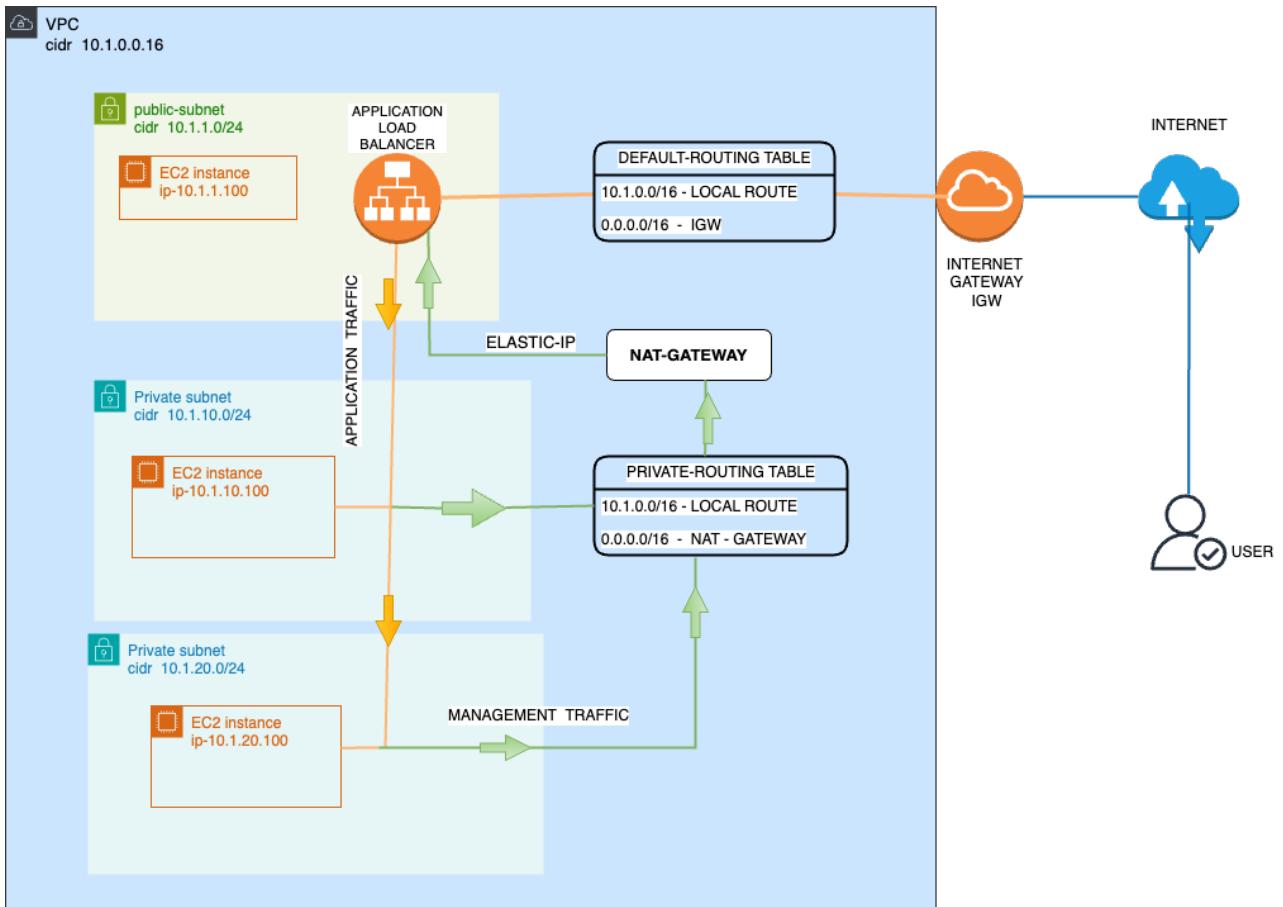


## DEPLOYING A 3-TIRE WEB APPLICATION USING TERRAFORM CODE

### 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.

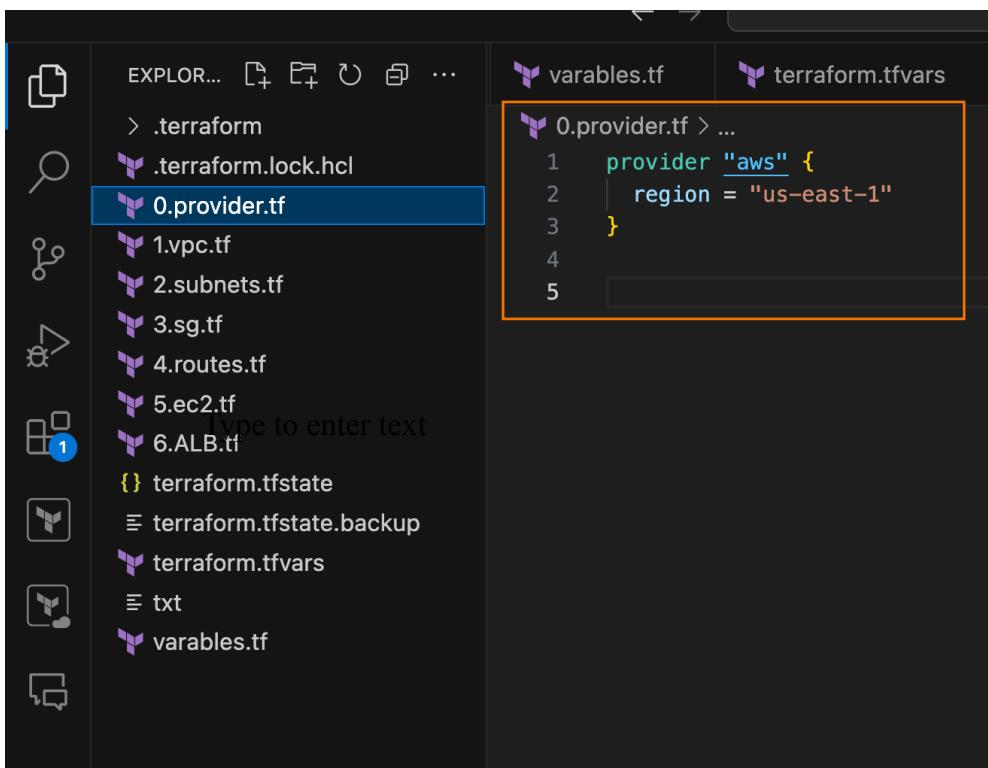
## 5. Terraform apply:

“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.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, it shows a tree view of the project structure. The file `0.provider.tf` is selected and highlighted in blue.
- Editor:** On the right, the content of the `0.provider.tf` file is displayed:

```
provider "aws" {  
  region = "us-east-1"  
}
```
- Terminal:** At the bottom, there is a terminal window with the following text:

```
