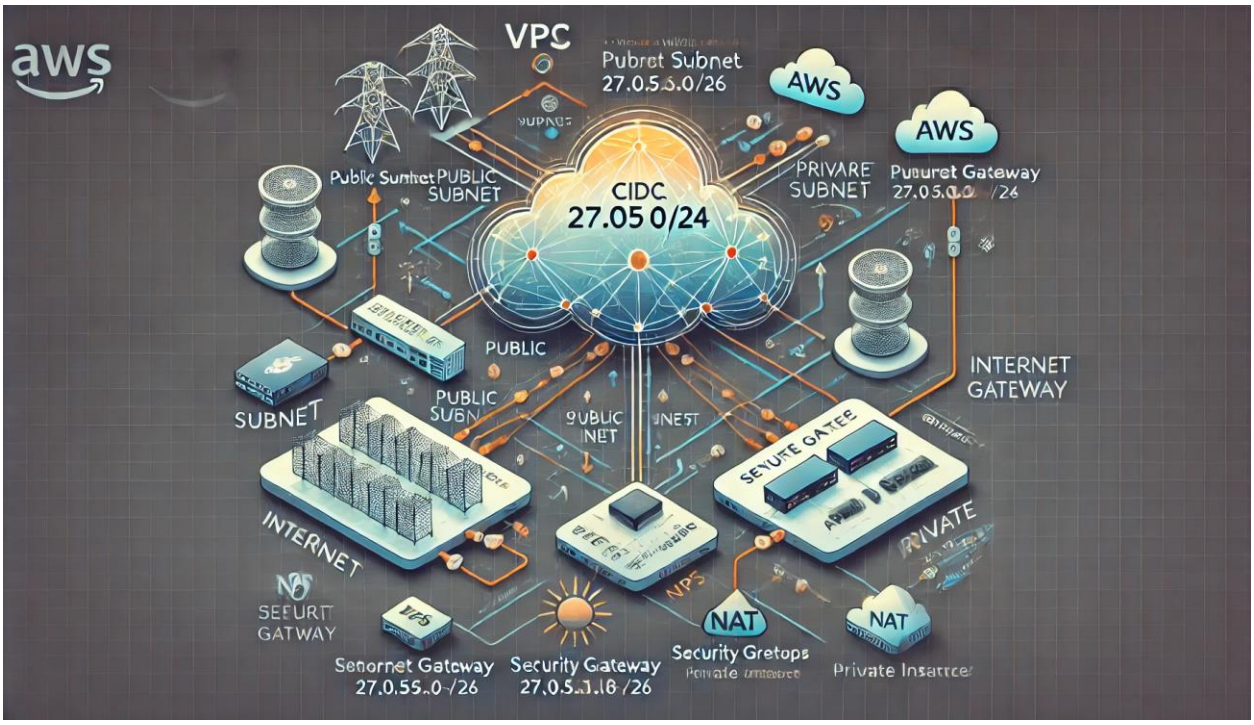
cprime

Automated AWS Infrastructure Deployment with Terraform: Secure Public and Private Subnets with Reverse Proxy



NAME : T MANOJ

EMPID : LYAKE2KHS

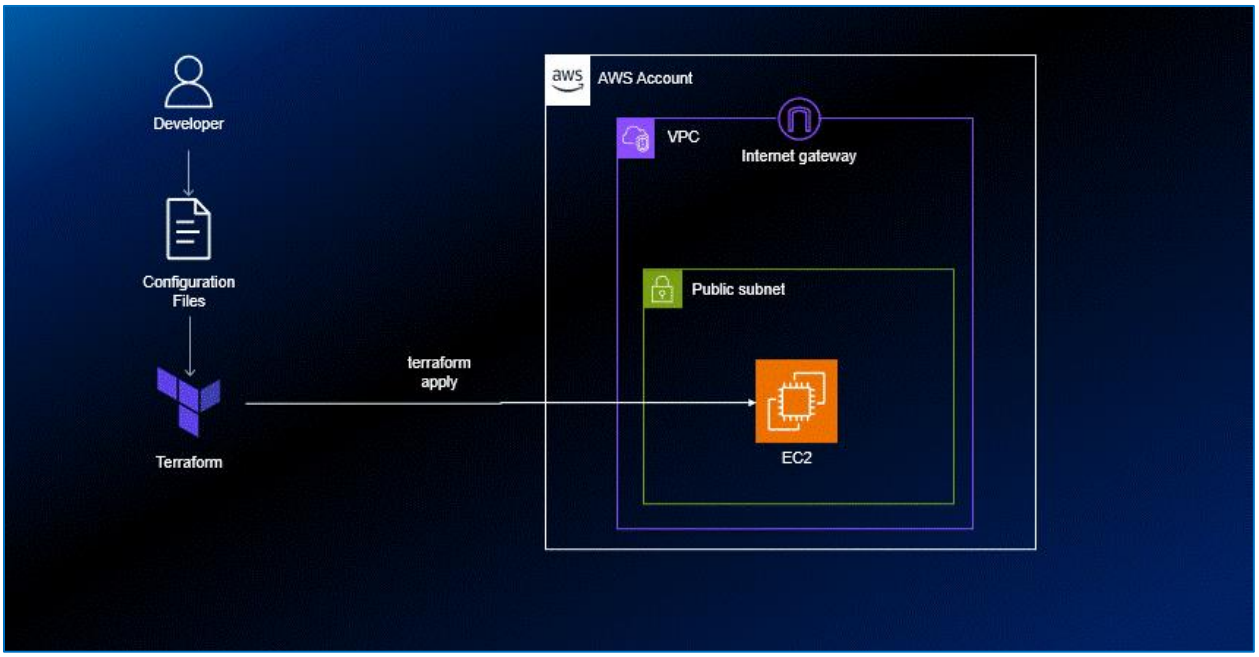
Objective

The primary objective of this Terraform configuration is to automate the deployment of a secure, scalable, and highly available AWS infrastructure. It aims to create a Virtual Private Cloud (VPC) with both public and private subnets, enabling the isolation of resources based on their accessibility requirements. The configuration sets up an Internet Gateway to allow public resources to communicate with the internet and a NAT Gateway to provide private resources with secure internet access. Additionally, it deploys EC2 instances in both subnets: a public instance that acts as a reverse proxy to securely route traffic to a private instance hosting a web server. By leveraging Terraform, this setup ensures consistency, reproducibility, and ease of management, making it ideal for hosting web applications, APIs, or any workload that requires a combination of public and private resources.

Introduction

In today's cloud-centric world, building a secure and scalable infrastructure is critical for hosting applications and services. This Terraform configuration provides a robust solution for deploying AWS resources in a structured and automated manner. It creates a VPC with clearly defined public and private subnets, ensuring that sensitive resources remain isolated while still allowing controlled access. The public subnet hosts an EC2 instance that serves as a reverse proxy, enabling secure communication with a private instance in the private subnet. The private instance, which is not directly accessible from the internet, hosts a web server and can only be reached through the public instance. Security groups are configured to enforce strict access controls, allowing only necessary traffic to flow between resources. This setup not only enhances security but also provides flexibility and scalability, making it suitable for a wide range of use cases, from web application hosting to development and testing environments. By automating the deployment process, this Terraform configuration reduces manual effort, minimizes errors, and ensures consistent infrastructure across different environments.

System Architecture



Service Description

The Terraform configuration provisions the following AWS services:

1. Virtual Private Cloud (VPC)

- A logically isolated network is created with a defined CIDR block (`27.0.50.0/24`).
- The VPC serves as the foundation for all other resources.

2. Subnets

- **Public Subnet**:
  - CIDR block: `27.0.50.0/26`.
  - Hosts resources that need to be publicly accessible (e.g., the public EC2 instance).
- **Private Subnet**:
  - CIDR block: `27.0.50.128/26`.
  - Hosts resources that should not be directly accessible from the internet (e.g., the private EC2 instance).

3. Internet Gateway

- Attached to the VPC to enable internet access for resources in the public subnet.

4. NAT Gateway

- Placed in the public subnet to allow resources in the private subnet to access the internet securely.

5. Route Tables

- **Public Route Table**:
  - Routes traffic from the public subnet to the internet via the Internet Gateway.
- **Private Route Table**:
  - Routes traffic from the private subnet to the internet via the NAT Gateway.

6. Elastic IP (EIP)

- Allocated for use with the NAT Gateway.

7. Security Groups

- **Public Security Group**:
  - Allows inbound traffic on ports 22 (SSH), 80 (HTTP), and 443 (HTTPS).
  - Allows all outbound traffic.
- **Private Security Group**:
  - Allows inbound SSH traffic only from the public instance.
  - Allows all outbound traffic.

