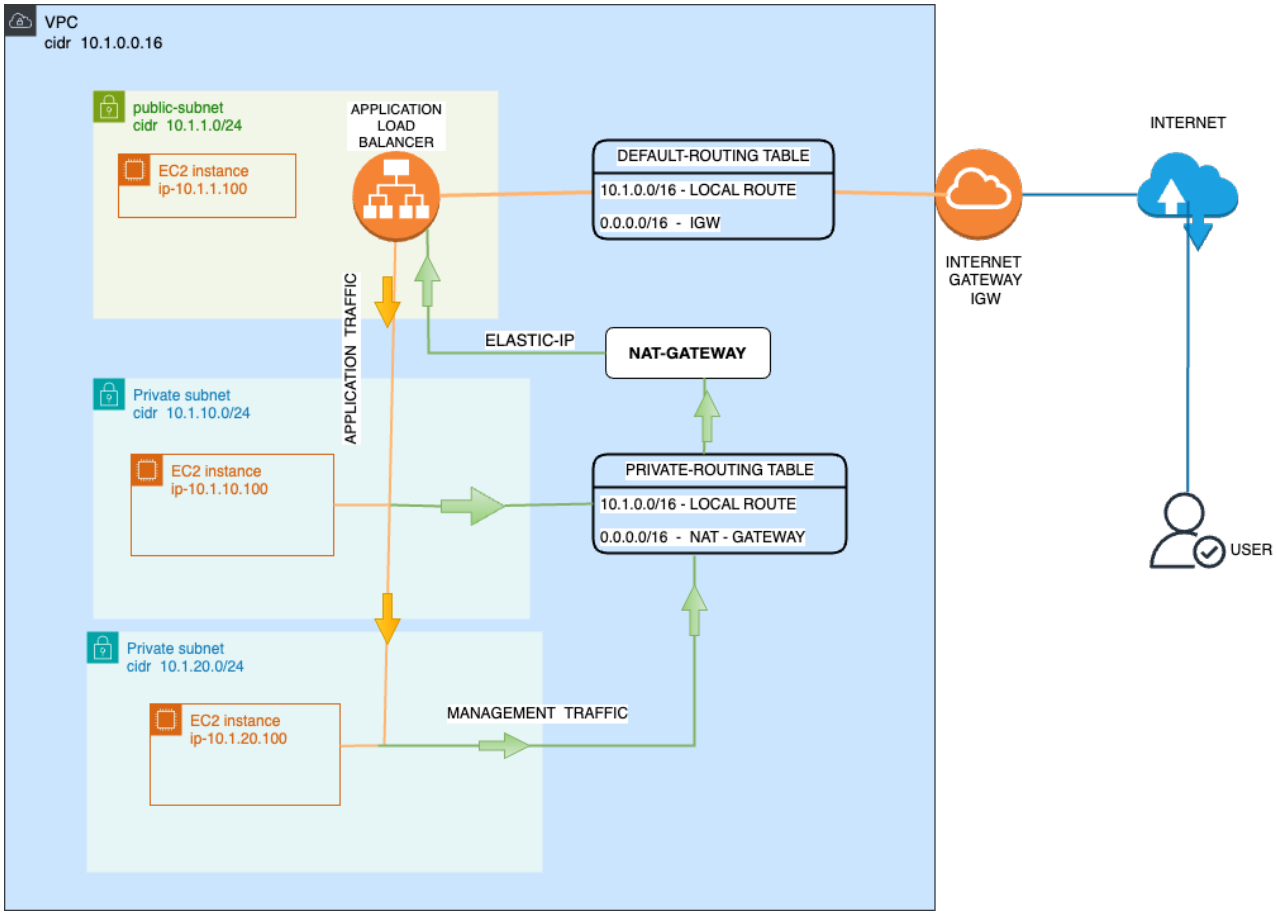
M.sai revanth

AWS project-02

DEPLOYING A 3-TIRE WEB APPLICATION USING TERRAFORM CODE

Architecture of 3-tire web application:



In this project we are going to deploy a 3-tire web application using terraform code.

- AWS services we are going to use in this project are:

1. VPC
2. EC-2
3. Application load balancer.

- Terraform code which is used in this project are gathered form reading online blogs and taking guidance from my mentor and making changes in the source code.
- Initially you need to be in familiar with some of the terraform codes, like.

1. **Terraform init:**

   with this command terraform will install all the required plugins.

2. **Terraform validate:**

   It verify the code syntax

3. **Terraform plan:**

   creates a dry-run, determining what actions are necessary to achieve the desired state defined in the Terraform configuration files.

4. **Terraform FMT:**

   This will format the code.

