Infrastructure as Code (IaC) Using Terraform on AWS

This project provides the Automation of AWS Infrastructure using Terraform software. I have been asked to build an infrastructure safely and efficiently.

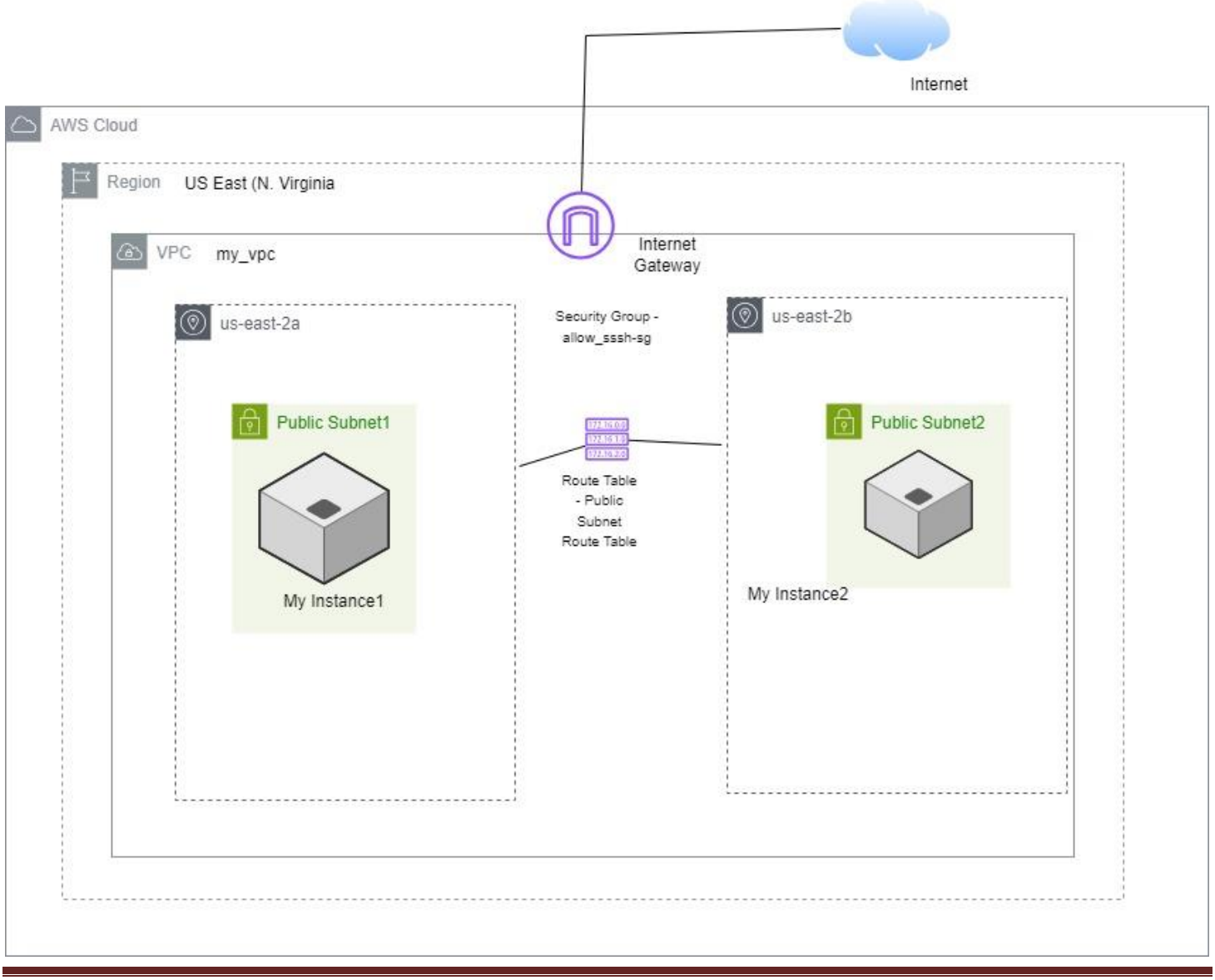
The company Requirements:

