CASE STUDY -CREATING AN ARCHITECTURE USING TERRAFORM ON AWS

You work as a DevOps Engineer in a leading Software Company.
You have been asked to build an infrastructure safely and efficiently.

The company Requirements:
1. Use AWS cloud Provider and the software to be installed is Apache2.
2. Use Ubuntu AMI.
The company wants the Architecture to have the following services:
1. Create a template with a VPC, 2 subnets and 1 instance in each subnet.
2. Attach Security groups, internet gateway and network interface to the instance.

==========

Let us install Terraform in our local system first.
Go to the Terraform downloads page: https://www.terraform.io/downloads.html
Download and extract the appropriate version of Terraform for your Windows system (32-bit or 64-bit).

Now, we will open terminal or command Prompt, go to the path where we have created the folder for terraform and type terraform init command. Whenever you are starting to use terraform, first you need to run this command to tell terraform which provider you are using.

Now in the path where terraform is installed, we will create a file called provider.tf and will give provider details there.

```
provider "aws" {
  region     = "us-east-1"
  access_key = "PUT-YOUR-ACCESS-KEY-HERE"
  secret_key = "PUT-YOUR-SECRET-KEY-HERE"
}
```

Let's break down the individual components:

1. provider: This keyword tells Terraform that we are defining a provider configuration block. In this case, we are setting up an AWS provider.

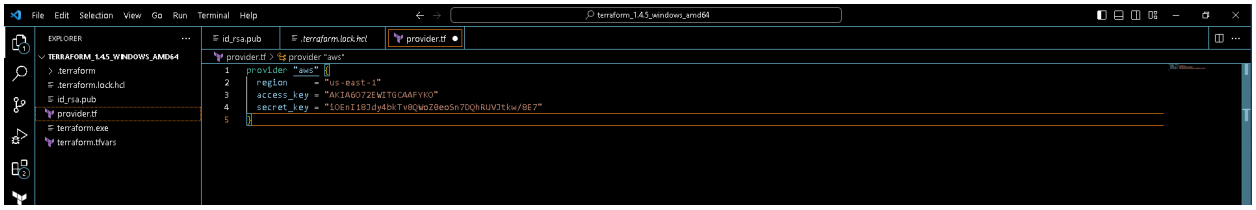
2. "aws": This is the name of the provider we are configuring. Terraform supports many different providers for various cloud platforms and services.

3. region: This is a configuration setting for the AWS provider, specifying which region we want to use. In this case, we are using the us-east-1 region, which is one of the most commonly used regions in AWS.

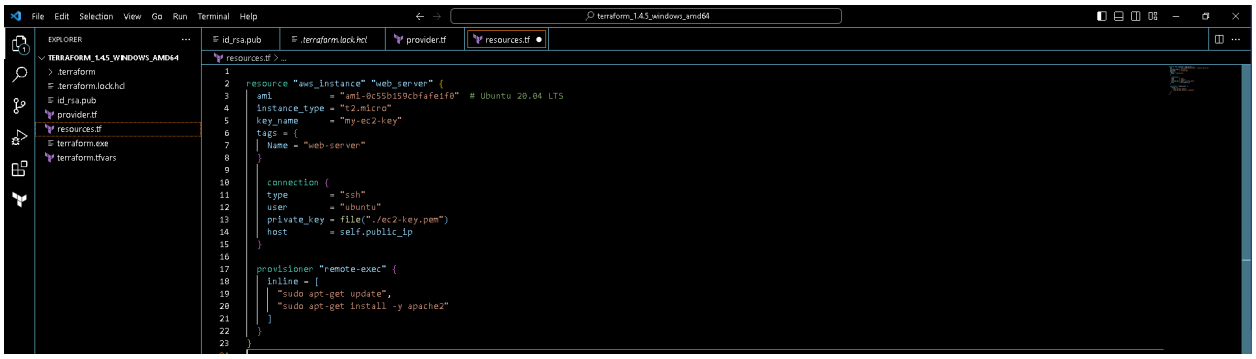
4. access_key: This is a configuration setting for the AWS provider, specifying the access key credential for an AWS account. Access keys are used to authenticate with AWS APIs and services.

5. secret_key: This is a configuration setting for the AWS provider, specifying the secret key credential for an AWS account. Secret keys are used to authenticate with AWS APIs and services, and should be kept confidential.

Together, these settings provide the necessary configuration for Terraform to authenticate with an AWS account and interact with resources in the us-east-1 region using the specified access and secret keys.



Let us move to a further path where we install apache2 in ubuntu AMI.
Create a terraform.tf file where we will define all resources.




```
resource "aws_instance" "web_server" {
  ami           = "ami-0c55b159cbfafe1f0"  # Ubuntu 20.04 LTS
  instance_type = "t2.micro"
  key_name      = "ec2-key"
  tags = {
    Name = "for-assignment"
  }

  connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("./ec2-key.pem")
    host        = self.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y apache2"
    ]
  }
}
```

This Terraform code creates an AWS EC2 instance in the default VPC of the default region (us-east-1) running Ubuntu 20.04 LTS (AMI ID: ami-0c55b159cbfafe1f0) with an instance type of t2.micro.

It also sets the name tag of the EC2 instance to "web-server" and specifies an EC2 key pair named "ec2-key" for SSH access.