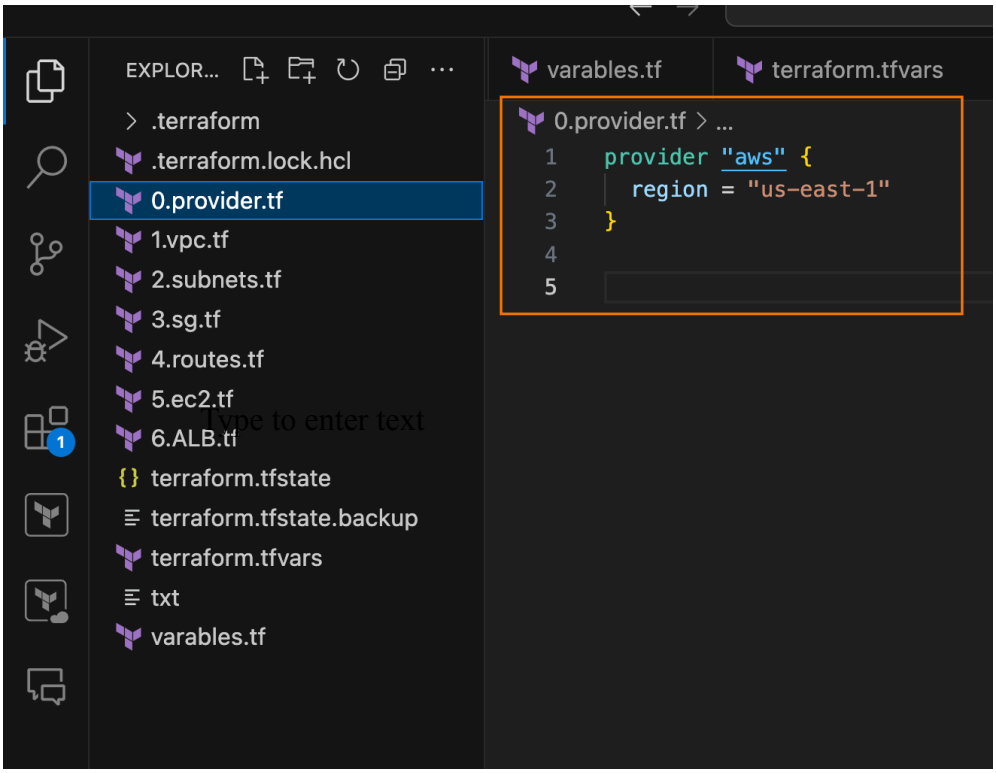
5. **Terraform apply:**

   “terraform apply” his will apply the code and execute the code in console

   “terraform apply -auto —approve”. It will apply without prompting you for permissions.

Steps:

1. Terraform can used to create infrastructure in any cloud (IaaC) services we need to declare AWS as our provider.

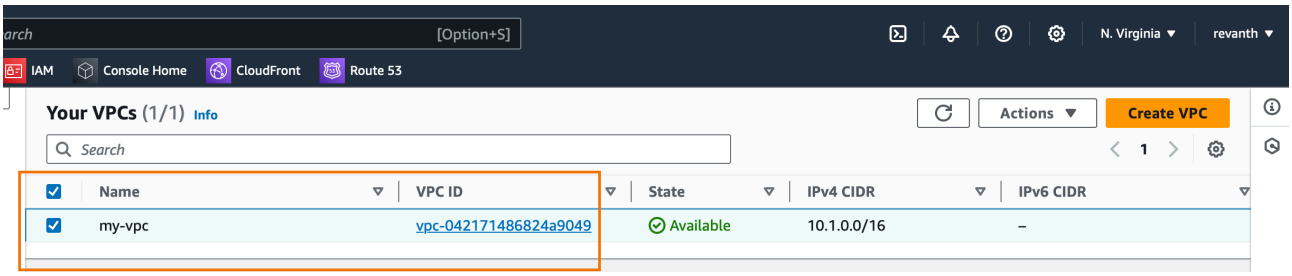


2. Let’s write a code for VPC by giving name and required cidr block.

```
resource "aws_vpc" "vpc" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = true

  tags = {
    Name = "my-vpc"
  }
}
```

3. Deploy the code using terraform.

- Terraform init
- Terraform validate
- Terraform plan
- Terraform apply -auto -approve

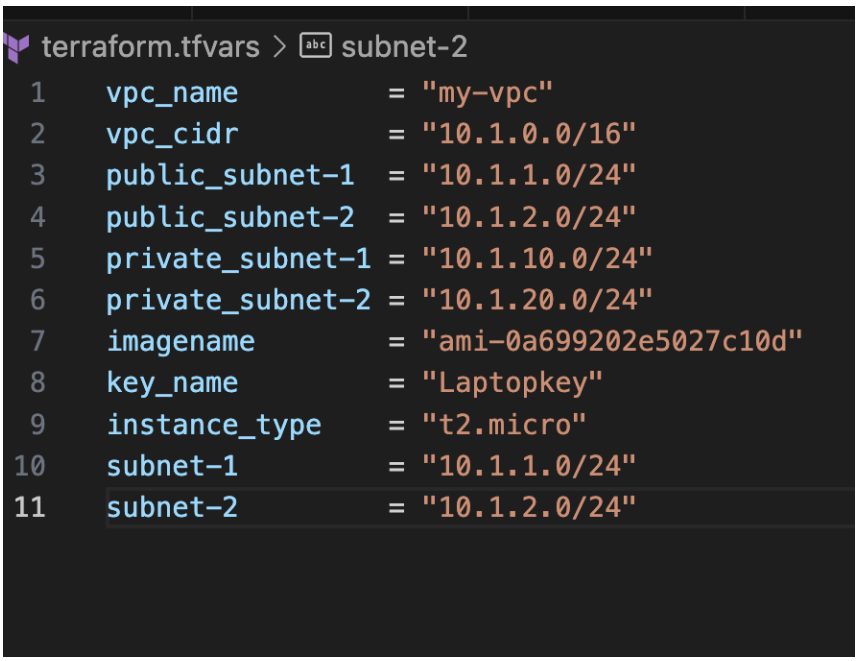


4. Form the above code the name and cidr of VPC is taken from variables files.

varables.tf > variable "subnet-2"

