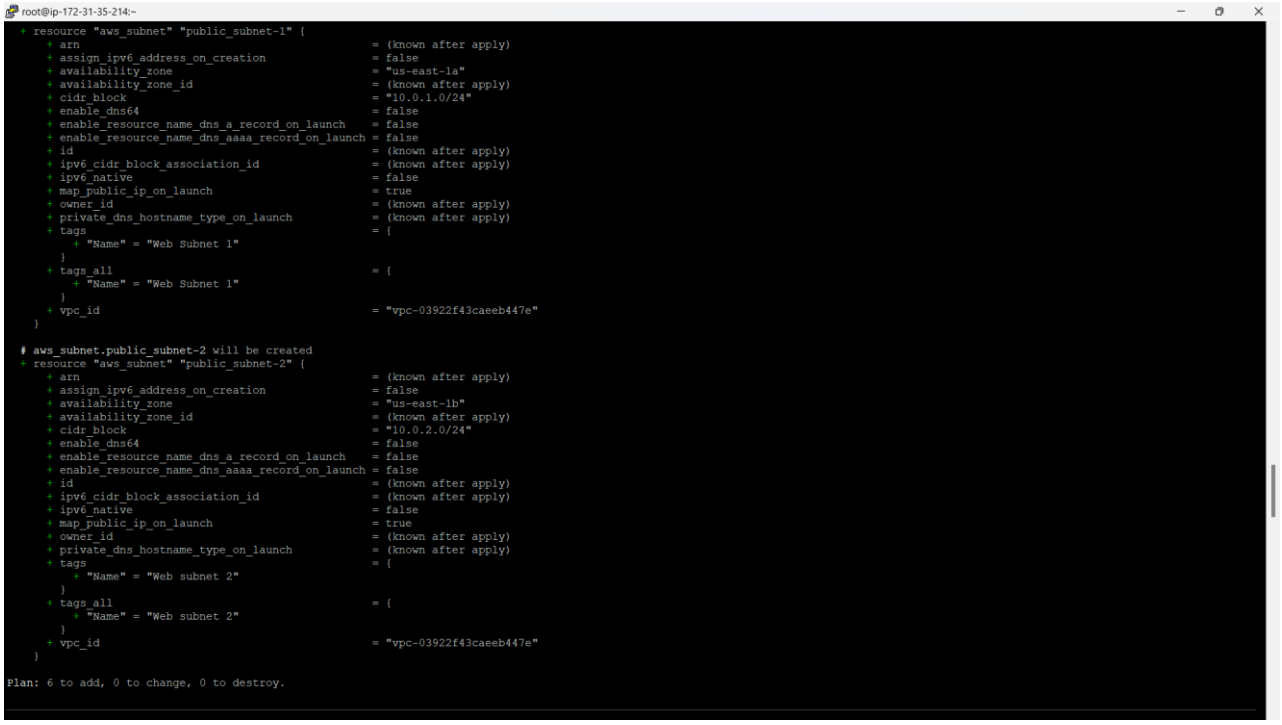
```
resource "aws_subnet" "public_subnet-1" {
vpc_id ="${aws_vpc.demovpc.id}"
cidr_block = "${var.subnet1_cidr}"
map_public_ip_on_launch = true
availability_zone = "us-east-1a"
tags={
Name="Web Subnet 1"
}
}
resource "aws_subnet" "public_subnet-2" {
  vpc_id                  = "${aws_vpc.demovpc.id}"
  cidr_block              = "${var.subnet2_cidr}"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1b"
  tags = {
    Name = "Web subnet 2"
  }
}
resource "aws_subnet" "application_subnet-1" {
  vpc_id                  = "${aws_vpc.demovpc.id}"
  cidr_block              = "${var.subnet3_cidr}"
  map_public_ip_on_launch = false
  availability_zone       = "us-east-1a"
  tags = {
    Name = "Application Subnet 1"
  }
}
resource "aws_subnet" "application-subnet-2" {
  vpc_id                  ="${aws_vpc.demovpc.id}"
  cidr_block              = "${var.subnet4_cidr}"
  map_public_ip_on_launch = false
  availability_zone       = "us-east-1b"
  tags = {
    Name = "Application Subnet 2"
  }
}
resource "aws_subnet" "database-subnet-1" {
  vpc_id            ="${aws_vpc.demovpc.id}"
  cidr_block        = "${var.subnet5_cidr}"
  availability_zone = "us-east-1a"
  tags = {
    Name = "Database Subnet 1"
  }
}
resource "aws_subnet" "database-subnet-2" {
  vpc_id            ="${aws_vpc.demovpc.id}"
  cidr_block        = "${var.subnet6_cidr}"
  availability_zone = "us-east-1b"
  tags = {
    Name = "Database subnet 2"
  }
```

Each subnet needs its own IP range, so you define a CIDR block (like 10.0.1.0/24). This places the subnet in a specific availability zone. This is set to true to assign public IPs to instances launched in this subnet, making it a public subnet. Adds a tag (Name = "My Subnet") for easier identification in the AWS console. After creating an subnet.tf file we need to run the terraform plan command.

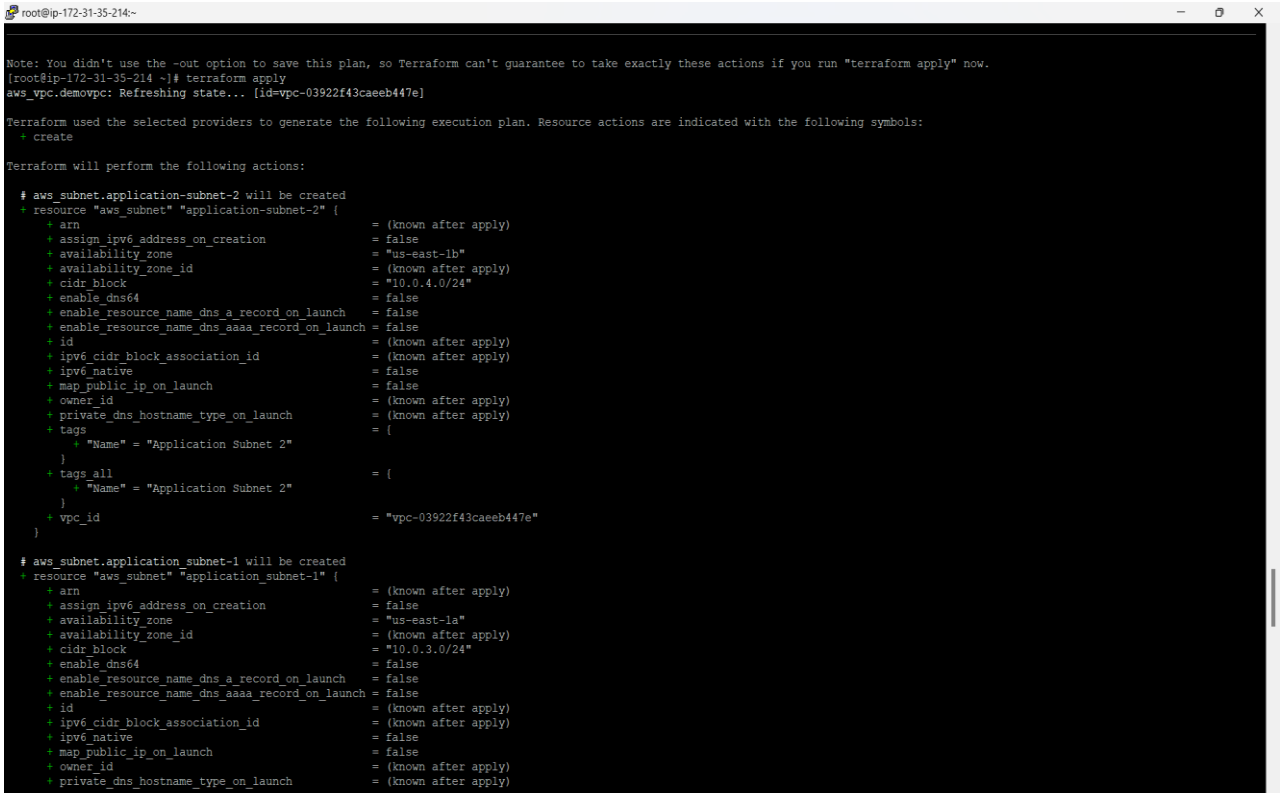


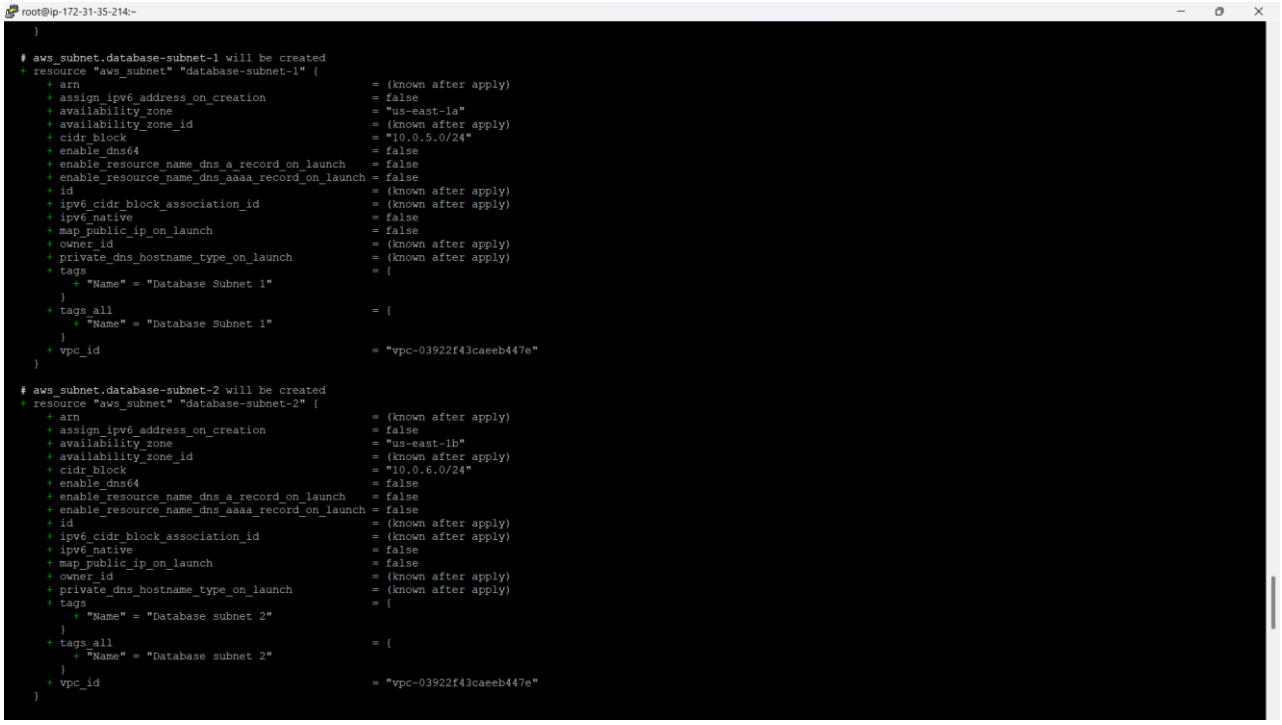
It will check every subnet configuration is correct or not.



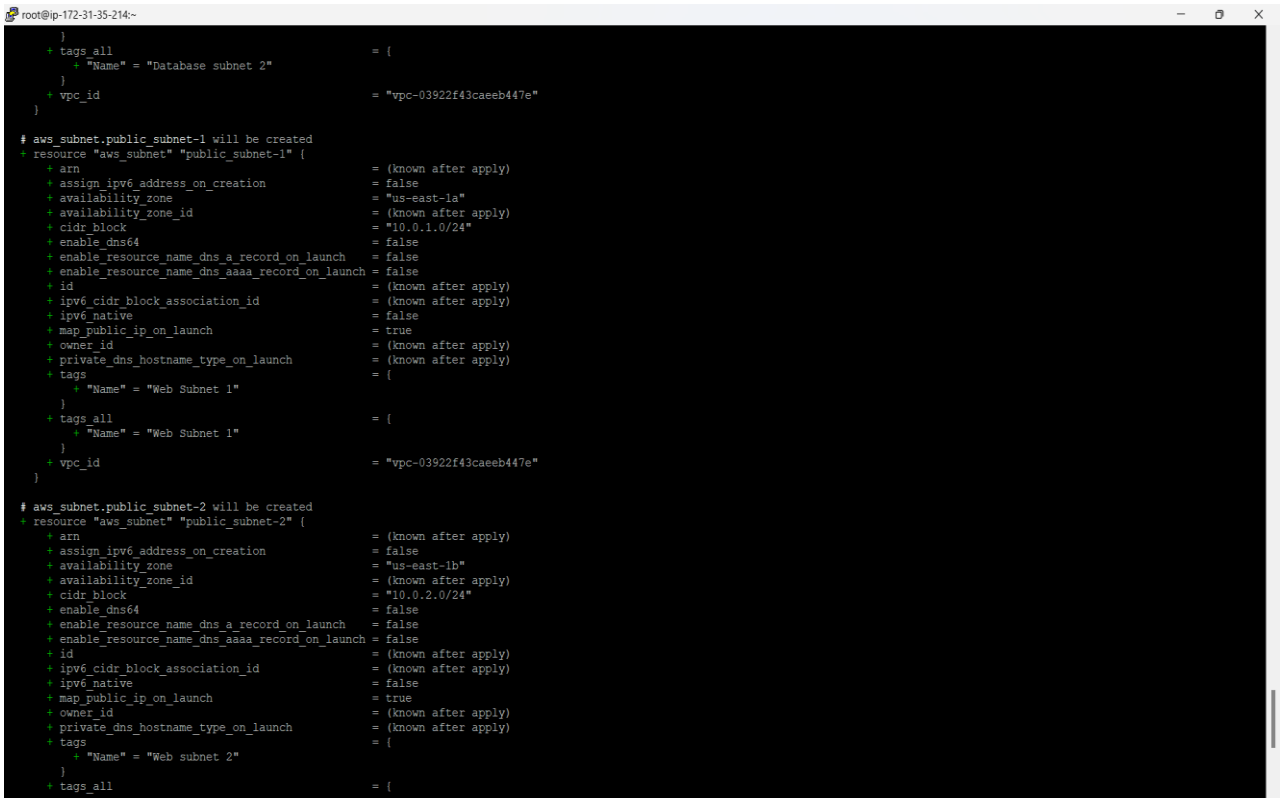


After successfully completion of terraform plan then we need to apply using terraform apply command we can see in the below image.



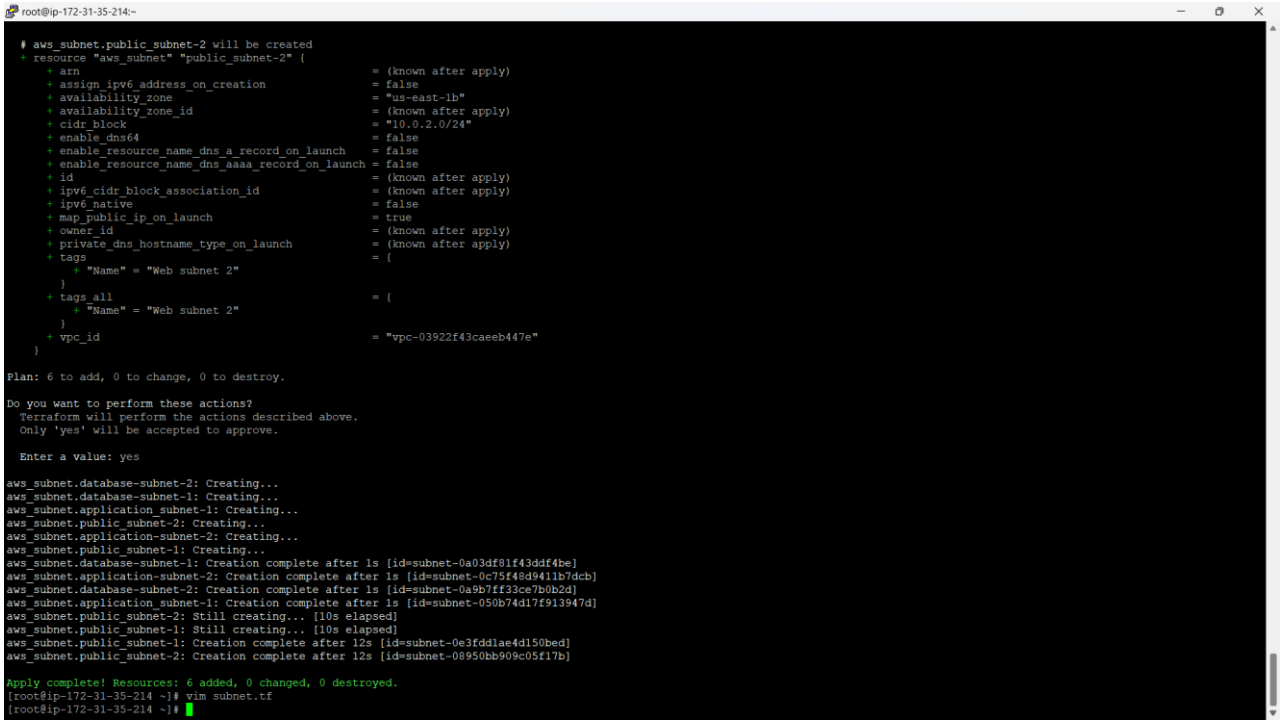


```
root@ip-172-31-35-214:~
  }

  # aws_subnet.database-subnet-1 will be created
  + resource "aws_subnet" "database-subnet-1" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1a"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.5.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags                                           = {
          + "Name" = "Database Subnet 1"
        }
      + tags_all                                       = {
          + "Name" = "Database Subnet 1"
        }
      + vpc_id                                         = "vpc-03922f43caeeb447e"
    }

  # aws_subnet.database-subnet-2 will be created
  + resource "aws_subnet" "database-subnet-2" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1b"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.6.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags                                           = {
          + "Name" = "Database subnet 2"
        }
      + tags_all                                       = {
          + "Name" = "Database subnet 2"
        }
      + vpc_id                                         = "vpc-03922f43caeeb447e"
    }
```

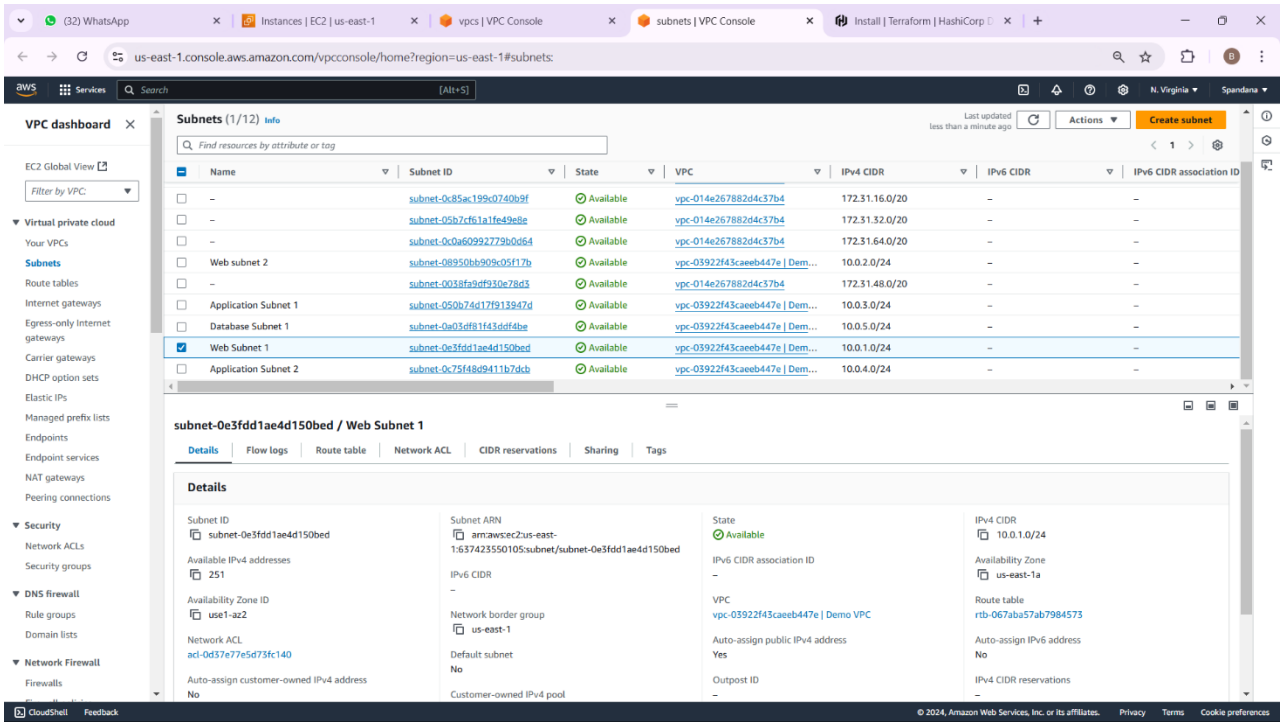


```
root@ip-172-31-35-214:~
      }
    + tags_all                                       = {
        + "Name" = "Database subnet 2"
      }
    + vpc_id                                         = "vpc-03922f43caeeb447e"
  }

  # aws_subnet.public_subnet-1 will be created
  + resource "aws_subnet" "public_subnet-1" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1a"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.1.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = true
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags                                           = {
          + "Name" = "Web Subnet 1"
        }
      + tags_all                                       = {
          + "Name" = "Web Subnet 1"
        }
      + vpc_id                                         = "vpc-03922f43caeeb447e"
    }

  # aws_subnet.public_subnet-2 will be created
  + resource "aws_subnet" "public_subnet-2" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1b"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.2.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = true
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags                                           = {
          + "Name" = "Web subnet 2"
        }
      + tags_all                                       = {
```

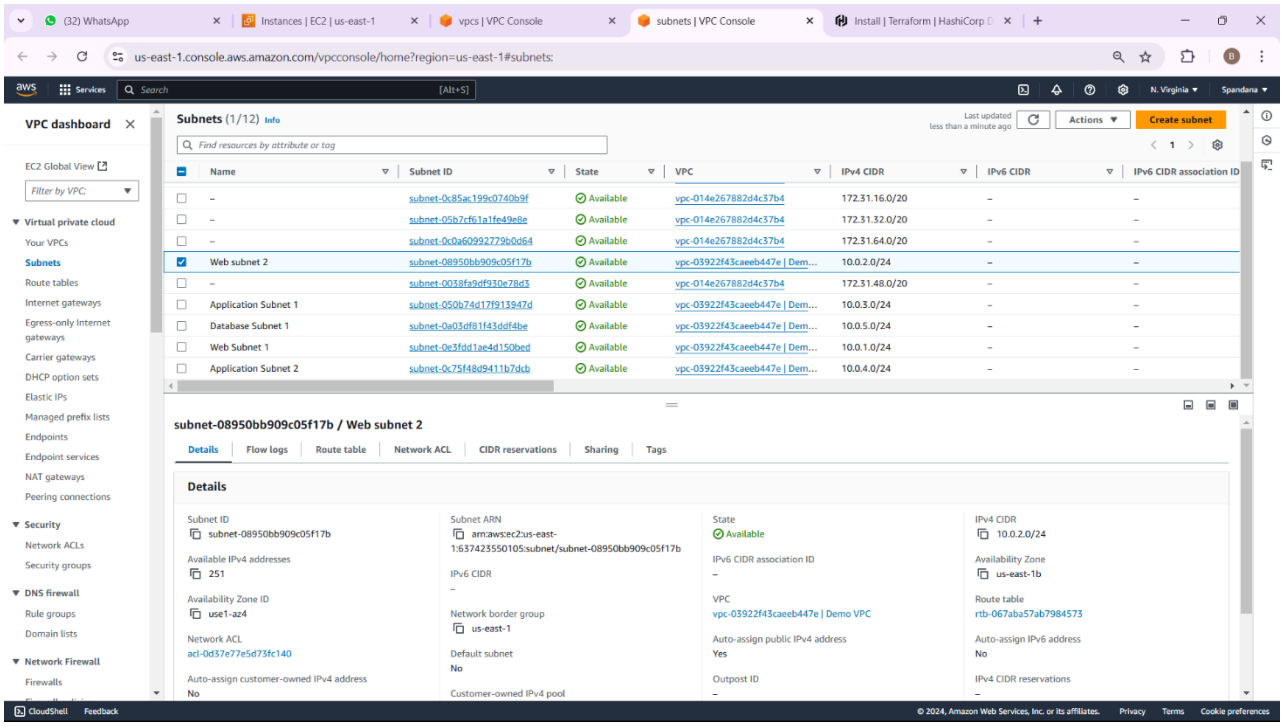
In the below image we can able to see that terraform apply was successful. Then we need to go AWS account and search for VPC in that we can able to see subnets click on that and check in the subnets.



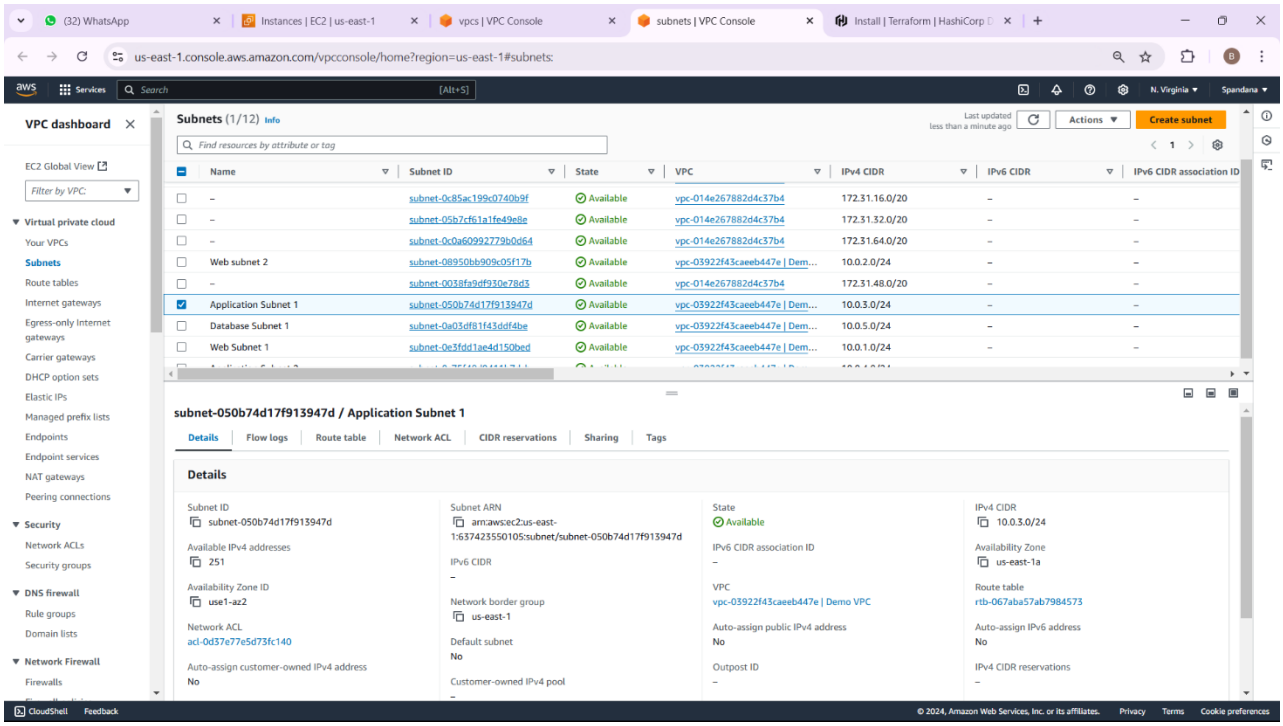
We can able to see that our first web subnet in the below image.