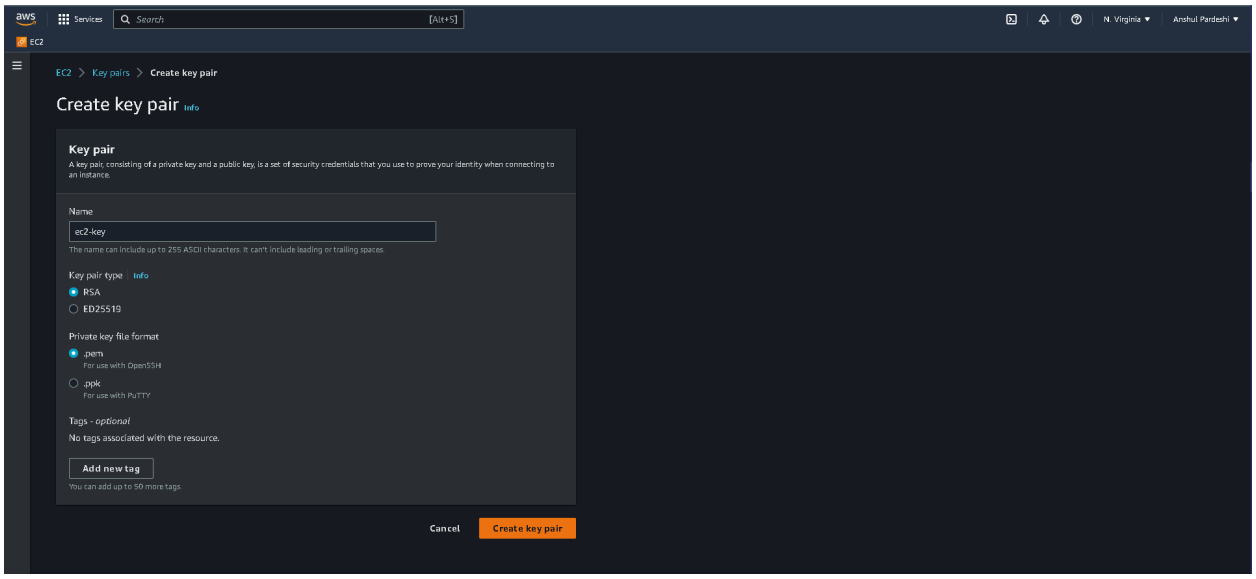
The connection block specifies how to connect to the EC2 instance using SSH. The type is set to ssh, the user is set to ubuntu (the default user for the Ubuntu AMI), the private_key is set to the path of the private key file for the EC2 key pair, and the host is set to self.public_ip, which is the public IP address of the EC2 instance.

The provisioner block specifies a remote-exec provisioner, which runs the specified commands on the EC2 instance after it's launched. In this case, it updates the package manager and installs Apache2.

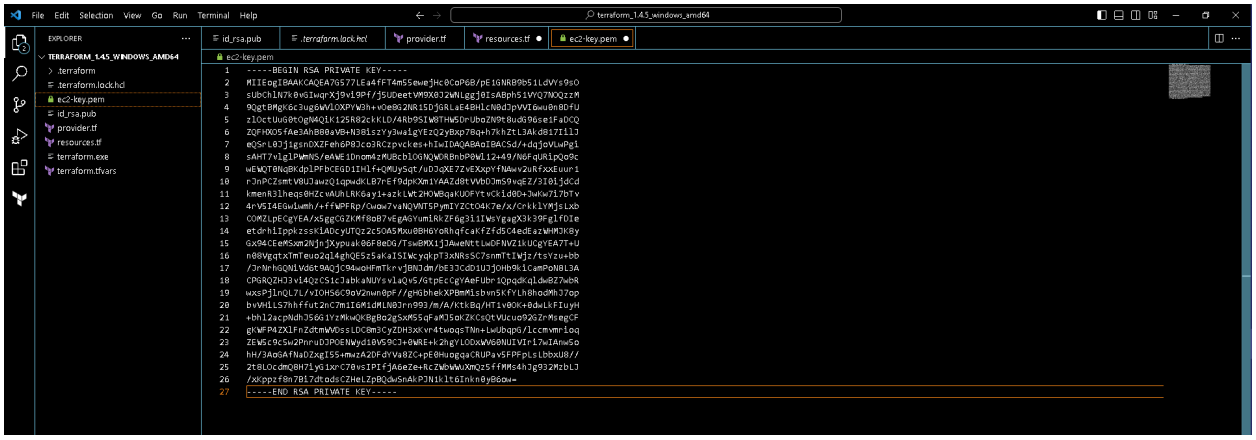
When you apply this code using terraform apply, Terraform will create an EC2 instance with the specified configuration and automatically install Apache2 on it.

We need “ec2-key.pem” in the terraform folder as a file specified in the code.
WE need to create that key in AWS and copy the key details here.

Create a keypair in AWS with same details.



Copy the key and paste it in the terraform folder.



Now all the above steps that we performed will create ec2 with ubuntu ami and install apache2 on it.

That is not our end goal.

Let us create a VPC with 2 subnets first. Then we will create ec2 instance in those subnets with apache2 installed in them.

Let us create new file “networking.tf” in which we will create a VPC with 2 subnets.

Let us create a VPC first.

```
# Create a VPC
resource "aws_vpc" "forassignment_vpc" {
  cidr_block = "10.0.0.0/16"
  region = "us-east-1"

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

This Terraform code creates an AWS VPC (Virtual Private Cloud) with a specified CIDR block of 10.0.0.0/16, which means that the VPC will have a total of 65,536 IP addresses available for use.

The aws_vpc resource type is used to define the VPC. The cidr_block parameter specifies the IP address range for the VPC. The tags parameter is used to attach metadata to the VPC resource, in this case a name tag of "forassignment-vpc".

Now let us create two subnets in that VPC.

```
# Create two subnets in the VPC
resource "aws_subnet" "forassignment_subnet_1" {
  vpc_id     = aws_vpc.forassignment_vpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "forassignment-subnet-1"
  }
}

resource "aws_subnet" "forassignment_subnet_2" {
  vpc_id     = aws_vpc.forassignment_vpc.id
  cidr_block = "10.0.2.0/24"
  availability_zone = "us-east-1b"

  tags = {
    Name = "forassignment-subnet-2"
  }
}
```

This Terraform code creates two subnets within the VPC that was created in the previous code block.

The aws_subnet resource type is used to define each subnet, and the vpc_id parameter is used to specify the ID of the VPC where the subnets should be created. The cidr_block parameter specifies the IP address range for each subnet. The availability_zone parameter specifies the availability zone where each subnet should be created. Note that you can create up to 20 subnets in a VPC, and each subnet must be associated with a different availability zone.

In this code, we are creating two subnets with CIDR blocks of 10.0.1.0/24 and 10.0.2.0/24 respectively, and associating them with availability zones us-east-1a and us-east-1b respectively.

The tags parameter is used to attach metadata to each subnet resource, in this case a name tag of "forassignment-subnet-1" and "forassignment-subnet-2" respectively.

Now what we want is to create two instances in previous two subnets. Let us install apache2 in them too.

What we will do for this is we will use previous code of creating instance which was created in resource.tf and add subnet details to it.

```
resource "aws_instance" "web_server-1" {
  ami           = "ami-0c55b159cbfafe1f0"  # Ubuntu 20.04 LTS
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.forassignment_subnet_1.id
  key_name      = "ec2-key"

  tags = {
    Name = "web-server-1"
  }

    connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("./ec2-key.pem")
    host        = self.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y apache2"
    ]
  }
}
```

We will copy paste this code again, this time just the subnet id will be second one.