$env:AWS_ACCESS_KEY_ID=""  
$env:AWS_SECRET_ACCESS_KEY="/MQcoslJ+iaV  
| Type to enter text
```

The first two lines are redacted with orange boxes.

```
2  
3  $env:AWS_ACCESS_KEY_ID=""  
4  $env:AWS_SECRET_ACCESS_KEY="/MQcoslJ+iaV  
5  | Type to enter text
```

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.

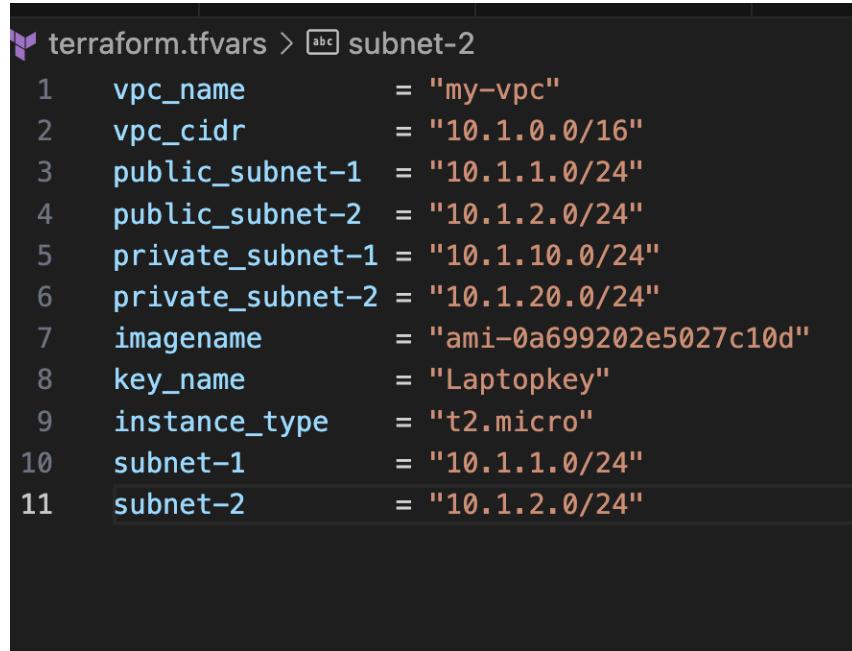
- Terraform init
- Terraform validate
- Terraform plan
- Terraform apply -auto -approve

Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR
my-vpc	vpc-042171486824a9049	Available	10.1.0.0/16	-

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

```
variables.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.