1. Use AWS cloud Provider and the software to be installed is Apache2
2. Use Ubuntu AMI

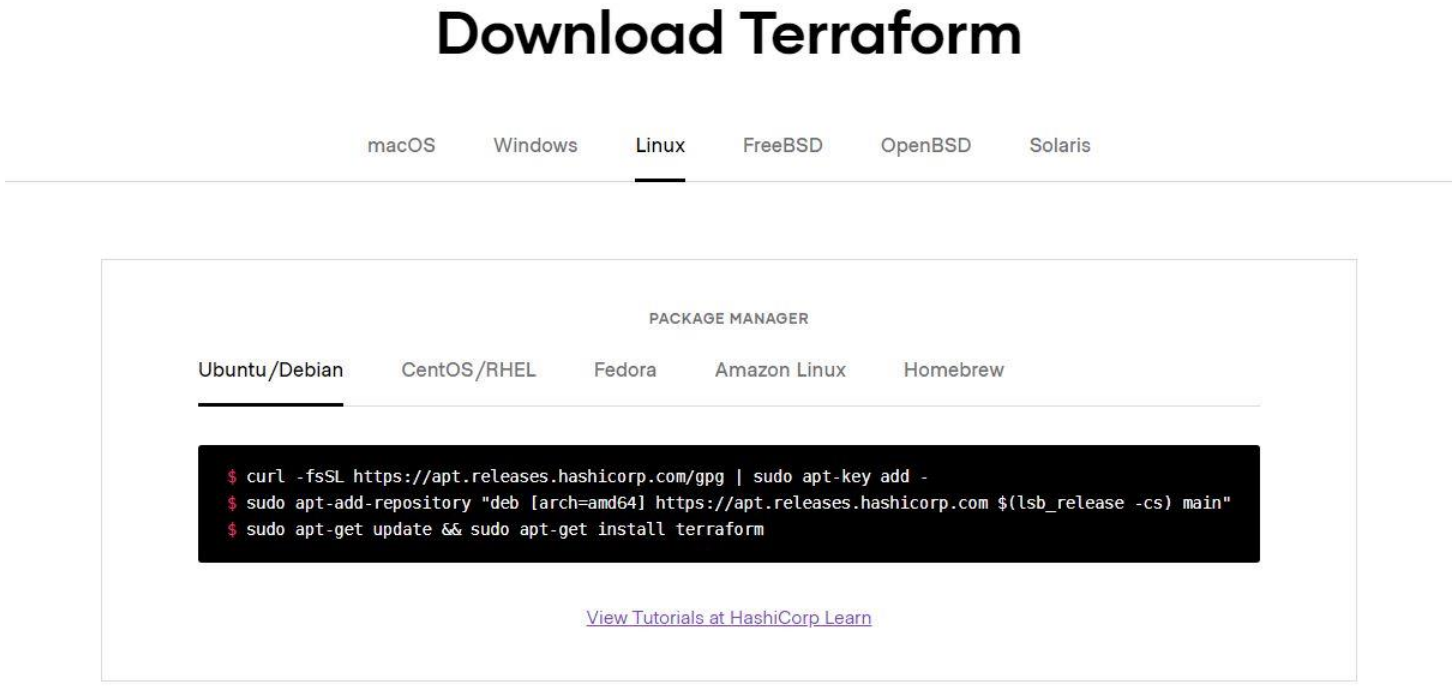
The company wants the Architecture to have the following services:

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

First I need to develop the architecture diagram



Step 1. Create and Install Terraform on AWS EC2 instance (Bastion Host). Latest download information for Linux Ubuntu , Mac, Windows, etc. is available at: [Install | Terraform | HashiCorp Developer]



Verify Install:

```
ubuntu@ip-172-31-44-68:~$ terraform --version
Terraform v1.2.4
on linux_amd64
ubuntu@ip-172-31-44-68:~$
```

Terraform installed correctly.