```
resource "aws_instance" "web_server-2" {
  ami           = "ami-0c55b159cbfafe1f0"  # Ubuntu 20.04 LTS
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.forassignment_subnet_2.id
  key_name      = "ec2-key"

  tags = {
    Name = "web-server-2"
  }

    connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("./ec2-key.pem")
    host        = self.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y apache2"
    ]
  }
}
```

This will create another instance in the second subnet of VPC that we have created. Instance will use ubuntu AMI and install apache2 to it.

Now, we want to attach Security groups, internet gateway and network interface to the instance.

Let us create Security Groups first, then we will attach them to ec2 instances that we have created.

```
# Create a security group for the instances
resource "aws_security_group" "forassignment_sg" {
  name_prefix = "forassignment-sg"
  vpc_id      = aws_vpc.forassignment_vpc.id

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

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

  tags = {
    Name = "forassignment-sg"
  }
}
```

Above Terraform code creates an AWS security group resource named "forassignment_sg". It is associated with the VPC "aws_vpc.forassignment_vpc" created earlier. The security group allows inbound traffic on ports 22 and 80 for TCP protocol from all IP addresses, and allows all outbound traffic. It also has a name tag "forassignment-sg".

The name_prefix parameter is used to specify a prefix for the security group's name, allowing Terraform to generate a unique name for the resource.

The ingress block defines the inbound rules for the security group, while the egress block defines the outbound rules. The cidr_blocks parameter is used to specify the source or destination IP ranges for the traffic allowed by the security group.

This security group can be associated with instances to restrict network traffic to and from those instances.

Let us create an Internet Gateway.

```
# Create an internet gateway for the VPC
resource "aws_internet_gateway" "forassignment_igw" {
  vpc_id = aws_vpc.forassignment_vpc.id

  tags = {
    Name = "forassignment-igw"
  }
}
```

This Terraform code creates an AWS internet gateway resource named forassignment_igw that is associated with the forassignment_vpc VPC created earlier.

An internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in the VPC and the internet. It serves as the entry and exit point for traffic going in and out of the VPC.

The code sets the vpc_id attribute to the ID of the VPC resource created earlier using the aws_vpc resource block. Additionally, the code assigns a name tag to the internet gateway resource for easier identification in the AWS console.

Let us create a Network Interface now.

```
# Create a network interface for each instance
resource "aws_network_interface" "forassignment_1" {
  subnet_id       = aws_subnet.forassignment_subnet_1.id
  security_groups = [aws_security_group.forassignment_sg.id]

  tags = {
    Name = "forassignment-nic-1"
  }
}

resource "aws_network_interface" "forassignment_2" {
  subnet_id       = aws_subnet.forassignment_subnet_2.id
  security_groups = [aws_security_group.forassignment_sg.id]

  tags = {
    Name = "forassignment-nic-2"
  }
}
```

This Terraform code creates two network interfaces, aws_network_interface.forassignment_1 and aws_network_interface.forassignment_2, that are associated with the two subnets created earlier. Each network interface is associated with the security group created earlier aws_security_group.forassignment_sg. The subnet_id parameter specifies the ID of the subnet to which the network interface should be attached, while security_groups specifies the IDs of the security groups associated with the network interface.

The tags parameter is optional and allows you to assign metadata to the network interfaces. In this case, it assigns a name tag to each interface to make them easily identifiable

Now let us attach Security Group to previous resource of instance and network interface too.
We will add this field to the code.

```
  security_groups = [aws_security_group.forassignment_sg.id]
  network_interface {
    device_index          = 0
    network_interface_id = aws_network_interface.forassignment_1.id
  }
```

This code block is configuring an EC2 instance with a security group and a network interface.

The security_groups parameter is specifying the ID of the security group created earlier that will be associated with the instance.

The network_interface block is configuring the primary network interface of the instance. The device_index parameter is set to 0 to indicate that it is the primary network interface, and the network_interface_id parameter is set to the ID of the network interface created earlier that will be attached to the instance.

We also need to create route table to associate internet gateway to subents.

```
resource "aws_route" "forassignment_route" {
  route_table_id            = aws_route_table.forassignment_route_table.id
  destination_cidr_block    = "0.0.0.0/0"
  gateway_id                = aws_internet_gateway.forassignment_igw.id
  depends_on                = [aws_internet_gateway.forassignment_igw]
}

# Associate the subnet with the route table
resource "aws_route_table_association"
"forassignment_subnet_association-1" {
  subnet_id      = aws_subnet.forassignment_subnet_1.id
  route_table_id = aws_route_table.forassignment_route_table.id
}

resource "aws_route_table_association"
"forassignment_subnet_association-2" {
  subnet_id      = aws_subnet.forassignment_subnet_1.id
  route_table_id = aws_route_table.forassignment_route_table.id
}
```

This Terraform code creates a route in the route table to direct all traffic with a destination of "0.0.0.0/0" (i.e. all internet traffic) to the internet gateway created earlier, using the aws_route resource.
Then, it associates the subnets created earlier with the route table using the aws_route_table_association resource. The first aws_route_table_association associates aws_subnet.forassignment_subnet_1 with aws_route_table.forassignment_route_table, and the second aws_route_table_association associates aws_subnet.forassignment_subnet_2 with aws_route_table.forassignment_route_table.

By doing this, any traffic sent from instances launched in these subnets that is destined for the internet will be routed through the internet gateway.
Let us summarize the whole process.

We have provider.tf.

Provider.tf:

```
provider "aws" {
  region     = "us-east-1"
  access_key = "<accesskey>"
  secret_key = "<Secret access key>"
}
```

Resources.tf:

```
resource "aws_instance" "web_server-1" {
  ami           = "ami-0c55b159cbfafe1f0"  # Ubuntu 20.04 LTS
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.forassignment_subnet_1.id
  key_name      = "ec2-key"
  associate_public_ip_address = true

  tags = {
    Name = "web-server-1"
  }

  security_groups = [aws_security_group.forassignment_sg.id]

    connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("./ec2-key.pem")
    host        = self.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y apache2"
    ]
  }
```