```

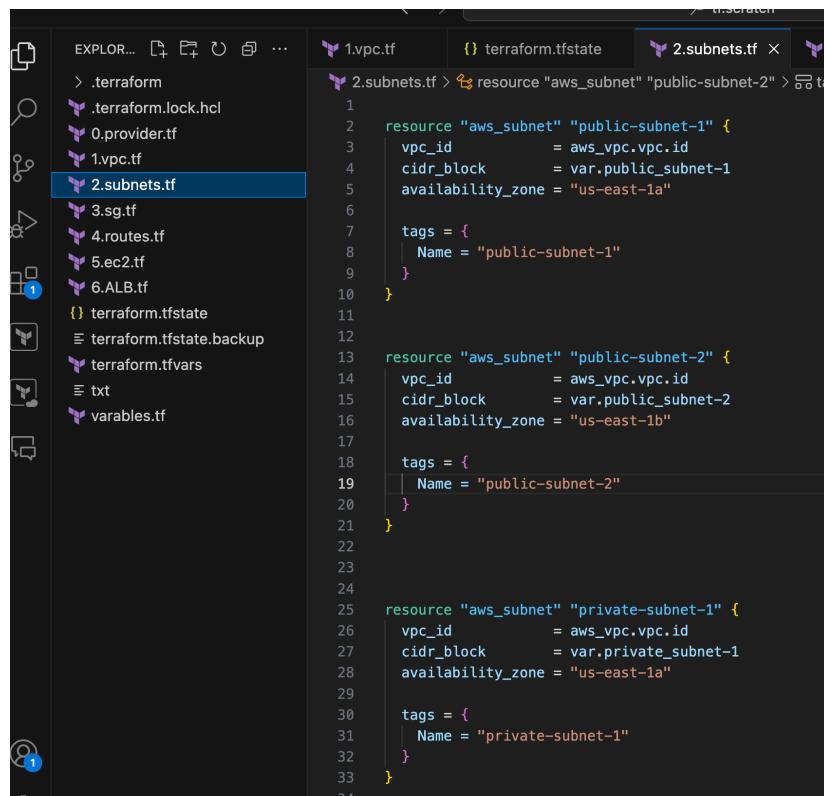
terraform.tfvars > [abc] subnet-2
1   vpc_name          = "my-vpc"
2   vpc_cidr           = "10.1.0.0/16"
3   public_subnet-1    = "10.1.1.0/24"
4   public_subnet-2    = "10.1.2.0/24"
5   private_subnet-1   = "10.1.10.0/24"
6   private_subnet-2   = "10.1.20.0/24"
7   imagename          = "ami-0a699202e5027c10d"
8   key_name            = "Laptopkey"
9   instance_type       = "t2.micro"
10  subnet-1            = "10.1.1.0/24"
11  subnet-2            = "10.1.2.0/24"

```

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