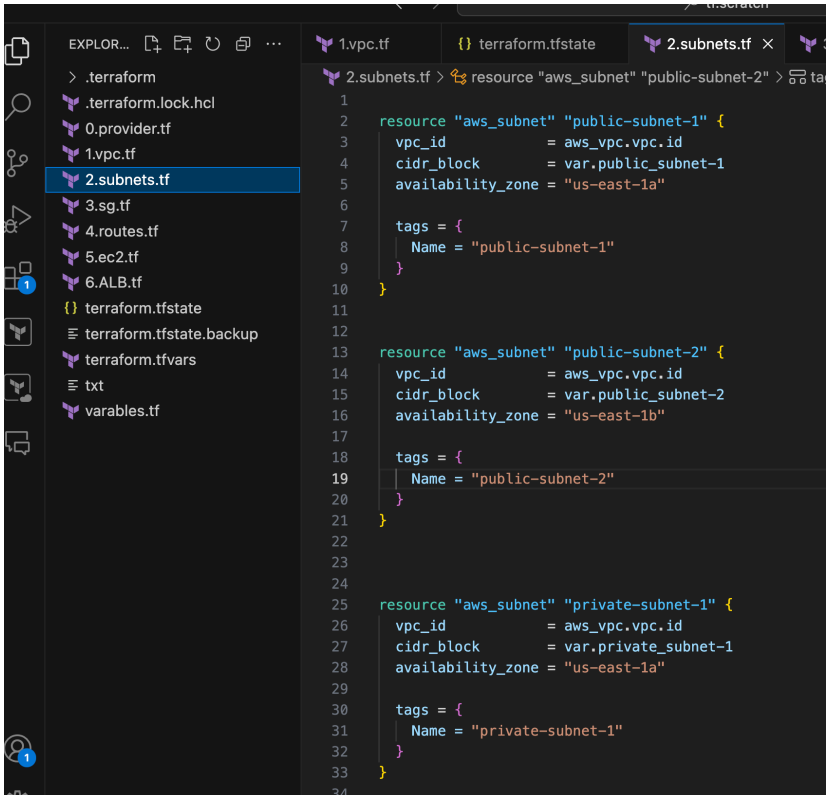
```
1   variable "vpc_name" {}
2   variable "vpc_cidr" {}
3   variable "public_subnet-1" {}
4   variable "public_subnet-2" {}
5   variable "private_subnet-1" {}
6   variable "private_subnet-2" {}
7   variable "imagename" {}
8   variable "key_name" {}
9   variable "instance_type" {}
10  variable "subnet-1" {}
11  variable "subnet-2" {}
```

5. We don’t store our cidrs and names in variables file they are stored in **TFVARS** file.

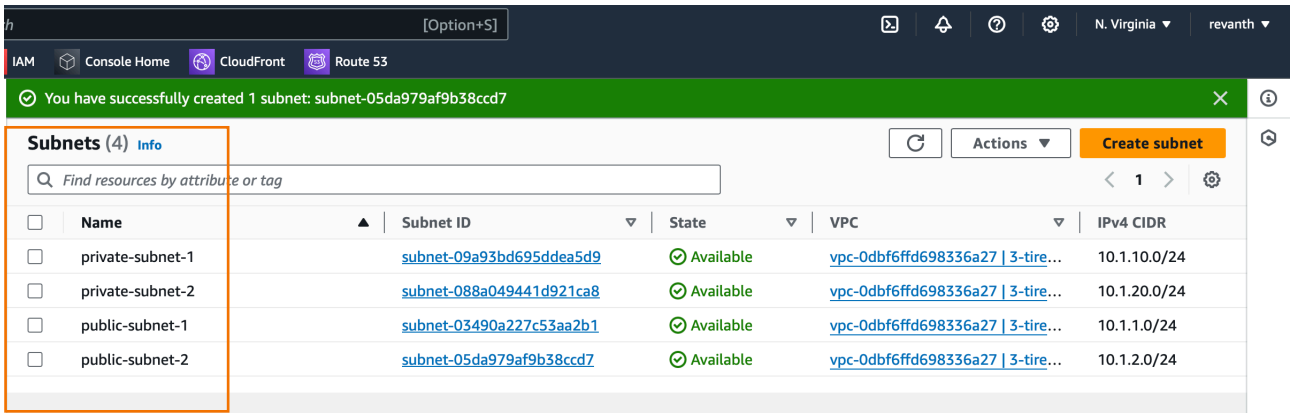


6. All the resources which we are going to create will take values from this file.
7. Now create subnets.

- 2- public of cidr 10.1.1.0/24 and 10.1.2.0/24
- 2-private of cidr 10.1.10.0/24 and 10.1.20.0/24