```
}

resource "aws_instance" "web_server-2" {
  ami           = "ami-0c55b159cbfafe1f0"  # Ubuntu 20.04 LTS
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.forassignment_subnet_2.id
  key_name      = "ec2-key"
  associate_public_ip_address = true

  tags = {
    Name = "web-server-2"
  }
  security_groups = [aws_security_group.forassignment_sg.id]

    connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("./ec2-key.pem")
    host        = self.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y apache2"
    ]
  }
}
```

ec2-key.pem:

-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA7G577LEa4fFT4m55ewejHc0CoP6B/pE1GNRB9b51LdVYs9sO
sUbChIN7k0vGIwqrXj9vi9Pf/j5UDeetVM9X0J2WNLggj0IsABph51VYQ7NOQzzM
9QgtBMgK6c3ug6WVlOXPYW3h+vOe8G2NR15DjGRLaE4BHlcN0dJpVVl6wu0n8DfU
zlOctUuG0tOgN4QiK125R82ckKLD/4Rb9SlW8THW5DrUboZN9t8udG96se1FaDCQ
ZQFHXO5fAe3AhB80aVB+N38iszYy3waigYEzQ2yBxp78q+h7khZtL3Akd817lilJ
eQSrL0Jj1gsnDXZFeh6P8Jco3RCzpvckes+hlwIDAQABAoIBACSd/+dqjoVLwPgi
sAHT7vlglPWmNS/eAWE1Dnom4zMUBcblOGNQWDRBnbP0Wl12+49/N6FqURipQo9c
wEWQT0NqBKdplPFbCEGD1lHlf+QMUySqt/uDJqXE7ZvEXxpYfNAwv2uRfxxEuur1
rJnPCZsmtV8UJawzQ1qpwdKLB7rEf9dpKXm1YAAZd8tVVbDJmS9vqEZ/3l0ijdCd

-----END RSA PRIVATE KEY-----

Networking.tf:

```
# Create a VPC
resource "aws_vpc" "forassignment_vpc" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "forassignment-vpc"
  }
}

# Create two subnets in the VPC
resource "aws_subnet" "forassignment_subnet_1" {
  vpc_id     = aws_vpc.forassignment_vpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "forassignment-subnet-1"
  }
}

resource "aws_subnet" "forassignment_subnet_2" {
  vpc_id     = aws_vpc.forassignment_vpc.id
  cidr_block = "10.0.2.0/24"
  availability_zone = "us-east-1b"

  tags = {
    Name = "forassignment-subnet-2"
  }
}

# Create a security group for the instances
resource "aws_security_group" "forassignment_sg" {
  name_prefix = "forassignment-sg"
  vpc_id      = aws_vpc.forassignment_vpc.id

  ingress {
    from_port   = 22
```

```
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

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

  tags = {
    Name = "forassignment-sg"
  }
}

# Create an internet gateway for the VPC
resource "aws_internet_gateway" "forassignment_igw" {
  vpc_id = aws_vpc.forassignment_vpc.id

  tags = {
    Name = "forassignment-igw"
  }
}

# Create a network interface for each instance
resource "aws_network_interface" "forassignment_1" {
  subnet_id       = aws_subnet.forassignment_subnet_1.id
  security_groups = [aws_security_group.forassignment_sg.id]

  tags = {
    Name = "forassignment-nic-1"
```

```
  }
}

resource "aws_network_interface" "forassignment_2" {
  subnet_id       = aws_subnet.forassignment_subnet_2.id
  security_groups = [aws_security_group.forassignment_sg.id]

  tags = {
    Name = "forassignment-nic-2"
  }
}

# Create a route table for the subnet
resource "aws_route_table" "forassignment_route_table-1" {
  vpc_id = aws_vpc.forassignment_vpc.id

  tags = {
    Name = "forassignment-route-table"
  }
}

# Add a route to the internet gateway in the route table
resource "aws_route" "forassignment_route-1" {
  route_table_id            =
aws_route_table.forassignment_route_table-1.id
  destination_cidr_block    = "0.0.0.0/0"
  gateway_id                = aws_internet_gateway.forassignment_igw.id
  depends_on                = [aws_internet_gateway.forassignment_igw]
}

# Associate the subnet with the route table
resource "aws_route_table_association"
"forassignment_subnet_association-1" {
  subnet_id      = aws_subnet.forassignment_subnet_1.id
  route_table_id = aws_route_table.forassignment_route_table-1.id
}

# Create a route table for the subnet2
resource "aws_route_table" "forassignment_route_table-2" {
  vpc_id = aws_vpc.forassignment_vpc.id
```

```
  tags = {
    Name = "forassignment-route-table"
  }
}

# Add a route to the internet gateway in the route table
resource "aws_route" "forassignment_route-2" {
  route_table_id            =
aws_route_table.forassignment_route_table-2.id
  destination_cidr_block    = "0.0.0.0/0"
  gateway_id                = aws_internet_gateway.forassignment_igw.id
  depends_on                = [aws_internet_gateway.forassignment_igw]
}

# Associate the subnet with the route table
resource "aws_route_table_association"
"forassignment_subnet_association-2" {
  subnet_id      = aws_subnet.forassignment_subnet_2.id
  route_table_id = aws_route_table.forassignment_route_table-2.id
}
```

Let us run: Terraform Plan

```
C:\Users\anpan\Downloads\terraform_1.4.5_windows_amd64>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_instance.web_server-1 will be created
  + resource "aws_instance" "web_server-1" {
      + ami                                  = "ami-0c55b159cbfafe1f0"
      + 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)
      + 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_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = "ec2-key"
      + 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)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "web-server-1"
        }
      + tags_all                             = {
          + "Name" = "web-server-1"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)

      + network_interface {
          + delete_on_termination = false
          + device_index          = 0
          + network_card_index    = 0
          + network_interface_id  = (known after apply)
        }
    }

  # aws_instance.web_server-2 will be created
  + resource "aws_instance" "web_server-2" {
```

```
# aws_instance.web_server-2 will be created
+ resource "aws_instance" "web_server-2" {
    + ami                                  = "ami-0c55b159cbfafe1f0"
    + 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)
    + 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_state                       = (known after apply)
    + instance_type                        = "t2.micro"
    + ipv6_address_count                   = (known after apply)
    + ipv6_addresses                       = (known after apply)
    + key_name                             = "ec2-key"
    + 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)
    + public_ip                            = (known after apply)
    + secondary_private_ips                = (known after apply)
    + security_groups                      = (known after apply)
    + source_dest_check                    = true
    + subnet_id                            = (known after apply)
    + tags                                 = {
        + "Name" = "web-server-2"
      }
    + tags_all                             = {
        + "Name" = "web-server-2"
      }
    + tenancy                              = (known after apply)
    + user_data                            = (known after apply)
    + user_data_base64                     = (known after apply)
    + user_data_replace_on_change          = false
    + vpc_security_group_ids               = (known after apply)
  }