8. EC2 Instances

- **Public Instance**:
  - Hosts a reverse proxy to forward traffic to the private instance.
  - Accessible via a public IP address.
- **Private Instance**:
  - Hosts a web server and is accessible only through the public instance.
  - Does not have a public IP address.

9. User Data

- **Public Instance**:
  - Installs and configures Apache HTTP Server as a reverse proxy.
  - Sets up a web page with a link to the private instance.
- **Private Instance**:
  - Installs and configures Apache HTTP Server.

- Hosts a simple web page.

Steps to Deploy the Infrastructure

1. **Install Terraform**:
    a. Download and install Terraform from the official website.
    b. Verify the installation by running: `terraform -version`

```
manoj@IND-140:~/terraf$ terraform -v
Terraform v1.11.2
on linux_amd64
+ provider registry.terraform.io/hashicorp/aws v5.91.0
manoj@IND-140:~/terraf$
```

2. **Set Up AWS Credentials**:
    a. Ensure you have an AWS account and IAM credentials with sufficient permissions to create VPCs, subnets, instances, etc.
    b. Configure your AWS credentials using the AWS CLI or by setting environment variables:
    ```
    export AWS_ACCESS_KEY_ID="your-access-key-id"
    export AWS_SECRET_ACCESS_KEY="your-secret-access-key"
    export AWS_DEFAULT_REGION="us-east-2"
    ```

```
manoj@IND-140: ~/hello
manoj@IND-140:~/hello$ aws configure
AWS Access Key ID [****************36NG]:
AWS Secret Access Key [****************m3kk]:
Default region name [us-east-1]:
Default output format [None]:
manoj@IND-140:~/hello$ ll
total 36
drwxr-xr-x  3 manoj manoj 4096 Mar 15 11:12 ./
drwxr-x--- 15 manoj manoj 4096 Mar 14 18:21 ../
drwxr-xr-x  3 manoj manoj 4096 Mar 14 17:53 .terraform/
-rw-r--r--  1 manoj manoj 1377 Mar 15 10:59 .terraform.lock.hcl
-rw-r--r--  1 manoj manoj 4759 Mar 15 11:01 main.tf
-rw-r--r--  1 manoj manoj  455 Mar 15 11:00 reverse_proxy.sh.tpl
-rw-r--r--  1 manoj manoj  182 Mar 14 18:46 terraform.tfstate
-rw-r--r--  1 manoj manoj 2957 Mar 14 18:46 terraform.tfstate.backup
manoj@IND-140:~/hello$
```

3. **Prepare the Terraform Configuration**:
    a. Save the provided Terraform configuration in a file named `main.tf`.

```
drwxr-x--- 16 manoj manoj  4096 Mar 15 17:48 ../
drwxr-xr-x  3 manoj manoj  4096 Mar 15 17:51 .terraform/
-rw-r--r--  1 manoj manoj  1406 Mar 15 17:51 .terraform.lock.hcl
-rw-r--r--  1 manoj manoj  5403 Mar 15 18:03 main.tf
-rw-r--r--  1 manoj manoj   182 Mar 15 19:11 terraform.tfstate
-rw-r--r--  1 manoj manoj 31658 Mar 15 19:10 terraform.tfstate.backup
```

breakdown of the provided Terraform configuration

Step 1: AWS Provider Configuration

```
provider "aws" {
  region = "us-east-2"
}
```

- This block specifies the AWS provider and sets the region to `us-east-2`.
- Terraform will use this region to create all resources unless overridden.

Step 2: Create a VPC

```
resource "aws_vpc" "myvpc" {
  cidr_block       = "27.0.50.0/24"
  instance_tenancy = "default"

  tags = {
    Name = "TERRAFORMVPC"
  }
}
```

- A VPC (Virtual Private Cloud) is created with the CIDR block `27.0.50.0/24`.
- The `instance_tenancy` is set to `default`, meaning instances will run on shared hardware.
- A tag `Name = "TERRAFORMVPC"` is added for identification.

Step 3: Create a Public Subnet

```
resource "aws_subnet" "pubsubnet" {
  vpc_id     = aws_vpc.myvpc.id
  cidr_block = "27.0.50.0/26"
  tags = {
    Name = "PUBLICSUBNET"
  }
}
```

- A public subnet is created within the VPC with the CIDR block `27.0.50.0/26`.
- The subnet is associated with the VPC using `vpc_id = aws_vpc.myvpc.id`.
- A tag `Name = "PUBLICSUBNET"` is added.

Step 4: Create an Internet Gateway

```
resource "aws_internet_gateway" "my-igw" {
  vpc_id = aws_vpc.myvpc.id

  tags = {
```

```
    Name = "INETGATEWAY"
  }
}
```

- An Internet Gateway (IGW) is created and attached to the VPC.
- This allows resources in the public subnet to communicate with the internet.

Step 5: Create a Public Route Table

```
resource "aws_route_table" "pubroutetable" {
  vpc_id = aws_vpc.myvpc.id

  tags = {
    Name = "PUBLICROUTETABLE"
  }
}
```

- A route table is created for the public subnet.
- It is associated with the VPC.

Step 6: Add a Route to the Internet

```
resource "aws_route" "public_internet_route" {
  route_table_id         = aws_route_table.pubroutetable.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.my-igw.id
}
```

- A route is added to the public route table to allow traffic to the internet (`0.0.0.0/0`).
- The route points to the Internet Gateway.

Step 7: Associate the Public Subnet with the Route Table

```
resource "aws_route_table_association" "pubsubnetroutetable" {
  subnet_id      = aws_subnet.pubsubnet.id
  route_table_id = aws_route_table.pubroutetable.id
}
```

- The public subnet is associated with the public route table.
- This ensures that instances in the public subnet can access the internet.

Step 8: Create an Elastic IP for the NAT Gateway

```
resource "aws_eip" "elasticip" {
  domain = "vpc"
}
```

- An Elastic IP (EIP) is created for use with the NAT Gateway.
- The `domain = "vpc"` ensures the EIP is allocated for use within the VPC.

Step 9: Create a NAT Gateway

```
resource "aws_nat_gateway" "NATGATEWAY" {
  allocation_id = aws_eip.elasticip.id
  subnet_id     = aws_subnet.pubsubnet.id

  tags = {
    Name = "NATGATEWAY"
  }
}
```

- A NAT Gateway is created in the public subnet.
- It uses the Elastic IP created earlier.
- The NAT Gateway allows instances in the private subnet to access the internet.

Step 10: Create a Private Subnet

```
resource "aws_subnet" "pvtsubnet" {
  vpc_id     = aws_vpc.myvpc.id
  cidr_block = "27.0.50.128/26"
  tags = {
    Name = "PRIVATESUBNET"
  }
}
```

- A private subnet is created with the CIDR block `27.0.50.128/26`.
- It is associated with the VPC.

Step 11: Create a Private Route Table

```
resource "aws_route_table" "pvtroutetable" {
  vpc_id = aws_vpc.myvpc.id

  tags = {
    Name = "PRIVATEROUTETABLE"
  }
}
```

- A route table is created for the private subnet.

Step 12: Add a Route to the NAT Gateway

```
resource "aws_route" "private_nat_route" {
  route_table_id         = aws_route_table.pvtroutetable.id
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id         = aws_nat_gateway.NATGATEWAY.id
}
```

- A route is added to the private route table to allow traffic to the internet (`0.0.0.0/0`).

- The route points to the NAT Gateway.

Step 13: Associate the Private Subnet with the Route Table

```
resource "aws_route_table_association" "pvtroutetableassociation" {
  subnet_id      = aws_subnet.pvtsubnet.id
  route_table_id = aws_route_table.pvtroutetable.id
}
```

- The private subnet is associated with the private route table.
- This ensures that instances in the private subnet can access the internet via the NAT Gateway.

Step 14: Create a Security Group for the Public Instance

```
resource "aws_security_group" "public_sg" {
  name        = "public_sg"
  description = "Allow HTTP, HTTPS, and SSH inbound traffic and all
outbound traffic"
  vpc_id      = aws_vpc.myvpc.id

  tags = {
    Name = "PUBLICSG"
  }
}
```

- A security group is created for the public instance.
- It allows inbound traffic on ports 22 (SSH), 80 (HTTP), and 443 (HTTPS).
- All outbound traffic is allowed.