8. Deploy subnets.



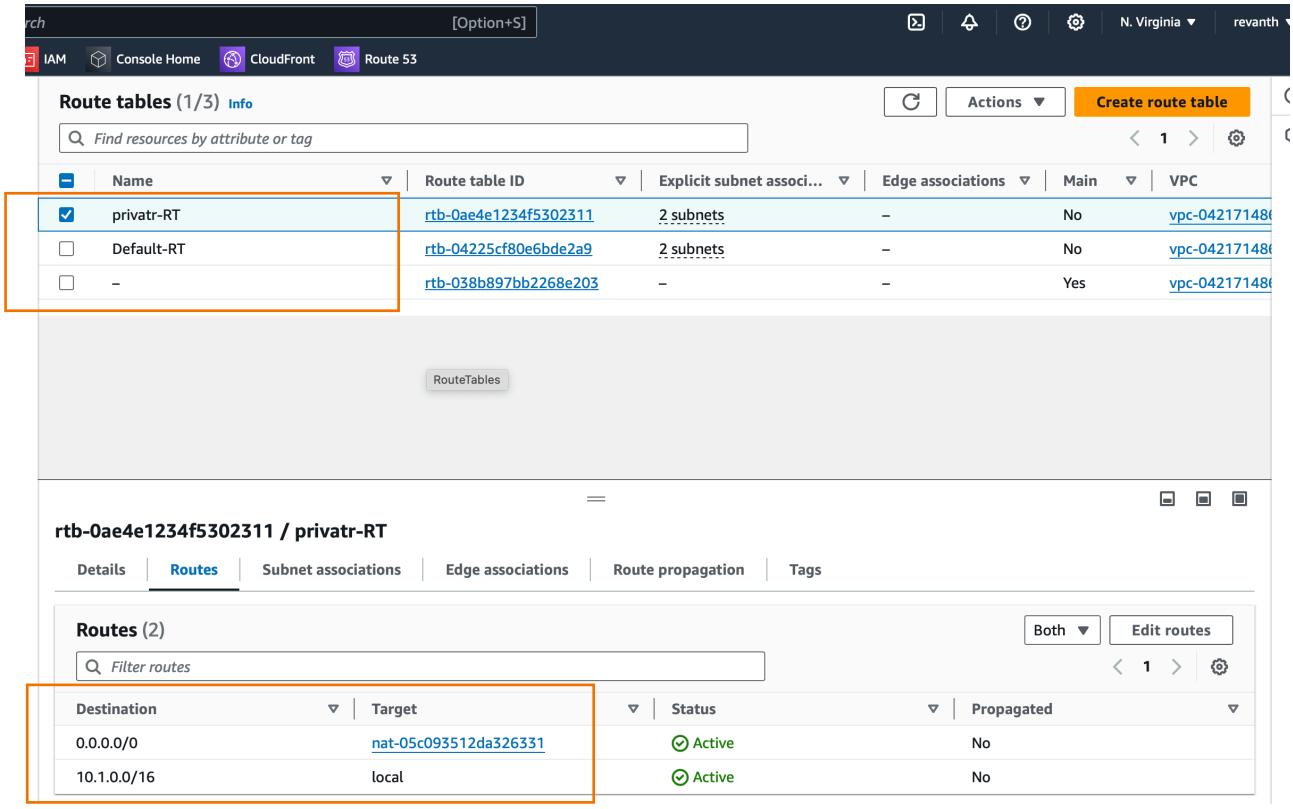
9. Create routing table public-RT and private-RT and add public and private subnets with respectively.

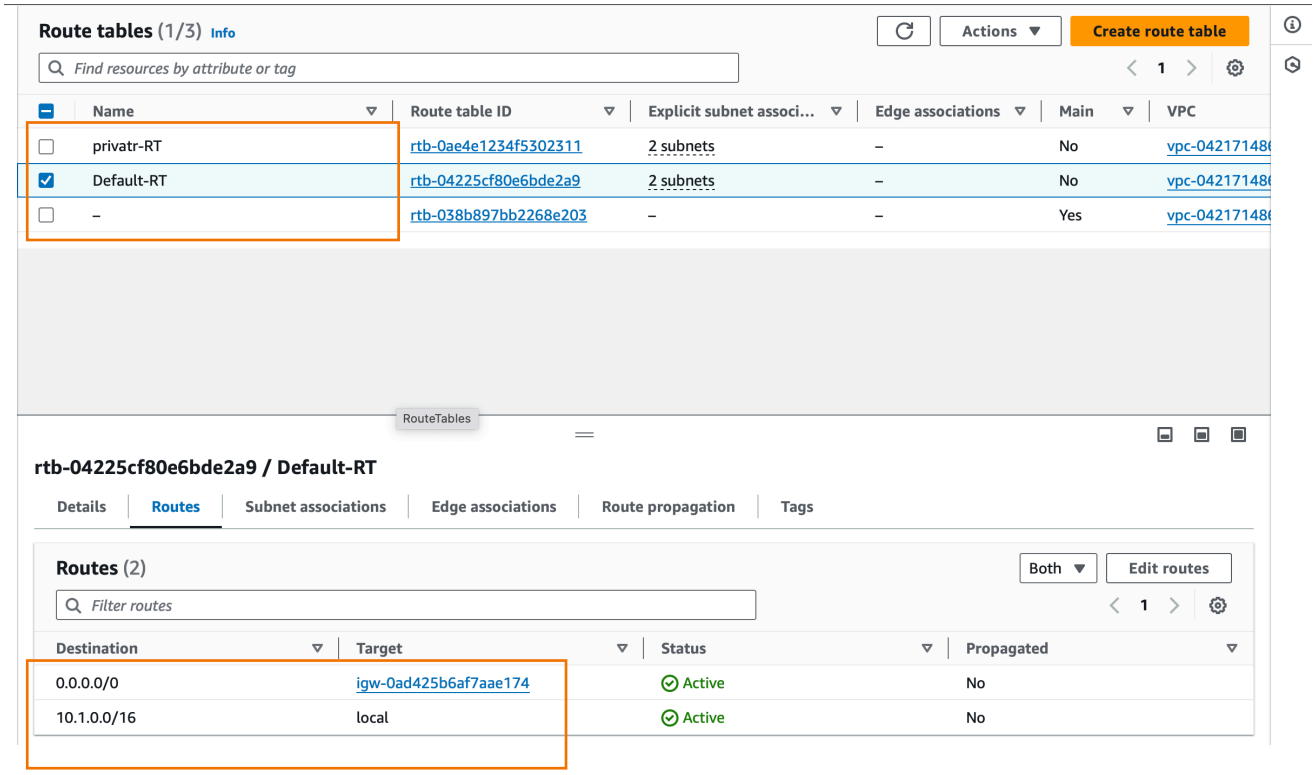
4.routes.tf > resource "aws_route_table" "RT" > tags

```
###### public RT
resource "aws_route_table" "RT" {
  vpc_id = aws_vpc.vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.IGW.id
  }
  tags = {
    Name = "Default-RT"
  }
}

resource "aws_route_table_association" "rt-1" {
  subnet_id      = aws_subnet.public-subnet-1.id
  route_table_id = aws_route_table.RT.id
}
resource "aws_route_table_association" "rt-2" {
  subnet_id      = aws_subnet.public-subnet-2.id
  route_table_id = aws_route_table.RT.id
}
```