# aws_internet_gateway.forassignment_igw will be created
+ resource "aws_internet_gateway" "forassignment_igw" {
    + arn      = (known after apply)
    + id       = (known after apply)
    + owner_id = (known after apply)
    + tags     = {
        + "Name" = "forassignment-igw"
      }
    + tags_all = {
        + "Name" = "forassignment-igw"
      }
    + vpc_id   = (known after apply)
  }

# aws_network_interface.forassignment_1 will be created
+ resource "aws_network_interface" "forassignment_1" {
```

```
# aws_network_interface.forassignment_1 will be created
+ resource "aws_network_interface" "forassignment_1" {
    + arn                       = (known after apply)
    + id                        = (known after apply)
    + interface_type            = (known after apply)
    + ipv4_prefix_count         = (known after apply)
    + ipv4_prefixes             = (known after apply)
    + ipv6_address_count        = (known after apply)
    + ipv6_address_list         = (known after apply)
    + ipv6_address_list_enabled = false
    + ipv6_addresses            = (known after apply)
    + ipv6_prefix_count         = (known after apply)
    + ipv6_prefixes             = (known after apply)
    + mac_address               = (known after apply)
    + outpost_arn               = (known after apply)
    + owner_id                  = (known after apply)
    + private_dns_name          = (known after apply)
    + private_ip                = (known after apply)
    + private_ip_list           = (known after apply)
    + private_ip_list_enabled   = false
    + private_ips               = (known after apply)
    + private_ips_count         = (known after apply)
    + security_groups           = (known after apply)
    + source_dest_check         = true
    + subnet_id                 = (known after apply)
    + tags                      = {
        + "Name" = "forassignment-nic-1"
      }
    + tags_all                  = {
        + "Name" = "forassignment-nic-1"
      }
  }

# aws_network_interface.forassignment_2 will be created
+ resource "aws_network_interface" "forassignment_2" {
    + arn                       = (known after apply)
    + id                        = (known after apply)
    + interface_type            = (known after apply)
    + ipv4_prefix_count         = (known after apply)
    + ipv4_prefixes             = (known after apply)
    + ipv6_address_count        = (known after apply)
    + ipv6_address_list         = (known after apply)
    + ipv6_address_list_enabled = false
    + ipv6_addresses            = (known after apply)
    + ipv6_prefix_count         = (known after apply)
    + ipv6_prefixes             = (known after apply)
    + mac_address               = (known after apply)
    + outpost_arn               = (known after apply)
    + owner_id                  = (known after apply)
    + private_dns_name          = (known after apply)
    + private_ip                = (known after apply)
    + private_ip_list           = (known after apply)
    + private_ip_list_enabled   = false
    + private_ips               = (known after apply)
    + private_ips_count         = (known after apply)
    + security_groups           = (known after apply)
    + source_dest_check         = true
    + subnet_id                 = (known after apply)
    + tags                      = {
        + "Name" = "forassignment-nic-2"
      }
    + tags_all                  = {
        + "Name" = "forassignment-nic-2"
      }
  }
```

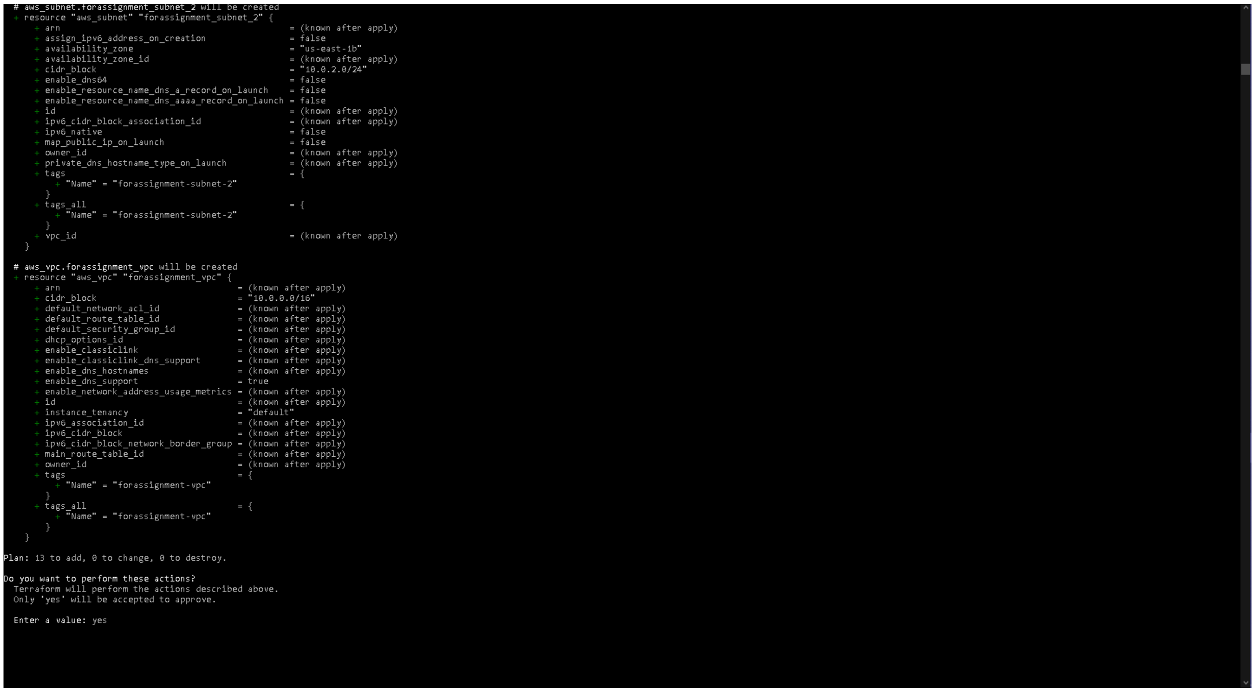
```
# aws_network_interface.forassignment_2 will be created
+ resource "aws_network_interface" "forassignment_2" {
    + arn                       = (known after apply)
    + id                        = (known after apply)
    + interface_type            = (known after apply)
    + ipv4_prefix_count         = (known after apply)
    + ipv4_prefixes             = (known after apply)
    + ipv6_address_count        = (known after apply)
    + ipv6_address_list         = (known after apply)
    + ipv6_address_list_enabled = false
    + ipv6_addresses            = (known after apply)
    + ipv6_prefix_count         = (known after apply)
    + ipv6_prefixes             = (known after apply)
    + mac_address               = (known after apply)
    + outpost_arn               = (known after apply)
    + owner_id                  = (known after apply)
    + private_dns_name          = (known after apply)
    + private_ip                = (known after apply)
    + private_ip_list           = (known after apply)
    + private_ip_list_enabled   = false
    + private_ips               = (known after apply)
    + private_ips_count         = (known after apply)
    + security_groups           = (known after apply)
    + source_dest_check         = true
    + subnet_id                 = (known after apply)
    + tags                      = {
        + "Name" = "forassignment-nic-2"
      }
    + tags_all                  = {
        + "Name" = "forassignment-nic-2"
      }
  }