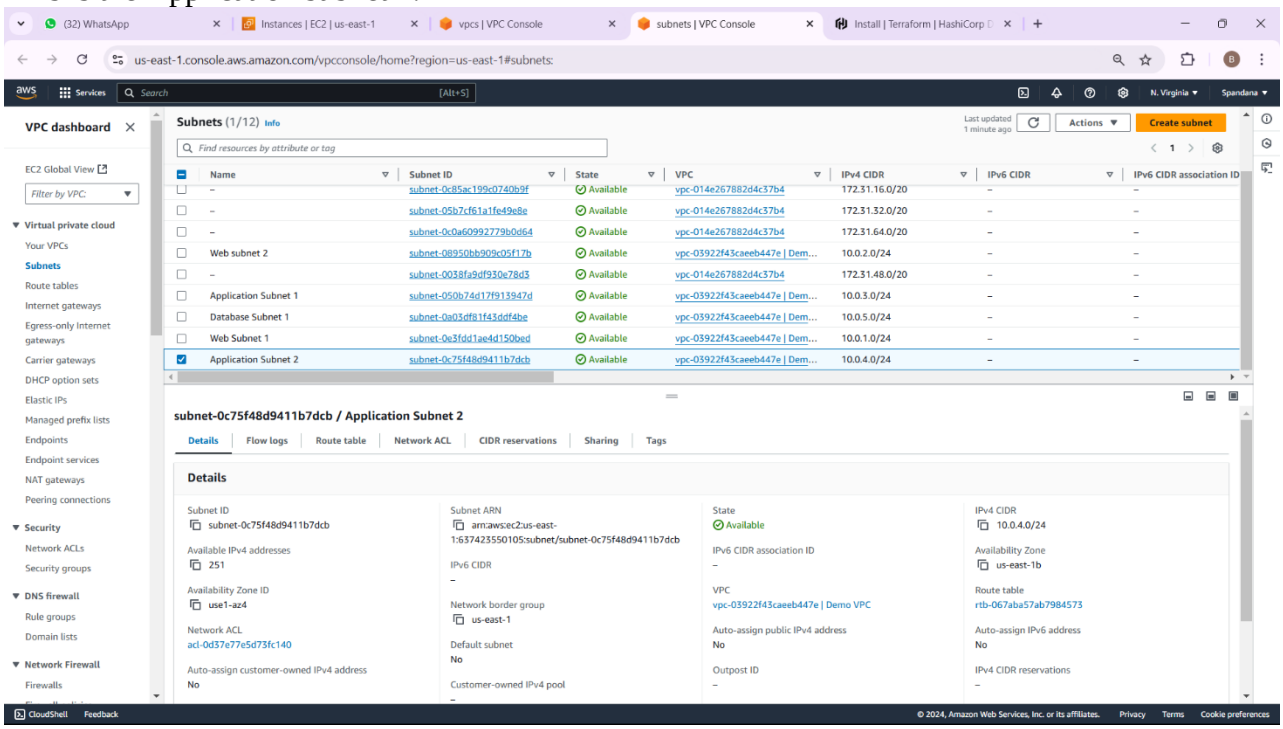
This is the Web subnet 2.



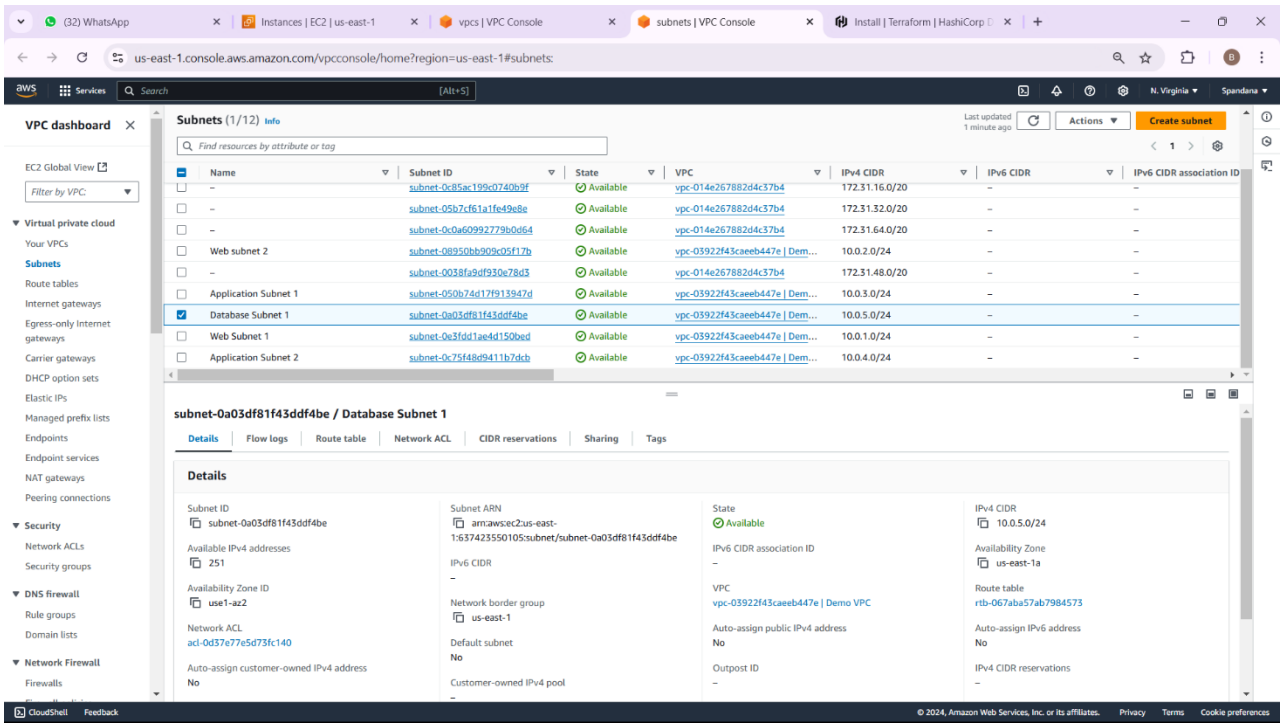
Application subnet 1 has been created successfully as shown in below image.



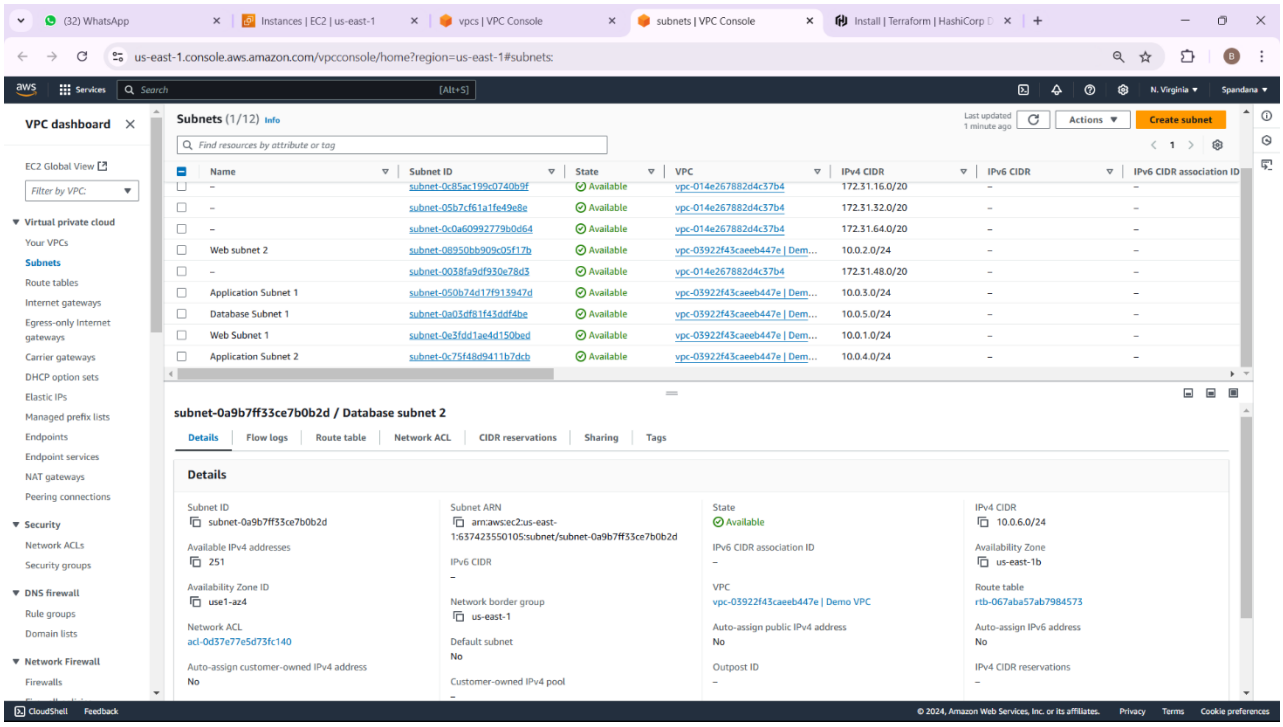
This is the Application subnet 2.



In the below image we can able to see Database subnet 1.



This is the Database Subnet 2.



STEP 4: CREATE INTERNET GATE WAY

To create an Internet Gateway in Terraform, you need to define it in a file, such as igw.tf. This file contains the code to create the Internet Gateway and attach it to your VPC. The Internet Gateway acts as a connection point, allowing instances in your VPC to communicate with the internet.

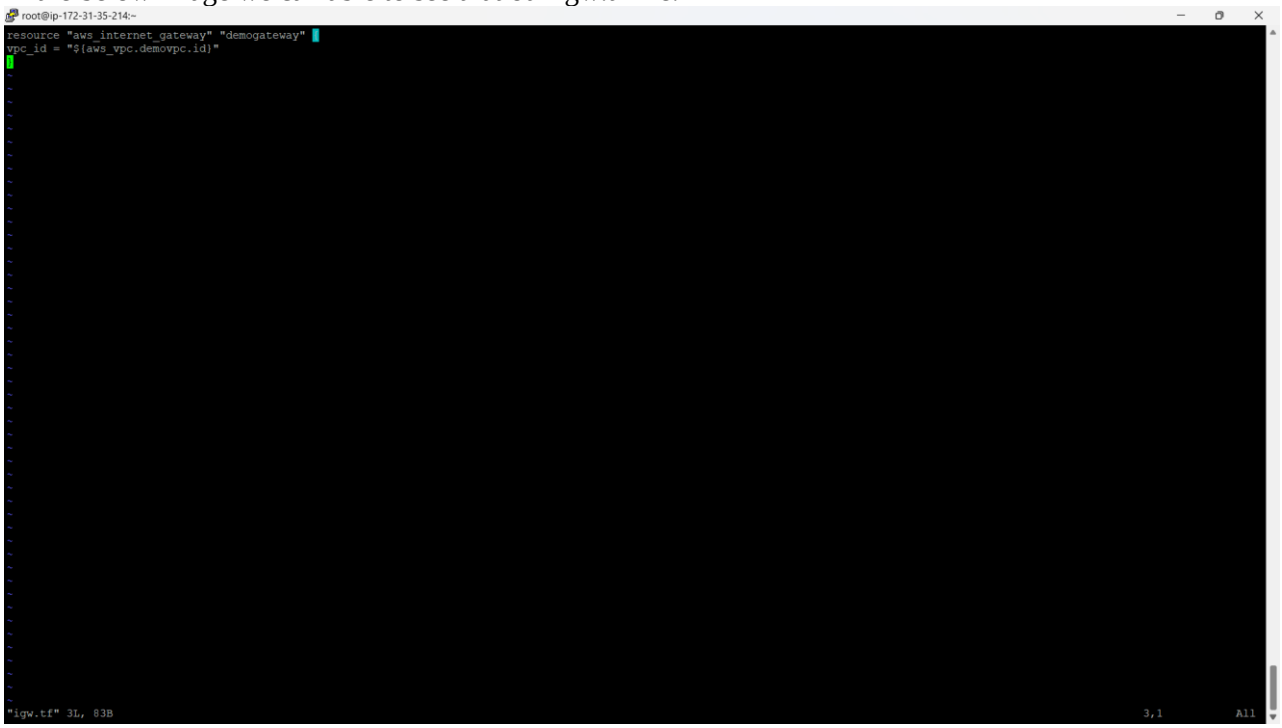
1. **aws_internet_gateway**:

   This is the Terraform resource used to create the Internet Gateway.
2. **vpc_id**:

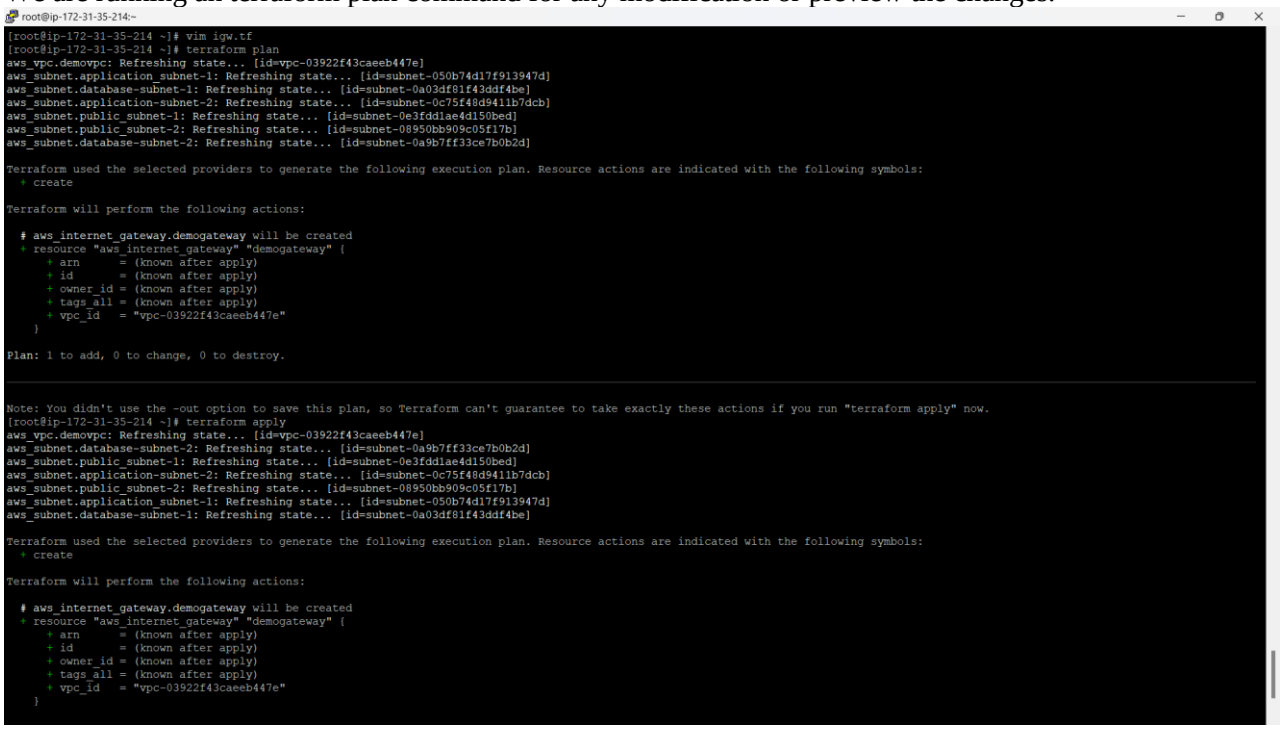
   This connects the Internet Gateway to the specific VPC you've already created (referenced as aws_vpc.my_vpc.id).
3. **Tags**:

   Adding a Name tag makes it easier to find and manage the Internet Gateway in the AWS Console.

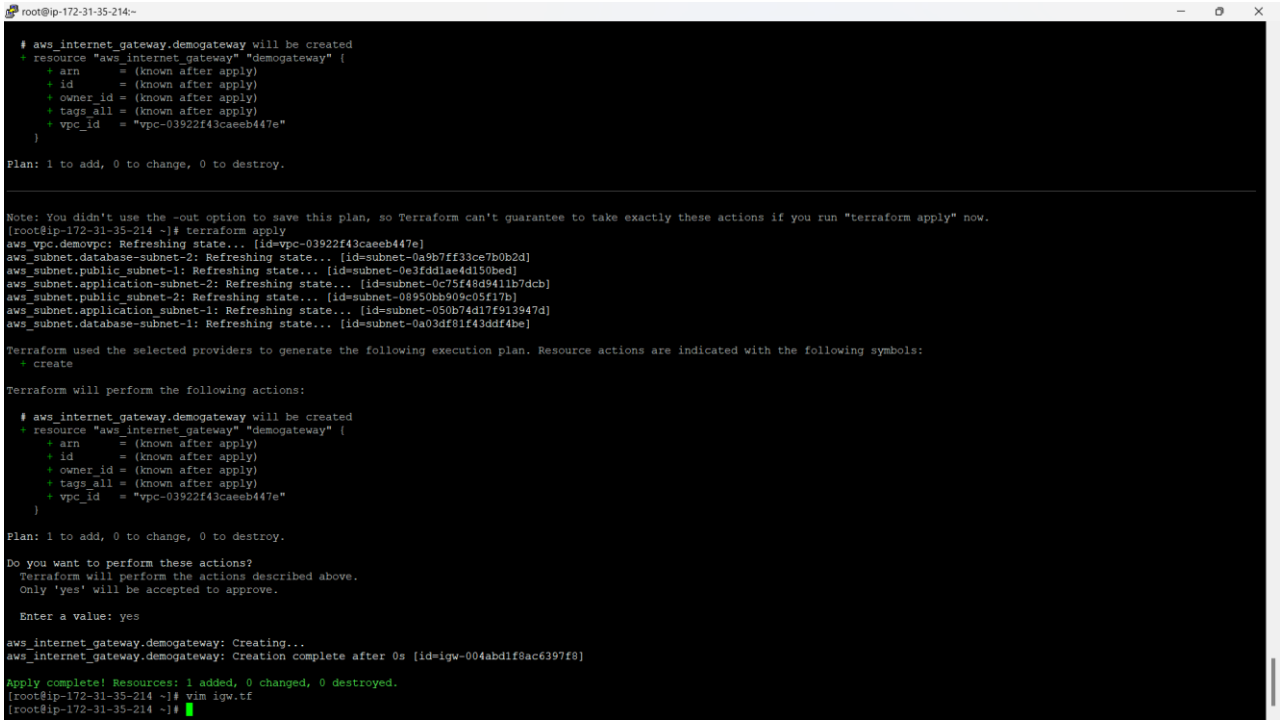
In the below image we can able to see that our igw.tf file.



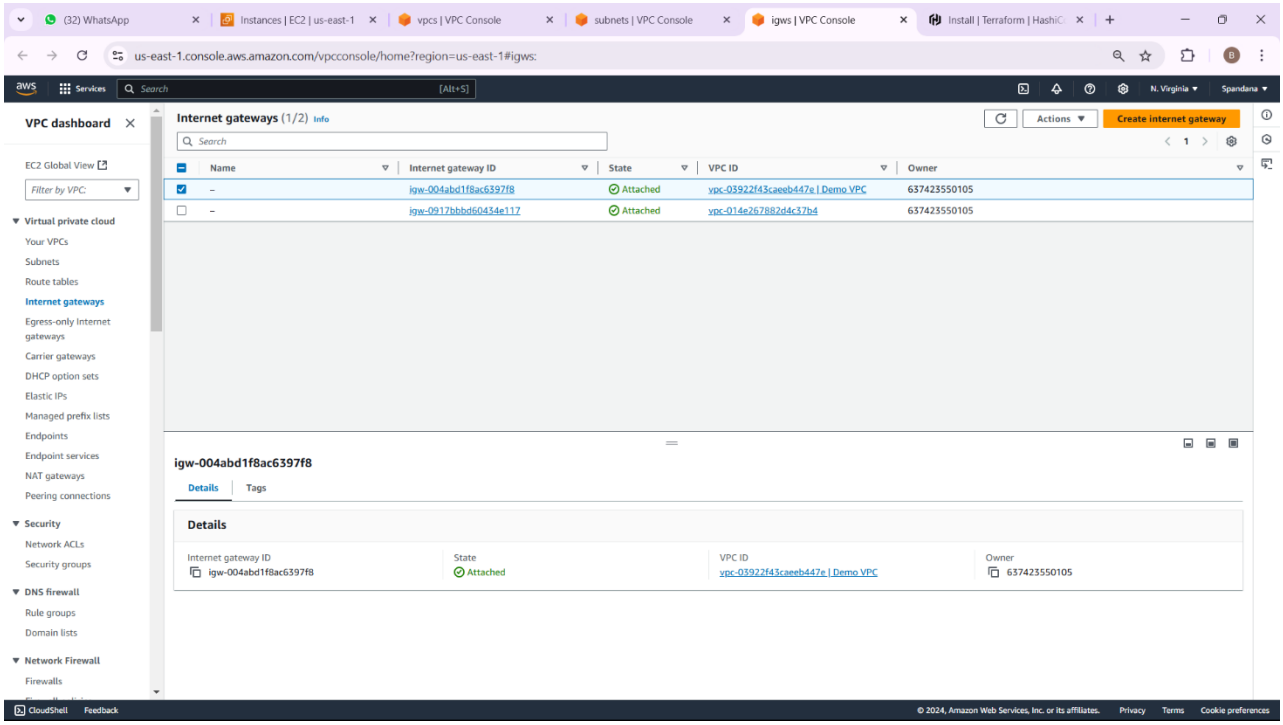
We are running an terraform plan command for any modification or preview the changes.



In the below image we can able to see that **terraform apply** for the applying the changes to create a internet gateway.



In the below image we can able to see that our Internet gateway has been created successfully and attached to VPC.



STEP 5: CREATING THE ROUTING TABLE

A Route Table is like a map that guides the flow of network traffic within your VPC (Virtual Private Cloud) in AWS. It decides where data should go when it leaves a resource, such as an EC2 instance. For example, it can direct traffic to the internet, other subnets, or a VPN connection. I have created an **route_table_public.tf** file to create a route table.

0.0.0.0/0 → Internet Gateway (for internet access).

Subnet CIDRs → Local traffic within the VPC.

Private IP ranges → NAT Gateway for outbound internet access without exposing resources.

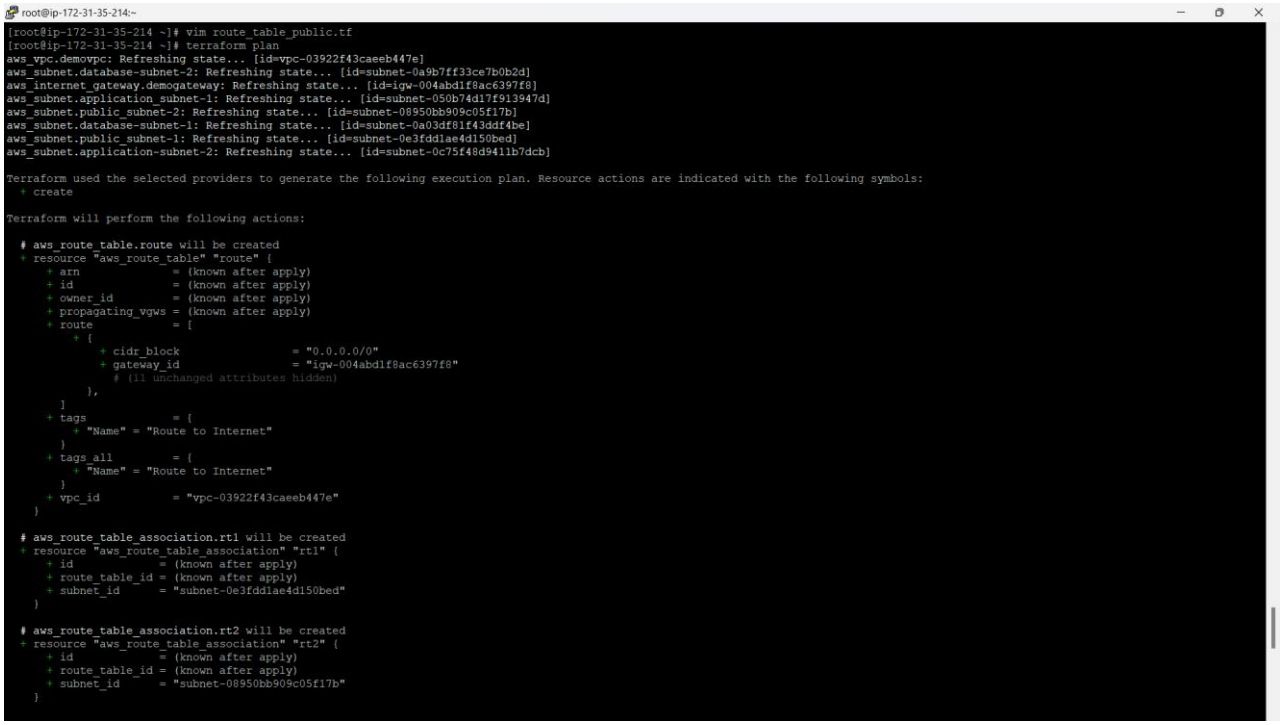
Control Traffic Flow: Ensures data reaches the correct destination.

Enables Internet Access: Connects subnets to the internet or other networks.

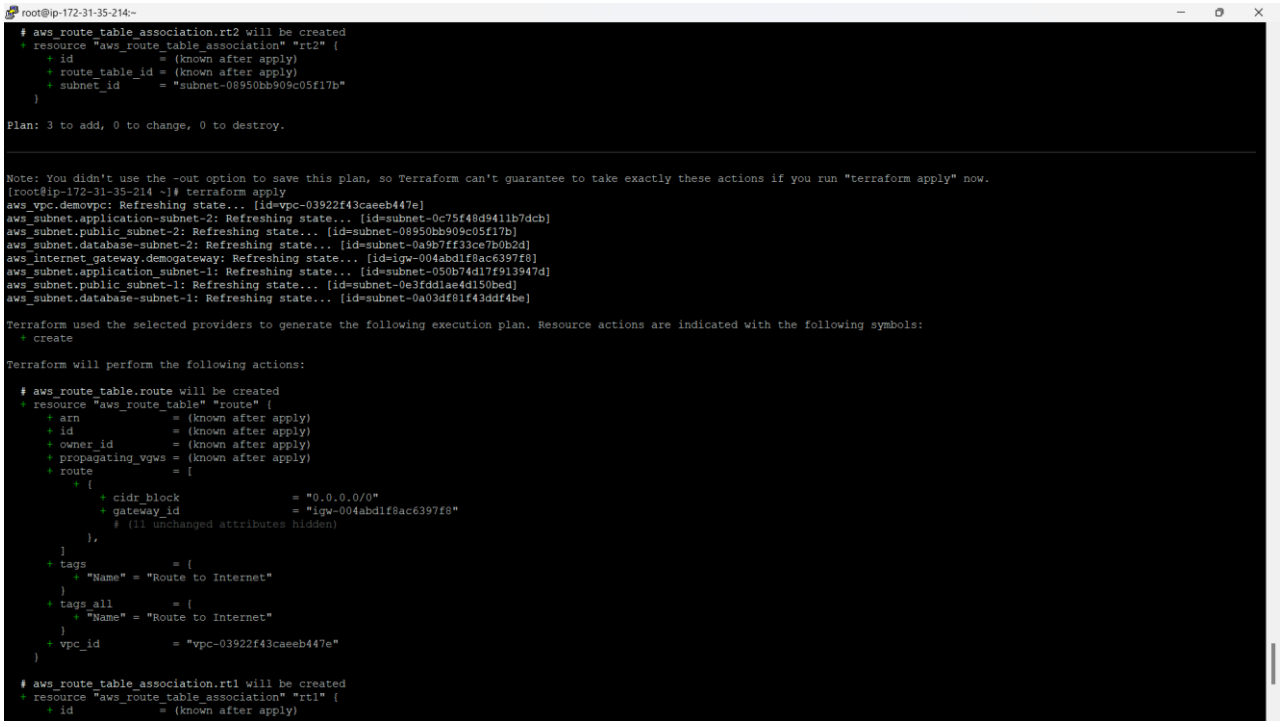
Network Segmentation: Helps isolate and secure parts of your network.

