Module 8: Terraform Assignment-4

Tasks To Be Performed:

1. Destroy the previous deployments

2. Create a VPC with the required components using Terraform

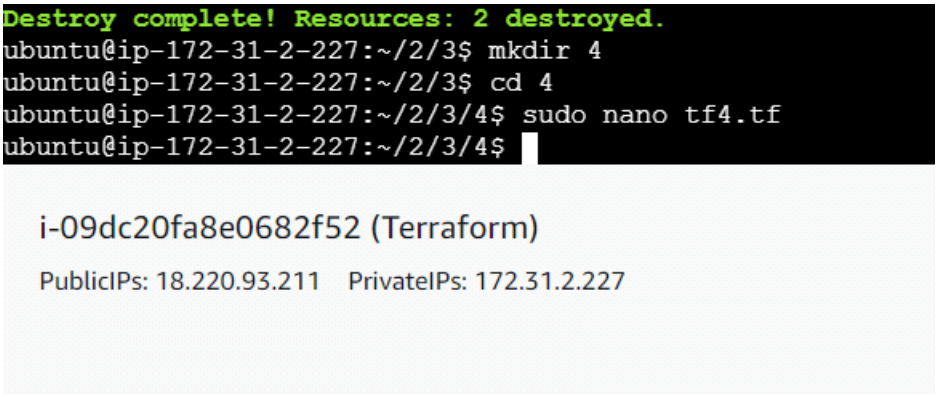
3. Deploy an EC2 instance inside the VPC

Solution:

Step 1: Destroy the Previous Deployments

Navigate to the directory containing your Terraform configuration files and run the following command to destroy the previous deployments:

terraform destroy



Step 2: Update the Terraform Configuration

Specify the provider

```
provider "aws" {
  region = "us-east-2"  # Specify your desired region
  access_key = " "
  secret_key = " "
}

# Create a VPC
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "my_vpc"
  }
}

# Create a subnet
resource "aws_subnet" "my_subnet" {
  vpc_id            = aws_vpc.my_vpc.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "us-east-2a"  # Specify your desired availability zone
  tags = {
    Name = "my_subnet"
  }
}
```

```
# Create an Internet Gateway
resource "aws_internet_gateway" "my_igw" {
  vpc_id = aws_vpc.my_vpc.id
  tags = {
    Name = "my_igw"
  }
}

# Create a route table
resource "aws_route_table" "my_route_table" {
  vpc_id = aws_vpc.my_vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.my_igw.id
  }
  tags = {
    Name = "my_route_table"
  }
}

# Associate the route table with the subnet
resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.my_subnet.id
  route_table_id = aws_route_table.my_route_table.id
```

```
}

# Create a security group
resource "aws_security_group" "my_sg" {
  vpc_id = aws_vpc.my_vpc.id

  ingress {
    from_port   = 22
    to_port     = 22
    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 = "my_sg"
  }
}
```

```
# Create an EC2 instance
resource "aws_instance" "my_instance" {
  ami                    = "ami-09040d770ffe2224f"  # Updated AMI ID
  instance_type          = "t2.micro"
  subnet_id              = aws_subnet.my_subnet.id
  vpc_security_group_ids = [aws_security_group.my_sg.id]

  tags = {
    Name = "my_instance"
  }
}

# Output the instance ID and public IP
output "instance_id" {
  value = aws_instance.my_instance.id
}

output "instance_public_ip" {
  value = aws_instance.my_instance.public_ip
}
```

```
# Create a VPC
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "my_vpc"
  }
}

# Create a subnet
resource "aws_subnet" "my_subnet" {
  vpc_id            = aws_vpc.my_vpc.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "us-east-2a"  # Specify your desired availability zone
  tags = {
    Name = "my_subnet"
  }
}
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify

i-09dc20fa8e0682f52 (Terraform)

PublicIPs: 18.220.93.211 PrivateIPs: 172.31.2.227

```
# Create an Internet Gateway
resource "aws_internet_gateway" "my_igw" {
  vpc_id = aws_vpc.my_vpc.id
  tags = {
    Name = "my_igw"
  }
}
```

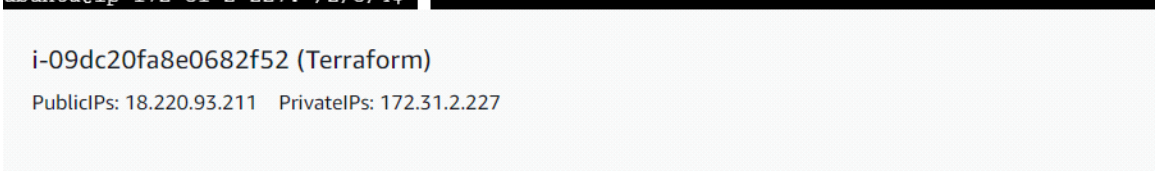
```
  GNU nano 7.2
# Create a route table
resource "aws_route_table" "my_route_table" {
  vpc_id = aws_vpc.my_vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.my_igw.id
  }
  tags = {
    Name = "my_route_table"
  }
}

# Associate the route table with the subnet
resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.my_subnet.id
  route_table_id = aws_route_table.my_route_table.id
}
```

```
# Create a security group
resource "aws_security_group" "my_sg" {
  vpc_id = aws_vpc.my_vpc.id

  ingress {
    from_port   = 22
    to_port     = 22
    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 = "my_sg"
  }
}
```

```
# Create an EC2 instance
resource "aws_instance" "my_instance" {
  ami                    = "ami-09040d770ffe2224f"  # Updated AMI ID
  instance_type          = "t2.micro"
  subnet_id              = aws_subnet.my_subnet.id
  vpc_security_group_ids = [aws_security_group.my_sg.id]

  tags = {
    Name = "my_instance"
  }
}

# Output the instance ID and public IP
output "instance_id" {
  value = aws_instance.my_instance.id
}

output "instance_public_ip" {
  value = aws_instance.my_instance.public_ip
}

^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute
```

Step 3: Initialize Terraform

```
ubuntu@ip-172-31-2-227:~/2/3/4$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.55.0...
- Installed hashicorp/aws v5.55.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
ubuntu@ip-172-31-2-227:~/2/3/4$
```



Step 4: Apply the New Configuration

Step 5: Verify the New Deployment