# aws_route.forassignment_route will be created
+ resource "aws_route" "forassignment_route" {
    + destination_cidr_block = "0.0.0.0/0"
    + gateway_id             = (known after apply)
    + id                     = (known after apply)
    + instance_id            = (known after apply)
    + instance_owner_id      = (known after apply)
    + network_interface_id   = (known after apply)
    + origin                 = (known after apply)
    + route_table_id         = (known after apply)
    + state                  = (known after apply)
  }

# aws_route_table.forassignment_route_table will be created
+ resource "aws_route_table" "forassignment_route_table" {
    + arn              = (known after apply)
    + id               = (known after apply)
    + owner_id         = (known after apply)
    + propagating_vgws = (known after apply)
    + route            = (known after apply)
    + tags             = {
        + "Name" = "forassignment-route-table"
      }
    + tags_all         = {
        + "Name" = "forassignment-route-table"
      }
    + vpc_id           = (known after apply)
  }

# aws_route_table_association.forassignment_subnet_association-1 will be created
+ resource "aws_route_table_association" "forassignment_subnet_association-1" {
    + id             = (known after apply)
```

```
# aws_route.forassignment_route will be created
+ resource "aws_route" "forassignment_route" {
    + destination_cidr_block = "0.0.0.0/0"
    + gateway_id             = (known after apply)
    + id                     = (known after apply)
    + instance_id            = (known after apply)
    + instance_owner_id      = (known after apply)
    + network_interface_id   = (known after apply)
    + origin                 = (known after apply)
    + route_table_id         = (known after apply)
    + state                  = (known after apply)
  }

# aws_route_table.forassignment_route_table will be created
+ resource "aws_route_table" "forassignment_route_table" {
    + arn              = (known after apply)
    + id               = (known after apply)
    + owner_id         = (known after apply)
    + propagating_vgws = (known after apply)
    + route            = (known after apply)
    + tags             = {
        + "Name" = "forassignment-route-table"
      }
    + tags_all         = {
        + "Name" = "forassignment-route-table"
      }
    + vpc_id           = (known after apply)
  }

# aws_route_table_association.forassignment_subnet_association-1 will be created
+ resource "aws_route_table_association" "forassignment_subnet_association-1" {
    + id             = (known after apply)
    + route_table_id = (known after apply)
    + subnet_id      = (known after apply)
  }

# aws_route_table_association.forassignment_subnet_association-2 will be created
+ resource "aws_route_table_association" "forassignment_subnet_association-2" {
    + id             = (known after apply)
    + route_table_id = (known after apply)
    + subnet_id      = (known after apply)
  }

# aws_security_group.forassignment_sg will be created
+ resource "aws_security_group" "forassignment_sg" {
    + arn                    = (known after apply)
    + description            = "Managed by Terraform"
    + egress                 = [
        + {
            + cidr_blocks      = [
                + "0.0.0.0/0",
              ]
            + description      = ""
            + from_port        = 0
            + ipv6_cidr_blocks = []
            + prefix_list_ids  = []
            + protocol         = "-1"
            + security_groups  = []
            + self             = false
            + to_port          = 0
          },
      ]
    + id                     = (known after apply)
    + ingress                = [
        + {
```

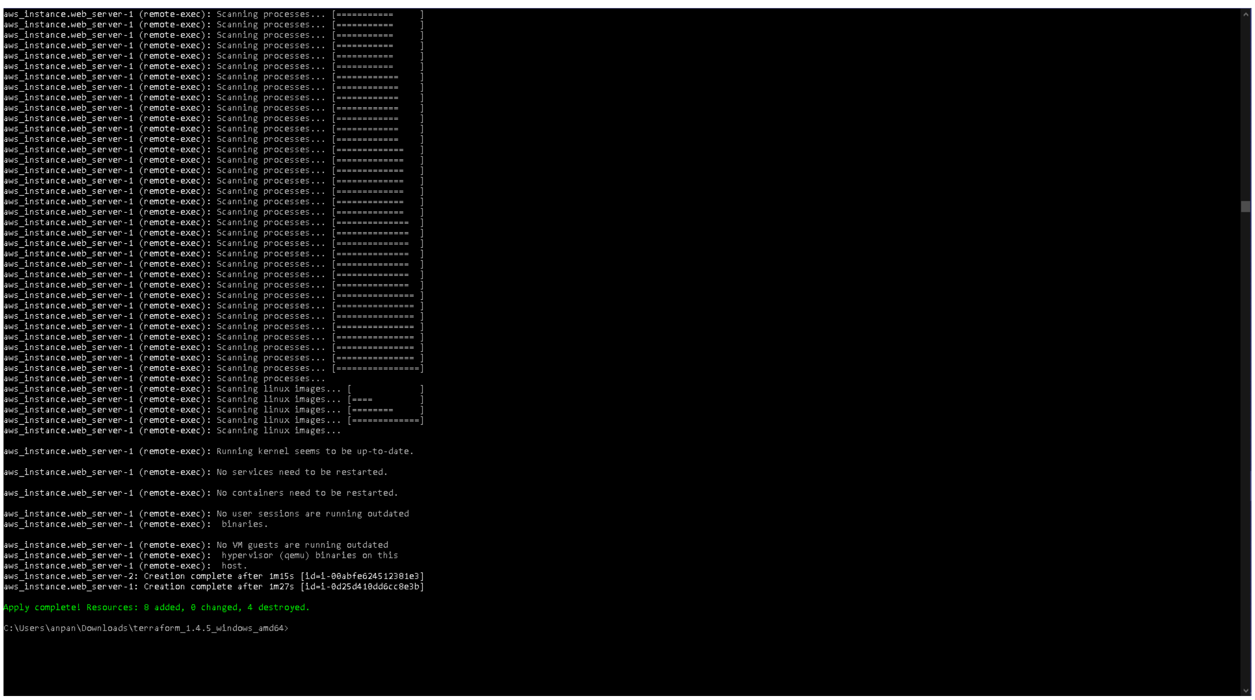
```
    + id                     = (known after apply)
    + ingress                = [
        + {
            + cidr_blocks      = [
                + "0.0.0.0/0",
              ]
            + description      = ""
            + from_port        = 22
            + ipv6_cidr_blocks = []
            + prefix_list_ids  = []
            + protocol         = "tcp"
            + security_groups  = []
            + self             = false
            + to_port          = 22
          },
        + {
            + cidr_blocks      = [
                + "0.0.0.0/0",
              ]
            + description      = ""
            + from_port        = 80
            + ipv6_cidr_blocks = []
            + prefix_list_ids  = []
            + protocol         = "tcp"
            + security_groups  = []
            + self             = false
            + to_port          = 80
          },
      ]
    + name                   = (known after apply)
    + name_prefix            = "forassignment-sg"
    + owner_id               = (known after apply)
    + revoke_rules_on_delete = false
    + tags                   = {
        + "Name" = "forassignment-sg"
      }
    + tags_all               = {
        + "Name" = "forassignment-sg"
      }
    + vpc_id                 = (known after apply)
  }