Step 15: Create a Security Group for the Private Instance

```
resource "aws_security_group" "private_sg" {
  name        = "private_sg"
  description = "Allow SSH inbound traffic from the public instance
```

```
and all outbound traffic"
  vpc_id      = aws_vpc.myvpc.id

  tags = {
    Name = "PRIVATESG"
  }
}
```

- A security group is created for the private instance.
- It allows inbound SSH traffic only from the public instance.
- All outbound traffic is allowed.

Step 16: Launch the Private Instance

```
resource "aws_instance" "private_instance" {
  ami             = "ami-0d0f28110d16ee7d6"
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.pvtsubnet.id
  security_groups = [aws_security_group.private_sg.id]

  user_data = <<-EOF
              #!/bin/bash
              sudo yum update -y
              sudo yum install -y httpd
              sudo systemctl enable httpd
              echo "<h1>This is the PRIVATE instance</h1>" >
/var/www/html/index.html
              systemctl enable httpd
              systemctl start httpd
              EOF

  tags = {
    Name = "PrivateInstance"
  }
}
```

- A private EC2 instance is launched in the private subnet.

- It uses the Amazon Linux 2 AMI (`ami-0d0f28110d16ee7d6`).
- The instance is associated with the private security group.
- User data installs and configures Apache HTTP Server.

Step 17: Launch the Public Instance

```
resource "aws_instance" "public_instance" {
  ami             = "ami-0d0f28110d16ee7d6"
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.pubsubnet.id
  security_groups = [aws_security_group.public_sg.id]
  associate_public_ip_address = true

  user_data = <<-EOF
              #!/bin/bash
              sudo yum update -y
              sudo yum install apache2 -y
              sudo yum install -y libapache2-mod-proxy-html libxml2-
dev
              sudo a2enmod proxy
              sudo a2enmod proxy_http
              sudo systemctl restart apache2
              # Set up reverse proxy to private instance
              sudo bash -c 'cat > /etc/apache2/sites-available/000-
default.conf <<EOL
              <VirtualHost *:80>
                  ServerAdmin webmaster@localhost
                  DocumentRoot /var/www/html

                  ProxyPass "/private"
"http://${aws_instance.private_instance.private_ip}/"
                  ProxyPassReverse "/private"
"http://${aws_instance.private_instance.private_ip}/"
                  ErrorLog /var/log/apache2/error.log
                  CustomLog /var/log/apache2/access.log combined
              </VirtualHost>
              EOL'
              sudo systemctl restart apache2
```

```
              echo "<h1>This is a Public Instance</h1><a
href='/private'>Go to Private Instance</a>" > /var/www/html/index.html
EOF

  tags = {
    Name = "PublicInstance"
  }
}
```

4. **Initialize Terraform**:
    a. Run the following command in the directory where `main.tf` is located:`terraform init`

    b. This command initializes the working directory and downloads the necessary provider plugins.

```
manoj@IND-140:~/hello$ terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.91.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
manoj@IND-140:~/hello$
```

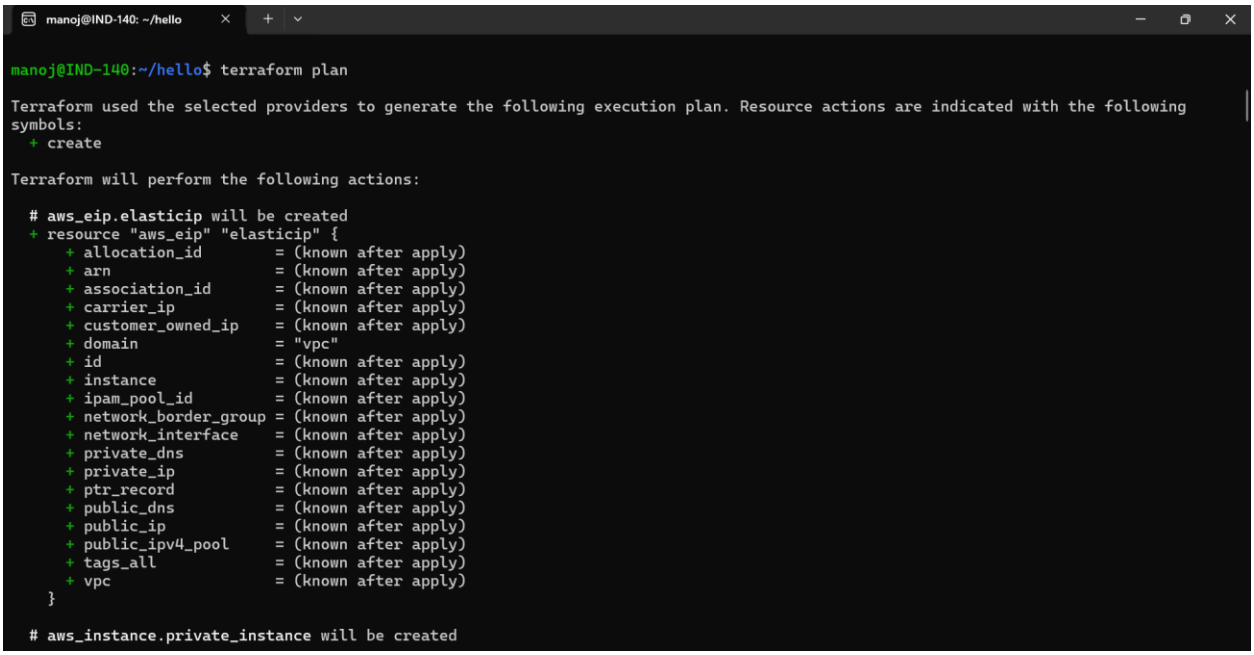
5. **Validate the Configuration**:
    a. Validate the Terraform configuration for syntax errors:`terraform validate`

```
manoj@IND-140:~/hello$ terraform validate
Success! The configuration is valid.

manoj@IND-140:~/hello$
```

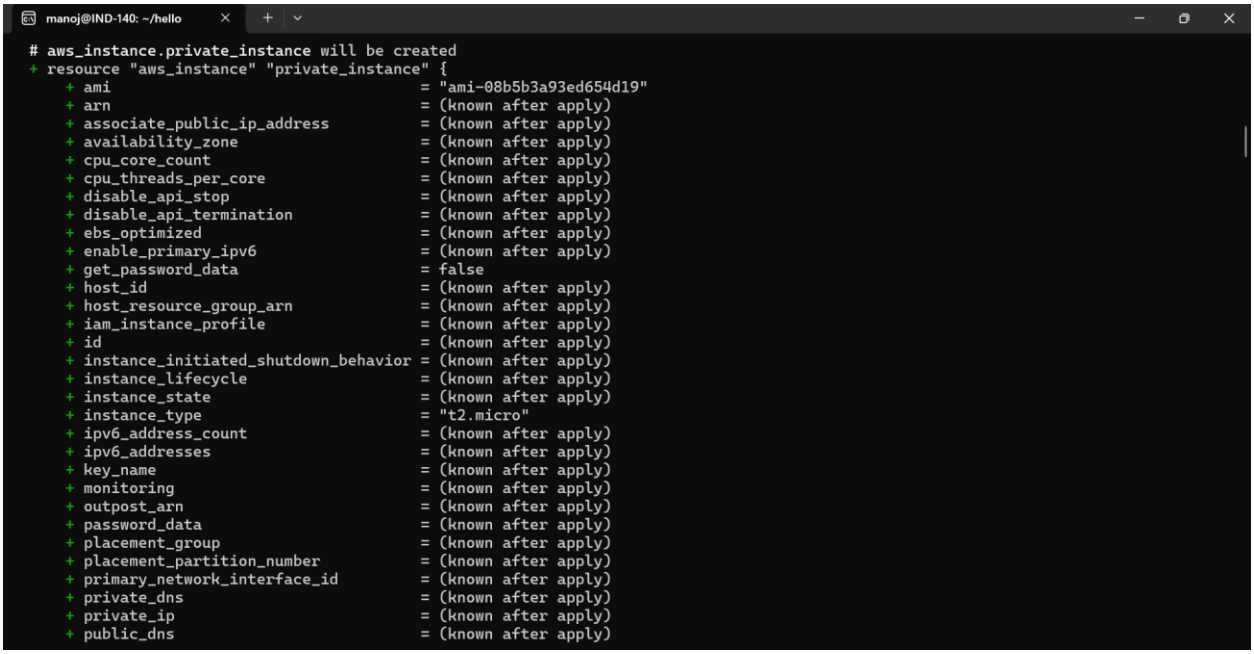
6. **Preview the Plan**:

a. Generate an execution plan to see what Terraform will do: `terraform plan`

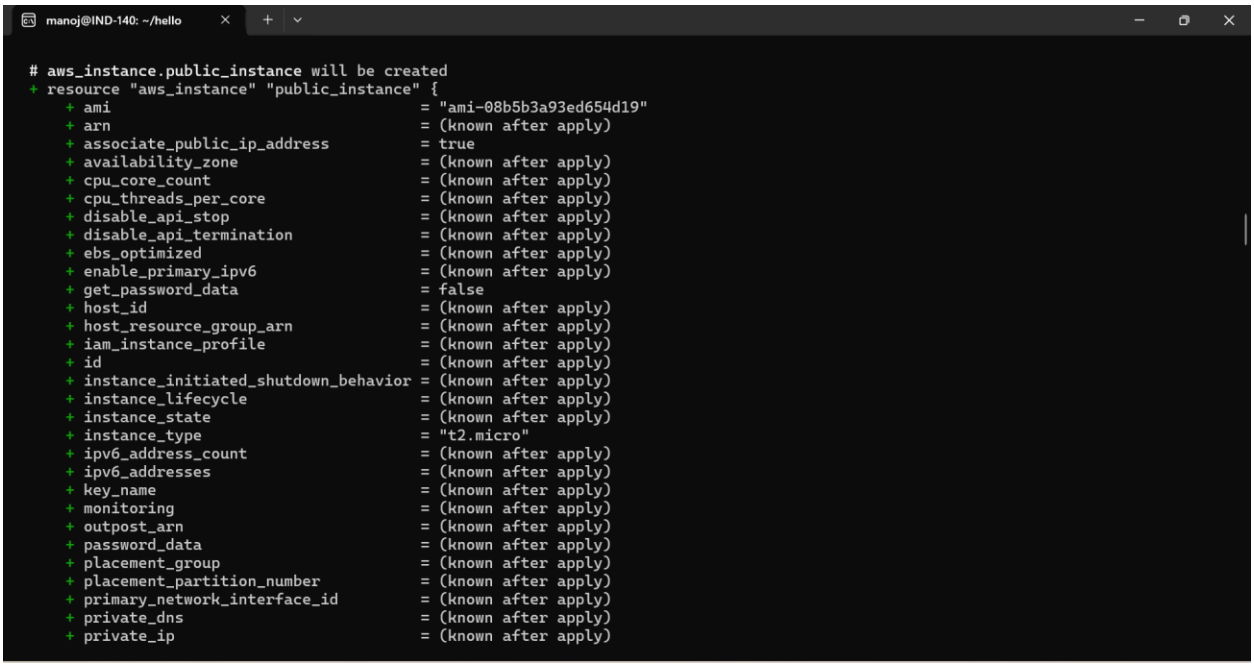
b. Review the plan to ensure it matches your expectations.

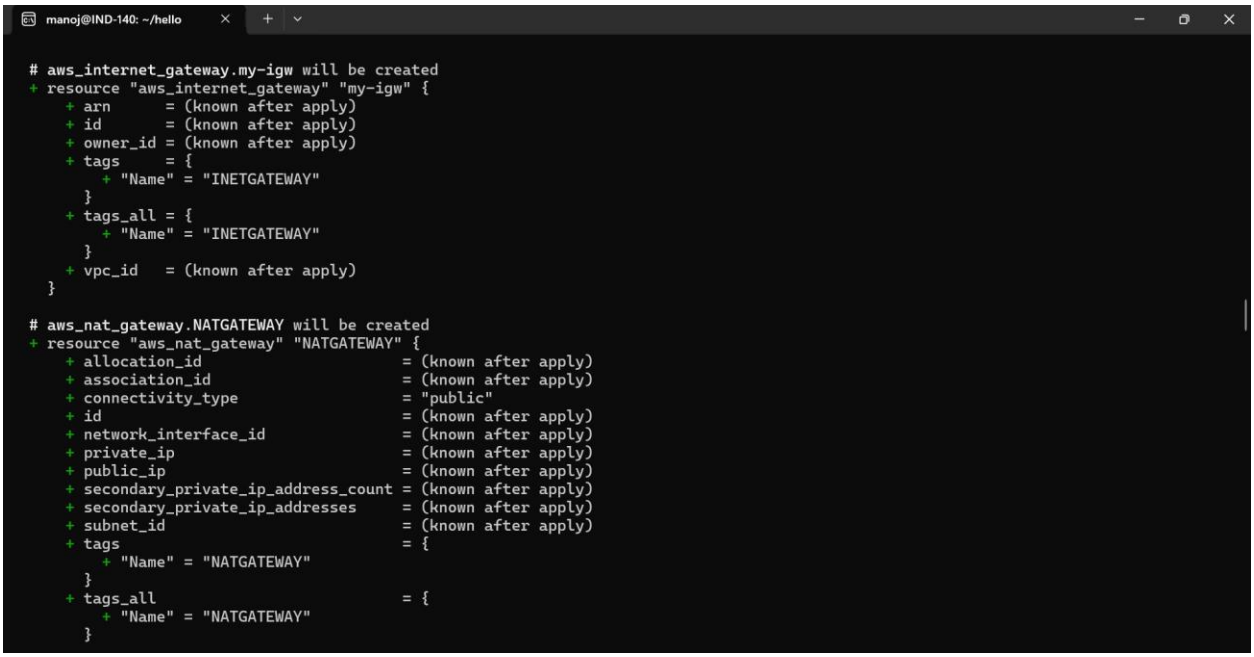


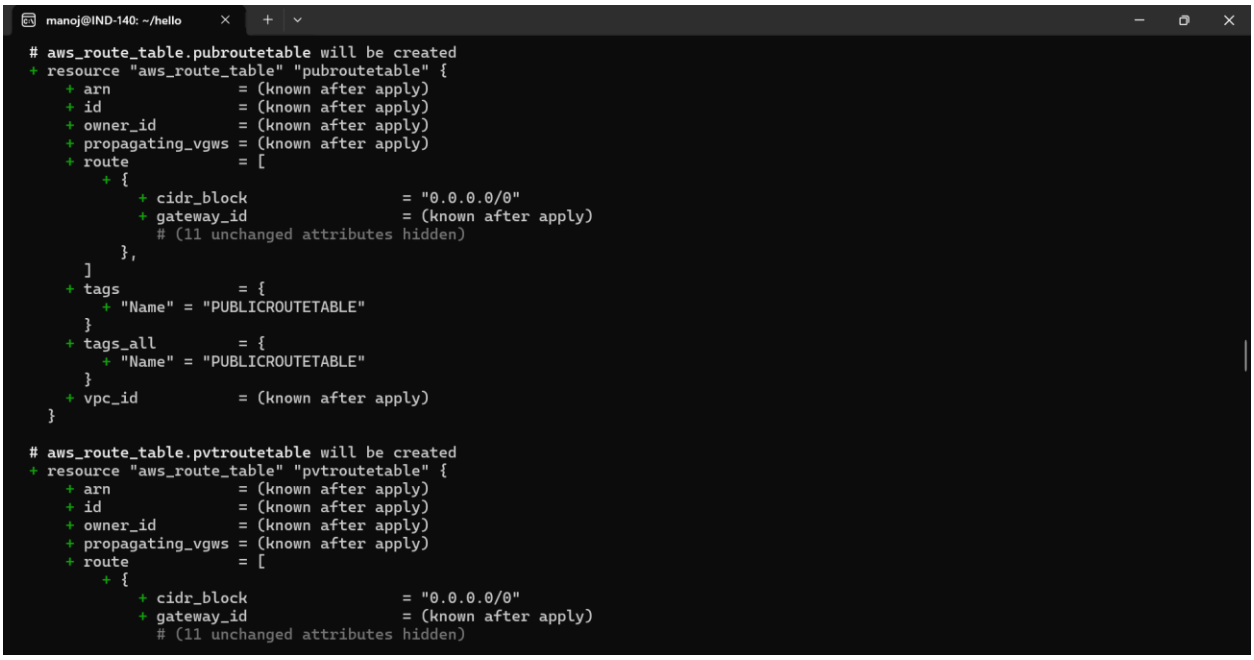
```
manoj@IND-140: ~/hello

manoj@IND-140:~/hello$ terraform plan

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_eip.elasticip will be created
  + resource "aws_eip" "elasticip" {
      + allocation_id        = (known after apply)
      + arn                  = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = "vpc"
      + id                   = (known after apply)
      + instance             = (known after apply)
      + ipam_pool_id         = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
      + ptr_record           = (known after apply)
      + public_dns           = (known after apply)
      + public_ip            = (known after apply)
      + public_ipv4_pool     = (known after apply)
      + tags_all             = (known after apply)
      + vpc                  = (known after apply)
    }

  # aws_instance.private_instance will be created
```