In simple terms, the route table is the brain behind how network traffic moves in and out of your VPC!

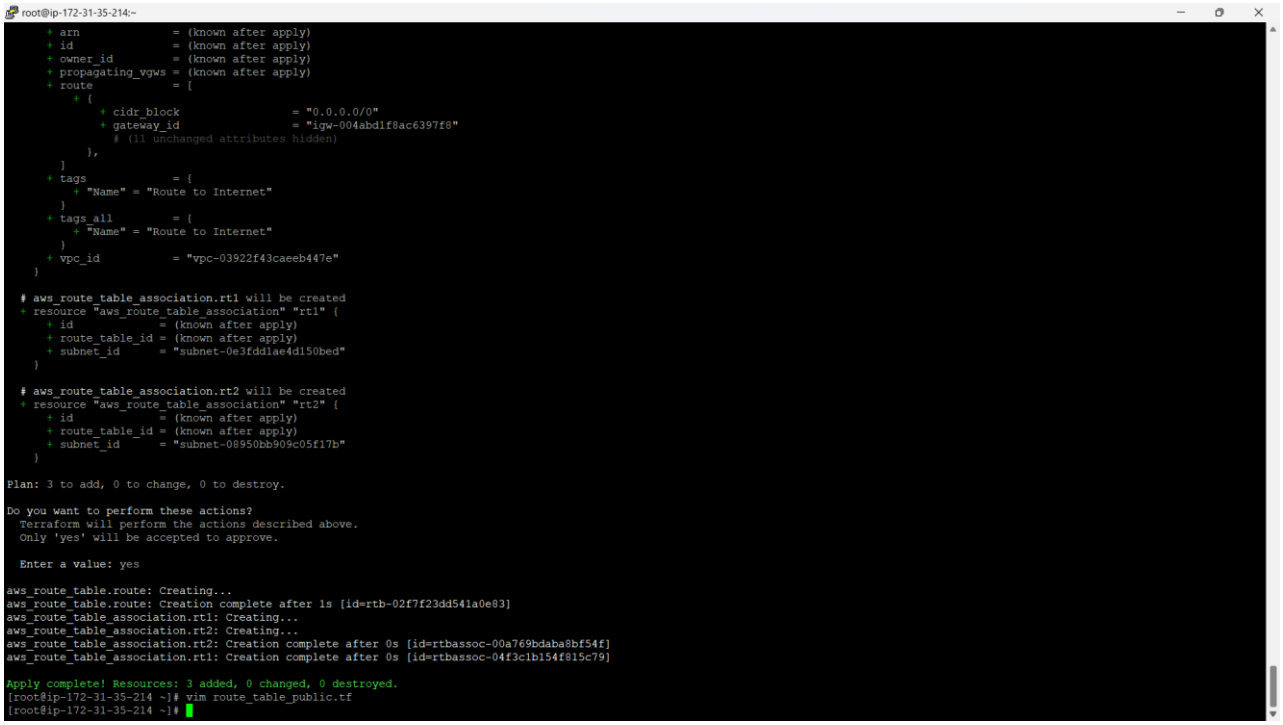
In the below image I have run the terraform plan command because I have review the changes and if any error occurs I can able to find it before applying.



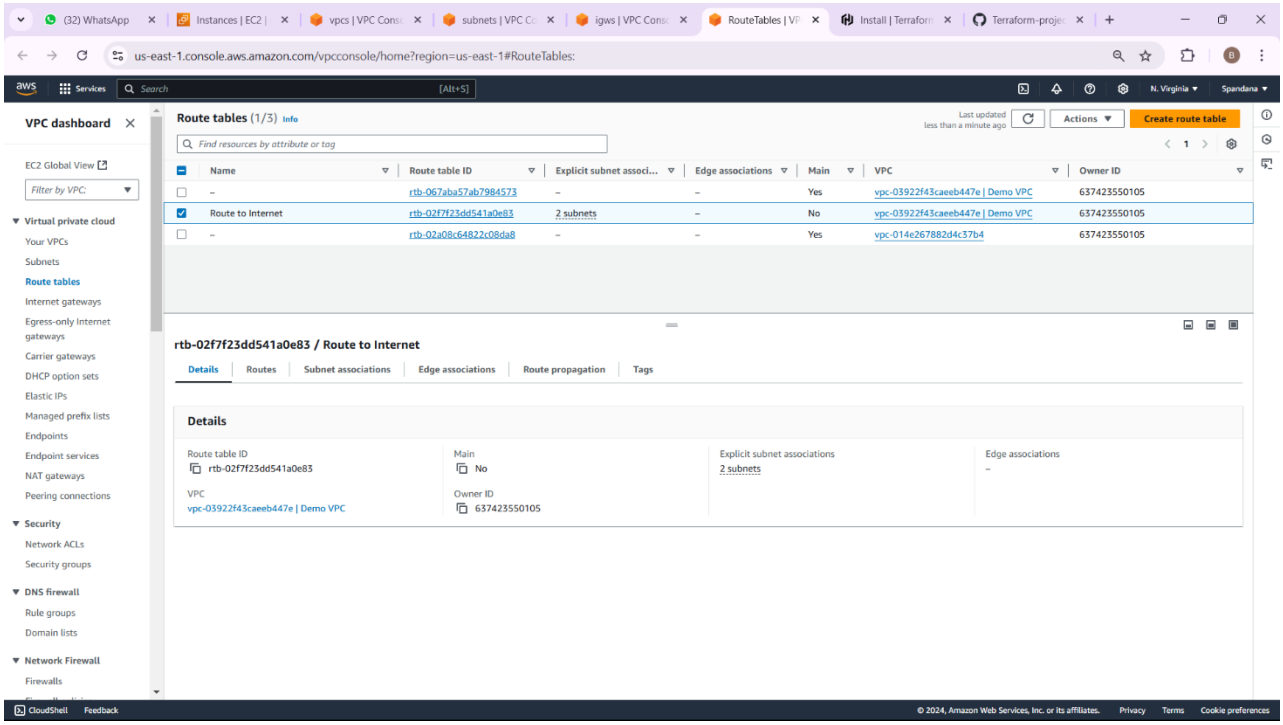
After completion of terraform plan there is no error or any mistake so it is ready for applying so I used terraform apply command to make changes and to create a route table.



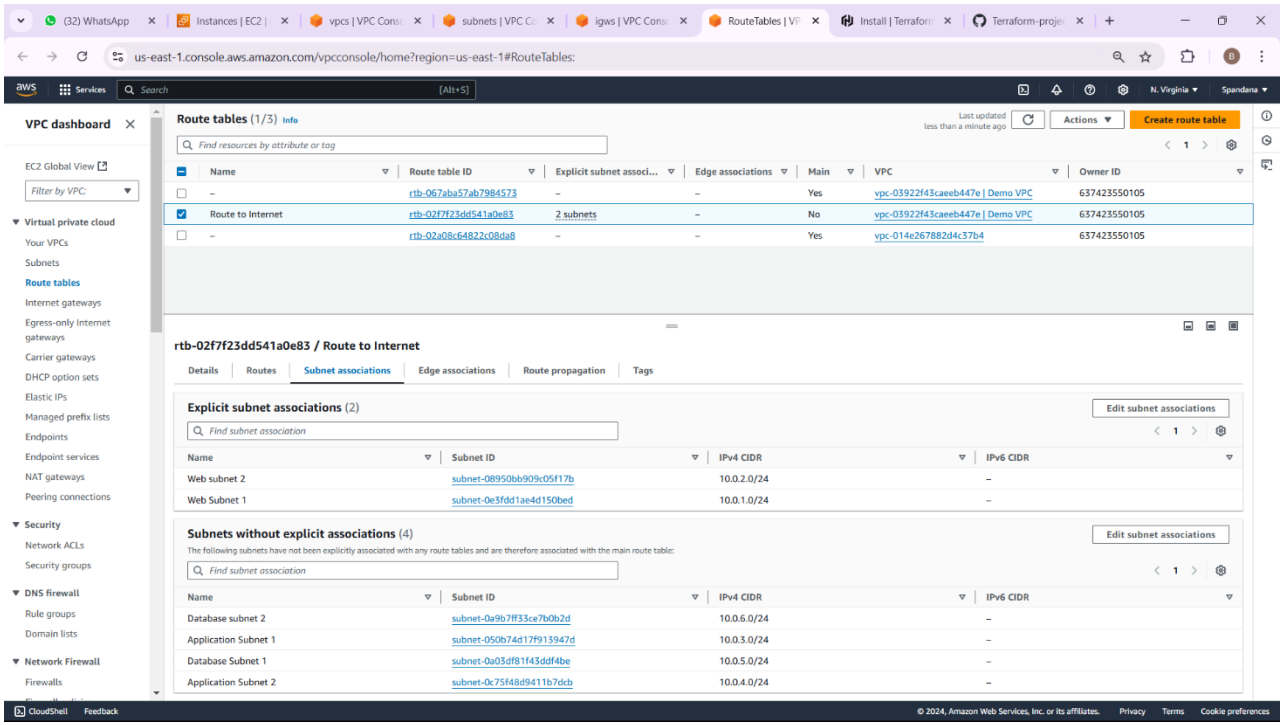
In the below image we can able to see that our apply is successful then we need to go to AWS VPC and then click on route table then we can able to see that our route table as been created successfully.



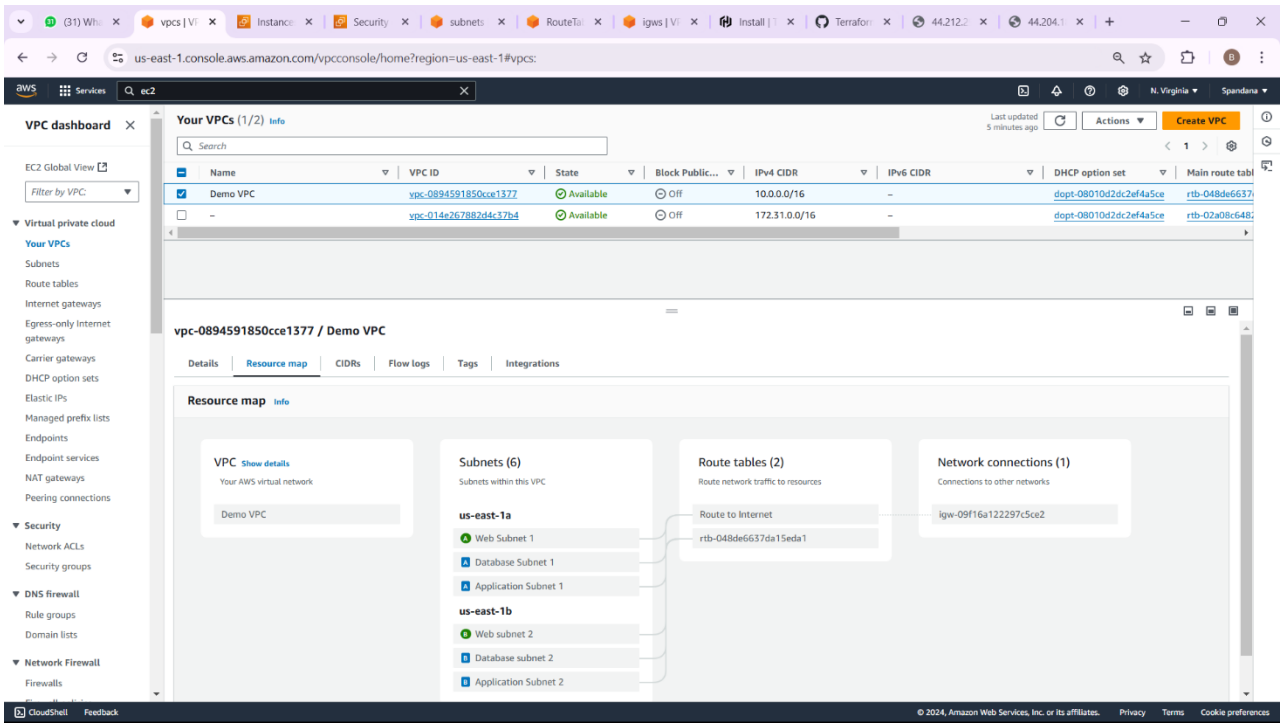
In the below image we can able to see that our route table as been created successfully.



In the below image we can able to see that the toute association.

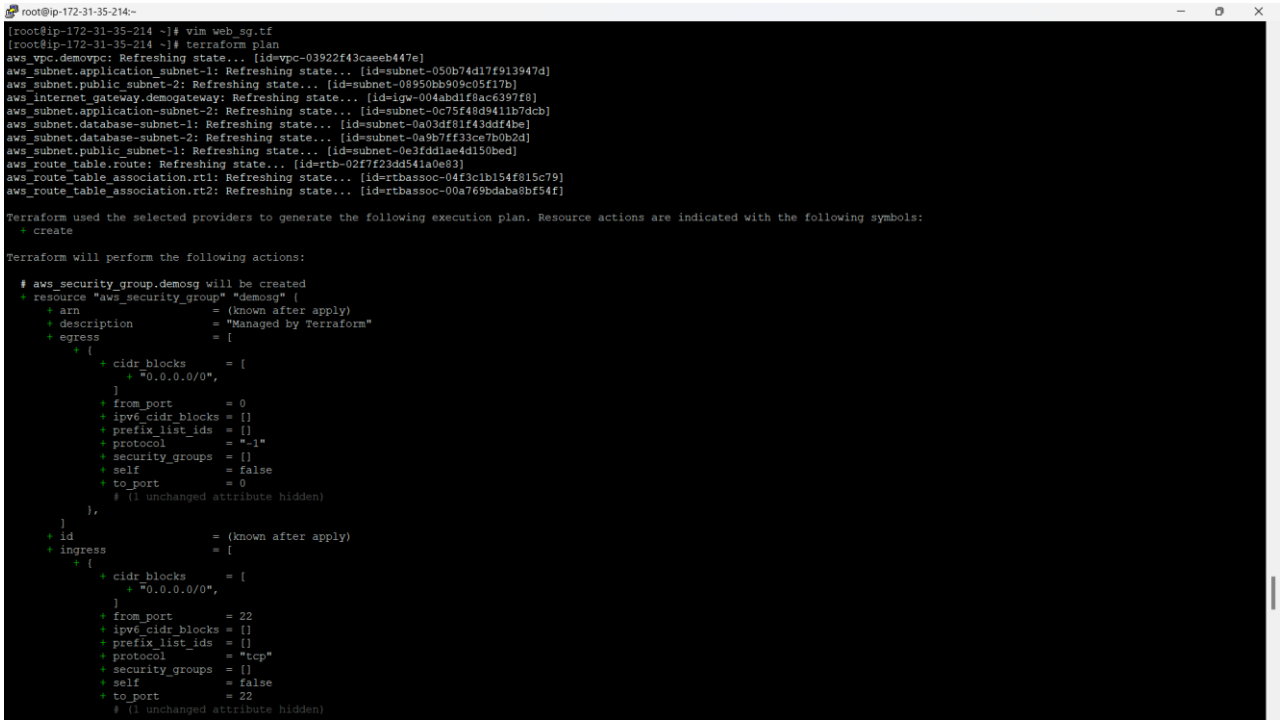


After creating VPC, Subnets, Internet gateway, Route table then go to VPC and click on VPC in that we can able to see Route map click on that and we can able to see a map for the creation of our VPC and subnet attachment to which route table and connected to which network connection.

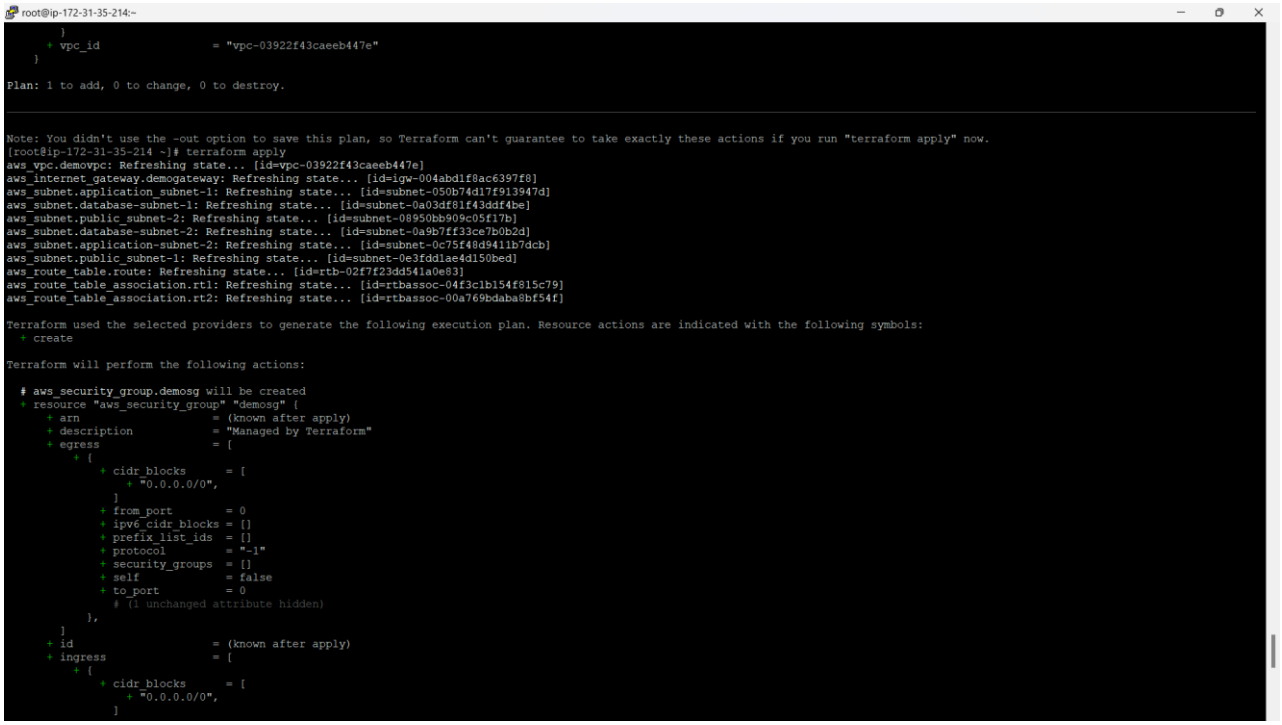


STEP 6: CREATING A SECURITY GROUP

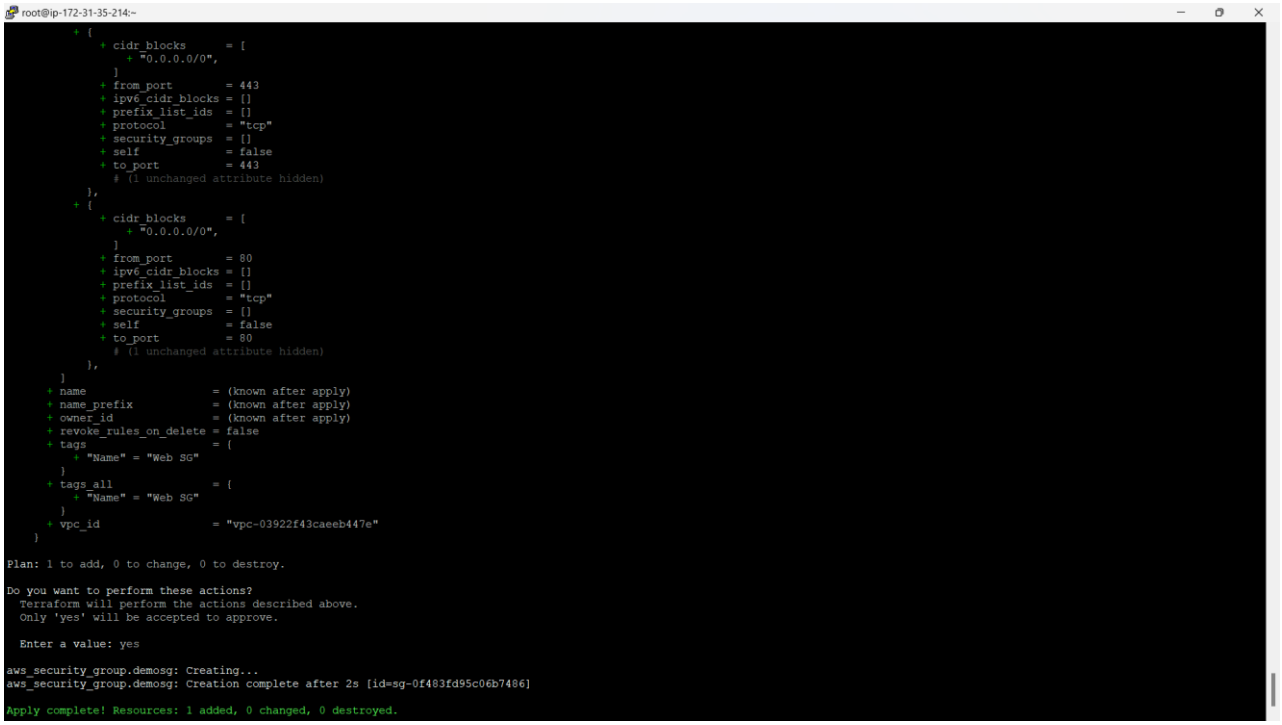
A Security Group in AWS acts like a virtual firewall for your resources, such as EC2 instances. It controls what kind of network traffic is allowed to or from these resources. Think of it as a set of rules that decides who can communicate with your resource and how. I have created a **web_sg.tf** file for the security group creation before the creation of ec2.



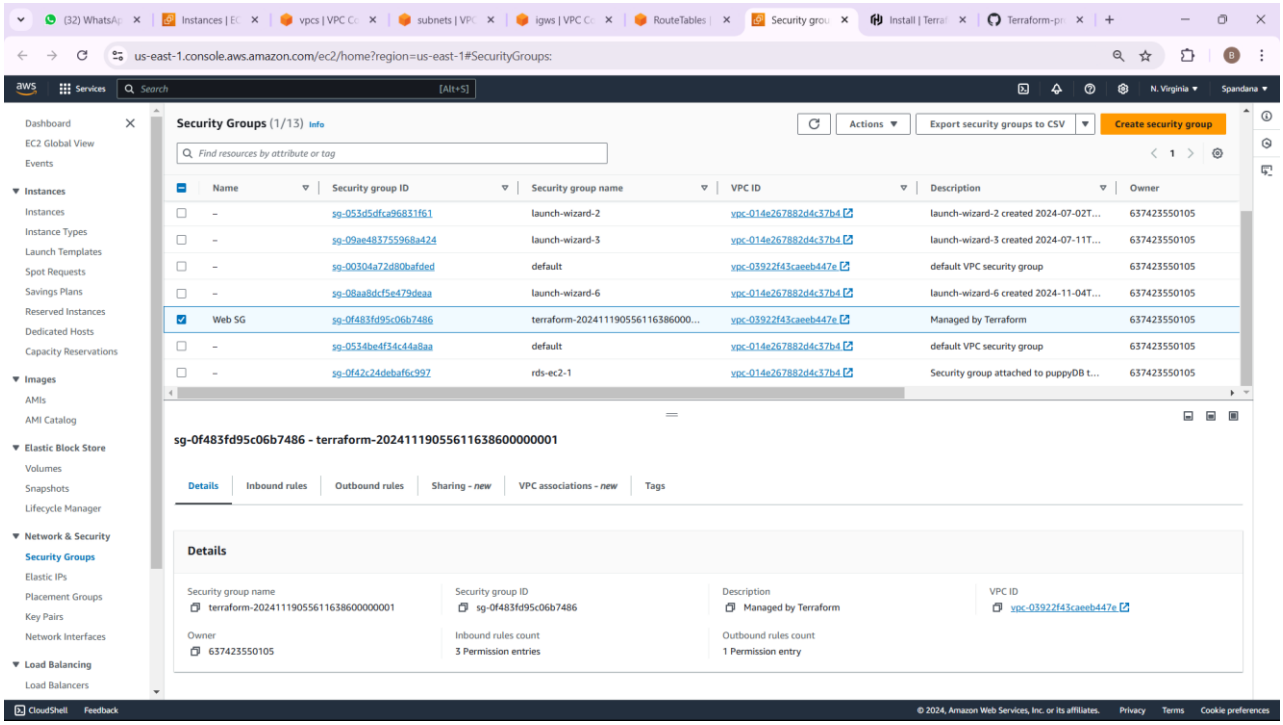
I have used terraform apply command to make changes and create an security group for an ec2 instance.



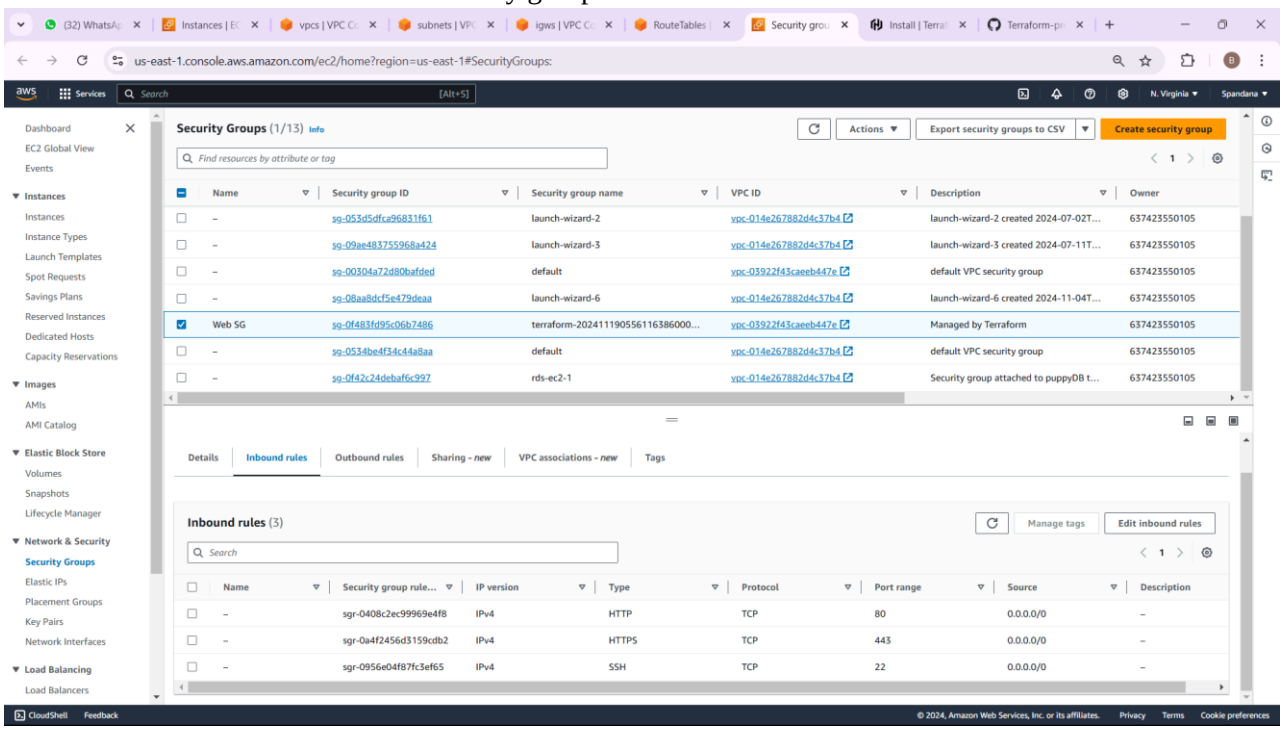
In the below image we can able to see that the terraform apply as been successfully applied. So now we can able to go and check in the security group we can able to see our new security group called **Web SG.**



In the below image we can able to see that our new security group has been created successfully.