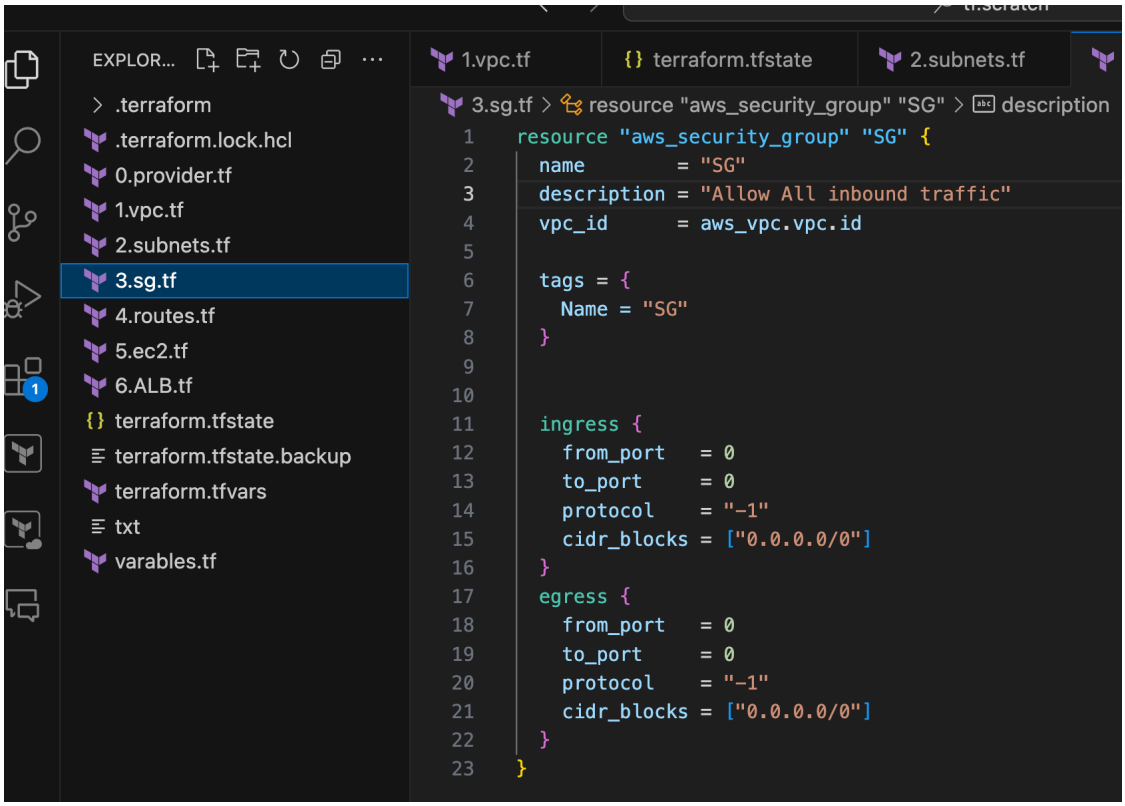
```
############ privatr RT
resource "aws_route_table" "pr-rt" {
  vpc_id = aws_vpc.vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_nat_gateway.nat.id
  }

  tags = {
    Name = "privatr-RT"
  }
}

resource "aws_route_table_association" "pr-rt-1" {
  subnet_id      = aws_subnet.private-subnet-1.id
  route_table_id = aws_route_table.pr-rt.id
}
resource "aws_route_table_association" "pr-rt-2" {
  subnet_id      = aws_subnet.private-subnet-2.id
  route_table_id = aws_route_table.pr-rt.id
}
```

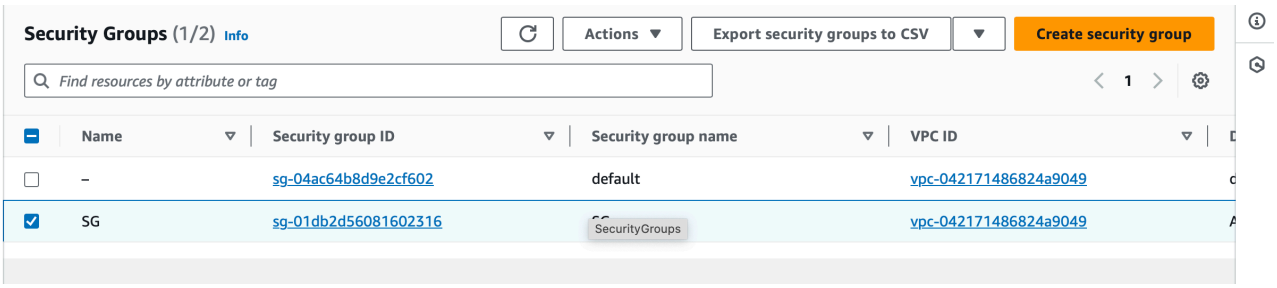
10. Deploy routing tables and routing table associations.





11. Creating security group by giving inbound and outbound rules as anywhere.





12. Lets create internet gateway and associate it in public routing table.

```
######### internet gateway
resource "aws_internet_gateway" "IGW" {
  vpc_id = aws_vpc.vpc.id

  tags = {
    Name = "IGW"
  }
}
```

13. Creating NAT gateway, before that elastic ip should be created and associate with nat gateway and add nat gateway in private routing table.

```
############ EIP
resource "aws_eip" "nat" {
  depends_on = [aws_internet_gateway.IGW]

  tags = {
    Name = "EIP"
  }
}
```

```
############ NAT gateway

resource "aws_nat_gateway" "nat" {
  allocation_id = aws_eip.nat.id
  subnet_id     = aws_subnet.public-subnet-1.id

  tags = {
    Name = "NAT"
  }
}
```

14. This concludes that when private servers need to connect with internet it goes through nat gateway to internet.
15. Nat gateway converts private ip address into public ip.
16. Let's create ec2 instance, in this we deploy 3 ec2 instance for that I have duplicated the code 3 times, but it is not a best practice.
17. Instead of duplicating the code we can use terraform count function.