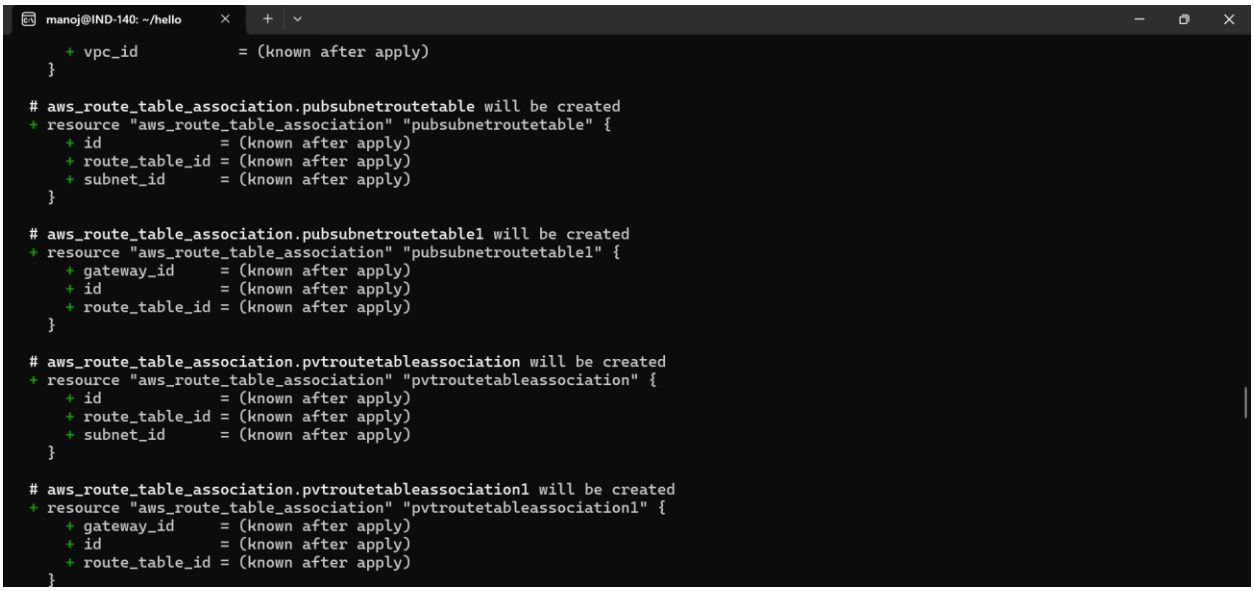
```
manoj@IND-140: ~/hello

  # aws_instance.private_instance will be created
  + resource "aws_instance" "private_instance" {
      + ami                                  = "ami-08b5b3a93ed654d19"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + enable_primary_ipv6                  = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
```


manoj@IND-140: ~/hello
# aws_instance.public_instance will be created
+ resource "aws_instance" "public_instance" {
+ ami = "ami-08b5b3a93ed654d19"
+ arn = (known after apply)
+ associate_public_ip_address = true
+ availability_zone = (known after apply)
+ cpu_core_count = (known after apply)
+ cpu_threads_per_core = (known after apply)
+ disable_api_stop = (known after apply)
+ disable_api_termination = (known after apply)
+ ebs_optimized = (known after apply)
+ enable_primary_ipv6 = (known after apply)
+ get_password_data = false
+ host_id = (known after apply)
+ host_resource_group_arn = (known after apply)
+ iam_instance_profile = (known after apply)
+ id = (known after apply)
+ instance_initiated_shutdown_behavior = (known after apply)
+ instance_lifecycle = (known after apply)
+ instance_state = (known after apply)
+ instance_type = "t2.micro"
+ ipv6_address_count = (known after apply)
+ ipv6_addresses = (known after apply)
+ key_name = (known after apply)
+ monitoring = (known after apply)
+ outpost_arn = (known after apply)
+ password_data = (known after apply)
+ placement_group = (known after apply)
+ placement_partition_number = (known after apply)
+ primary_network_interface_id = (known after apply)
+ private_dns = (known after apply)
+ private_ip = (known after apply)

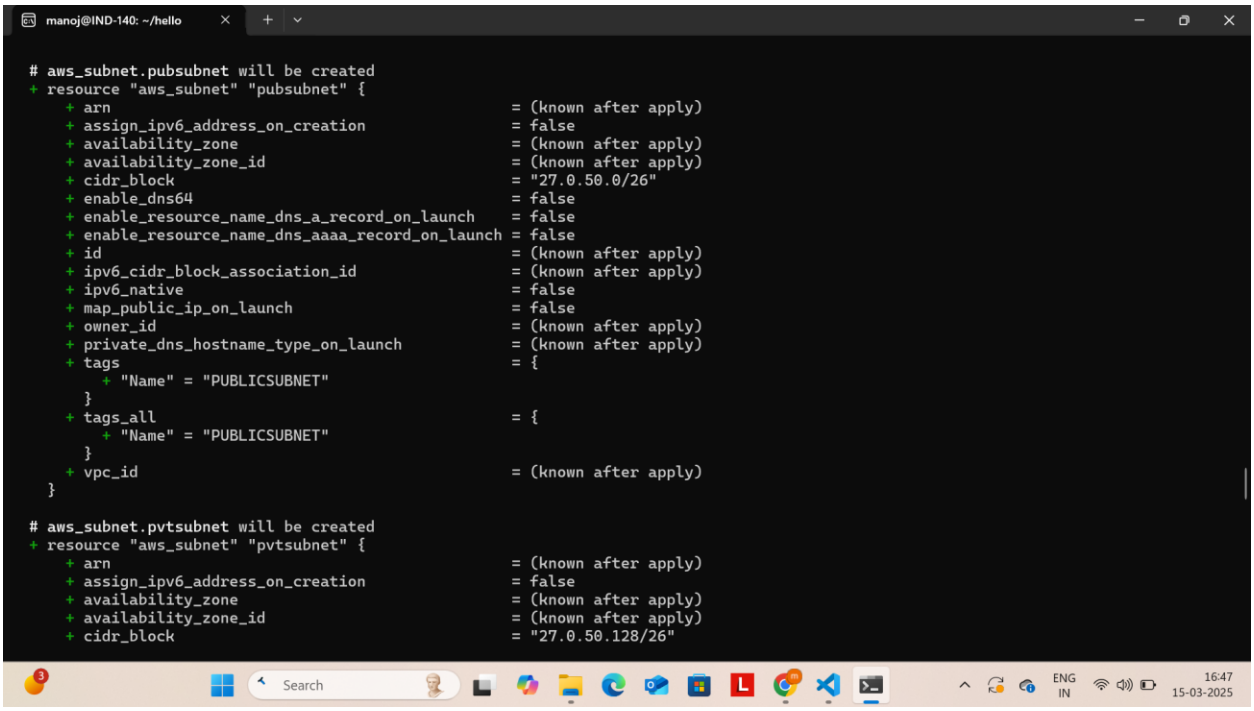

manoj@IND-140: ~/hello
# aws_internet_gateway.my-igw will be created
+ resource "aws_internet_gateway" "my-igw" {
+ arn = (known after apply)
+ id = (known after apply)
+ owner_id = (known after apply)
+ tags = {
+ "Name" = "INETGATEWAY"
}
+ tags_all = {
+ "Name" = "INETGATEWAY"
}
+ vpc_id = (known after apply)
}
# aws_nat_gateway.NATGATEWAY will be created
+ resource "aws_nat_gateway" "NATGATEWAY" {
+ allocation_id = (known after apply)
+ association_id = (known after apply)
+ connectivity_type = "public"
+ id = (known after apply)
+ network_interface_id = (known after apply)
+ private_ip = (known after apply)
+ public_ip = (known after apply)
+ secondary_private_ip_address_count = (known after apply)
+ secondary_private_ip_addresses = (known after apply)
+ subnet_id = (known after apply)
+ tags = {
+ "Name" = "NATGATEWAY"
}
+ tags_all = {
+ "Name" = "NATGATEWAY"
}