```

2.subnets.tf > 2.subnets.tf
1
2 resource "aws_subnet" "public-subnet-1" {
3   vpc_id      = aws_vpc.vpc.id
4   cidr_block  = var.public_subnet-1
5   availability_zone = "us-east-1a"
6
7   tags = {
8     Name = "public-subnet-1"
9   }
10
11
12 resource "aws_subnet" "public-subnet-2" {
13   vpc_id      = aws_vpc.vpc.id
14   cidr_block  = var.public_subnet-2
15   availability_zone = "us-east-1b"
16
17   tags = {
18     Name = "public-subnet-2"
19   }
20
21
22
23
24 resource "aws_subnet" "private-subnet-1" {
25   vpc_id      = aws_vpc.vpc.id
26   cidr_block  = var.private_subnet-1
27   availability_zone = "us-east-1a"
28
29   tags = {
30     Name = "private-subnet-1"
31   }
32
33
34

```

## 8. Deploy subnets.

The screenshot shows the AWS VPC Subnets page. At the top, there is a success message: "You have successfully created 1 subnet: subnet-05da979af9b38cccd7". Below this, a table lists four subnets:

Name	Subnet ID	State	VPC	IPv4 CIDR
private-subnet-1	subnet-09a93bd695dde45d9	Available	vpc-0dbf6ffd698336a27   3-tire...	10.1.10.0/24
private-subnet-2	subnet-088a049441d921ca8	Available	vpc-0dbf6ffd698336a27   3-tire...	10.1.20.0/24
public-subnet-1	subnet-03490a227c53aa2b1	Available	vpc-0dbf6ffd698336a27   3-tire...	10.1.1.0/24
public-subnet-2	subnet-05da979af9b38cccd7	Available	vpc-0dbf6ffd698336a27   3-tire...	10.1.2.0/24

## 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
1 ##### public RT
2 resource "aws_route_table" "RT" {
3   vpc_id = aws_vpc.vpc.id
4
5   route {
6     cidr_block = "0.0.0.0/0"
7     gateway_id = aws_internet_gateway.IGW.id
8   }
9   tags = {
10    Name = "Default-RT"
11  }
12}

13
14 resource "aws_route_table_association" "rt-1" {
15   subnet_id      = aws_subnet.public-subnet-1.id
16   route_table_id = aws_route_table.RT.id
17 }
18 resource "aws_route_table_association" "rt-2" {
19   subnet_id      = aws_subnet.public-subnet-2.id
20   route_table_id = aws_route_table.RT.id
21 }

#####
22
23 ##### privatr RT
24 resource "aws_route_table" "pr-rt" {
25   vpc_id = aws_vpc.vpc.id
26
27   route {
28     cidr_block = "0.0.0.0/0"
29     gateway_id = aws_nat_gateway.nat.id
30   }
31
32   tags = {
33     Name = "privatr-RT"
34   }
35 }

36
37 resource "aws_route_table_association" "pr-rt-1" {
38   subnet_id      = aws_subnet.private-subnet-1.id
39   route_table_id = aws_route_table.pr-rt.id
40 }
41 resource "aws_route_table_association" "pr-rt-2" {
42   subnet_id      = aws_subnet.private-subnet-2.id
43   route_table_id = aws_route_table.pr-rt.id
44 }

```

## 10. Deploy routing tables and routing table associations.

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC
<input checked="" type="checkbox"/> privatr-RT	<a href="#">rtb-0ae4e1234f5302311</a>	2 subnets	-	No	<a href="#">vpc-04217148</a>
<input type="checkbox"/> Default-RT	<a href="#">rtb-04225cf80e6bde2a9</a>	2 subnets	-	No	<a href="#">vpc-04217148</a>
<input type="checkbox"/> -	<a href="#">rtb-038b897bb2268e203</a>	-	-	Yes	<a href="#">vpc-04217148</a>

**rtb-0ae4e1234f5302311 / privatr-RT**

Details | **Routes** | Subnet associations | Edge associations | Route propagation | Tags

**Routes (2)**

Filter routes

Destination	Target	Status	Propagated
0.0.0.0/0	<a href="#">nat-05c093512da326331</a>	Active	No
10.1.0.0/16	local	Active	No

**Route tables (1/3) Info**

Find resources by attribute or tag

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC
<input type="checkbox"/> privatr-RT	<a href="#">rtb-0ae4e1234f5302311</a>	2 subnets	-	No	<a href="#">vpc-04217148</a>
<input checked="" type="checkbox"/> Default-RT	<a href="#">rtb-04225cf80e6bde2a9</a>	2 subnets	-	No	<a href="#">vpc-04217148</a>
<input type="checkbox"/> -	<a href="#">rtb-038b897bb2268e203</a>	-	-	Yes	<a href="#">vpc-04217148</a>