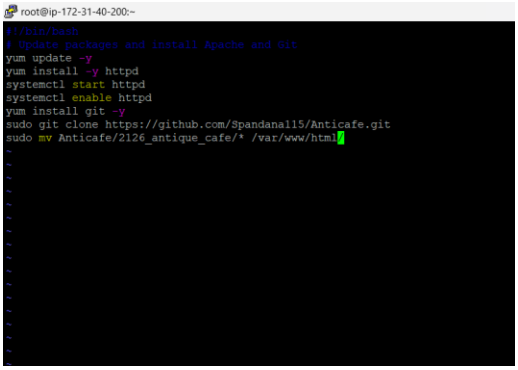


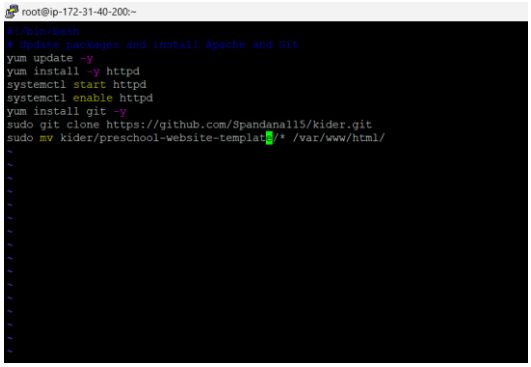
Below are the inbound rules for the security group that we have created.



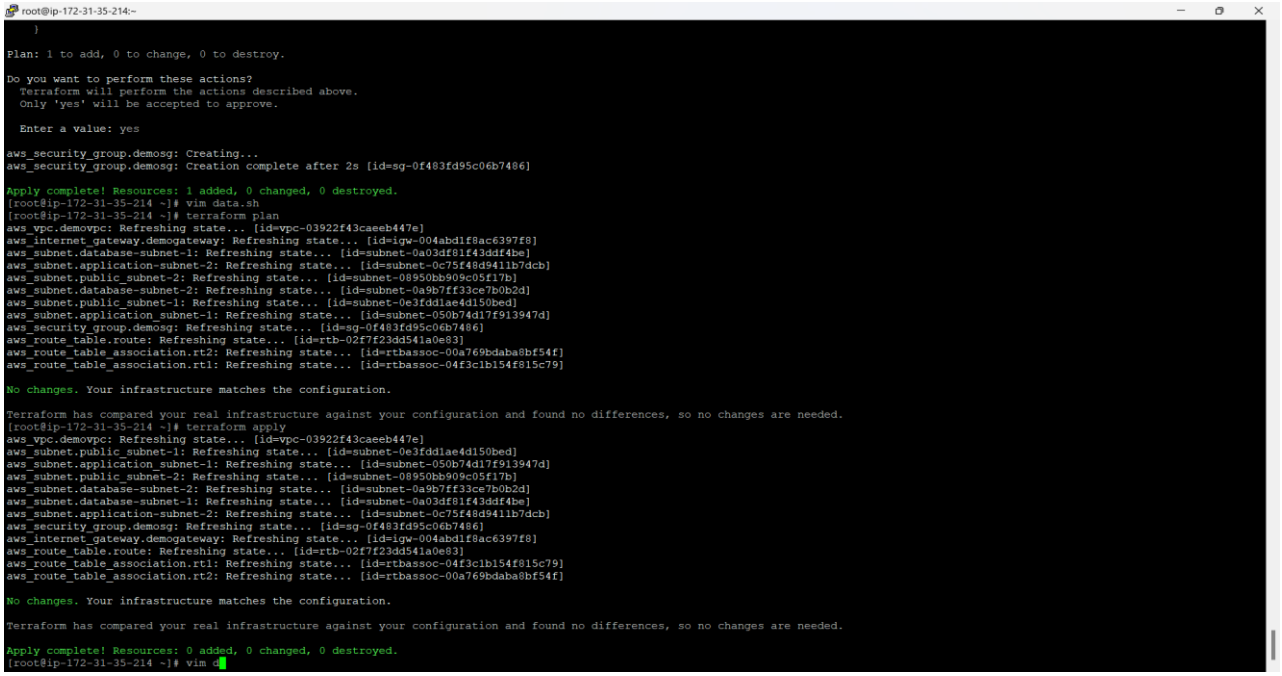
STEP 7: CREATING AN USER DATA

We are creating an data.sh because if we want to create an instance. The data.sh file is a shell script that you can use to automate the setup of your EC2 instance when it starts. By including this file in the User Data section, you ensure that the necessary software and configurations are automatically applied without having to manually log into the instance after it's launched.

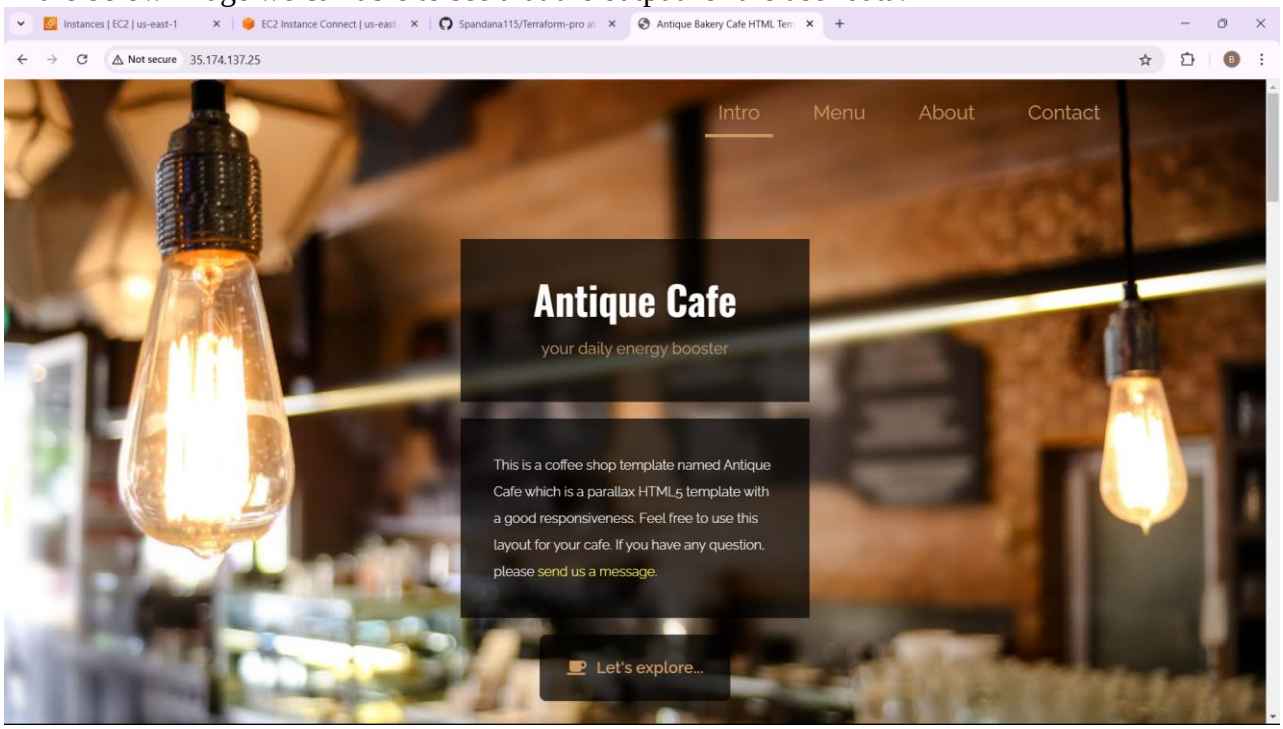




I have used terraform plan and apply for the data.sh.



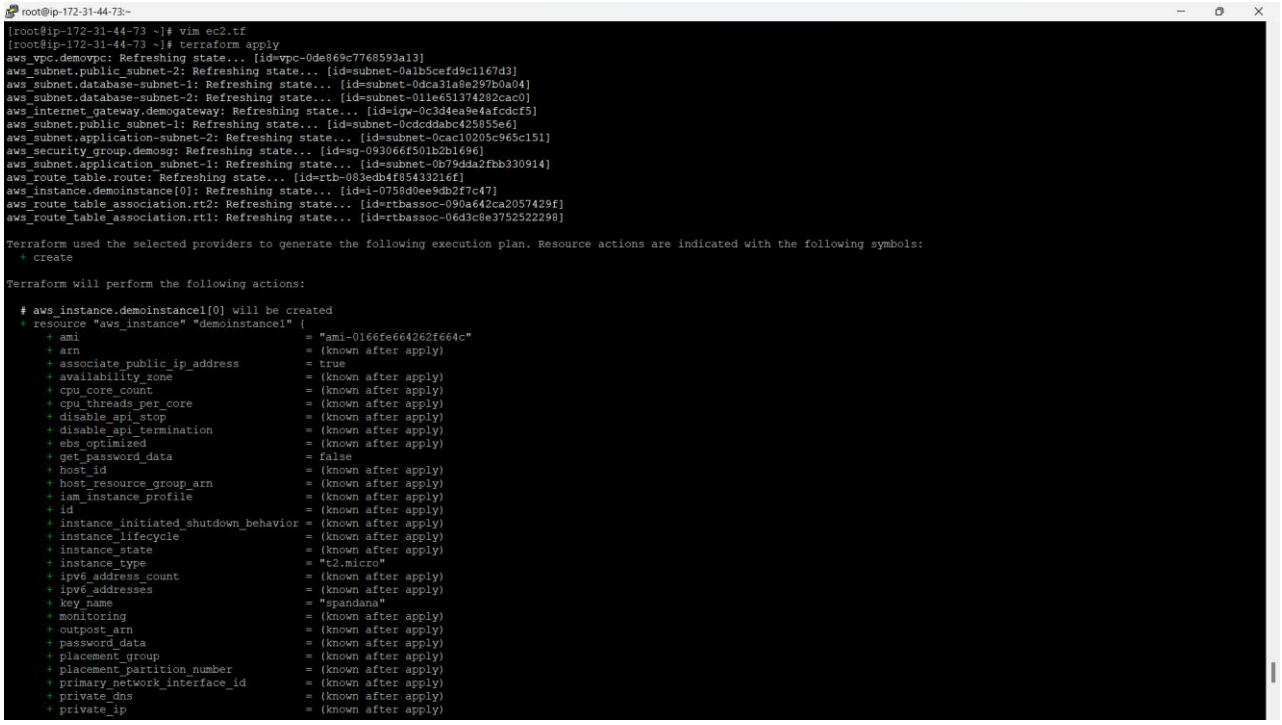
In the below image we can able to see that the output for the user data.

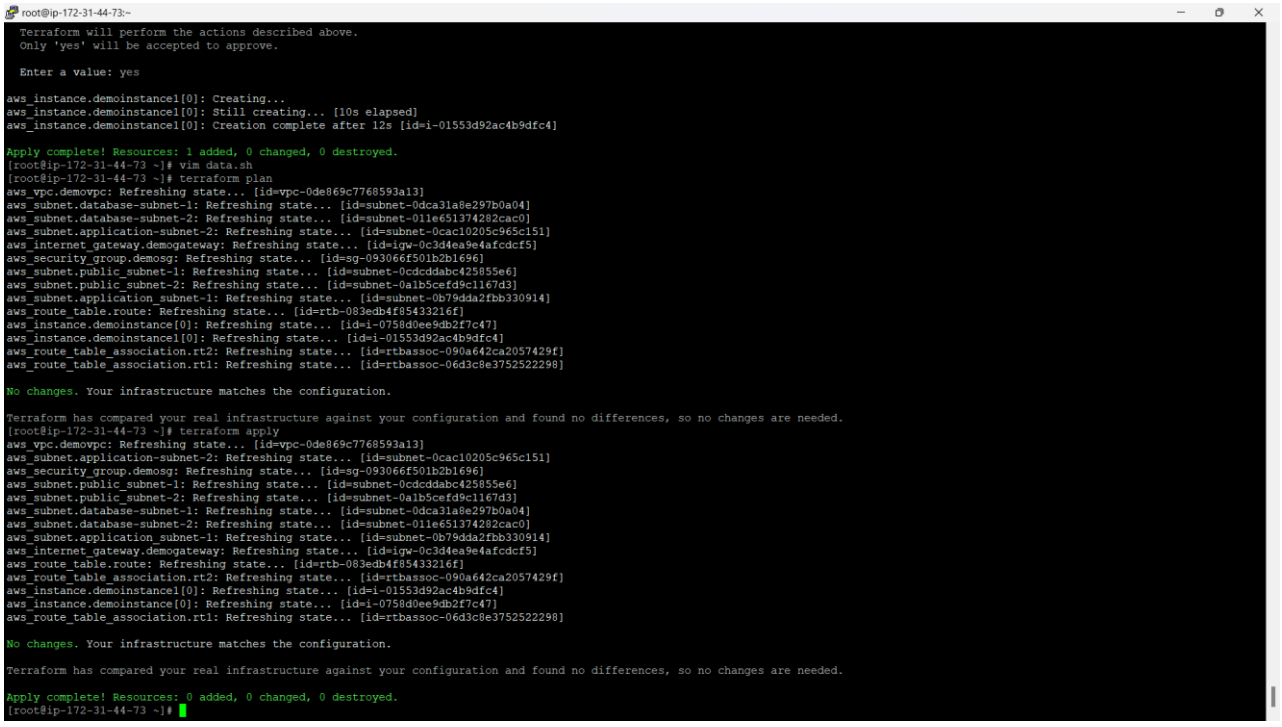


STEP 8: CREATING AN EC2 INSTANCES

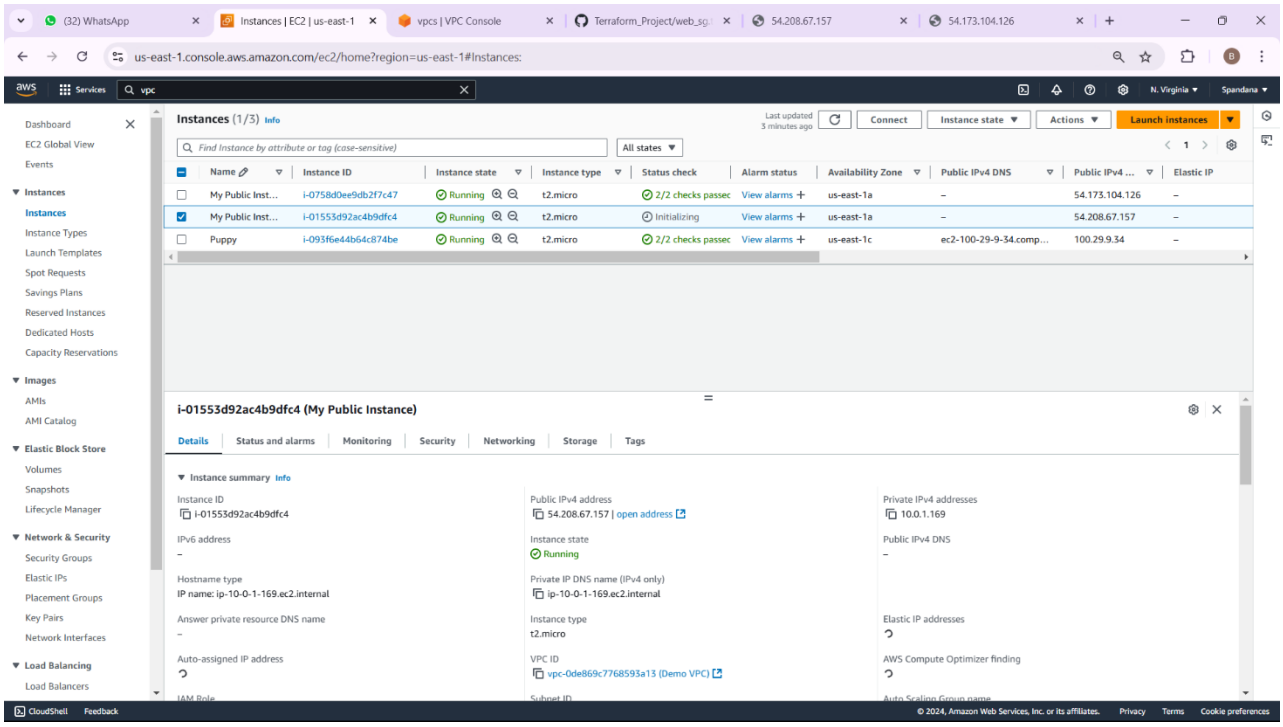
To create an EC2 instance in AWS using Terraform, we need to define the configuration in a file **ec2.tf** file. This file contains all the details about your EC2 instance, such as the Amazon Machine Image (AMI), instance type, security groups, and other settings. I have created 2 instances.

After creating an ec2.tf file then i have run a terraform plan command and terraform apply command to check and create an ec2 instances based on our configuration.

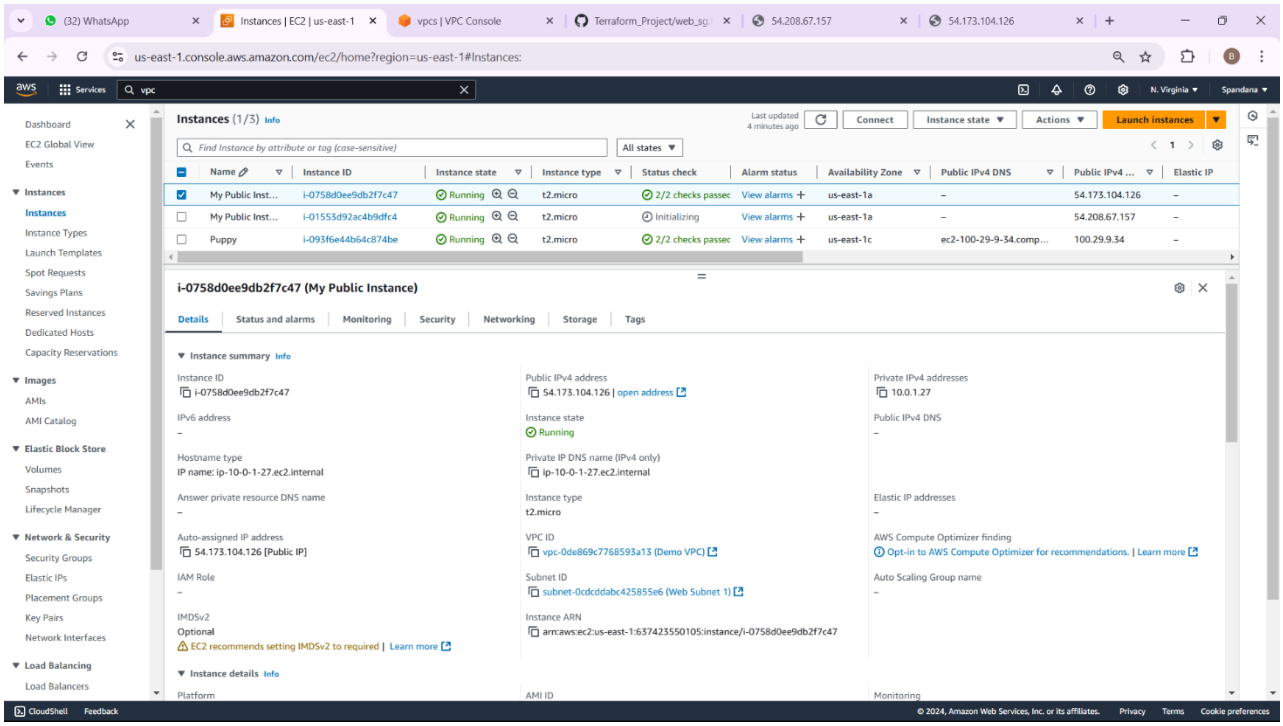




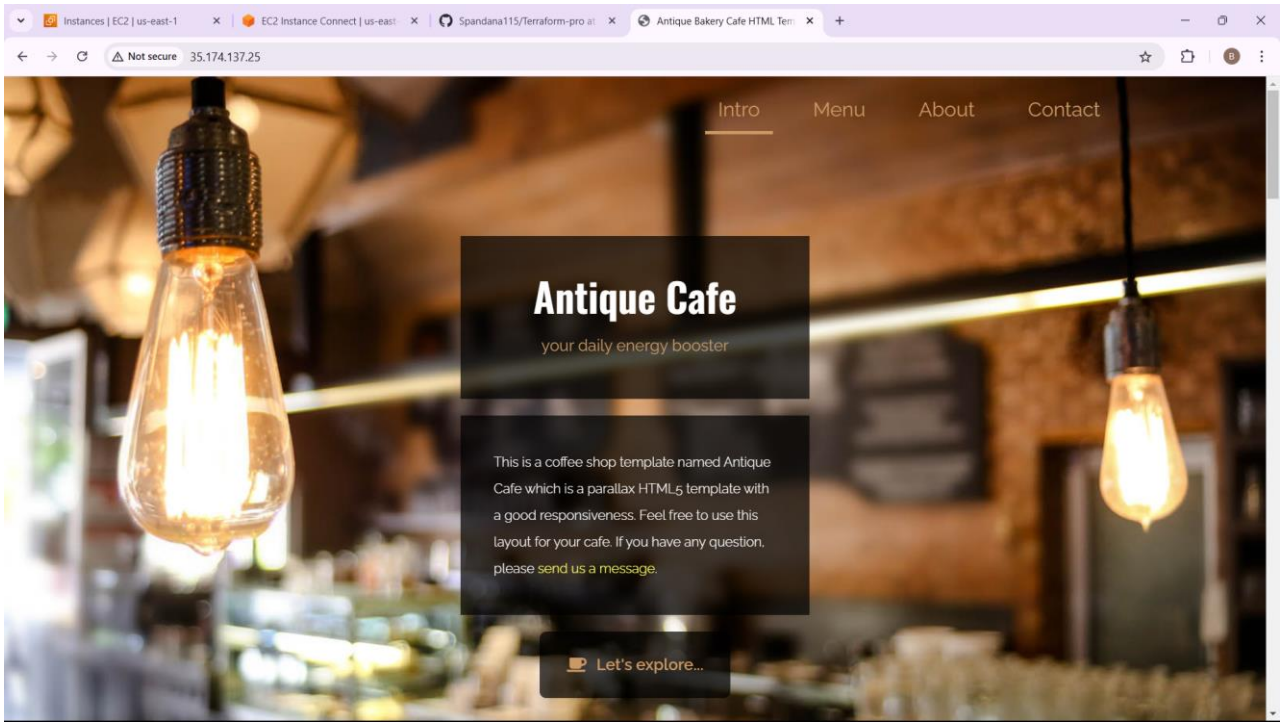
In the below image we can able to see that our newly created 2 instances has been created successfully.the below image is the first instance details.



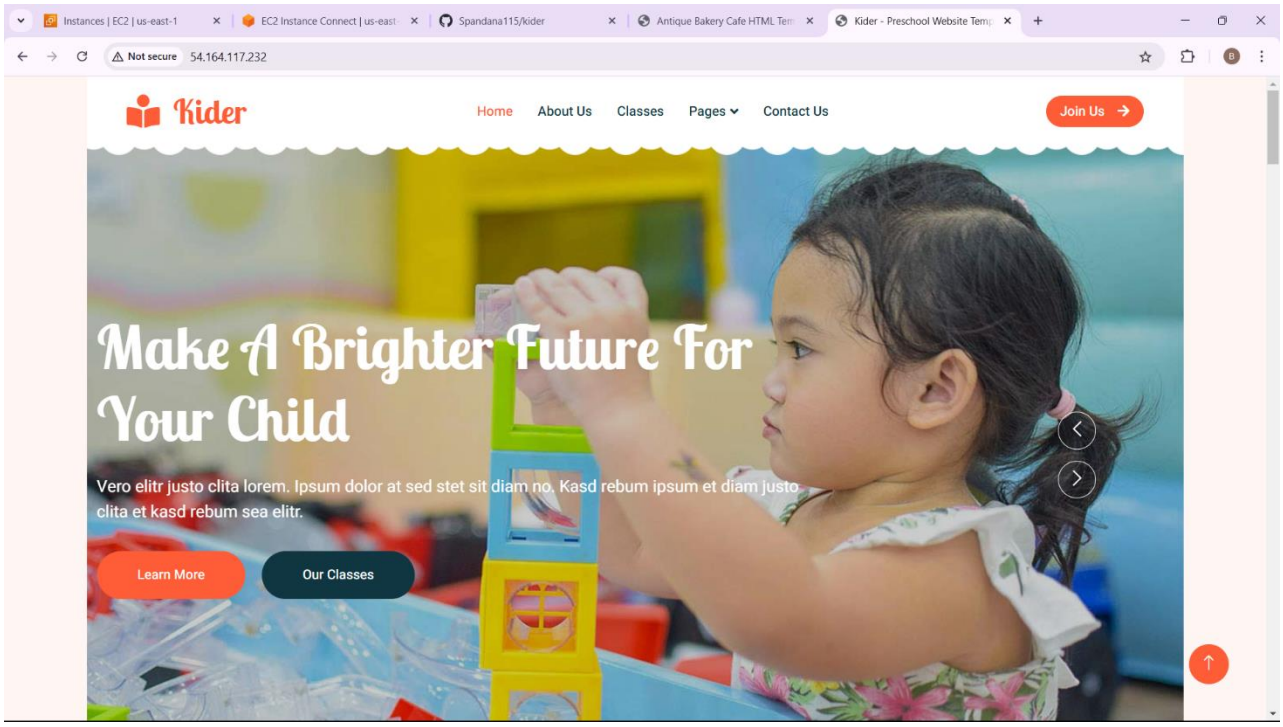
In the below image we can see that second instance details.



After creating an instances if we want to check that user data is working or not we need to copy the public IP of the instance and paster it in browser it throughs an error because it is https:// we need to remove the “s” http:// and again reload we can able to see the user data content.