# aws_subnet.forassignment_subnet_1 will be created
+ resource "aws_subnet" "forassignment_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                        = false
    + owner_id                                       = (known after apply)
    + private_dns_hostname_type_on_launch            = (known after apply)
    + tags                                           = {
        + "Name" = "forassignment-subnet-1"
      }
    + tags_all                                       = {
        + "Name" = "forassignment-subnet-1"
      }
    + vpc_id                                         = (known after apply)
  }
```

```
# aws_subnet.forassignment_subnet_2 will be created
+ resource "aws_subnet" "forassignment_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                        = false
    + owner_id                                       = (known after apply)
    + private_dns_hostname_type_on_launch            = (known after apply)
    + tags                                           = {
        + "Name" = "forassignment-subnet-2"
      }
    + tags_all                                       = {
        + "Name" = "forassignment-subnet-2"
      }
    + vpc_id                                         = (known after apply)
  }

# aws_vpc.forassignment_vpc will be created
+ resource "aws_vpc" "forassignment_vpc" {
    + arn                                  = (known after apply)
    + cidr_block                           = "10.0.0.0/16"
    + 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_classiclink                   = (known after apply)
    + enable_classiclink_dns_support       = (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" = "forassignment-vpc"
      }
    + tags_all                             = {
        + "Name" = "forassignment-vpc"
      }
  }

Plan: 13 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.