**rtb-04225cf80e6bde2a9 / Default-RT**

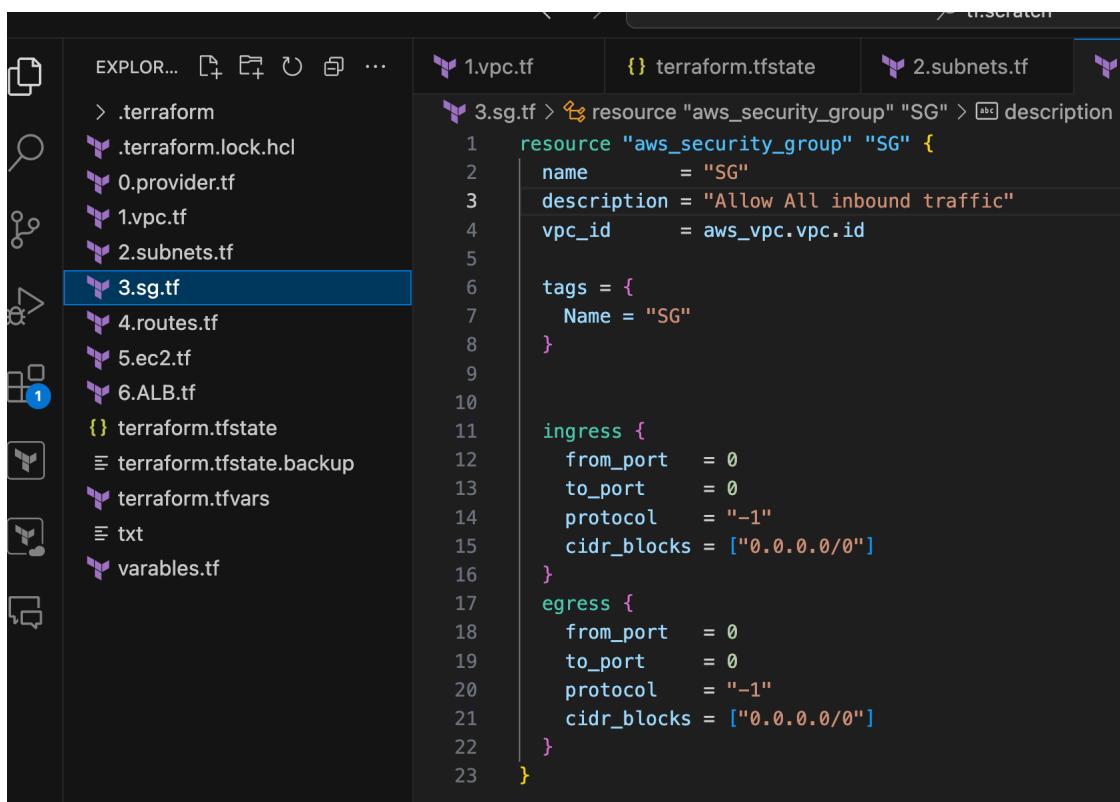
Details | **Routes** | Subnet associations | Edge associations | Route propagation | Tags

**Routes (2)**

Filter routes

Destination	Target	Status	Propagated
0.0.0.0/0	<a href="#">igw-0ad425b6af7aae174</a>	Active	No
10.1.0.0/16	local	Active	No

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



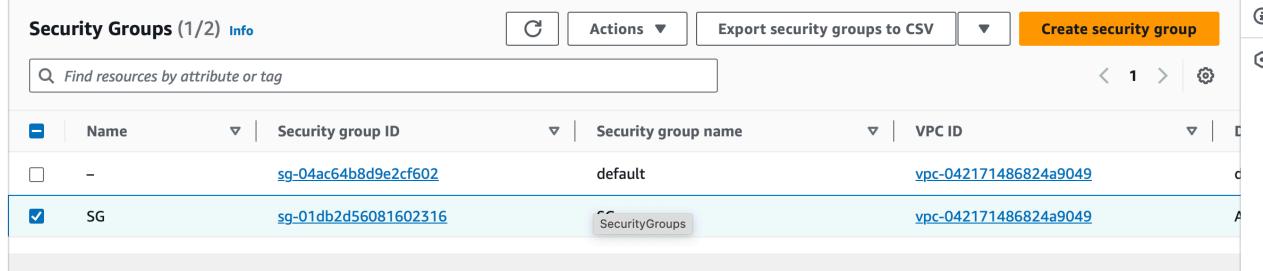
The screenshot shows a code editor with a sidebar containing file navigation. The main pane displays a Terraform configuration file named 3.sg.tf. The code defines an AWS Security Group (SG) named "SG" with the following properties:

```
resource "aws_security_group" "SG" {
  name            = "SG"
  description     = "Allow All inbound traffic"
  vpc_id          = aws_vpc.vpc.id

  tags = {
    Name = "SG"
  }

  ingress {
    from_port      = 0
    to_port        = 0
    protocol       = "-1"
    cidr_blocks   = ["0.0.0.0/0"]
  }

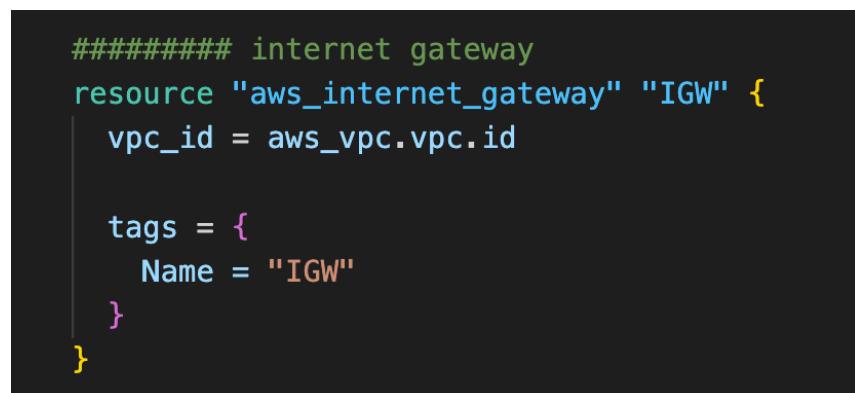
  egress {
    from_port      = 0
    to_port        = 0
    protocol       = "-1"
    cidr_blocks   = ["0.0.0.0/0"]
  }
}
```