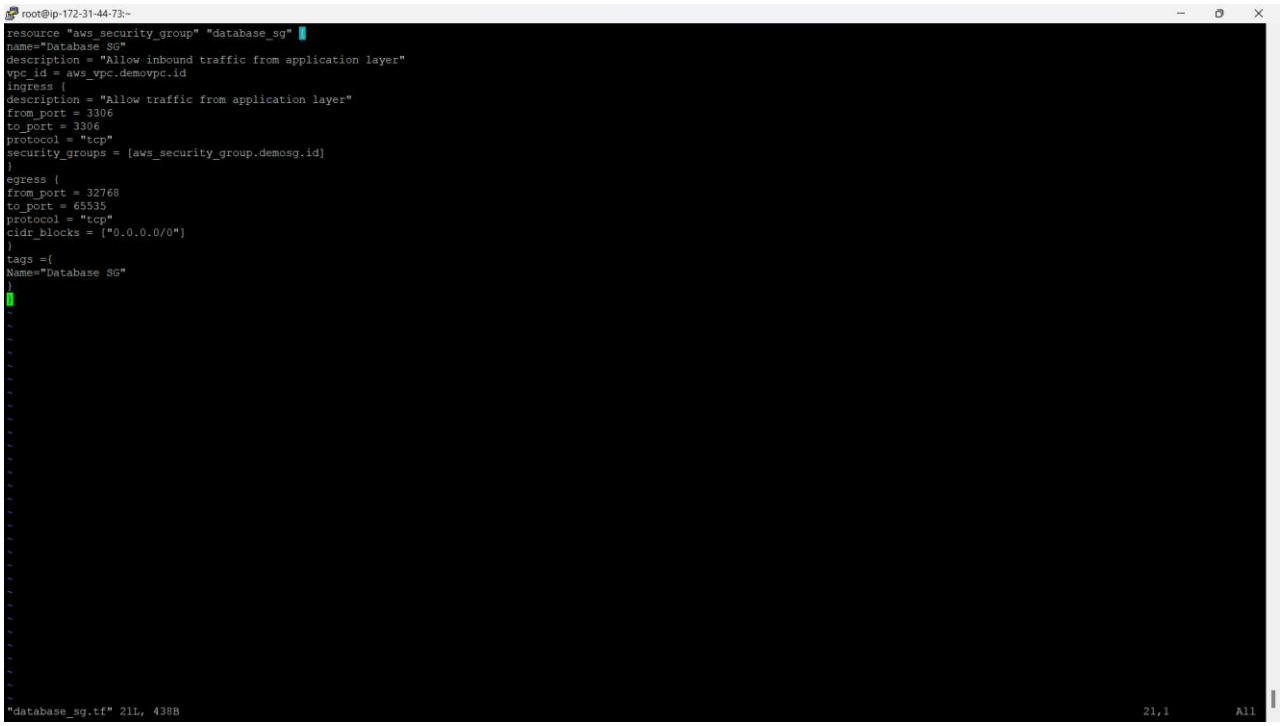


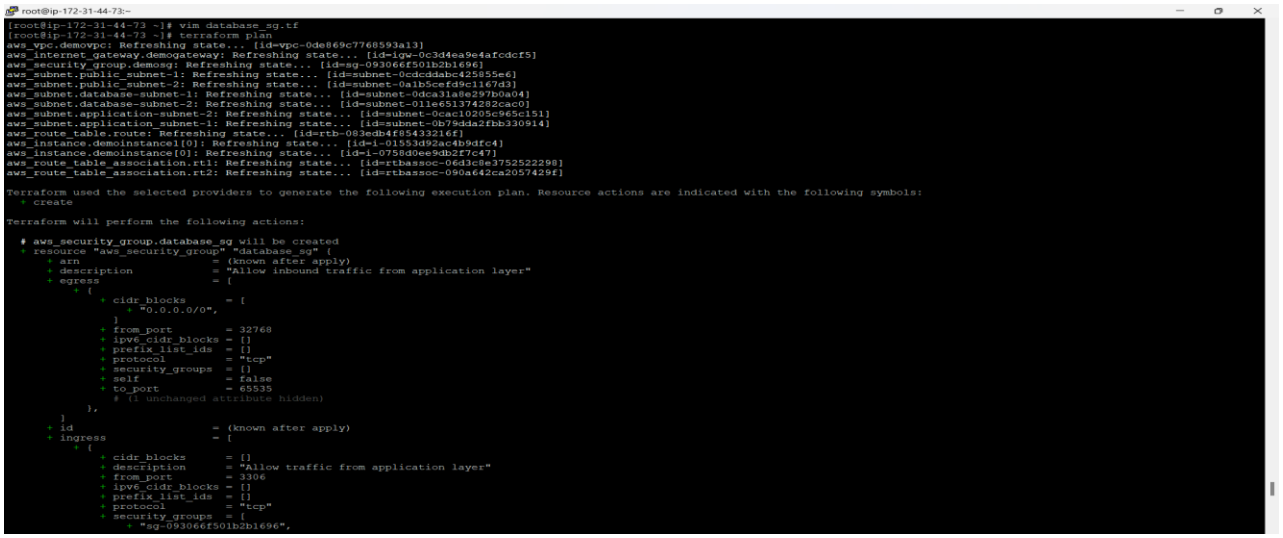
We need to do the same process for the second instance then we can able to see like a below image.



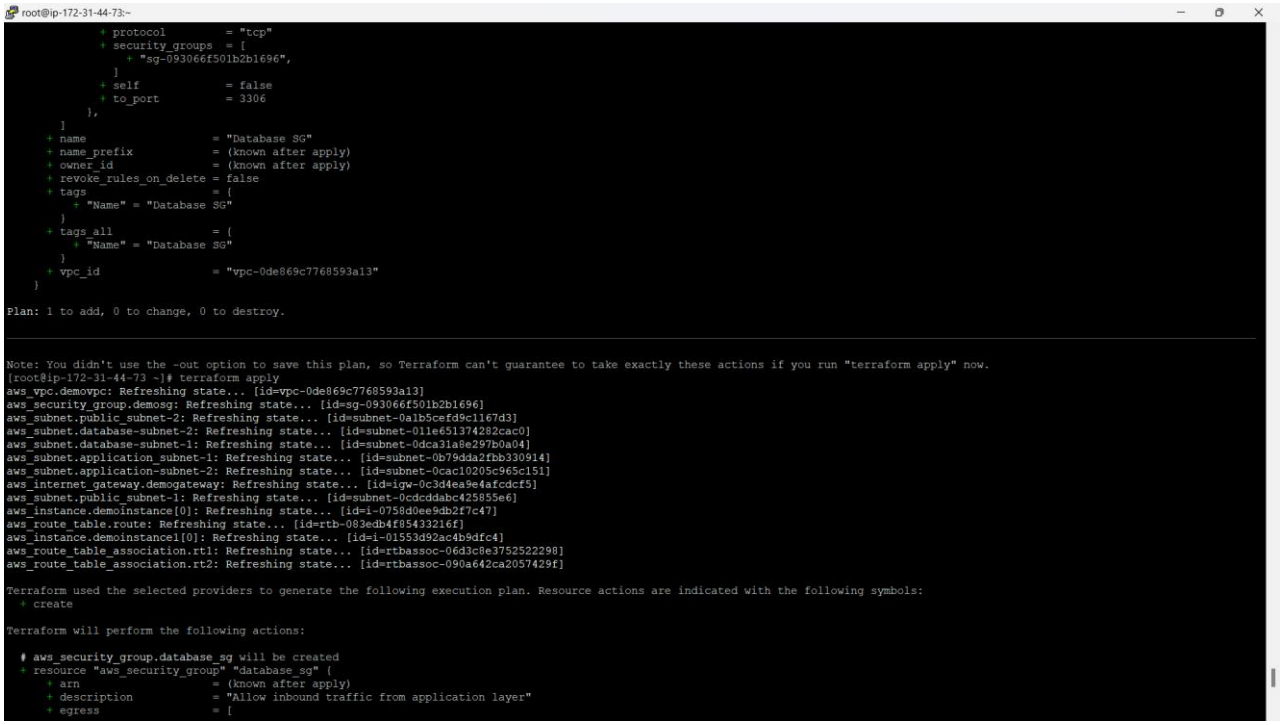
STEP 9: CREATING A FILE FOR SECURITY GROUP FOR THE DATABASE TIER

The Security Group for the Database Tier is like a gatekeeper. It allows only specific resources (like your Application Tier) to access the database. We set rules to allow only trusted traffic (e.g., from specific IP addresses or subnets) and block all others. This ensures your database is protected and only accessible by the parts of your system that need it. So because of that I have created an **database_sg.tf** file.

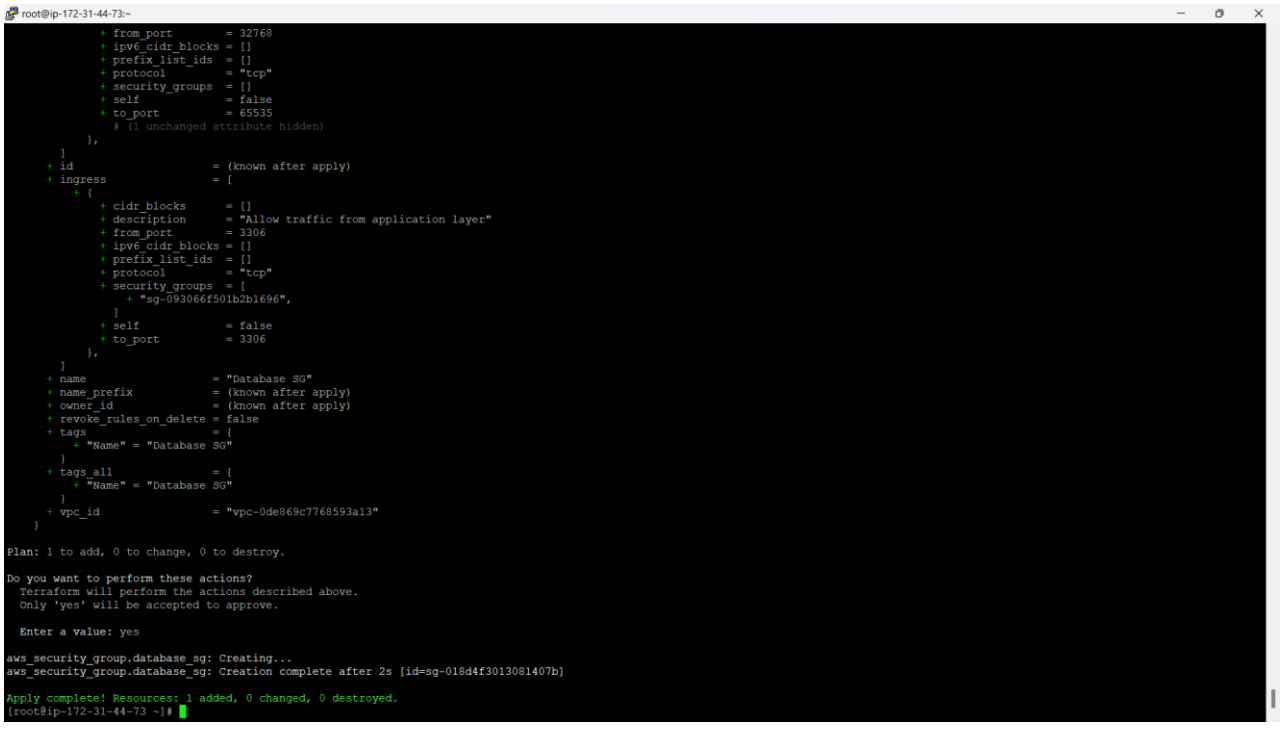




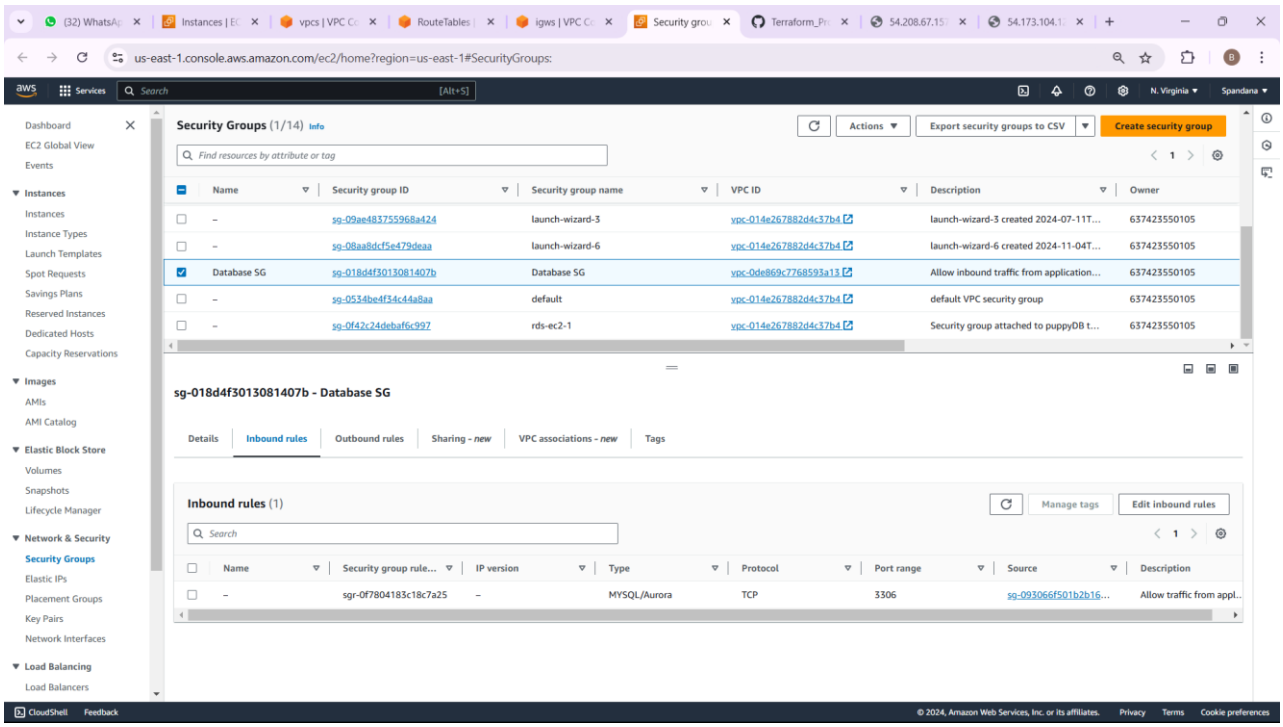
I have create a Security Group that allows only specific sources (like the Application Tier) to access the database on specific port.



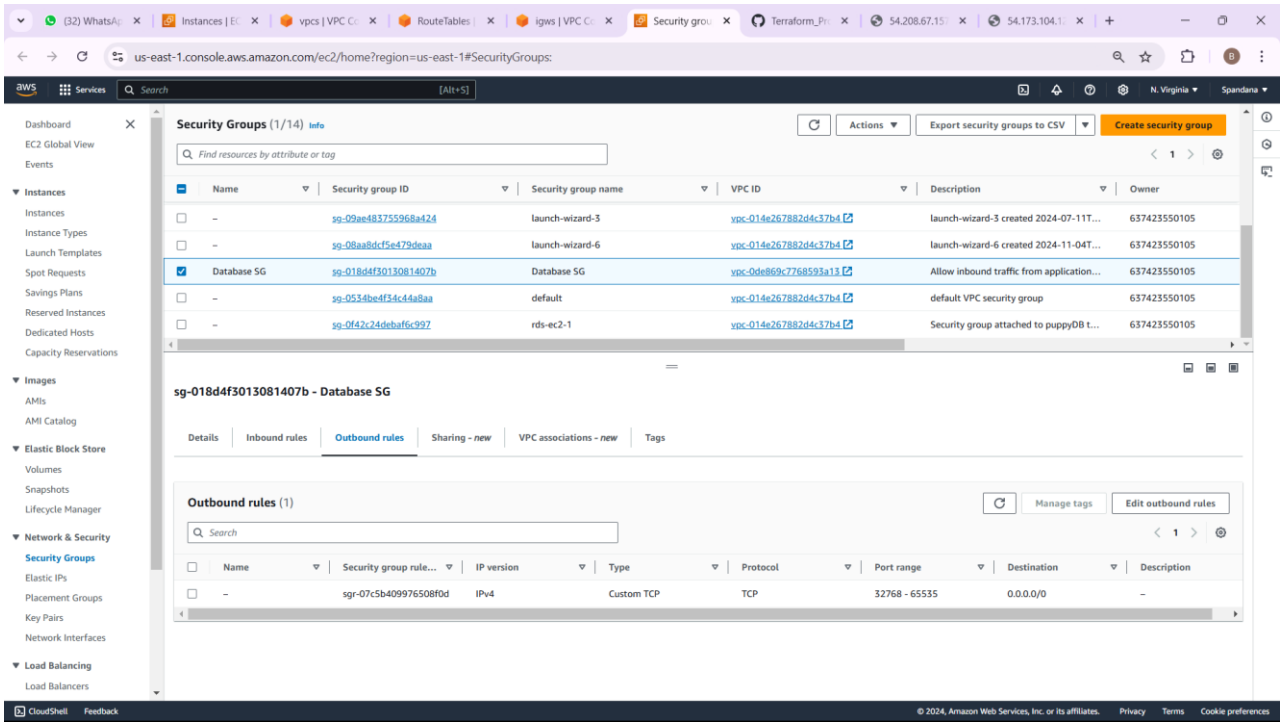
I have applied it successfully using terraform apply command.



In the below image we can able to see that our database security group has been created successfully. We can able to see the inbound rules.

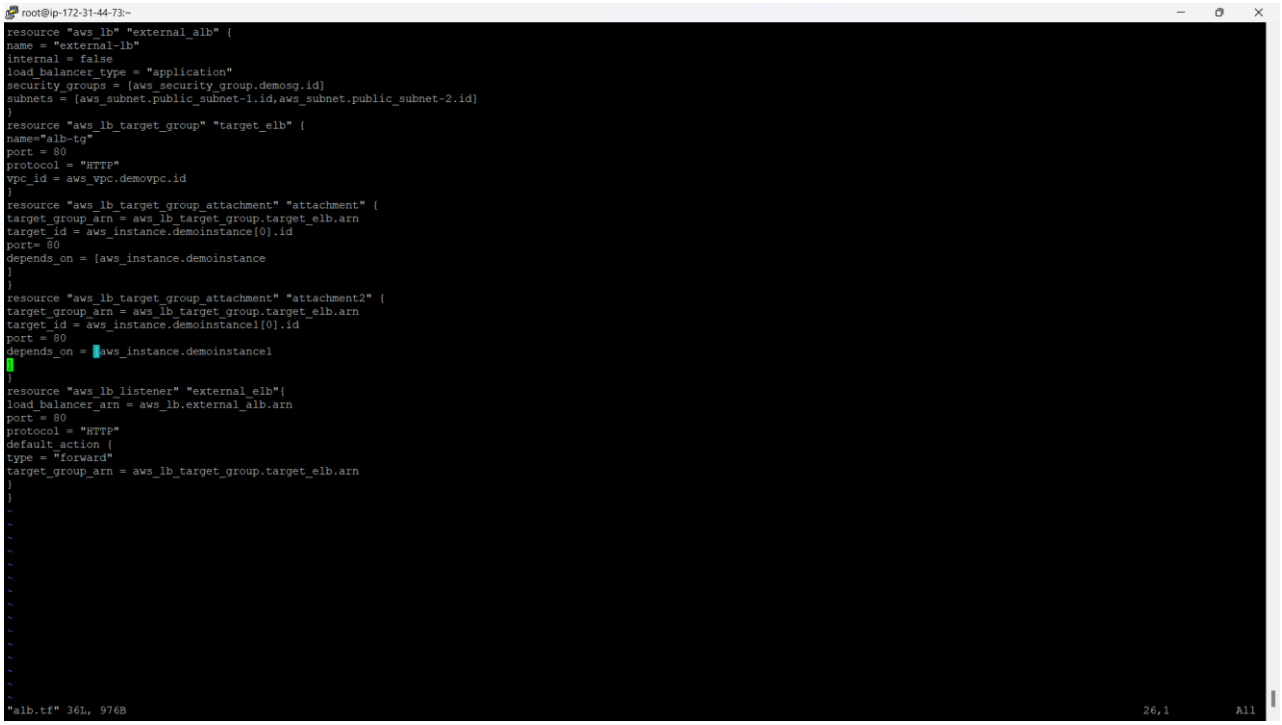


In the below image we can able to see that the outbound rules for the Database security group.



STEP 10: CREATE A FILE FOR THE APPLICATION LOAD BALANCER

I have created an Application Load Balancer because it distribute the incoming traffic to various servers. So I have created an alb.tf file for the load balancer and also I have created a target group.



Load Balancer (aws_lb):

name: The name of the ALB.

internal: Whether the ALB is public (false) or private (true).

load_balancer_type: Specifies it's an application load balancer.

security_groups: The security group attached to the ALB to control traffic.

subnets: Defines the public subnets where the ALB will operate.

Listener (aws_lb_listener):

Specifies how the ALB listens for traffic (e.g., HTTP on port 80).

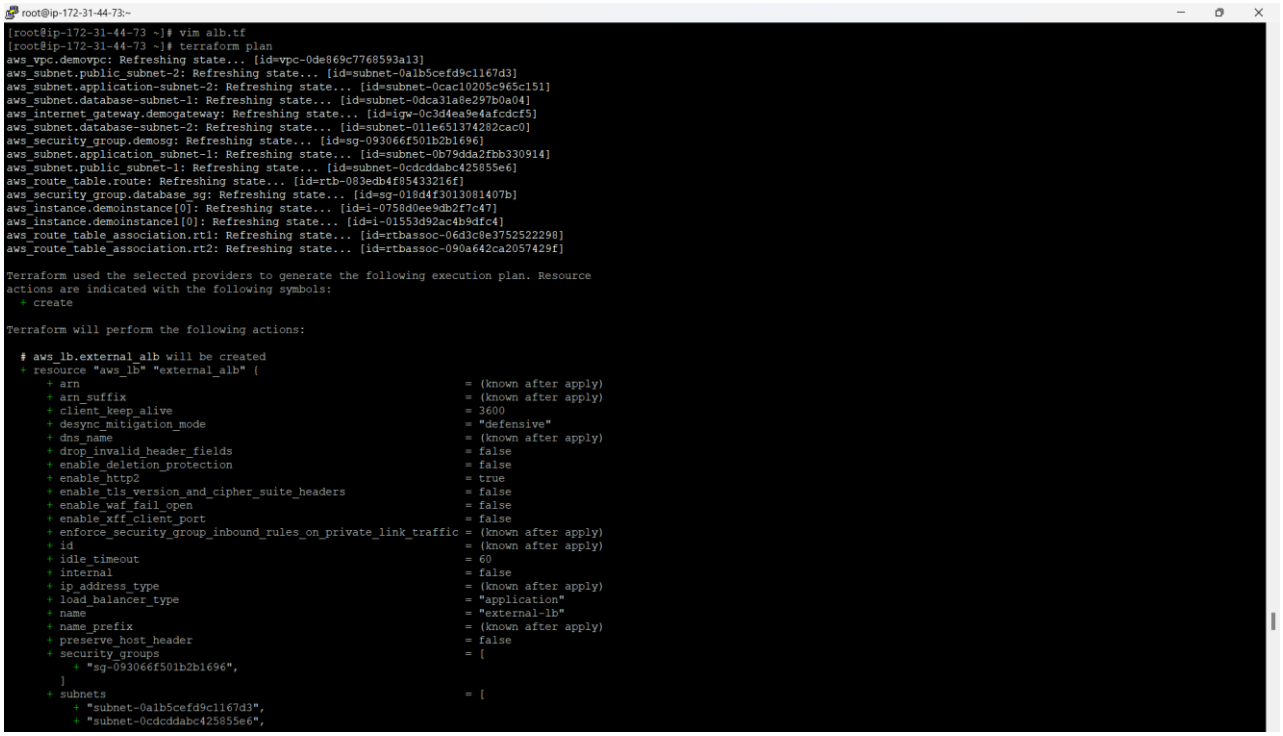
default_action: Forwards traffic to the target group.

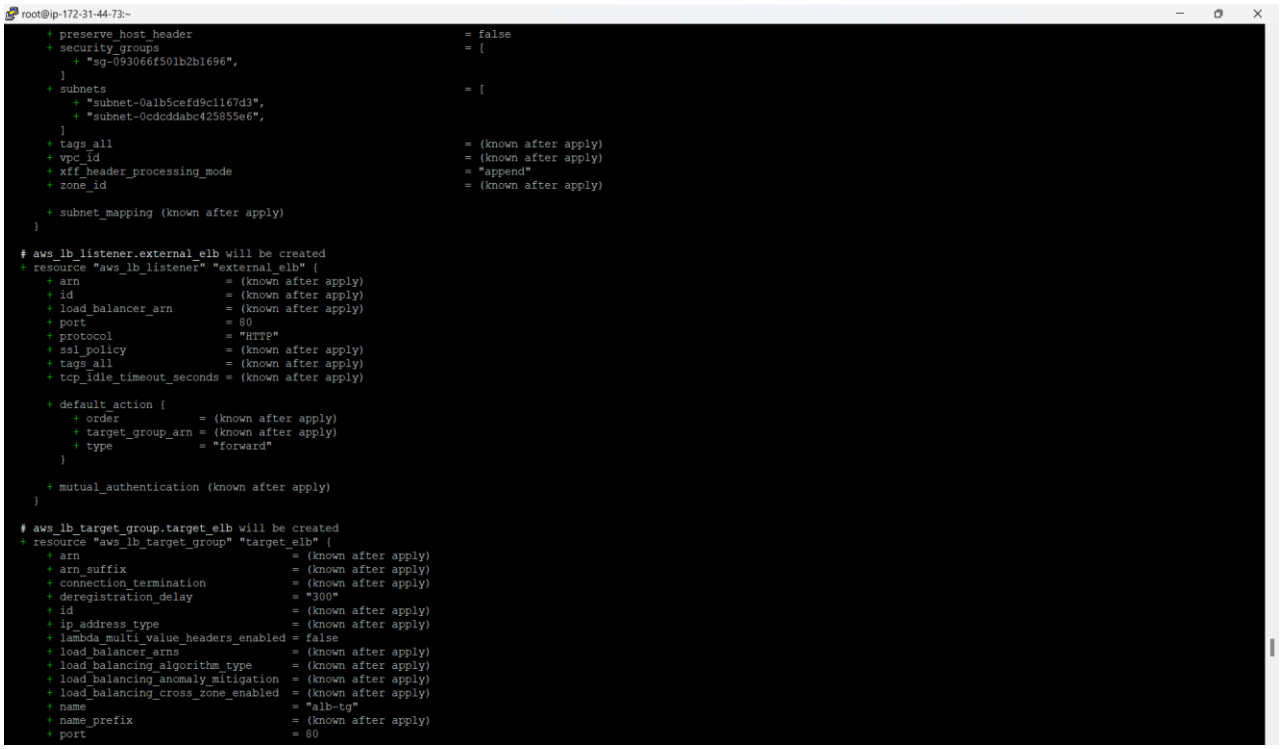
Target Group (aws_lb_target_group):

Groups the backend instances (e.g., EC2 or containers) the ALB routes traffic to.

Includes health checks to monitor the health of backend resources.