- Count = 3

18. By giving count function the code runs number of times given in the count value.
19. But I have duplicated the code three times, due to some reasons.
20. The code for ec2 instance as follows.

Public server-1 with ip of 10.1.1.1100.

```
5.ec2.tf > resource "aws_instance" "ec2-1"
resource "aws_instance" "ec2-1" {
  ami                         = var.imagename
  instance_type               = var.instance_type
  key_name                    = var.key_name
  private_ip                  = "10.1.1.100"
  subnet_id                   = aws_subnet.public-subnet-1.id
  vpc_security_group_ids      = [aws_security_group.SG.id]
  associate_public_ip_address = true

  tags = {
    Name = "web-server"
  }
  user_data = <<-EOF
  #!/bin/bash
  sudo su -
  amazon-linux-extras install nginx1.12 -y
  systemctl enable nginx --now
  echo "<div><h1>$(cat /etc/hostname)</h1></div>" >> /usr/share/nginx/html/index.html
EOF
}
```

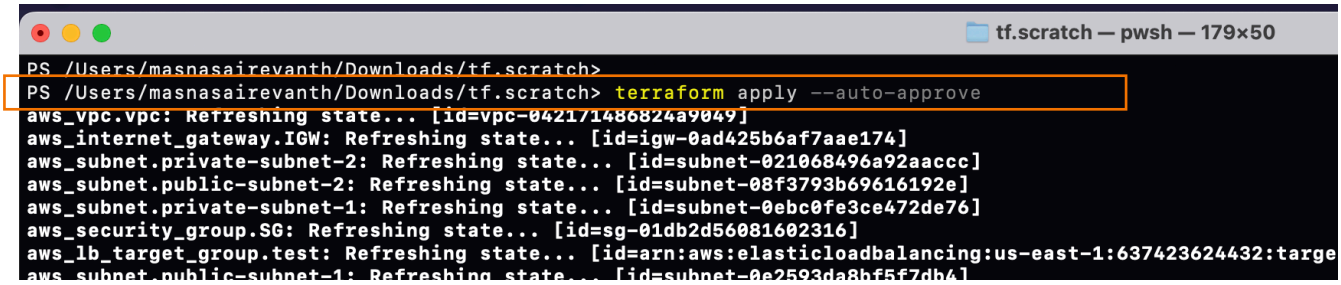
Private server-1 ip of 10.1.10.100.

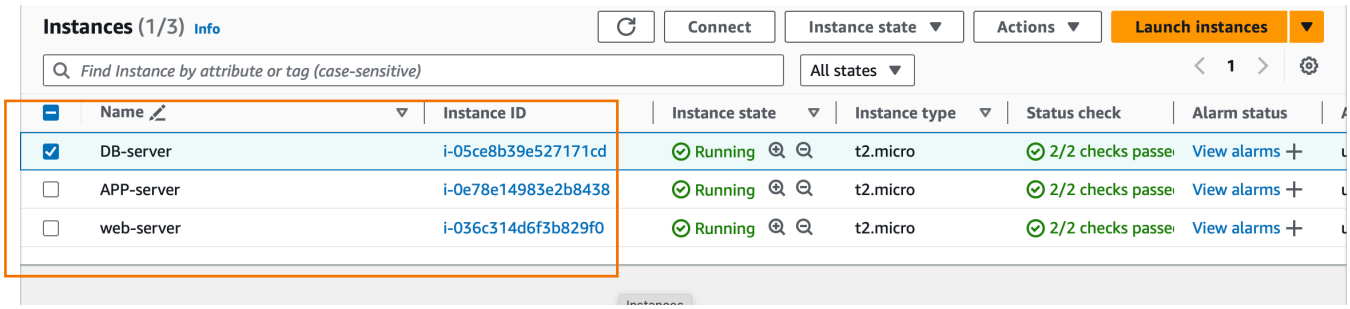
```

  ######## ec2
  resource "aws_instance" "ec2-2" {
    ami                         = var.imagename
    instance_type               = var.instance_type
    key_name                    = var.key_name
    private_ip                  = "10.1.10.100"
    subnet_id                   = aws_subnet.private-subnet-1.id
    vpc_security_group_ids      = [aws_security_group.SG.id]
    associate_public_ip_address = true

    tags = {
      Name = "APP-server"
    }
    user_data = <<-EOF
    #!/bin/bash
    sudo su -
    amazon-linux-extras install nginx1.12 -y
    systemctl enable nginx --now
    echo "<div><h1>$(cat /etc/hostname)</h1></div>" >> /usr/share/nginx/html/index.html
  EOF
  }

```

Private server-2 ip 10.1.20.100.

```
44
45  ######### ec2
46  resource "aws_instance" "ec2-3" {
47    ami                         = var.imagename
48    instance_type               = var.instance_type
49    key_name                    = var.key_name
50    private_ip                  = "10.1.20.100"
51    subnet_id                   = aws_subnet.private-subnet-2.id
52    vpc_security_group_ids      = [aws_security_group.SG.id]
53    associate_public_ip_address = true
54
55    tags = {
56      Name = "DB-server"
57    }
58    user_data = <<-EOF
59    #!/bin/bash
60    sudo su -
61    amazon-linux-extras install nginx1.12 -y
62    systemctl enable nginx --now
63    echo "<div><h1>$(cat /etc/hostname)</h1></div>" >> /usr/share/nginx/html/index.html
64  EOF
65  }
66
```

21. Deploy the terraform code.





22. Now create a APPLICATION LOAD BALANCER.
23. First we need target group in which we add our private instances as our targets.