The screenshot shows the AWS Management Console interface for Security Groups. It displays a list of two security groups:

Name	Security group ID	Security group name	VPC ID
-	sg-04ac64b8d9e2cf602	default	vpc-042171486824a9049
SG	sg-01db2d56081602316	SG	vpc-042171486824a9049

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



The screenshot shows a code editor with a dark theme displaying Terraform configuration for an Internet Gateway (IGW). The code defines an AWS Internet Gateway (IGW) with the following properties:

```
##### 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"
1   resource "aws_instance" "ec2-1" {
2     ami                  = var.imagename
3     instance_type        = var.instance_type
4     key_name             = var.key_name
5     private_ip           = "10.1.1.100"
6     subnet_id            = aws_subnet.public-subnet-1.id
7     vpc_security_group_ids = [aws_security_group.SG.id]
8     associate_public_ip_address = true
9
10    tags = {
11      Name = "web-server"
12    }
13    user_data = <<-EOF
14    #!/bin/bash
15    sudo su -
16    amazon-linux-extras install nginx1.12 -y
17    systemctl enable nginx --now
18    echo "<div><h1>$(> cat /etc/hostname)</h1></div>" >> /usr/share/nginx/html/index.html
19  EOF
20 }
```

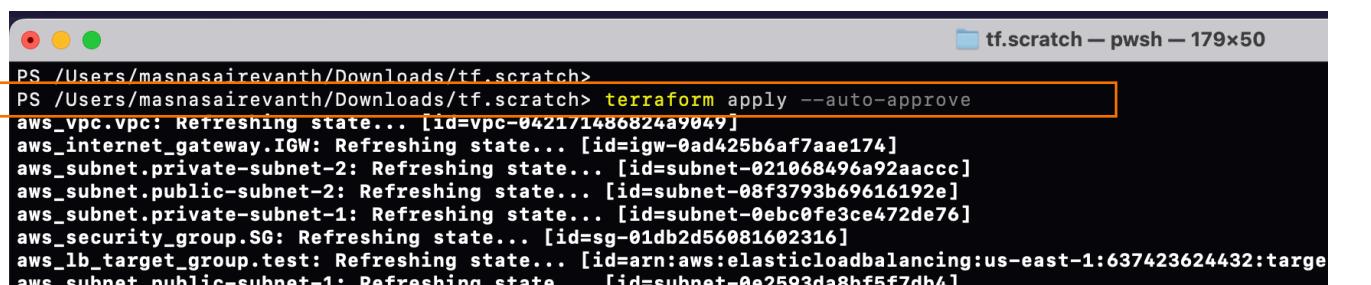
Private server-1 ip of 10.1.10.100.

```
21 ###### ec2
22
23 resource "aws_instance" "ec2-2" {
24   ami                  = var.imagename
25   instance_type        = var.instance_type
26   key_name             = var.key_name
27   private_ip           = "10.1.10.100"
28   subnet_id            = aws_subnet.private-subnet-1.id
29   vpc_security_group_ids = [aws_security_group.SG.id]
30   associate_public_ip_address = true
31
32   tags = {
33     Name = "APP-server"
34   }
35   user_data = <<-EOF
36   #!/bin/bash
37   sudo su -
38   amazon-linux-extras install nginx1.12 -y
39   systemctl enable nginx --now
40   echo "<div><h1>$(> cat /etc/hostname)</h1></div>" >> /usr/share/nginx/html/index.html
41 EOF
42 }
43
```