After creation of alb.tf then I have used terraform plan

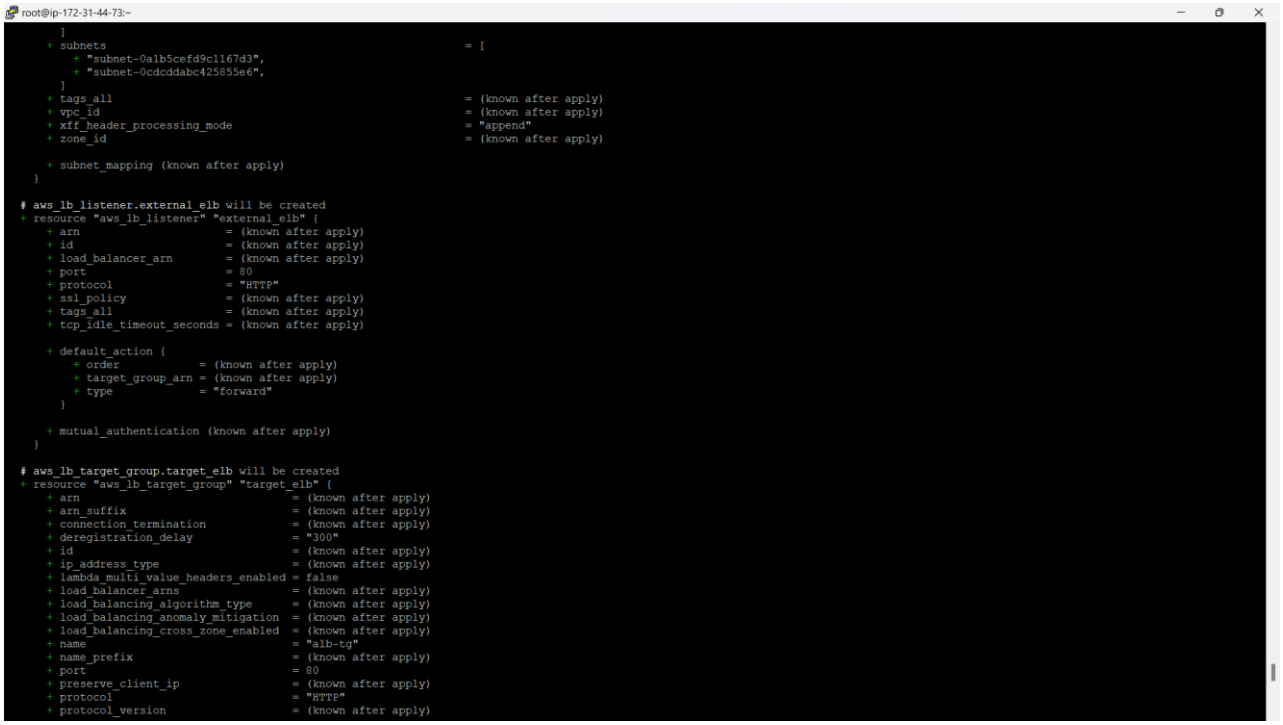


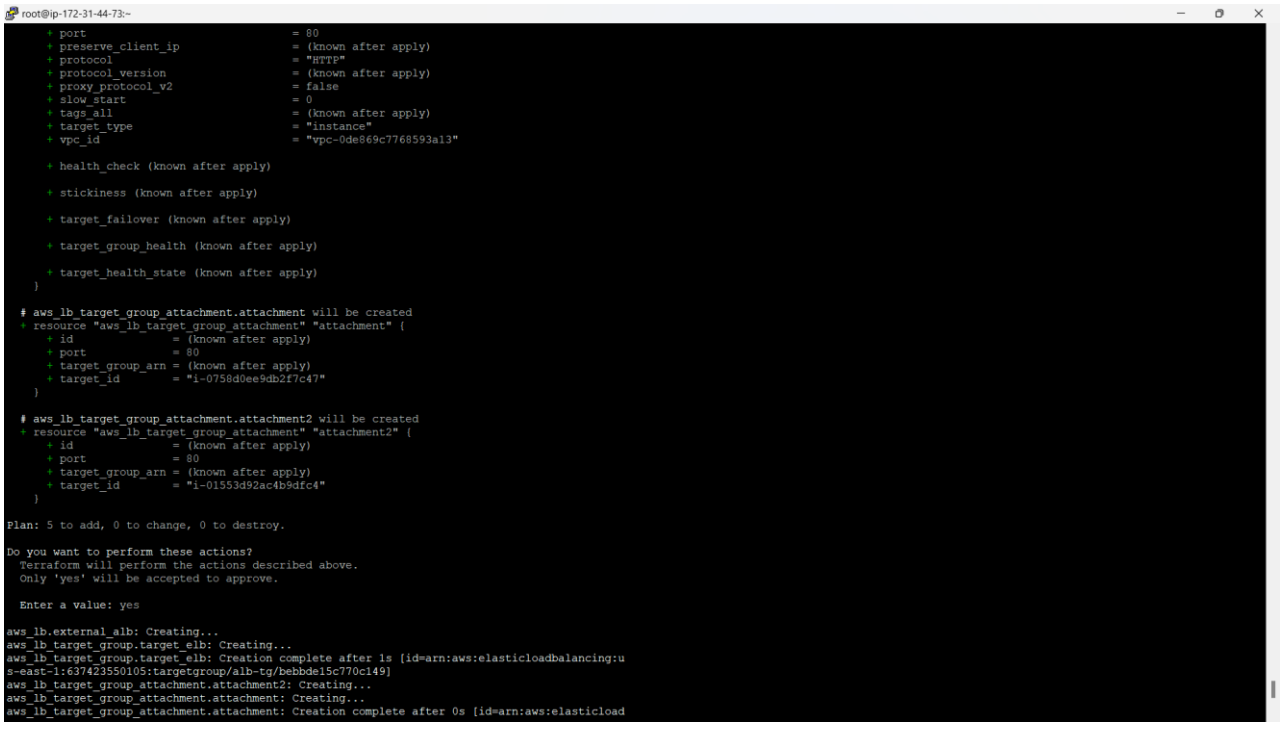


```
root@ip-172-31-44-73:~
      + arn_suffix                       = (known after apply)
      + connection_termination           = (known after apply)
      + deregistration_delay             = "300"
      + id                               = (known after apply)
      + ip_address_type                  = (known after apply)
      + lambda_multi_value_headers_enabled = false
      + load_balancer_arns               = (known after apply)
      + load_balancing_algorithm_type    = (known after apply)
      + load_balancing_anomaly_mitigation  = (known after apply)
      + load_balancing_cross_zone_enabled  = (known after apply)
      + name                             = "alb-tg"
      + name_prefix                      = (known after apply)
      + port                             = 80
      + preserve_client_ip               = (known after apply)
      + protocol                         = "HTTP"
      + protocol_version                 = (known after apply)
      + proxy_protocol_v2                = false
      + slow_start                       = 0
      + tags_all                         = (known after apply)
      + target_type                      = "instance"
      + vpc_id                           = "vpc-0de869c7768593a13"

      + health_check (known after apply)

      + stickiness (known after apply)

      + target_failover (known after apply)

      + target_group_health (known after apply)

      + target_health_state (known after apply)
    }

  # aws_lb_target_group_attachment.attachment will be created
  + resource "aws_lb_target_group_attachment" "attachment" {
      + id               = (known after apply)
      + port             = 80
      + target_group_arn = (known after apply)
      + target_id        = "i-0758d0ee9db2f7c47"
    }

  # aws_lb_target_group_attachment.attachment2 will be created
  + resource "aws_lb_target_group_attachment" "attachment2" {
      + id               = (known after apply)
      + port             = 80
      + target_group_arn = (known after apply)
      + target_id        = "i-01553d92ac4b9dfc4"
    }

Plan: 5 to add, 0 to change, 0 to destroy.
```

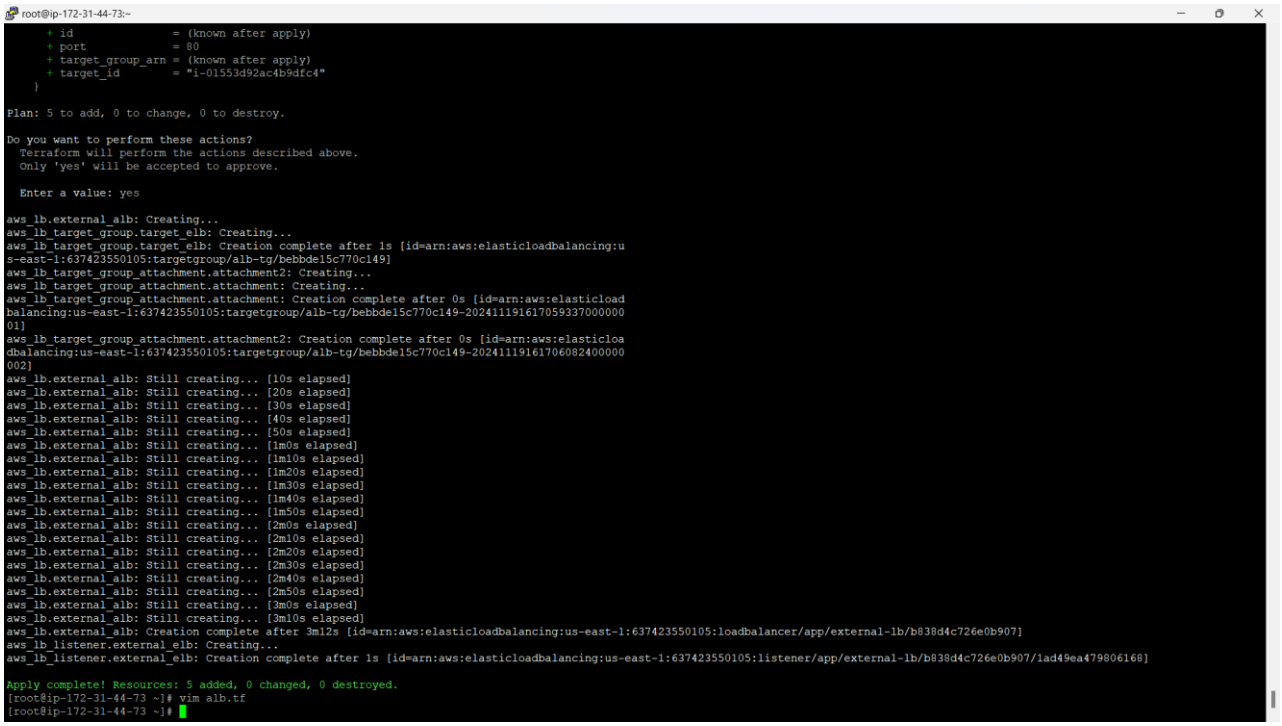
Now we need to perform an terraform apply for the creation of the application load balancer.

```
root@ip-172-31-44-73:~
Plan: 5 to add, 0 to change, 0 to destroy.


Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take
exactly these actions if you run "terraform apply" now.
[root@ip-172-31-44-73 ~]# terraform apply
aws_vpc.demovpc: Refreshing state... [id=vpc-0de869c7768593a13]
aws_security_group.demosg: Refreshing state... [id=sg-093066f501b2b1696]
aws_subnet.database-subnet-2: Refreshing state... [id=subnet-011e651374282cac0]
aws_internet_gateway.demogateway: Refreshing state... [id=igw-0c3d4ea9e4afcdcf5]
aws_subnet.database-subnet-1: Refreshing state... [id=subnet-0dca31a8e297b0a04]
aws_subnet.application_subnet-1: Refreshing state... [id=subnet-0b79dda2fbb330914]
aws_subnet.public_subnet-2: Refreshing state... [id=subnet-0a1b5cefd9c1167d3]
aws_subnet.public_subnet-1: Refreshing state... [id=subnet-0cdcddabc425855e6]
aws_subnet.application-subnet-2: Refreshing state... [id=subnet-0cac10205c965c151]
aws_security_group.database_sg: Refreshing state... [id=sg-018d4f3013081407b]
aws_route_table.route: Refreshing state... [id=rtb-083edb4f85433216f]
aws_instance.demoinstance[0]: Refreshing state... [id=i-0758d0ee9db2f7c47]
aws_instance.demoinstance1[0]: Refreshing state... [id=i-01553d92ac4b9dfc4]
aws_route_table_association.rt1: Refreshing state... [id=rtbassoc-06d3c8e3752522298]
aws_route_table_association.rt2: Refreshing state... [id=rtbassoc-090a642ca2057429f]

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_lb.external_alb will be created
  + resource "aws_lb" "external_alb" {
      + arn                                                        = (known after apply)
      + arn_suffix                                                 = (known after apply)
      + client_keep_alive                                          = 3600
      + desync_mitigation_mode                                     = "defensive"
      + dns_name                                                   = (known after apply)
      + drop_invalid_header_fields                                 = false
      + enable_deletion_protection                                 = false
      + enable_http2                                               = true
      + enable_tls_version_and_cipher_suite_headers                = false
      + enable_waf_fail_open                                       = false
      + enable_xff_client_port                                     = false
      + enforce_security_group_inbound_rules_on_private_link_traffic = (known after apply)
      + id                                                         = (known after apply)
      + idle_timeout                                               = 60
      + internal                                                   = false
      + ip_address_type                                            = (known after apply)
      + load_balancer_type                                         = "application"
      + name                                                       = "external-lb"
      + name_prefix                                                = (known after apply)
      + preserve_host_header                                       = false
      + security_groups                                            = [
```

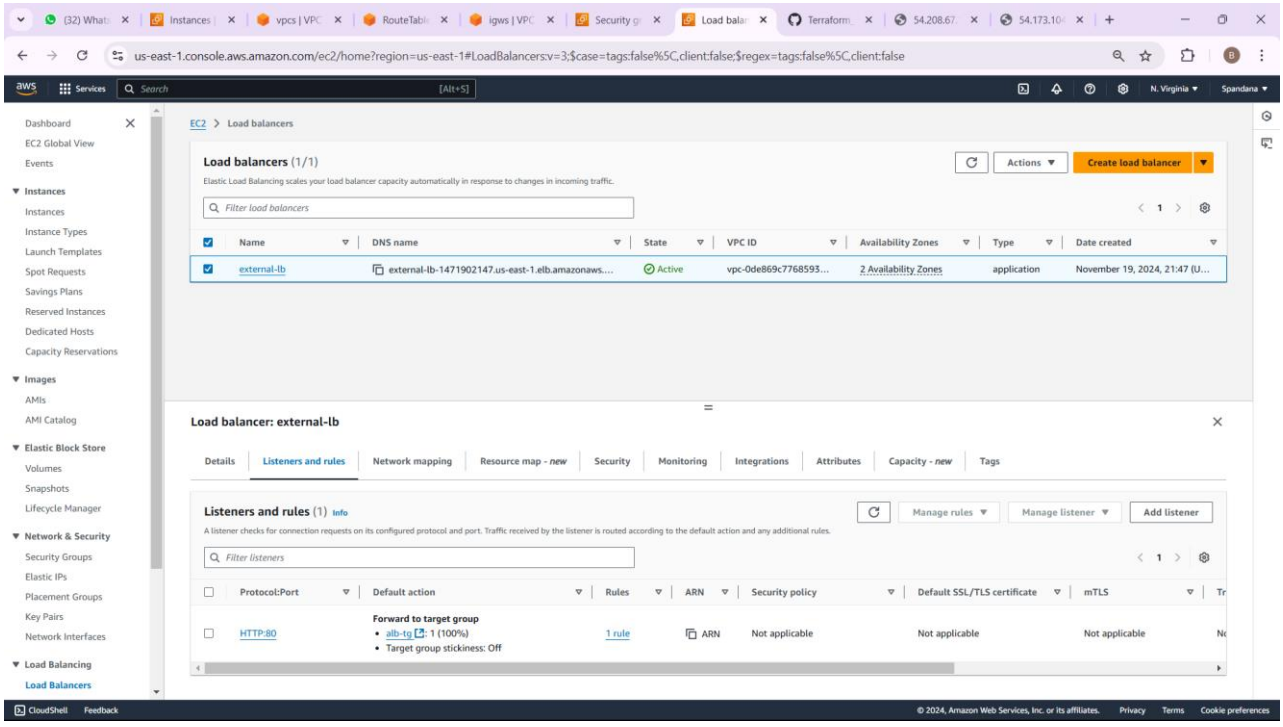

root@ip-172-31-44-73:~
]
+ subnets = [
+ "subnet-0a1b5cefd9c1167d3",
+ "subnet-0cdcddabc425855e6",
]
+ tags_all = (known after apply)
+ vpc_id = (known after apply)
+ xff_header_processing_mode = "append"
+ zone_id = (known after apply)
+ subnet_mapping (known after apply)
# aws_lb_listener.external_elb will be created
+ resource "aws_lb_listener" "external_elb" {
+ arn = (known after apply)
+ id = (known after apply)
+ load_balancer_arn = (known after apply)
+ port = 80
+ protocol = "HTTP"
+ ssl_policy = (known after apply)
+ tags_all = (known after apply)
+ tcp_idle_timeout_seconds = (known after apply)
+ default_action {
+ order = (known after apply)
+ target_group_arn = (known after apply)
+ type = "forward"
}
+ mutual_authentication (known after apply)
# aws_lb_target_group.target_elb will be created
+ resource "aws_lb_target_group" "target_elb" {
+ arn = (known after apply)
+ arn_suffix = (known after apply)
+ connection_termination = (known after apply)
+ deregistration_delay = "300"
+ id = (known after apply)
+ ip_address_type = (known after apply)
+ lambda_multi_value_headers_enabled = false
+ load_balancer_arns = (known after apply)
+ load_balancing_algorithm_type = (known after apply)
+ load_balancing_anomaly_mitigation = (known after apply)
+ load_balancing_cross_zone_enabled = (known after apply)
+ name = "alb-tg"
+ name_prefix = (known after apply)
+ port = 80
+ preserve_client_ip = (known after apply)
+ protocol = "HTTP"
+ protocol_version = (known after apply)


root@ip-172-31-44-73:~
+ port = 80
+ preserve_client_ip = (known after apply)
+ protocol = "HTTP"
+ protocol_version = (known after apply)
+ proxy_protocol_v2 = false
+ slow_start = 0
+ tags_all = (known after apply)
+ target_type = "instance"
+ vpc_id = "vpc-0de869c7768593a13"
+ health_check (known after apply)
+ stickiness (known after apply)
+ target_failover (known after apply)
+ target_group_health (known after apply)
+ target_health_state (known after apply)
# aws_lb_target_group_attachment.attachment will be created
+ resource "aws_lb_target_group_attachment" "attachment" {
+ id = (known after apply)
+ port = 80
+ target_group_arn = (known after apply)
+ target_id = "i-0758d0ee9db2f7c47"
# aws_lb_target_group_attachment.attachment2 will be created
+ resource "aws_lb_target_group_attachment" "attachment2" {
+ id = (known after apply)
+ port = 80
+ target_group_arn = (known after apply)
+ target_id = "i-01553d92ac4b9dfc4"
Plan: 5 to add, 0 to change, 0 to destroy.
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
Enter a value: yes
aws_lb.external_alb: Creating...
aws_lb_target_group.target_elb: Creating...
aws_lb_target_group.target_elb: Creation complete after 1s [id=arn:aws:elasticloadbalancing:u
s-east-1:637423550105:targetgroup/alb-tg/bebbde15c770c149]
aws_lb_target_group_attachment.attachment2: Creating...
aws_lb_target_group_attachment.attachment: Creating...
aws_lb_target_group_attachment.attachment: Creation complete after 0s [id=arn:aws:elasticload

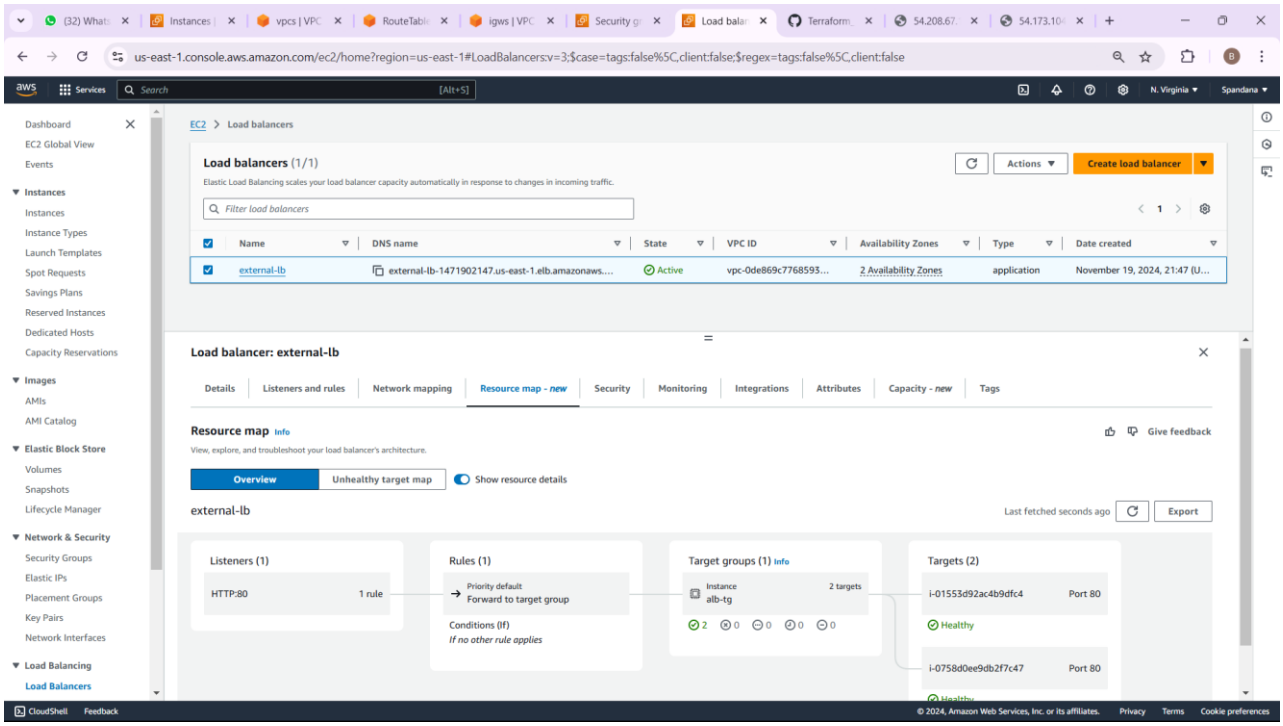
In the below image we can able to see that load balancer has been created successfully.



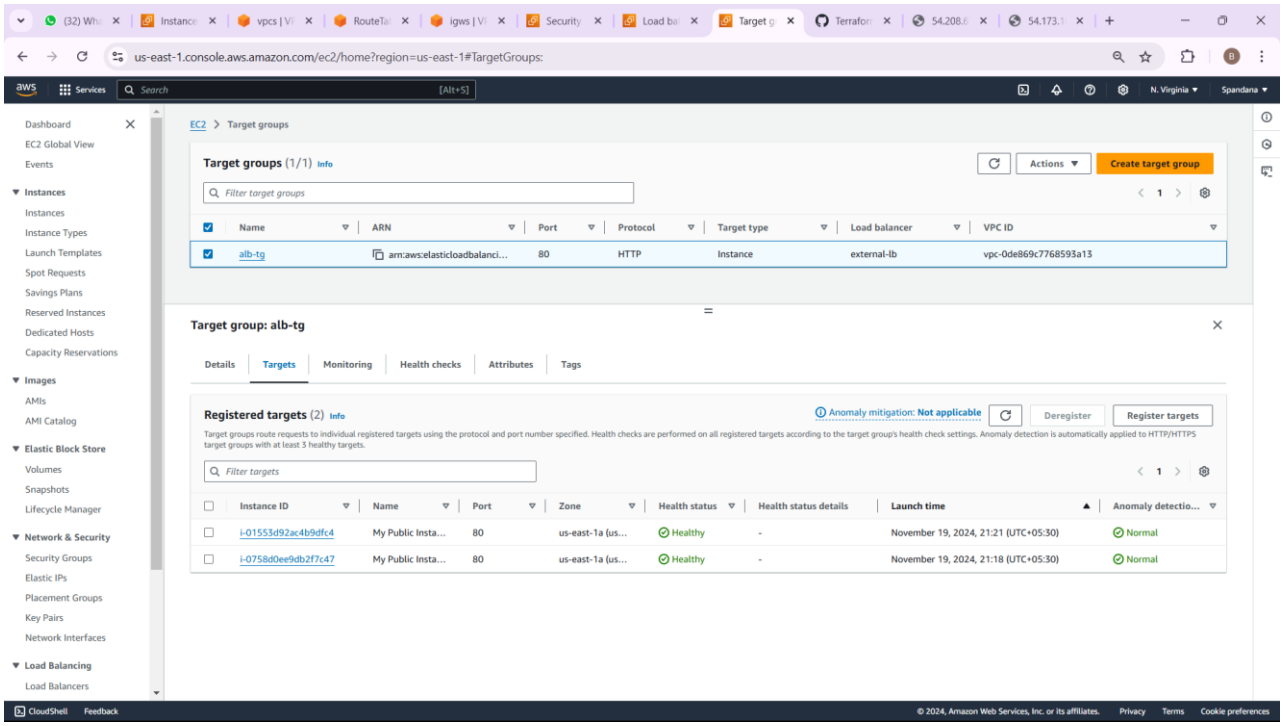
Now we can able to see the application load balancer in the below image.



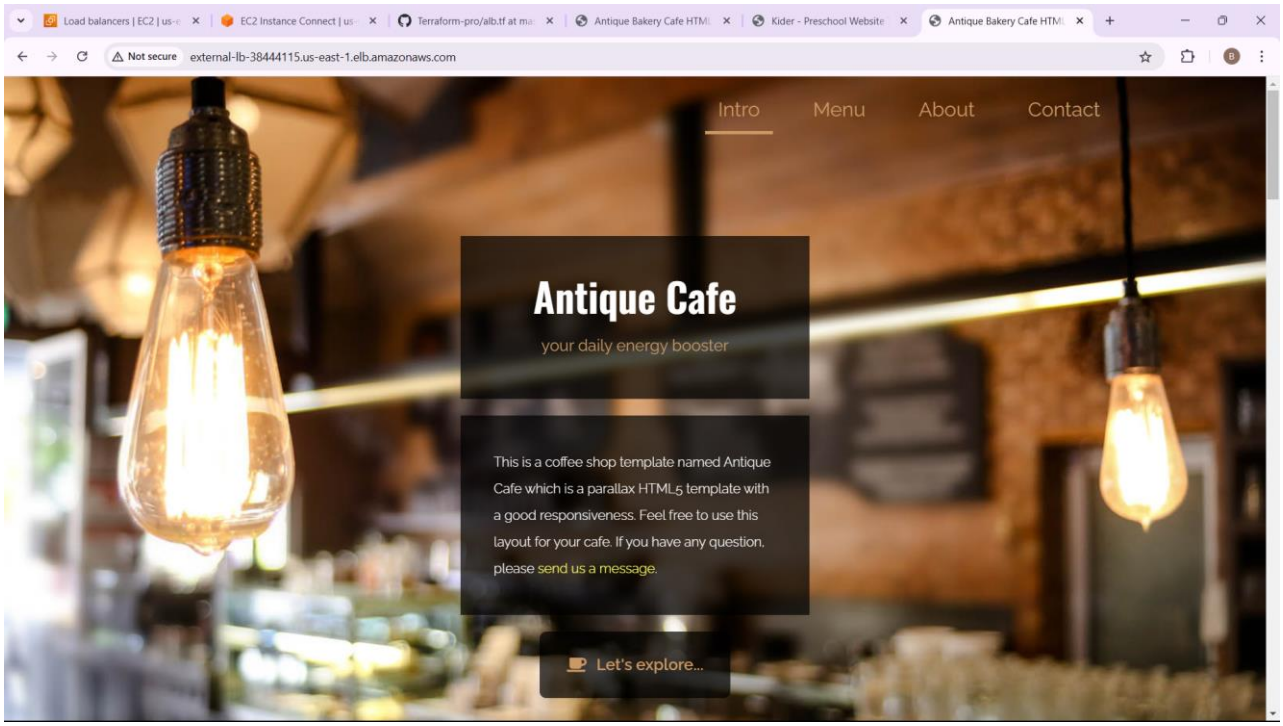
In the below image we can able to see the route map for the load balancer.