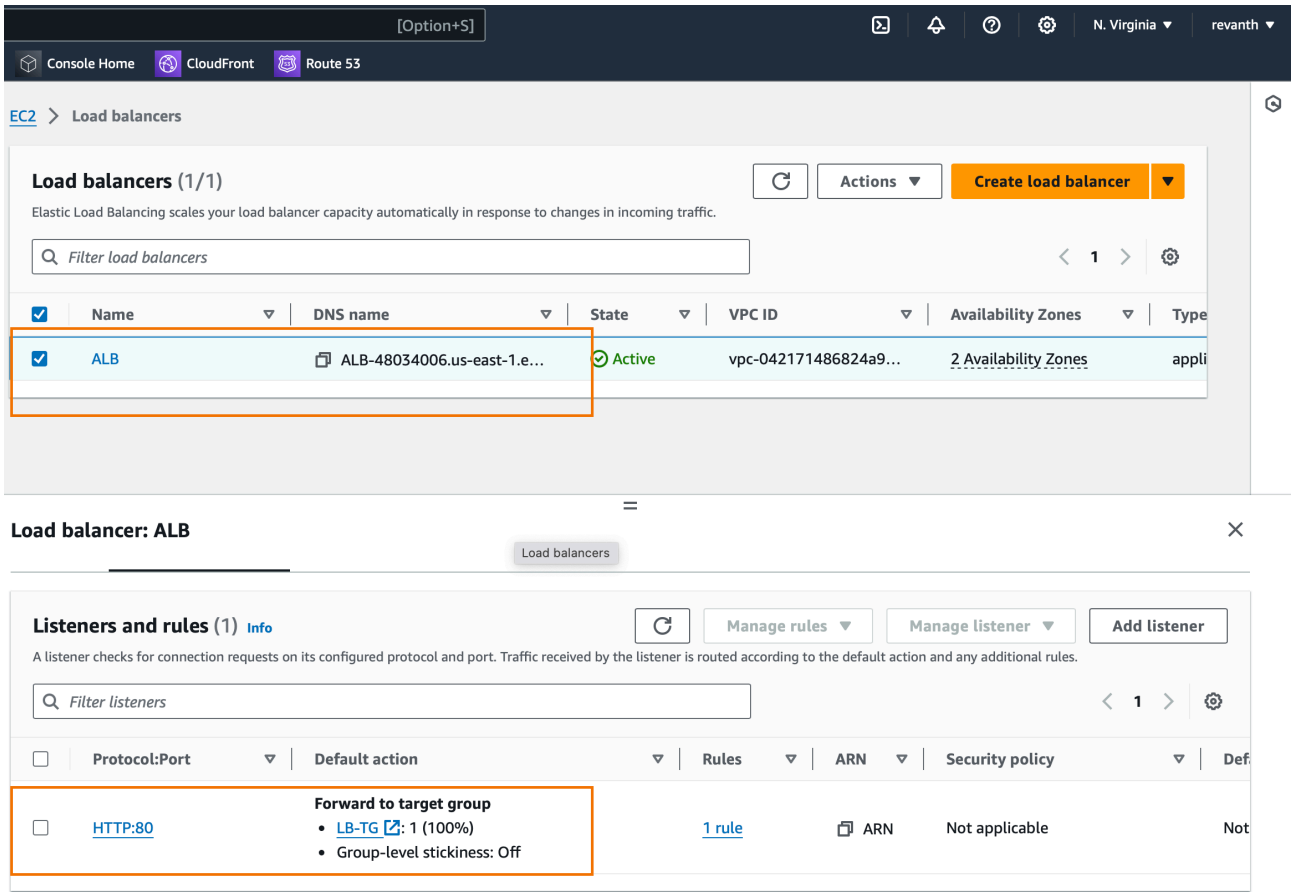
24. Target group attachment for adding private instance as a target.
25. Now create a load balancer in this we use application type.

26. Application load balancer we deploy in public subnet as internet facing.

```
###### ALB
resource "aws_lb" "example" {
  name               = "ALB"
  load_balancer_type = "application"
   security_groups    = [aws_security_group.SG.id]

  subnet_mapping {
    subnet_id = aws_subnet.public-subnet-1.id
    #allocation_id = aws_eip.example1.id
  }

  subnet_mapping {
    subnet_id = aws_subnet.public-subnet-2.id
    #allocation_id = aws_eip.example2.id
  }
}
```

27. Now add listeners in load balancer.

```
# ######## Listener
resource "aws_lb_listener" "alb_listener" {
  load_balancer_arn = aws_lb.example.arn
  port              = "80"
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.test.arn
  }
}
```