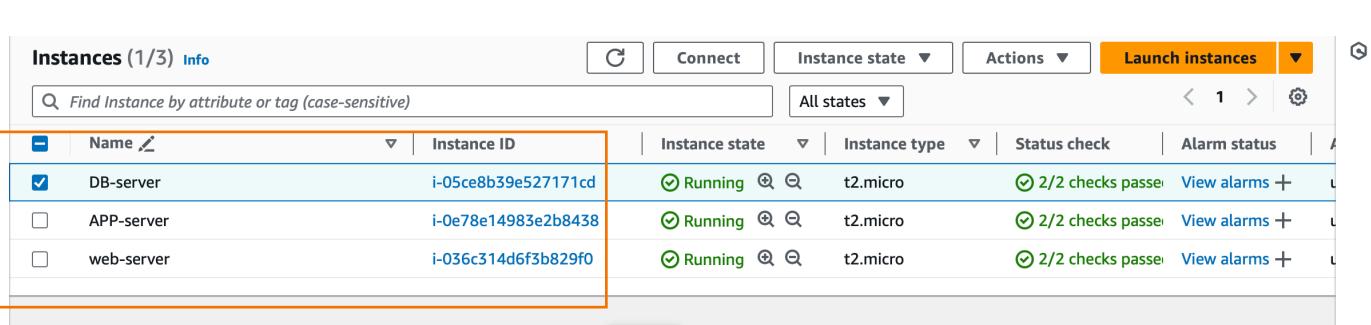
Private server-2 ip 10.1.20.100.

```
45
44
45 ##### ec2
46 resource "aws_instance" "ec2-3" {
47   ami           = var.image_name
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.



```
PS /Users/masnasairevanth/Downloads/tf.scratch>
PS /Users/masnasairevanth/Downloads/tf.scratch> terraform apply --auto-approve
aws_vpc.vpc: Refreshing state... [id=vpc-042171486824a9049]
aws_internet_gateway.IGW: Refreshing state... [id=igw-0ad425b6af7aae174]
aws_subnet.private-subnet-2: Refreshing state... [id=subnet-021068496a92aacc]
aws_subnet.public-subnet-2: Refreshing state... [id=subnet-08f3793b69616192e]
aws_subnet.private-subnet-1: Refreshing state... [id=subnet-0ebc0fe3ce472de76]
aws_security_group.SG: Refreshing state... [id=sg-01db2d56081602316]
aws_lb_target_group.test: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:637423624432:targetgroup/test/123456789012]
aws_subnet.public-subnet-1: Refreshing state... [id=subnet-0e2593da8bf5f7db41]
```



Name	Instance ID	Instance state	Instance type	Status check	Alarm status
DB-server	i-05ce8b39e527171cd	Running	t2.micro	2/2 checks passed	<a href="#">View alarms</a>
APP-server	i-0e78e14983e2b8438	Running	t2.micro	2/2 checks passed	<a href="#">View alarms</a>
web-server	i-036c314d6f3b829f0	Running	t2.micro	2/2 checks passed	<a href="#">View alarms</a>

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

```

6.ALB.tf > 6.ALB.tf
1 ##### instance Target-group
2 resource "aws_lb_target_group" "test" {
3   name      = "LB-TG"
4   target_type = "instance"
5   port      = 80
6   protocol    = "HTTP"
7   vpc_id      = aws_vpc.vpc.id
8   tags = {
9     Name = "TG"
10  }
11 }
12
13
14 ##### Attachment
15 resource "aws_lb_target_group_attachment" "ec2_attach-1" {
16   count      = 1
17   target_group_arn = aws_lb_target_group.test.arn
18   target_id    = aws_instance.ec2-2.id
19   port        = 80
20 }
21
22
23 ##### Attachment
24 resource "aws_lb_target_group_attachment" "ec2_attach-2" {
25   count      = 1
26   target_group_arn = aws_lb_target_group.test.arn
27   target_id    = aws_instance.ec2-3.id
28   port        = 80
29 }
30

```

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.

```
46 ##### ALB
47 resource "aws_lb" "example" {
48   name          = "ALB"
49   load_balancer_type = "application"
50   security_groups    = [aws_security_group.SG.id]
51
52   subnet_mapping {
53     subnet_id = aws_subnet.public-subnet-1.id
54     #allocation_id = aws_eip.example1.id
55   }
56
57   subnet_mapping {
58     subnet_id = aws_subnet.public-subnet-2.id
59     #allocation_id = aws_eip.example2.id
60   }
61 }
```

27. Now add listeners in load balancer.