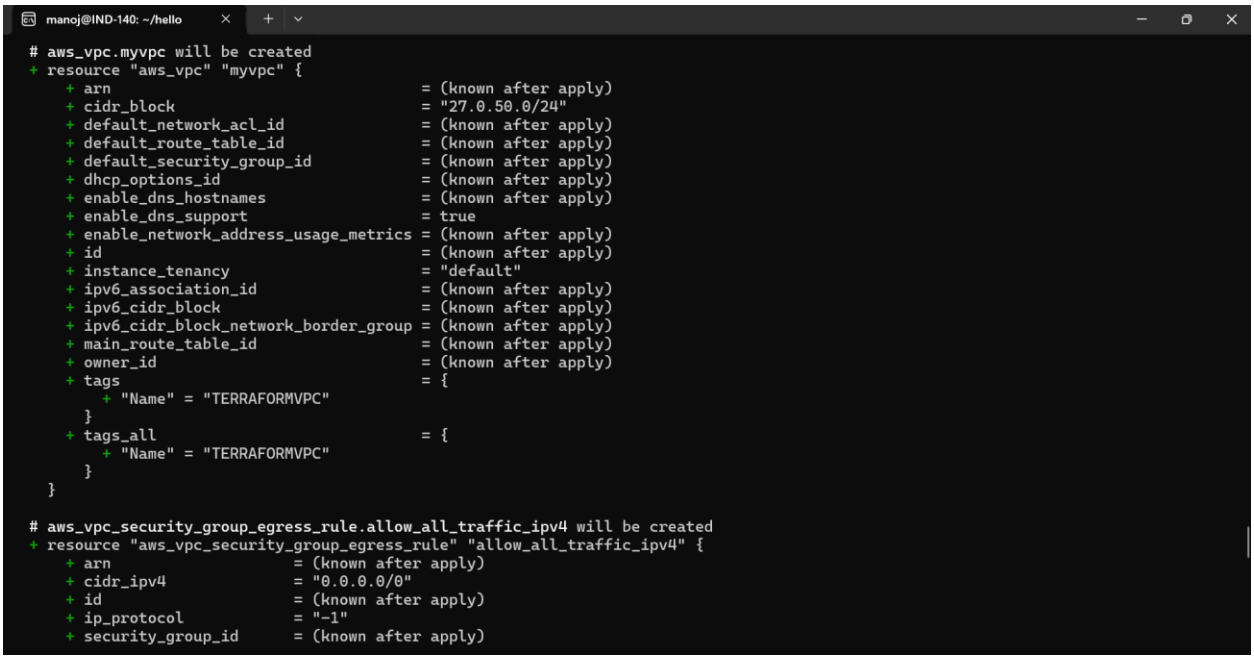


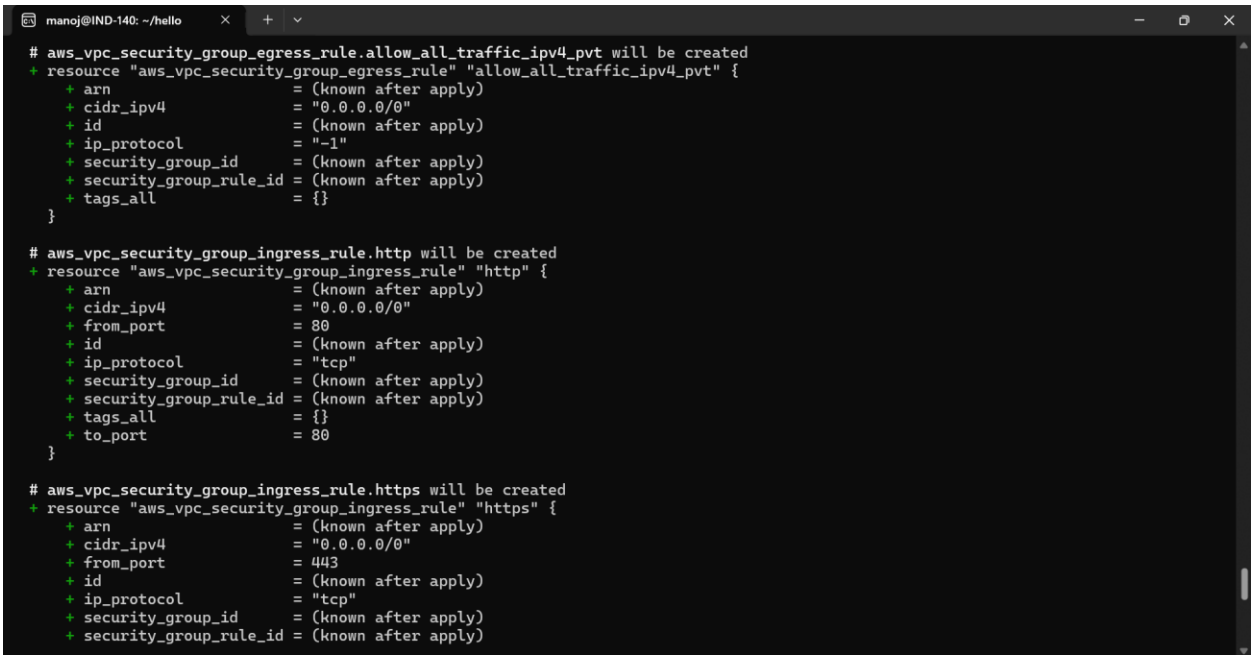
```
manoj@IND-140: ~/hello
# aws_route_table.pubroutetable will be created
+ resource "aws_route_table" "pubroutetable" {
    + arn              = (known after apply)
    + id               = (known after apply)
    + owner_id         = (known after apply)
    + propagating_vgws = (known after apply)
    + route            = [
        + {
            + cidr_block                = "0.0.0.0/0"
            + gateway_id                = (known after apply)
              # (11 unchanged attributes hidden)
          },
      ]
    + tags             = {
        + "Name" = "PUBLICROUTETABLE"
      }
    + tags_all         = {
        + "Name" = "PUBLICROUTETABLE"
      }
    + vpc_id           = (known after apply)
  }

# aws_route_table.pvtroutetable will be created
+ resource "aws_route_table" "pvtroutetable" {
    + arn              = (known after apply)
    + id               = (known after apply)
    + owner_id         = (known after apply)
    + propagating_vgws = (known after apply)
    + route            = [
        + {
            + cidr_block                = "0.0.0.0/0"
            + gateway_id                = (known after apply)
              # (11 unchanged attributes hidden)
```

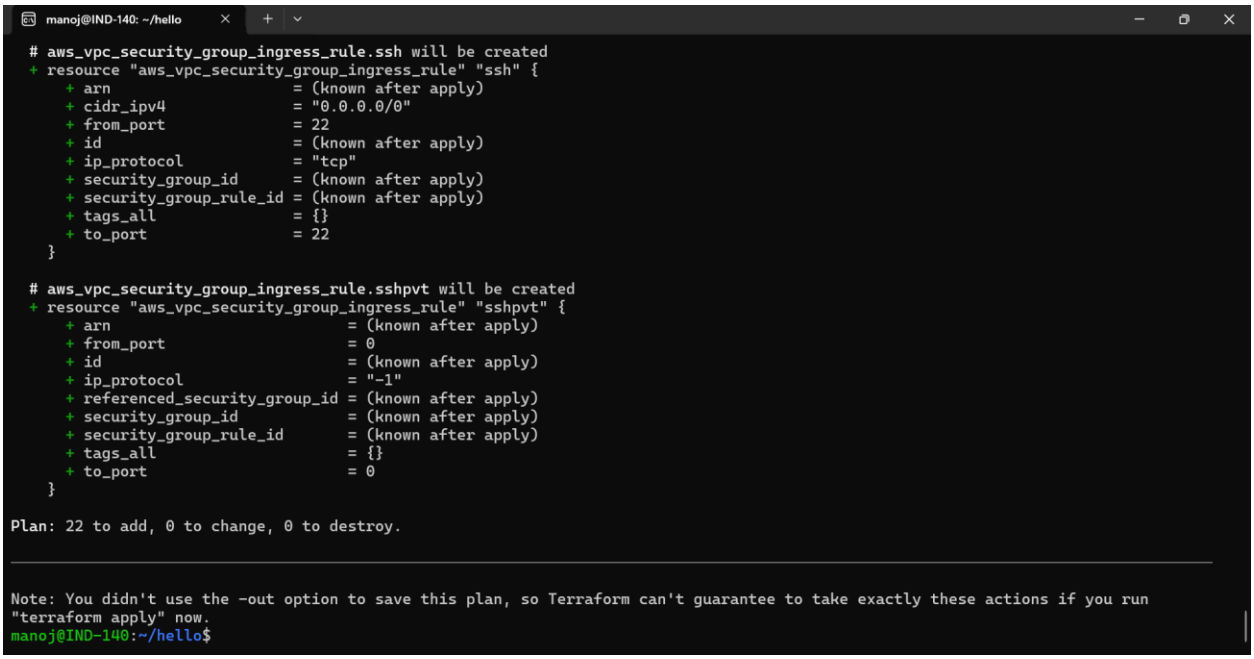


```
manoj@IND-140: ~/hello
    + vpc_id           = (known after apply)
  }

# aws_route_table_association.pubsubnetroutetable will be created
+ resource "aws_route_table_association" "pubsubnetroutetable" {
    + id             = (known after apply)
    + route_table_id = (known after apply)
    + subnet_id      = (known after apply)
  }

# aws_route_table_association.pubsubnetroutetable1 will be created
+ resource "aws_route_table_association" "pubsubnetroutetable1" {
    + gateway_id     = (known after apply)
    + id             = (known after apply)
    + route_table_id = (known after apply)
  }

# aws_route_table_association.pvtroutetableassociation will be created
+ resource "aws_route_table_association" "pvtroutetableassociation" {
    + id             = (known after apply)
    + route_table_id = (known after apply)
    + subnet_id      = (known after apply)
  }

# aws_route_table_association.pvtroutetableassociation1 will be created
+ resource "aws_route_table_association" "pvtroutetableassociation1" {
    + gateway_id     = (known after apply)
    + id             = (known after apply)
    + route_table_id = (known after apply)
  }
```