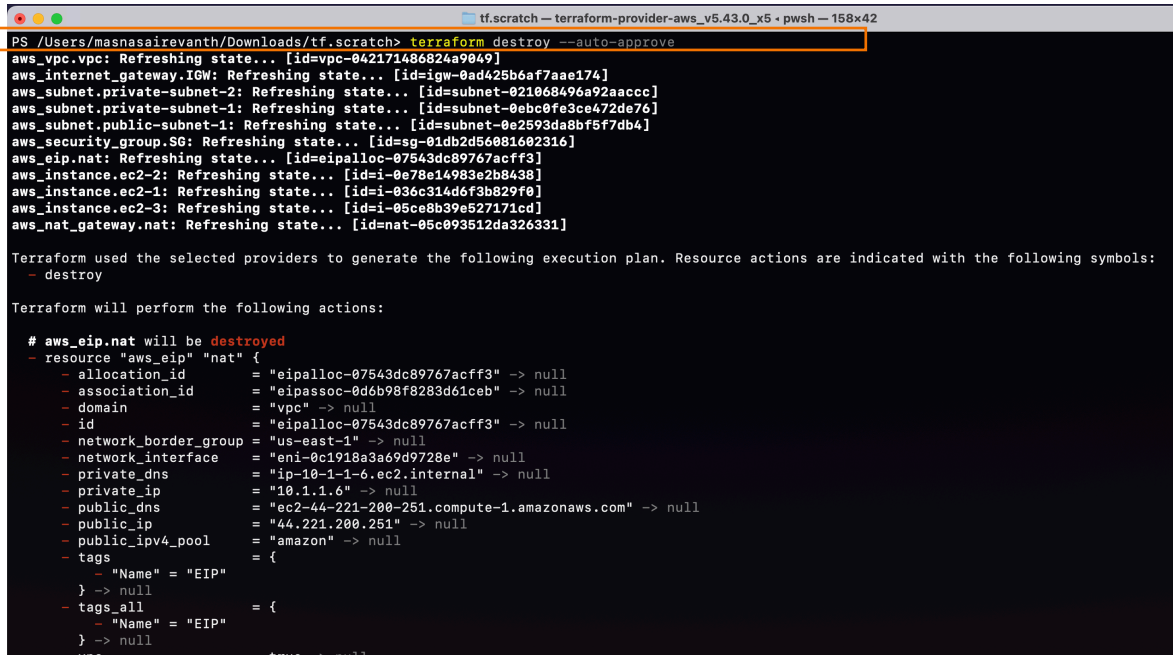


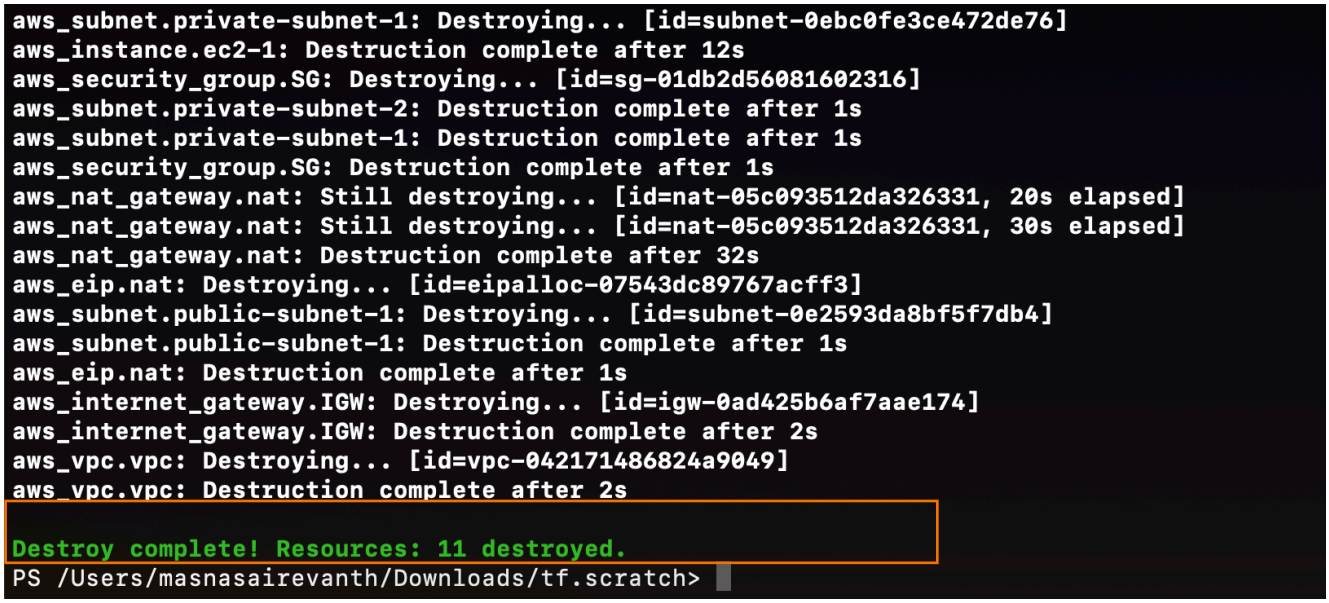
28. Whenever the traffic receives in lister it will forward it to load balancer.
29. Our configuration is set perfectly, let's test it.
30. Copy the load balancer dns name and past it in browser.
31. You will redirect to ip of 10.100 or 20.100 which is our private servers.

32. Lets run a small shell script to test load balancing.

```
^C
[root@ip-10-1-1-100 ~]# while true
> do
> curl -sL http://ALB-48034006.us-east-1.elb.amazonaws.com/ | grep -i 'ip-10-1'
> sleep 1
> done
<div><h1>ip-10-1-10-100.ec2.internal</h1></div>
<div><h1>ip-10-1-10-100.ec2.internal</h1></div>
<div><h1>ip-10-1-20-100.ec2.internal</h1></div>
<div><h1>ip-10-1-20-100.ec2.internal</h1></div>
<div><h1>ip-10-1-20-100.ec2.internal</h1></div>
<div><h1>ip-10-1-20-100.ec2.internal</h1></div>
<div><h1>ip-10-1-10-100.ec2.internal</h1></div>
<div><h1>ip-10-1-10-100.ec2.internal</h1></div>
<div><h1>ip-10-1-20-100.ec2.internal</h1></div>
^C
[root@ip-10-1-1-100 ~]#
```

33. This script will ping the load balancer ip every time.
34. After done with load balancer destroy the code, so every thing created using terraform code will be destroyed.





Conclusion:

- We can conclude that the load is balancing between 2 ip ranges which is 10.100 and 20.100, which is our APP-server and DB-server.
- This type of traffic is called as “application” traffic, user is connecting through the internet.
- When private servers need to get patches it will reach internet through NAT gateway, this traffic known as “management” traffic.

References:

- Guided by my mentor.
- Online articles "terraform.io".
- youtube videos.
- Other online articles which are related to terraform code.