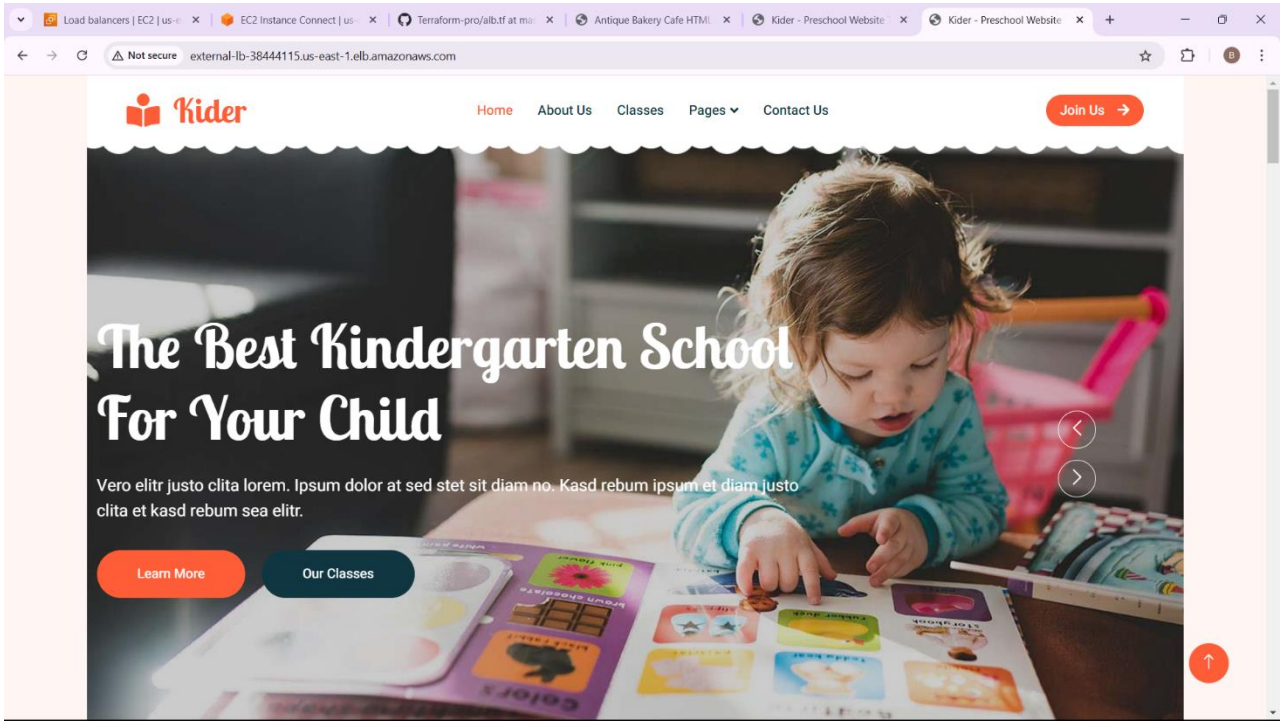
In the below image we can able to see that the target group for the load balancer and both the servers are healthy.



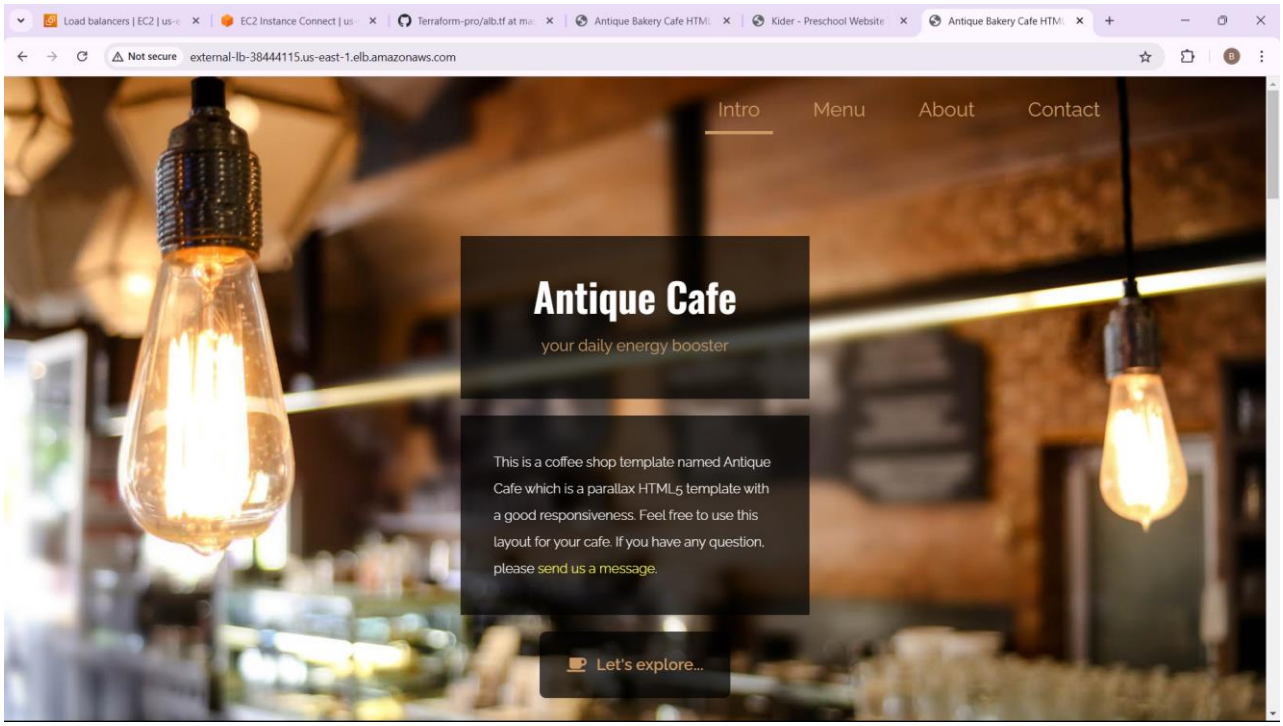
Copy the DNS of the load balancer and paste it in browser we can able to see the servers are working properly it will fluctuate the servers.



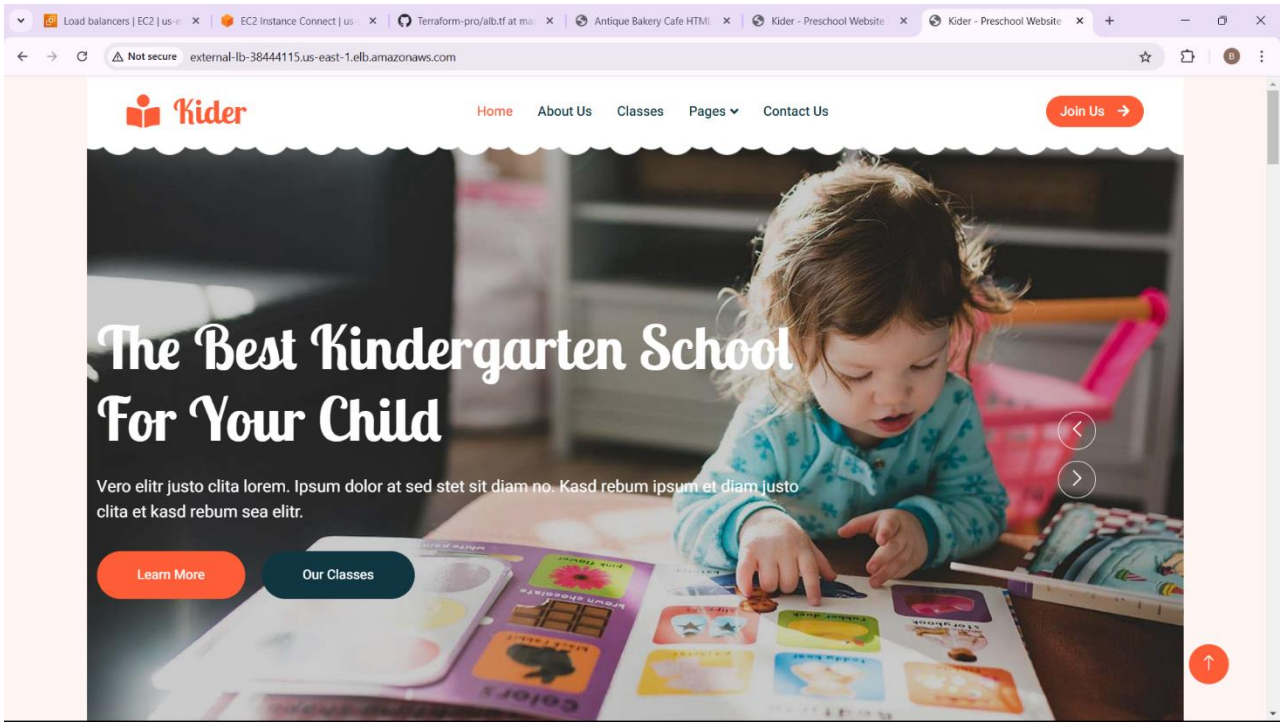
It shows the second server when I reload.



After reloading it shows the below web application

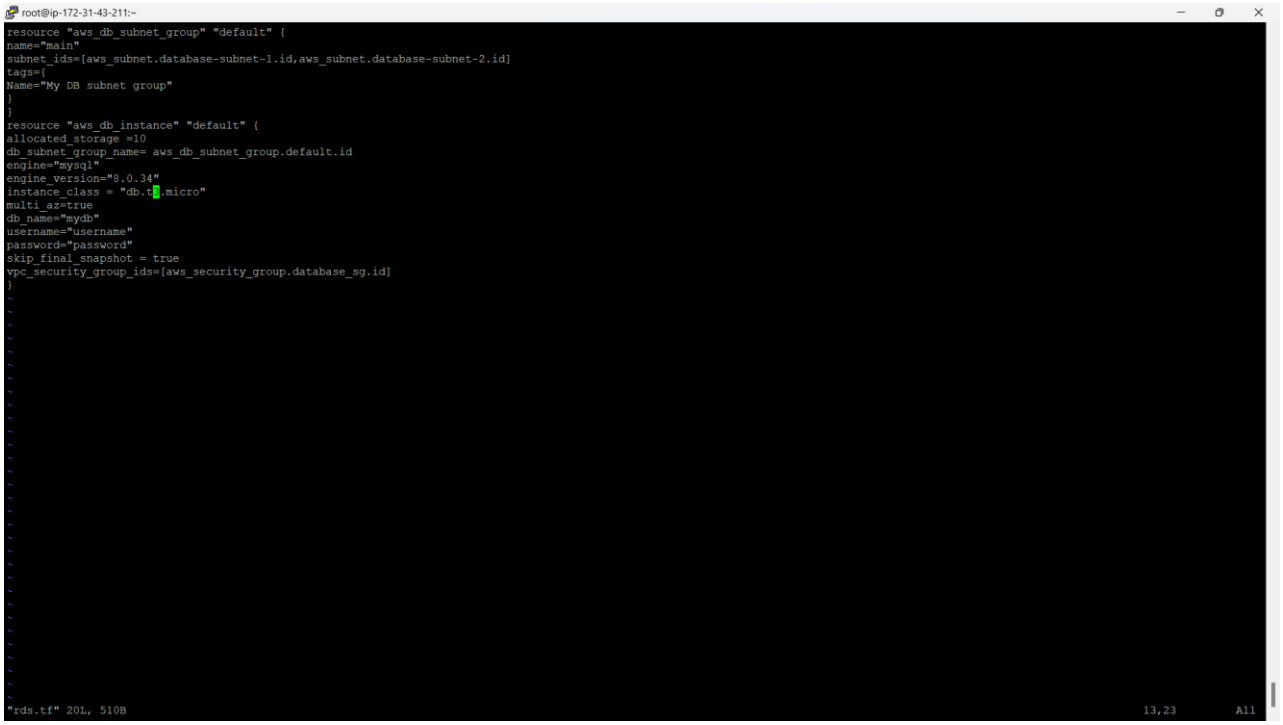


The load is distributing through two servers.

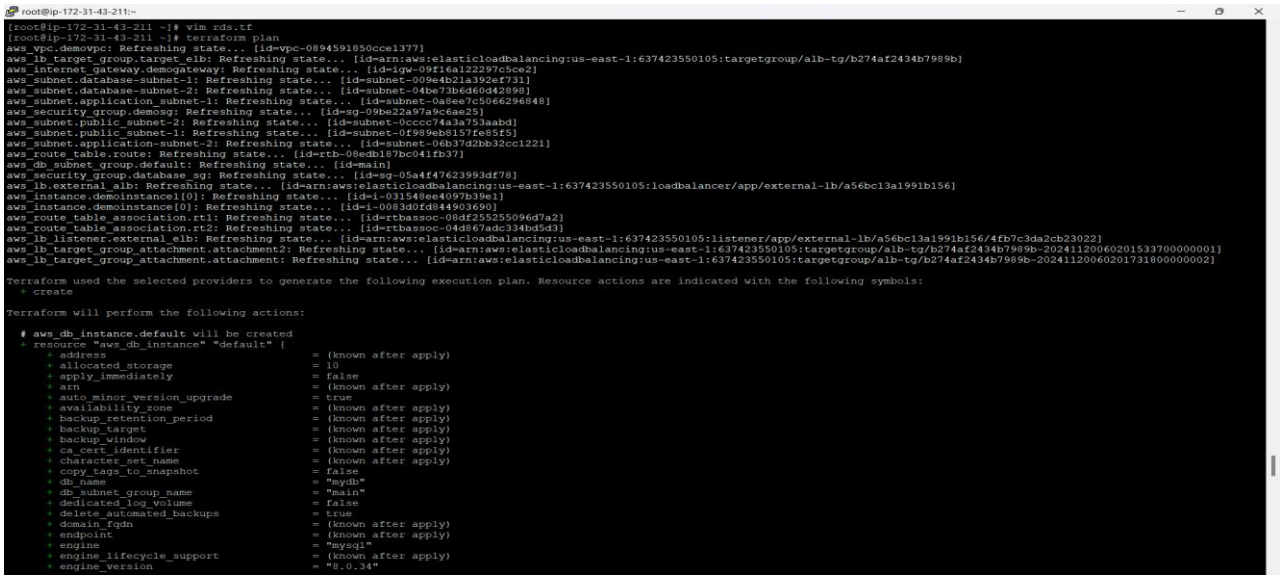


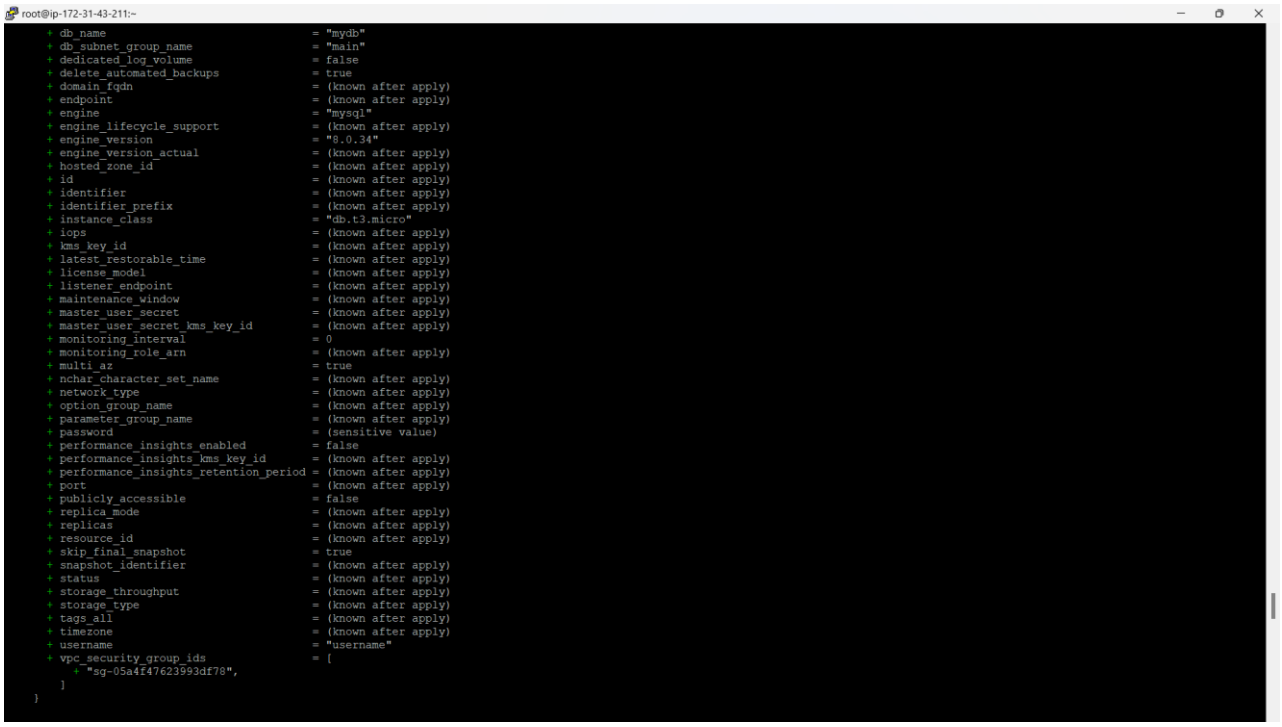
STEP 11: CREATING A FILE FOR AN RDS INSTANCE

The **rds.tf** file defines a managed database instance in AWS. It specifies the type of database, storage size, and networking rules. This setup ensures your database is secure and only accessible by the required application or resources.

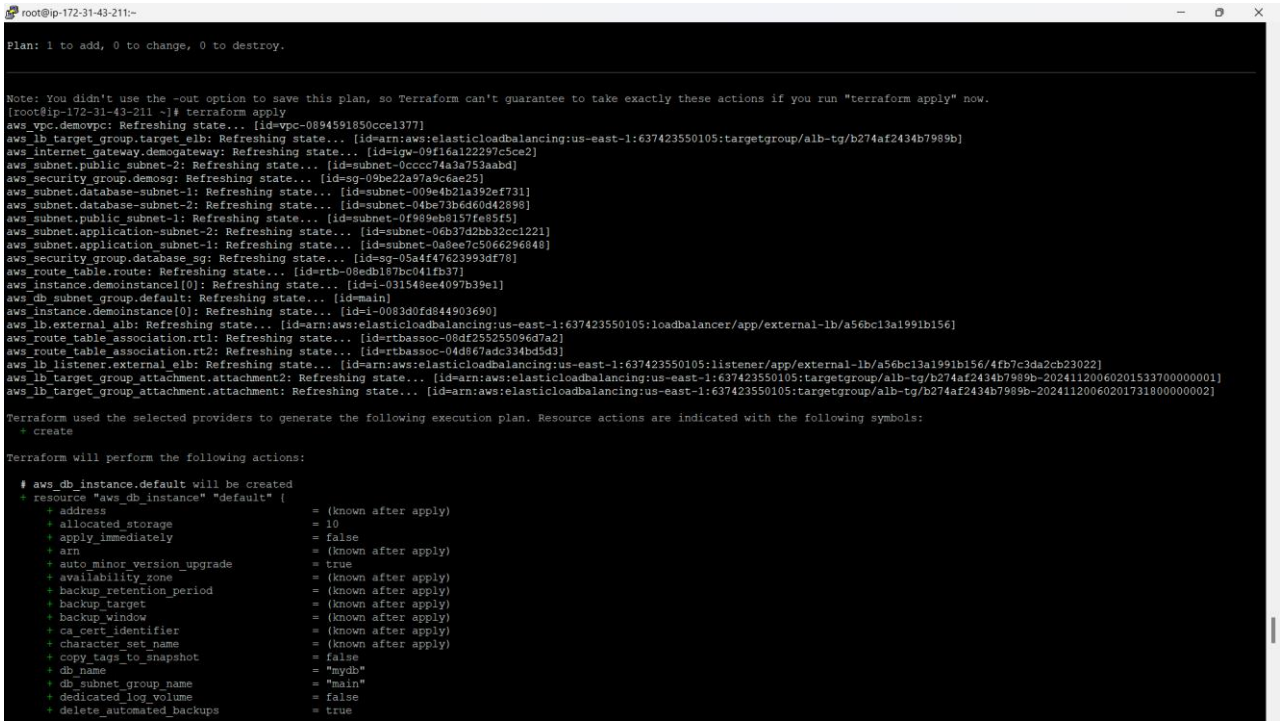


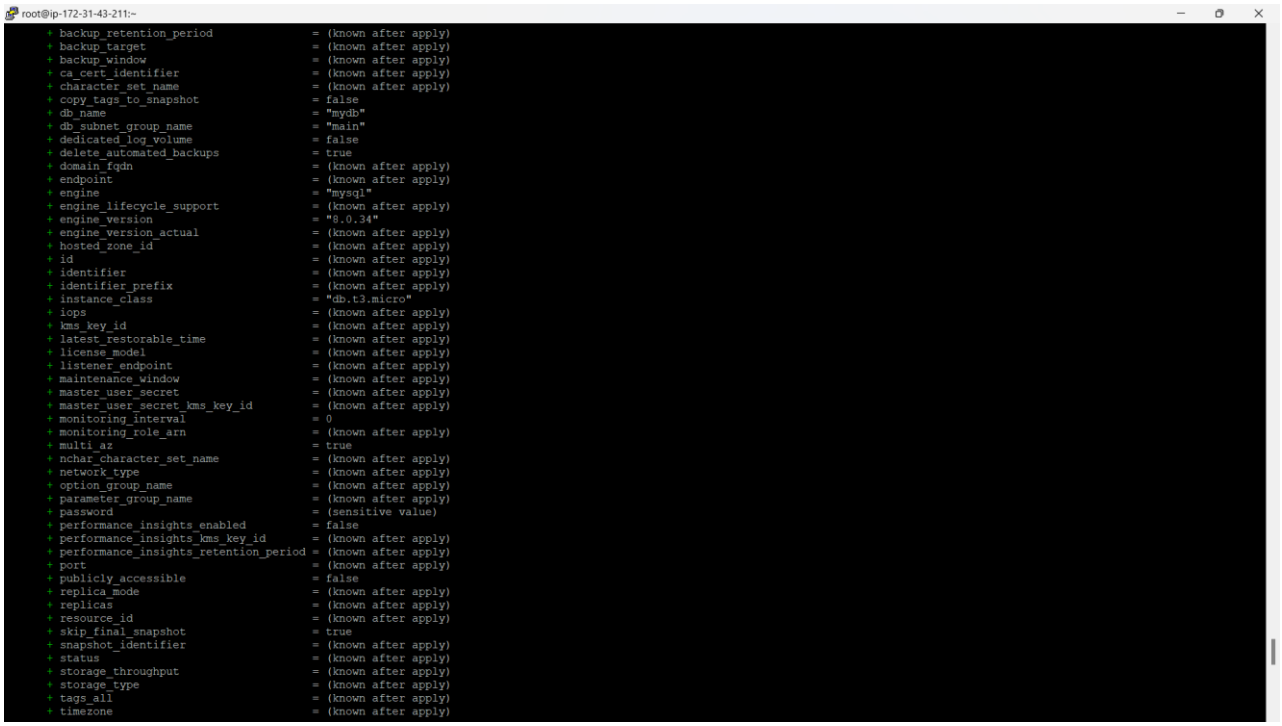
Then i have run the terraform plan command to check the configuration before creating an rds.

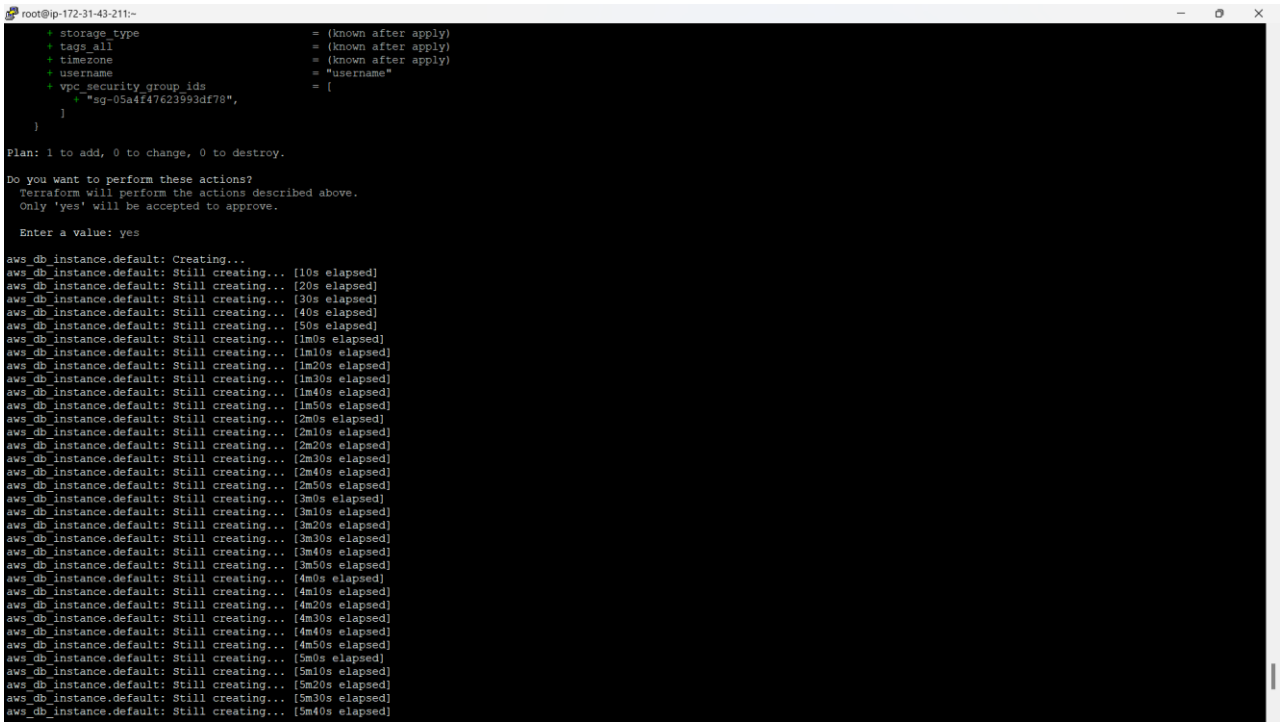




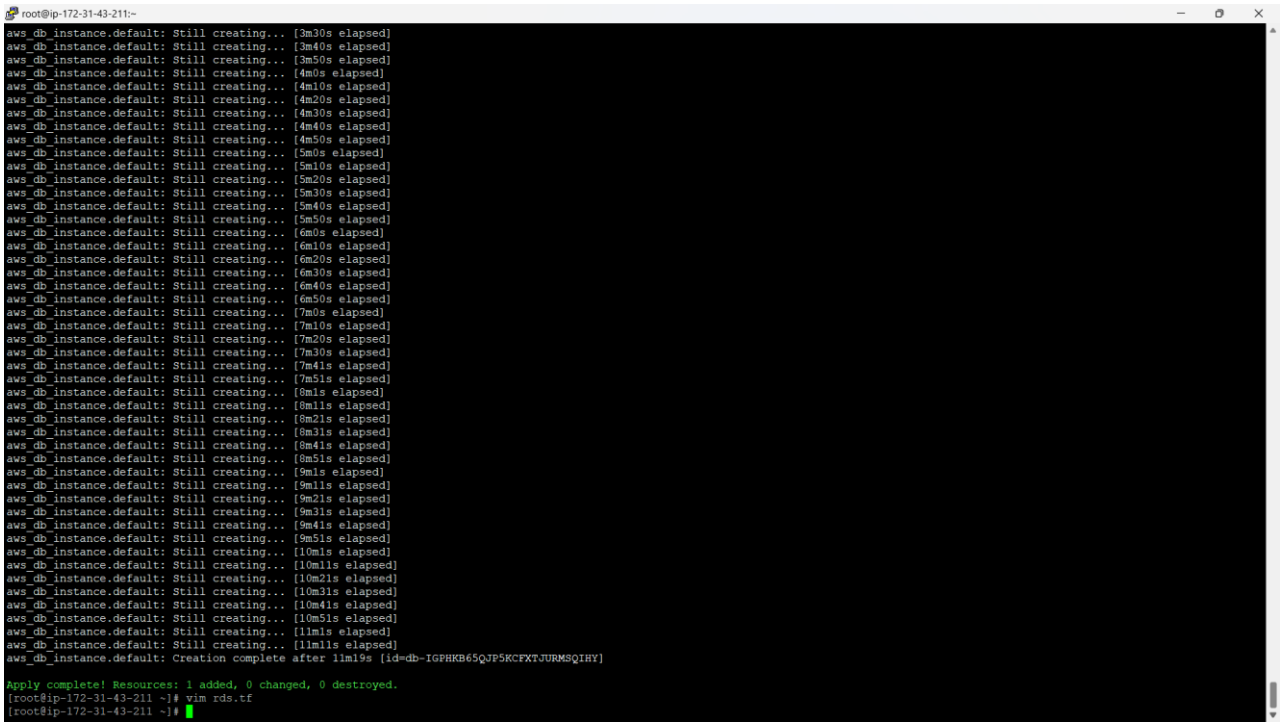
In the below image we can see terraform apply to create an rds.