manoj@IND-140: ~/hello
# aws_subnet.pubsubnet will be created
+ resource "aws_subnet" "pubsubnet" {
+ arn = (known after apply)
+ assign_ipv6_address_on_creation = false
+ availability_zone = (known after apply)
+ availability_zone_id = (known after apply)
+ cidr_block = "27.0.50.0/26"
+ 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" = "PUBLICSUBNET"
}
+ tags_all = {
+ "Name" = "PUBLICSUBNET"
}
+ vpc_id = (known after apply)
}
# aws_subnet.pvtsubnet will be created
+ resource "aws_subnet" "pvtsubnet" {
+ arn = (known after apply)
+ assign_ipv6_address_on_creation = false
+ availability_zone = (known after apply)
+ availability_zone_id = (known after apply)
+ cidr_block = "27.0.50.128/26"
Search
ENG
IN
16:47
15-03-2025

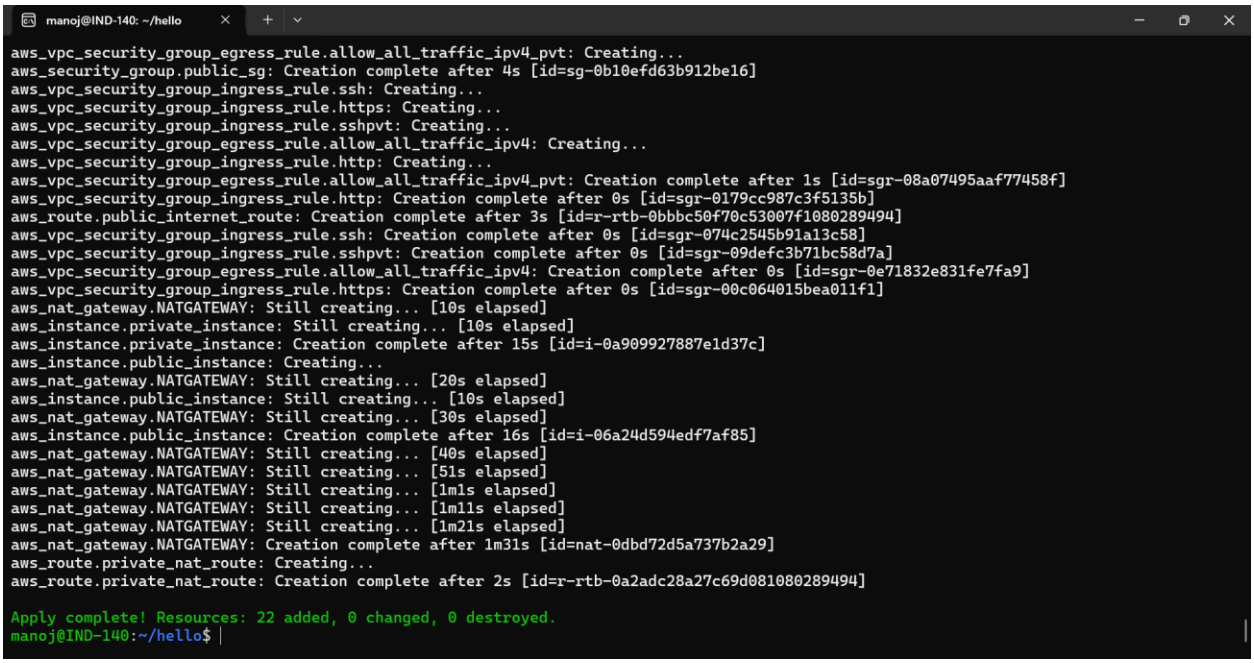

manoj@IND-140: ~/hello
# aws_vpc.myvpc will be created
+ resource "aws_vpc" "myvpc" {
+ arn = (known after apply)
+ cidr_block = "27.0.50.0/24"
+ default_network_acl_id = (known after apply)
+ default_route_table_id = (known after apply)
+ default_security_group_id = (known after apply)
+ dhcp_options_id = (known after apply)
+ enable_dns_hostnames = (known after apply)
+ enable_dns_support = true
+ enable_network_address_usage_metrics = (known after apply)
+ id = (known after apply)
+ instance_tenancy = "default"
+ ipv6_association_id = (known after apply)
+ ipv6_cidr_block = (known after apply)
+ ipv6_cidr_block_network_border_group = (known after apply)
+ main_route_table_id = (known after apply)
+ owner_id = (known after apply)
+ tags = {
+ "Name" = "TERRAFORMVPC"
}
+ tags_all = {
+ "Name" = "TERRAFORMVPC"
}
}
# aws_vpc_security_group_egress_rule.allow_all_traffic_ipv4 will be created
+ resource "aws_vpc_security_group_egress_rule" "allow_all_traffic_ipv4" {
+ arn = (known after apply)
+ cidr_ipv4 = "0.0.0.0/0"
+ id = (known after apply)
+ ip_protocol = "-1"
+ security_group_id = (known after apply)





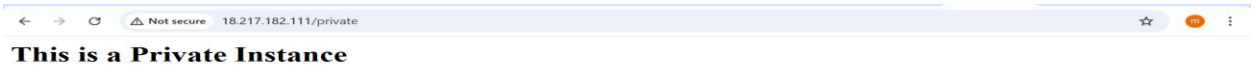
7. **Apply the Configuration**:
    a. Deploy the infrastructure by applying the configuration: `terraform apply`

    b. Terraform will show the execution plan again. Type `yes` to confirm and proceed.



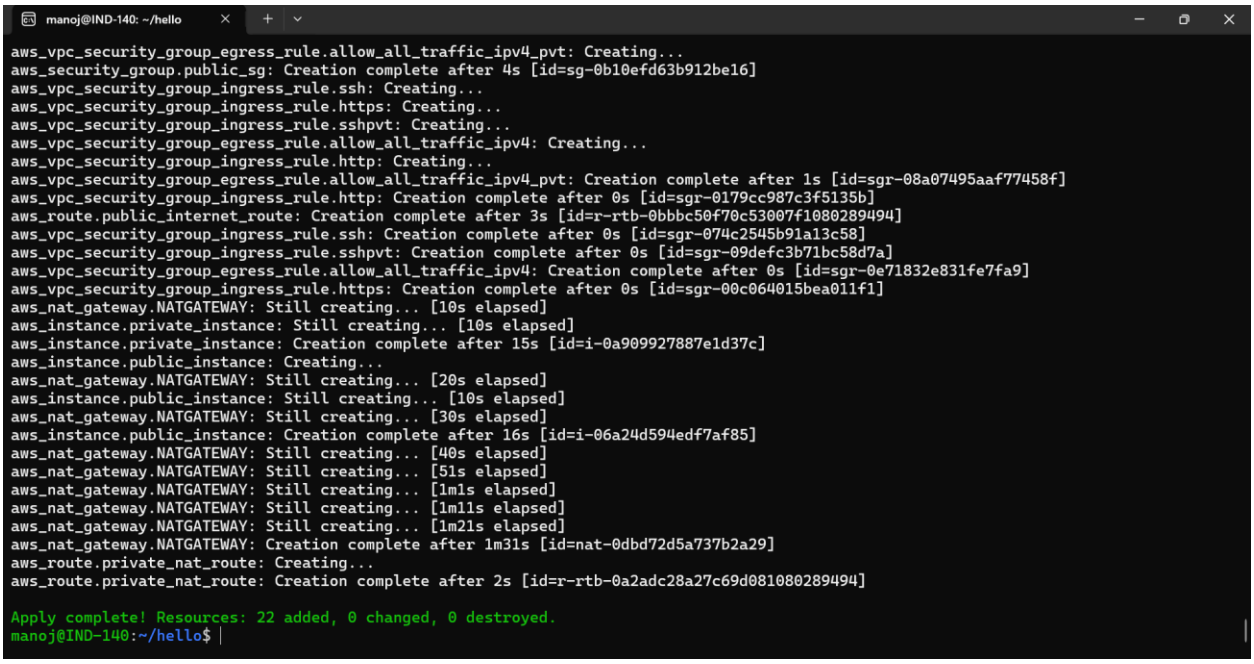
8. **Verify the Deployment**:
    a. Once the deployment is complete, Terraform will output the public IP address of the public instance.
    b. Access the public instance via its public IP in a web browser:`http://<public-ip>`

    c. Click the link to access the private instance through the reverse proxy.