```
52
53 # ###### Listener
54 resource "aws_lb_listener" "alb_listener" {
55   load_balancer_arn = aws_lb.example.arn
56   port            = "80"
57   protocol        = "HTTP"
58
59   default_action {
60     type          = "forward"
61     target_group_arn = aws_lb_target_group.test.arn
62   }
63 }
```

**Create New Distribution**

**Step 2: Select Origin**

**Origin**: Amazon S3

**Origin Path**: /

**HTTP Version**: HTTP 1.1

**TLS Policy**: CloudFront-managed TLS

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.

Welcome to nginx on Amazon Linux!

Website Administrator

This is the default index.html page that is distributed with nginx on Amazon Linux. It is located in /usr/share/nginx/html.  
You should now put your content in a location of your choice and edit the root configuration directive in the nginx configuration file /etc/nginx/nginx.conf.

Powered by nginx.

ip-10-1-20-100.ec2.internal

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.

```
PS /Users/masnasairevanth/Downloads/tf.scratch> terraform destroy --auto-approve
aws_vpc.vpc: Refreshing state... [id=vpc-0421714868249849]
aws_internet_gateway.IGW: Refreshing state... [id=igw-0ad425b6af7aae174]
aws_subnet.private-subnet-2: Refreshing state... [id=subnet-021068496a92aaccc]
aws_subnet.private-subnet-1: Refreshing state... [id=subnet-0ebc0fe3ce472de76]
aws_subnet.public-subnet-1: Refreshing state... [id=subnet-0e2593da8bf5f7db4]
aws_security_group.SG: Refreshing state... [id=sg-01db2d5681692316]
aws_eip.nat: Refreshing state... [id=eipalloc-07543dc89767acff3]
aws_instance.ec2-2: Refreshing state... [id=i-0e78e14983e2b8438]
aws_instance.ec2-1: Refreshing state... [id=i-036c314d6f3b829fe]
aws_instance.ec2-3: Refreshing state... [id=i-05ce8b39e527171cd]
aws_nat_gateway.nat: Refreshing state... [id=nat-05c093812da326331]

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

Terraform will perform the following actions:

# aws_eip.nat will be destroyed
- resource "aws_eip" "nat" {
  - allocation_id      = "eipalloc-07543dc89767acff3" -> null
  - association_id    = "eipassoc-0d6b98f8283d6iceb" -> null
  - domain            = "vpc" -> null
  - id                = "eipalloc-07543dc89767acff3" -> null
  - network_border_group = "us-east-1" -> null
  - network_interface = "eni-0c1918a3a69d9728e" -> null
  - private_dns       = "ip-10-1-1-6.ec2.internal" -> null
  - private_ip        = "10.1.1.6" -> null
  - public_dns         = "ec2-44-221-200-251.compute-1.amazonaws.com" -> null
  - public_ip          = "44.221.200.251" -> null
  - public_ipv4_pool   = "amazon" -> null
  - tags              = {
      - "Name" = "EIP"
    } -> null
  - tags_all           = {
      - "Name" = "EIP"
    } -> null
}
```

```
aws_subnet.private-subnet-1: Destroying... [id=subnet-0ebc0fe3ce472de76]
aws_instance.ec2-1: Destruction complete after 12s
aws_security_group.SG: Destroying... [id=sg-01db2d56081602316]
aws_subnet.private-subnet-2: Destruction complete after 1s
aws_subnet.private-subnet-1: Destruction complete after 1s
aws_security_group.SG: Destruction complete after 1s
aws_nat_gateway.nat: Still destroying... [id=nat-05c093512da326331, 20s elapsed]
aws_nat_gateway.nat: Still destroying... [id=nat-05c093512da326331, 30s elapsed]
aws_nat_gateway.nat: Destruction complete after 32s
aws_eip.nat: Destroying... [id=eipalloc-07543dc89767acff3]
aws_subnet.public-subnet-1: Destroying... [id=subnet-0e2593da8bf5f7db4]
aws_subnet.public-subnet-1: Destruction complete after 1s
aws_eip.nat: Destruction complete after 1s
aws_internet_gateway.IGW: Destroying... [id=igw-0ad425b6af7aae174]
aws_internet_gateway.IGW: Destruction complete after 2s
aws_vpc.vpc: Destroying... [id=vpc-042171486824a9049]
aws_vpc.vpc: Destruction complete after 2s
```

```
Destroy complete! Resources: 11 destroyed.
```

```
PS /Users/masnasairevanth/Downloads/tf.scratch>
```

## 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.