C:\Users\anpan\Downloads\terraform_1.4.5_windows_amd64>
```

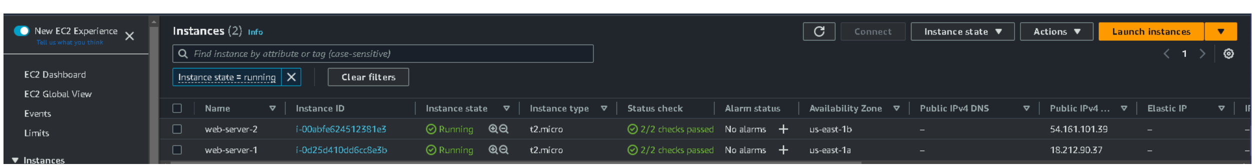
Run: terraform apply
Enter “yes”



The code ran successfully.

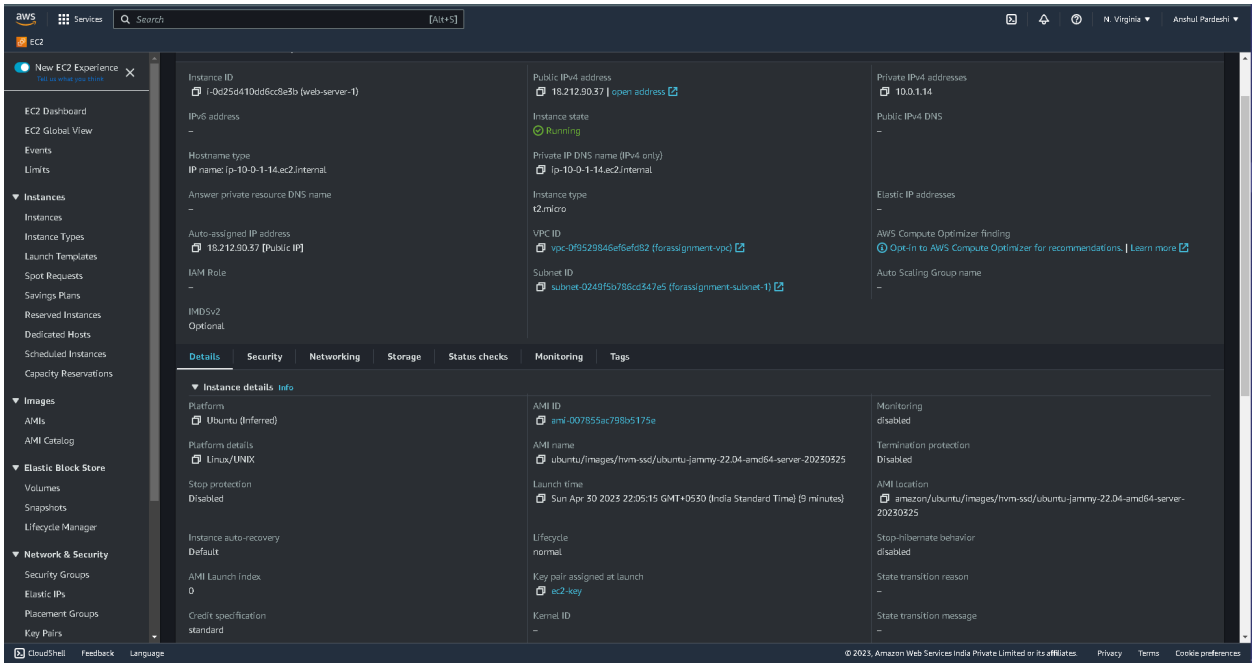


Two Instances are created.

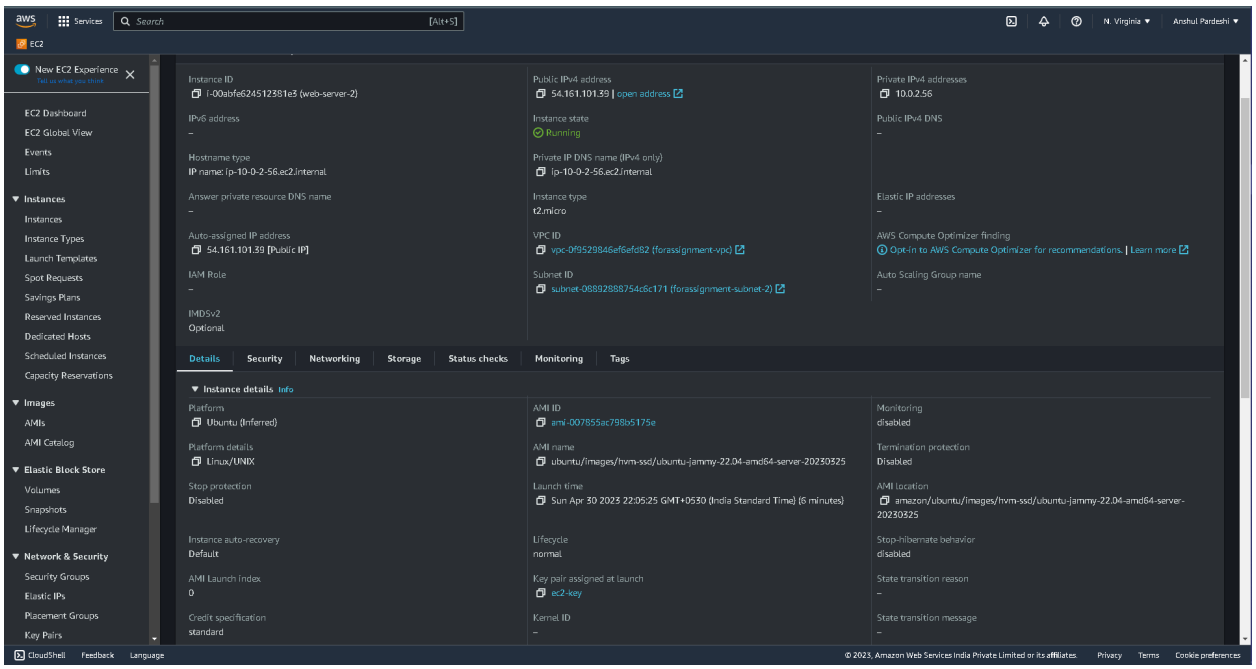


They are in the same subnet and VPC that we intended through code.

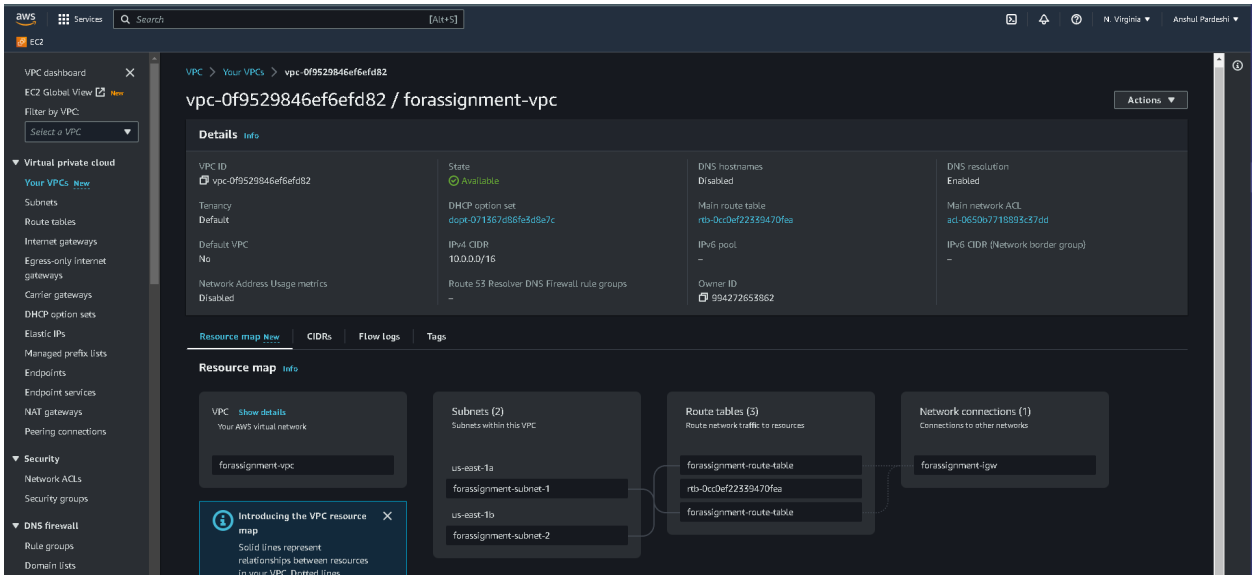
Instance-1



Instance-2



The VPC, subnet and routes:



Let us copy the public ip of both instances and paste it in browser to see if apache2 is working.

Instance-1

Instance-2


Not secure | 54.161.101.39
Apache2 Default Page
Ubuntu
It works!
This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should replace this file (located at /var/www/html/index.html) before continuing to operate your HTTP server.
If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.
Configuration Overview
Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is fully documented in /usr/share/doc/apache2/README.Debian.gz. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the manual if the apache2-doc package was installed on this server.
The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:
/etc/apache2/
|-- apache2.conf
| `-- ports.conf
|-- mods-enabled
| |-- *.load
| `-- *.conf
|-- conf-enabled
| `-- *.conf
|-- sites-enabled
| `-- *.conf
apache2.conf is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
ports.conf is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
Configuration files in the mods-enabled/, conf-enabled/ and sites-enabled/ directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers a2enmod, a2dismod, a2ensite, a2dissite, and a2enconf, a2disconf . See their respective man pages for detailed information.
The binary is called apache2 and is managed using systemd, so to start/stop the service use systemctl start apache2 and systemctl stop apache2, and use systemctl status apache2 and journalctl -u apache2 to check status. system and apache2ctl can also be used for service