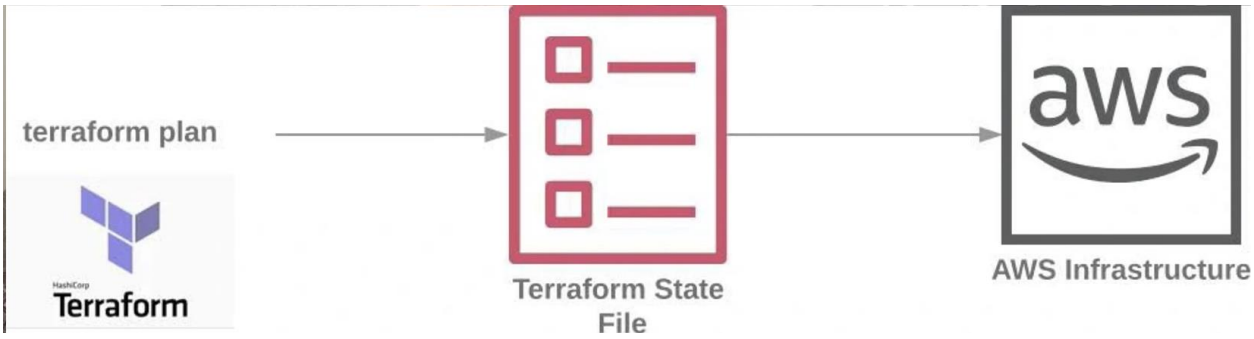


9. **Clean Up** :
    a. If you want to destroy the infrastructure to avoid incurring costs, run:`terraform destroy`

    b. Confirm the destruction by typing `yes`.



FINAL ARCHITECTURE :



Conclusion :

This Terraform configuration provides a **secure, scalable, and automated solution** for deploying AWS infrastructure. It is ideal for hosting web applications, APIs, or any workload that requires a combination of public and private resources. By leveraging Terraform, the infrastructure can be easily managed, modified, and reproduced acro ss different environments.

APPENDIX

SOURCECODE [HCL] :

```
provider "aws" {
  region = "us-east-2"
}

resource "aws_vpc" "myvpc" {
  cidr_block       = "27.0.50.0/24"
  instance_tenancy = "default"

  tags = {
    Name = "TERRAFORMVPC"
  }
}

resource "aws_subnet" "pubsubnet" {
  vpc_id     = aws_vpc.myvpc.id
  cidr_block = "27.0.50.0/26"
  tags = {
    Name = "PUBLICSUBNET"
```

```
  }
}

resource "aws_internet_gateway" "my-igw" {
  vpc_id = aws_vpc.myvpc.id

  tags = {
    Name = "INETGATEWAY"
  }
}

resource "aws_route_table" "pubroutetable" {
  vpc_id = aws_vpc.myvpc.id

  tags = {
    Name = "PUBLICROUTETABLE"
  }
}

resource "aws_route" "public_internet_route" {
  route_table_id         = aws_route_table.pubroutetable.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.my-igw.id
}

resource "aws_route_table_association" "pubsubnetroutetable" {
  subnet_id      = aws_subnet.pubsubnet.id
  route_table_id = aws_route_table.pubroutetable.id
}

resource "aws_eip" "elasticip" {
  domain = "vpc"
}

resource "aws_nat_gateway" "NATGATEWAY" {
  allocation_id = aws_eip.elasticip.id
  subnet_id     = aws_subnet.pubsubnet.id

  tags = {
    Name = "NATGATEWAY"
  }
}
```

```
resource "aws_subnet" "pvtsubnet" {
  vpc_id     = aws_vpc.myvpc.id
  cidr_block = "27.0.50.128/26"
  tags = {
    Name = "PRIVATESUBNET"
  }
}

resource "aws_route_table" "pvtroutetable" {
  vpc_id = aws_vpc.myvpc.id

  tags = {
    Name = "PRIVATEROUTETABLE"
  }
}

resource "aws_route" "private_nat_route" {
  route_table_id         = aws_route_table.pvtroutetable.id
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id         = aws_nat_gateway.NATGATEWAY.id
}

resource "aws_route_table_association" "pvtroutetableassociation" {
  subnet_id      = aws_subnet.pvtsubnet.id
  route_table_id = aws_route_table.pvtroutetable.id
}

resource "aws_security_group" "public_sg" {
  name        = "public_sg"
  description = "Allow HTTP, HTTPS, and SSH inbound traffic and all outbound
traffic"
  vpc_id      = aws_vpc.myvpc.id

  tags = {
    Name = "PUBLICSG"
  }
}

resource "aws_vpc_security_group_ingress_rule" "ssh" {
  security_group_id = aws_security_group.public_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 22
```

```
  ip_protocol       = "tcp"
  to_port           = 22
}

resource "aws_vpc_security_group_ingress_rule" "http" {
  security_group_id = aws_security_group.public_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 80
  ip_protocol       = "tcp"
  to_port           = 80
}

resource "aws_vpc_security_group_ingress_rule" "https" {
  security_group_id = aws_security_group.public_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 443
  ip_protocol       = "tcp"
  to_port           = 443
}

resource "aws_vpc_security_group_egress_rule" "allow_all_traffic_ipv4" {
  security_group_id = aws_security_group.public_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "-1"
}

resource "aws_security_group" "private_sg" {
  name        = "private_sg"
  description = "Allow SSH inbound traffic from the public instance and all
outbound traffic"
  vpc_id      = aws_vpc.myvpc.id

  tags = {
    Name = "PRIVATESG"
  }
}

resource "aws_vpc_security_group_ingress_rule" "sshpvt" {
  security_group_id            = aws_security_group.private_sg.id
  referenced_security_group_id = aws_security_group.public_sg.id
  from_port                    = 22
  to_port                      = 22
  ip_protocol                  = "tcp"
```

```
}

resource "aws_vpc_security_group_egress_rule" "allow_all_traffic_ipv4_pvt" {
  security_group_id = aws_security_group.private_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "-1"
}

resource "aws_instance" "private_instance" {
  ami             = "ami-0d0f28110d16ee7d6"
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.pvtsubnet.id
  security_groups = [aws_security_group.private_sg.id]



  user_data = <<-EOF
              #!/bin/bash
              sudo yum update -y
              sudo yum install -y httpd
              sudo systemctl enable httpd
              echo "<h1>This is the PRIVATE instance</h1>" >
/var/www/html/index.html
              systemctl enable httpd
              systemctl start httpd
              EOF

  tags = {
    Name = "PrivateInstance"
  }
}

resource "aws_instance" "public_instance" {
  ami             = "ami-0d0f28110d16ee7d6"
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.pubsubnet.id
  security_groups = [aws_security_group.public_sg.id]
  associate_public_ip_address = true

  user_data = <<-EOF
              #!/bin/bash
              sudo yum update -y
              sudo yum install apache2 -y
              sudo yum install -y libapache2-mod-proxy-html libxml2-dev
```

```
              sudo a2enmod proxy
              sudo a2enmod proxy_http
              sudo systemctl restart apache2
              # Set up reverse proxy to private instance
              sudo bash -c 'cat > /etc/apache2/sites-available/000-default.conf
<<EOL
              <VirtualHost *:80>
                  ServerAdmin webmaster@localhost
                  DocumentRoot /var/www/html

                  ProxyPass "/private"
"http://${aws_instance.private_instance.private_ip}/"
                  ProxyPassReverse "/private"
"http://${aws_instance.private_instance.private_ip}/"
                  ErrorLog /var/log/apache2/error.log
                  CustomLog /var/log/apache2/access.log combined
              </VirtualHost>
              EOL'
              sudo systemctl restart apache2
              echo "<h1>This is a Public Instance</h1><a href='/private'>Go to
Private Instance</a>" > /var/www/html/index.html
EOF

  tags = {
    Name = "PublicInstance"
  }
}
```