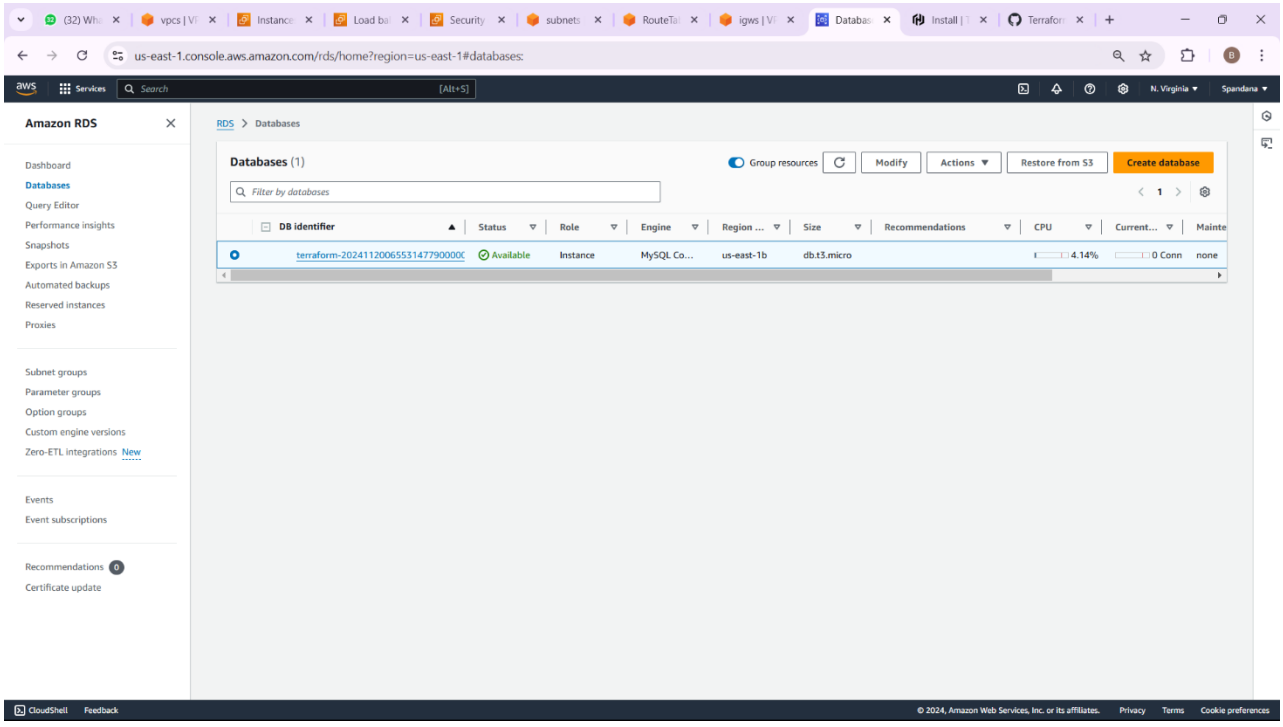

root@ip-172-31-43-211:~
+ backup_retention_period = (known after apply)
+ backup_target = (known after apply)
+ backup_window = (known after apply)
+ ca_cert_identifier = (known after apply)
+ character_set_name = (known after apply)
+ copy_tags_to_snapshot = false
+ db_name = "mydb"
+ db_subnet_group_name = "main"
+ dedicated_log_volume = false
+ delete_automated_backups = true
+ domain_fqdn = (known after apply)
+ endpoint = (known after apply)
+ engine = "mysql"
+ engine_lifecycle_support = (known after apply)
+ engine_version = "8.0.34"
+ engine_version_actual = (known after apply)
+ hosted_zone_id = (known after apply)
+ id = (known after apply)
+ identifier = (known after apply)
+ identifier_prefix = (known after apply)
+ instance_class = "db.t3.micro"
+ iops = (known after apply)
+ kms_key_id = (known after apply)
+ latest_restorable_time = (known after apply)
+ license_model = (known after apply)
+ listener_endpoint = (known after apply)
+ maintenance_window = (known after apply)
+ master_user_secret = (known after apply)
+ master_user_secret_kms_key_id = (known after apply)
+ monitoring_interval = 0
+ monitoring_role_arn = (known after apply)
+ multi_az = true
+ nchar_character_set_name = (known after apply)
+ network_type = (known after apply)
+ option_group_name = (known after apply)
+ parameter_group_name = (known after apply)
+ password = (sensitive value)
+ performance_insights_enabled = false
+ performance_insights_kms_key_id = (known after apply)
+ performance_insights_retention_period = (known after apply)
+ port = (known after apply)
+ publicly_accessible = false
+ replica_mode = (known after apply)
+ replicas = (known after apply)
+ resource_id = (known after apply)
+ skip_final_snapshot = true
+ snapshot_identifier = (known after apply)
+ status = (known after apply)
+ storage_throughput = (known after apply)
+ storage_type = (known after apply)
+ tags_all = (known after apply)
+ timezone = (known after apply)

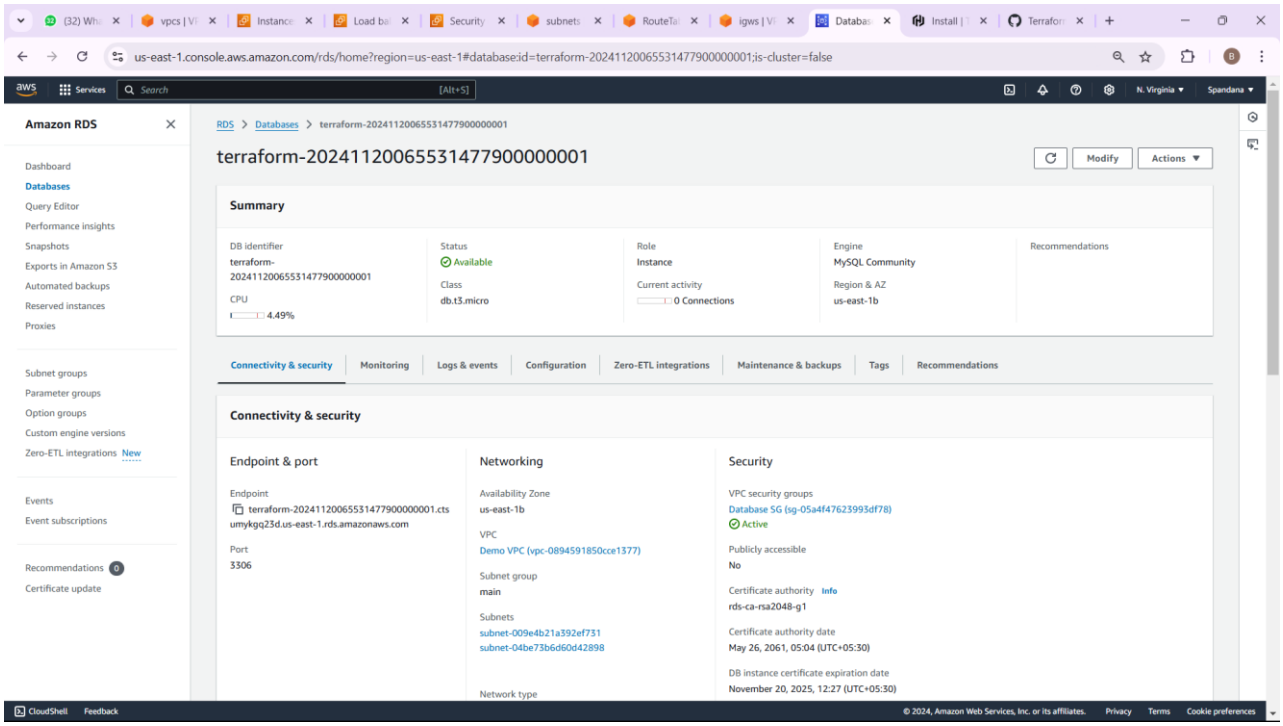

root@ip-172-31-43-211:~
+ storage_type = (known after apply)
+ tags_all = (known after apply)
+ timezone = (known after apply)
+ username = "username"
+ vpc_security_group_ids = [
+ "sg-05a4f47623993df78",
]
}
Plan: 1 to add, 0 to change, 0 to destroy.
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
Enter a value: yes
aws_db_instance.default: Creating...
aws_db_instance.default: Still creating... [10s elapsed]
aws_db_instance.default: Still creating... [20s elapsed]
aws_db_instance.default: Still creating... [30s elapsed]
aws_db_instance.default: Still creating... [40s elapsed]
aws_db_instance.default: Still creating... [50s elapsed]
aws_db_instance.default: Still creating... [1m0s elapsed]
aws_db_instance.default: Still creating... [1m10s elapsed]
aws_db_instance.default: Still creating... [1m20s elapsed]
aws_db_instance.default: Still creating... [1m30s elapsed]
aws_db_instance.default: Still creating... [1m40s elapsed]
aws_db_instance.default: Still creating... [1m50s elapsed]
aws_db_instance.default: Still creating... [2m0s elapsed]
aws_db_instance.default: Still creating... [2m10s elapsed]
aws_db_instance.default: Still creating... [2m20s elapsed]
aws_db_instance.default: Still creating... [2m30s elapsed]
aws_db_instance.default: Still creating... [2m40s elapsed]
aws_db_instance.default: Still creating... [2m50s elapsed]
aws_db_instance.default: Still creating... [3m0s elapsed]
aws_db_instance.default: Still creating... [3m10s elapsed]
aws_db_instance.default: Still creating... [3m20s elapsed]
aws_db_instance.default: Still creating... [3m30s elapsed]
aws_db_instance.default: Still creating... [3m40s elapsed]
aws_db_instance.default: Still creating... [3m50s elapsed]
aws_db_instance.default: Still creating... [4m0s elapsed]
aws_db_instance.default: Still creating... [4m10s elapsed]
aws_db_instance.default: Still creating... [4m20s elapsed]
aws_db_instance.default: Still creating... [4m30s elapsed]
aws_db_instance.default: Still creating... [4m40s elapsed]
aws_db_instance.default: Still creating... [4m50s elapsed]
aws_db_instance.default: Still creating... [5m0s elapsed]
aws_db_instance.default: Still creating... [5m10s elapsed]
aws_db_instance.default: Still creating... [5m20s elapsed]
aws_db_instance.default: Still creating... [5m30s elapsed]
aws_db_instance.default: Still creating... [5m40s elapsed]

Successfully created RDS.

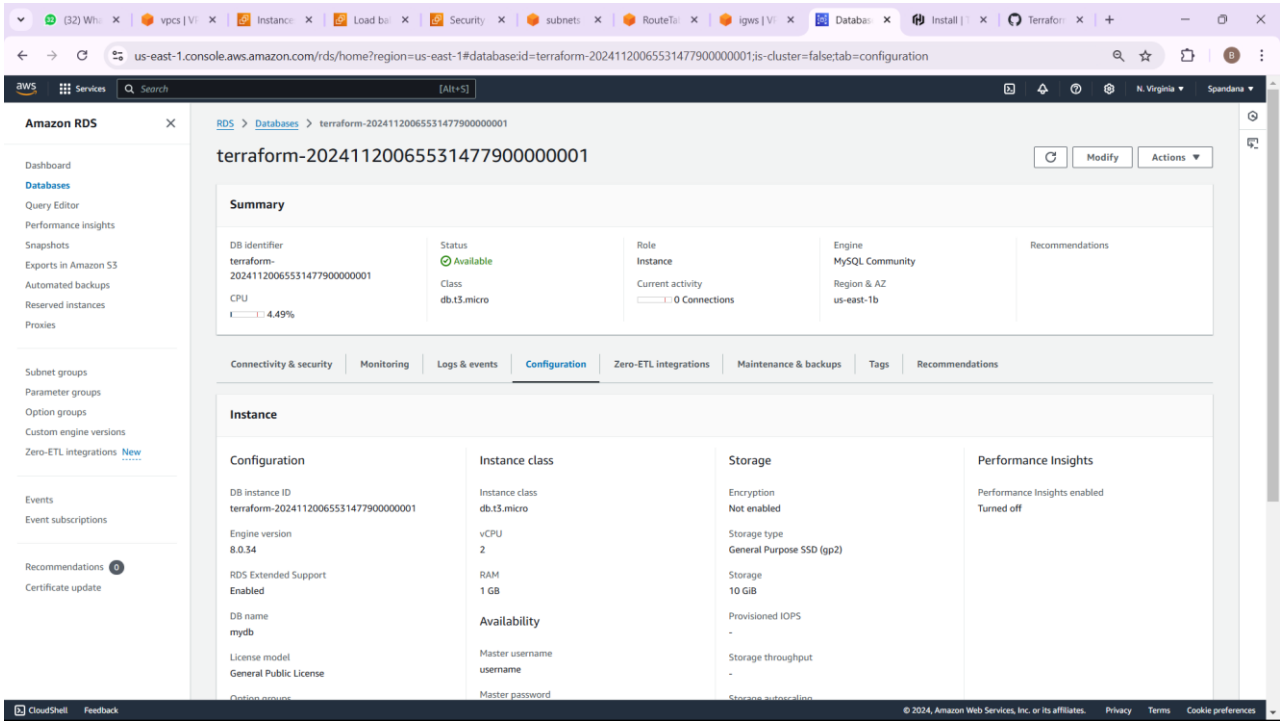


In the below image we can able to see that the database has been created successfully.



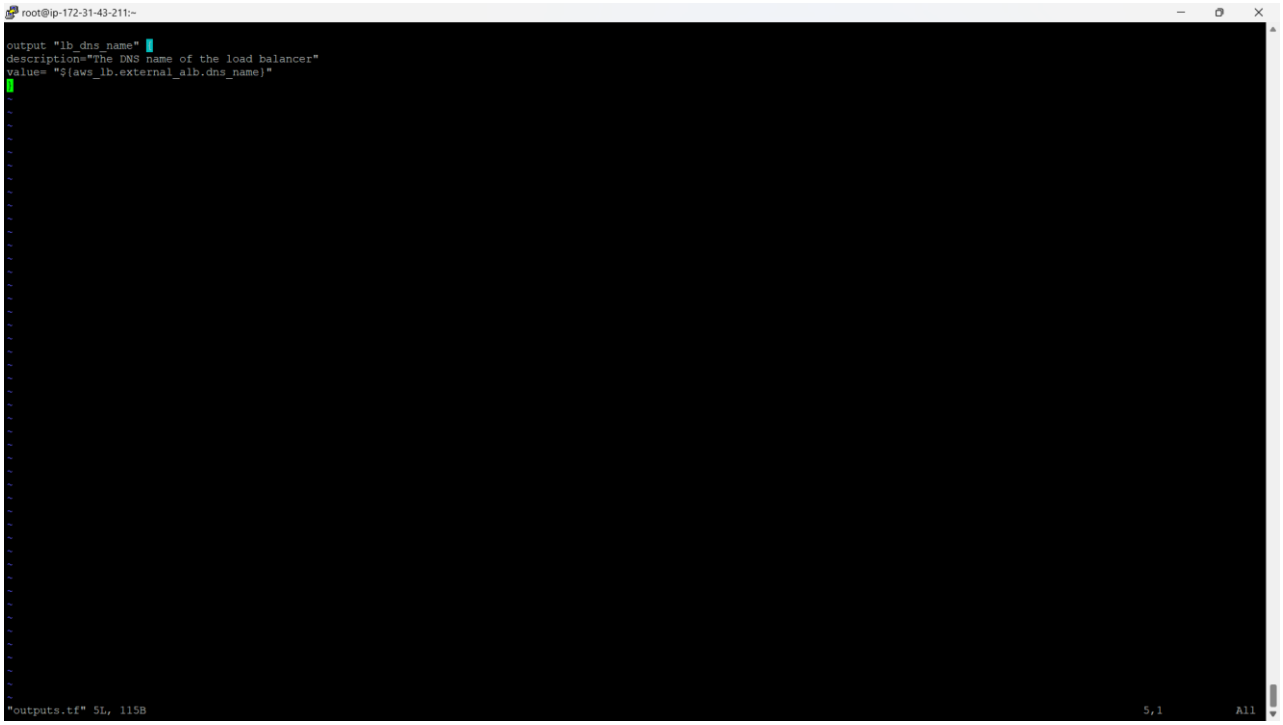


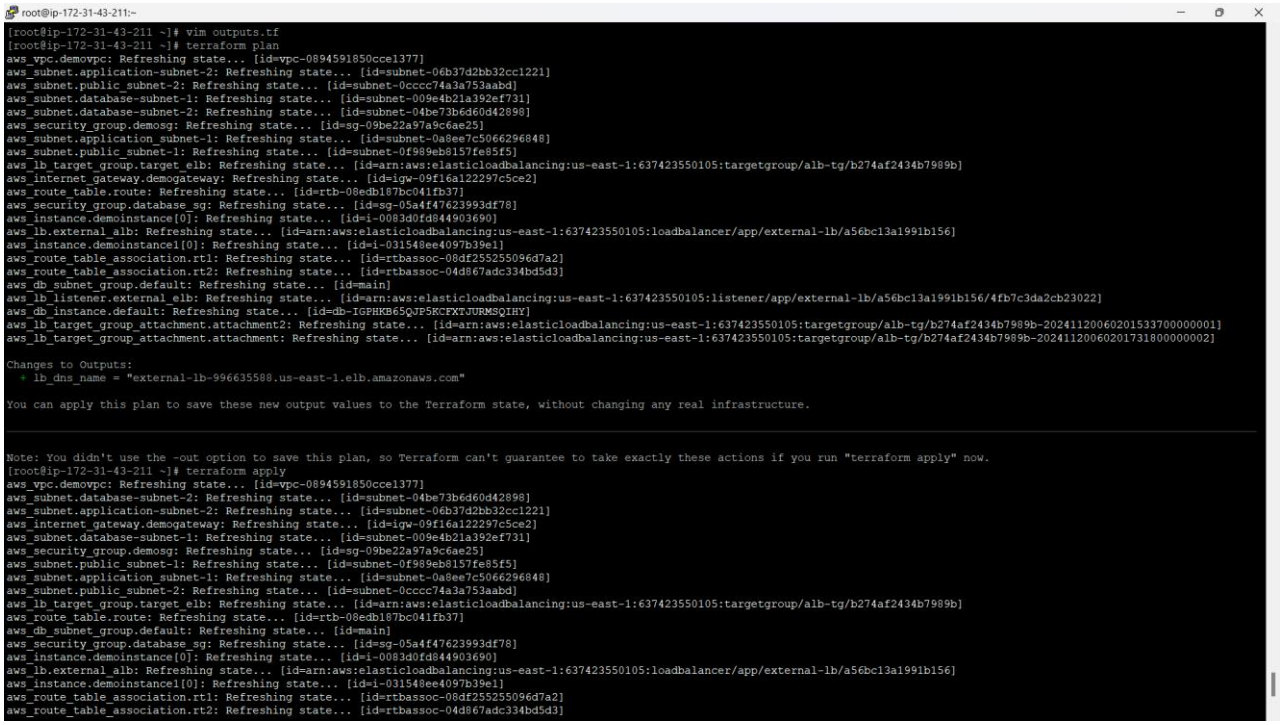
In the below image we can able to see that the configuration of the RDS.

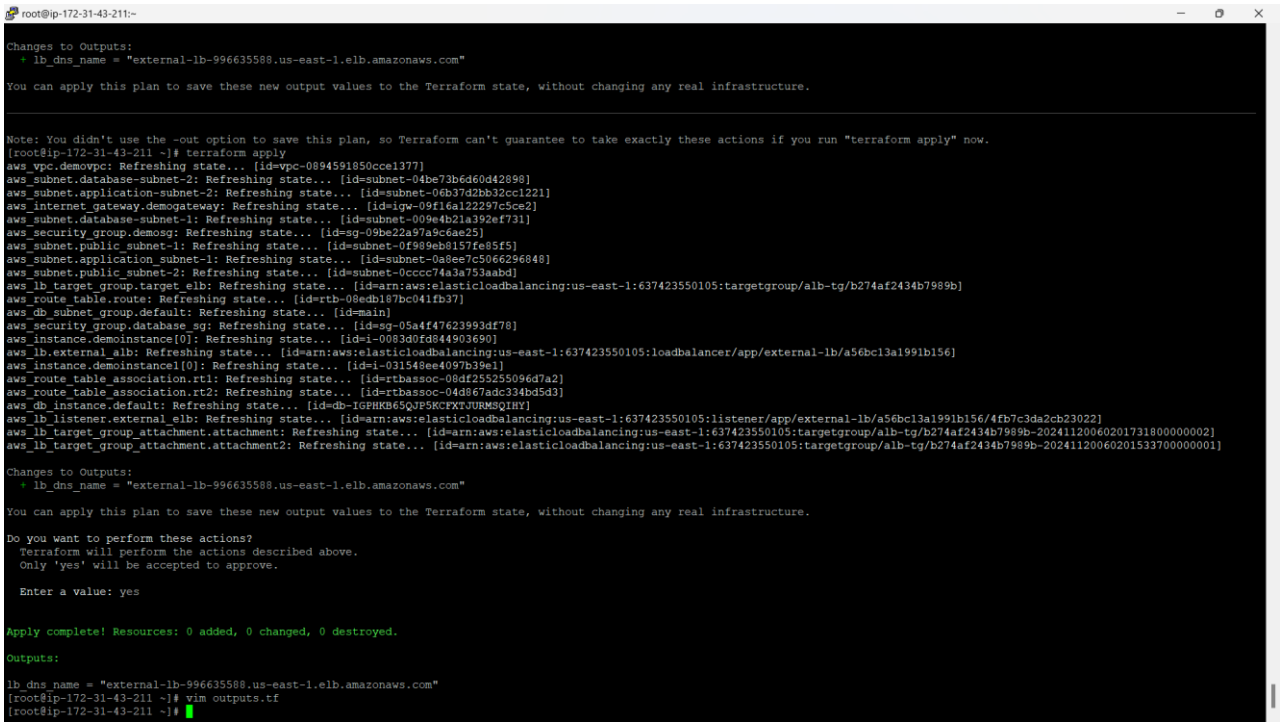


STEP 12: CREATING A FILE FOR OUTPUTS

The **outputs.tf** file in Terraform is used to display the results of specific resources or attributes after applying the configuration. This helps users easily retrieve important information, such as instance IP addresses, database connection endpoints, or load balancer DNS names.






root@ip-172-31-43-211:~
Changes to Outputs:
+ lb_dns_name = "external-lb-996635588.us-east-1.elb.amazonaws.com"
You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
[root@ip-172-31-43-211 ~]# terraform apply
aws_vpc.demovpc: Refreshing state... [id=vpc-0894591850cce1377]
aws_subnet.database-subnet-2: Refreshing state... [id=subnet-04be73b6d60d42898]
aws_subnet.application-subnet-2: Refreshing state... [id=subnet-06b37d2bb32cc1221]
aws_internet_gateway.demogateway: Refreshing state... [id=igw-09f16a122297c5ce2]
aws_subnet.database-subnet-1: Refreshing state... [id=subnet-009e4b21a392ef731]
aws_security_group.demosg: Refreshing state... [id=sg-09be22a97a9c6ae25]
aws_subnet.public_subnet-1: Refreshing state... [id=subnet-0f989eb8157fe85f5]
aws_subnet.application_subnet-1: Refreshing state... [id=subnet-0a8ee7c50662968d8]
aws_subnet.public_subnet-2: Refreshing state... [id=subnet-0cccc74a3a753aabd]
aws_lb_target_group.target_elb: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:637423550105:targetgroup/alb-tg/b274af2434b7989b]
aws_route_table.route: Refreshing state... [id=rtb-08edb187bc041fb37]
aws_db_subnet_group.default: Refreshing state... [id=main]
aws_security_group.database_sg: Refreshing state... [id=sg-05a4f47623993df78]
aws_instance.demoinstance[0]: Refreshing state... [id=i-0083d0fd844903690]
aws_lb.external_alb: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:637423550105:loadbalancer/app/external-lb/a56bc13a1991b156]
aws_instance.demoinstance1[0]: Refreshing state... [id=i-031548ee4097b39e1]
aws_route_table_association.rt1: Refreshing state... [id=rtbassoc-08df2552550096d7a2]
aws_route_table_association.rt2: Refreshing state... [id=rtbassoc-04d867adc334bd5d3]
aws_db_instance.default: Refreshing state... [id=db-IGPHKB65QJP5KCFXTJURM8QIHY]
aws_lb_listener.external_elb: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:637423550105:listener/app/external-lb/a56bc13a1991b156/4fb7c3da2cb23022]
aws_lb_target_group_attachment.attachment: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:637423550105:targetgroup/alb-tg/b274af2434b7989b-20241120060201731800000002]
aws_lb_target_group_attachment.attachment2: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:637423550105:targetgroup/alb-tg/b274af2434b7989b-20241120060201533700000001]
Changes to Outputs:
+ lb_dns_name = "external-lb-996635588.us-east-1.elb.amazonaws.com"
You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
Enter a value: yes
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
Outputs:
lb_dns_name = "external-lb-996635588.us-east-1.elb.amazonaws.com"
[root@ip-172-31-43-211 ~]# vim outputs.tf
[root@ip-172-31-43-211 ~]#