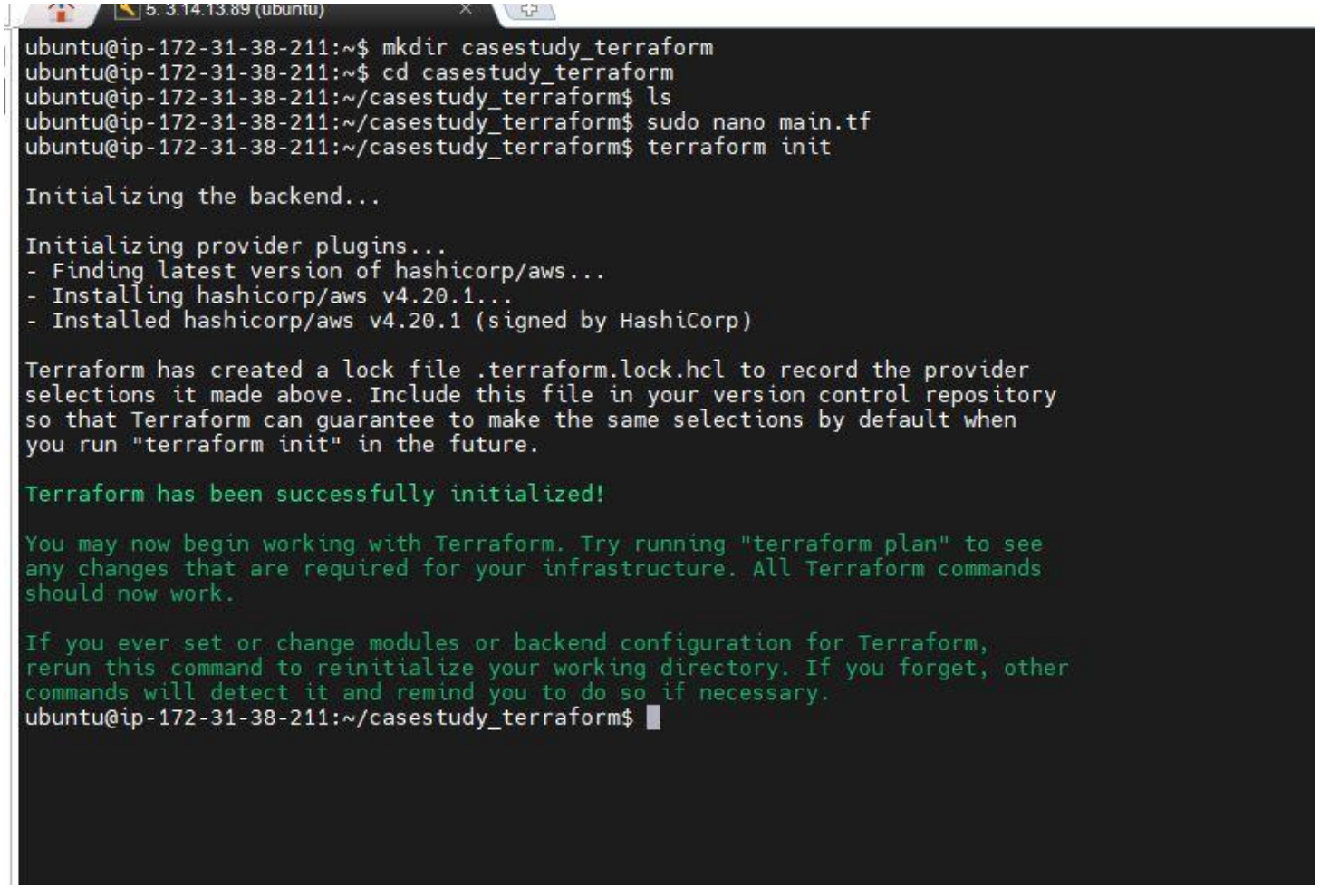
Step 2.

Create dir (**mkdir casestudy_terraform**) Initialize provider plugins using command **terraform init** and the below file (See below Linux screenshot)

main.tf file: (.tf file extension means it's a terraform file)

sudo nano main.tf

```
provider "aws"{

region = "us-east-2"

}
```

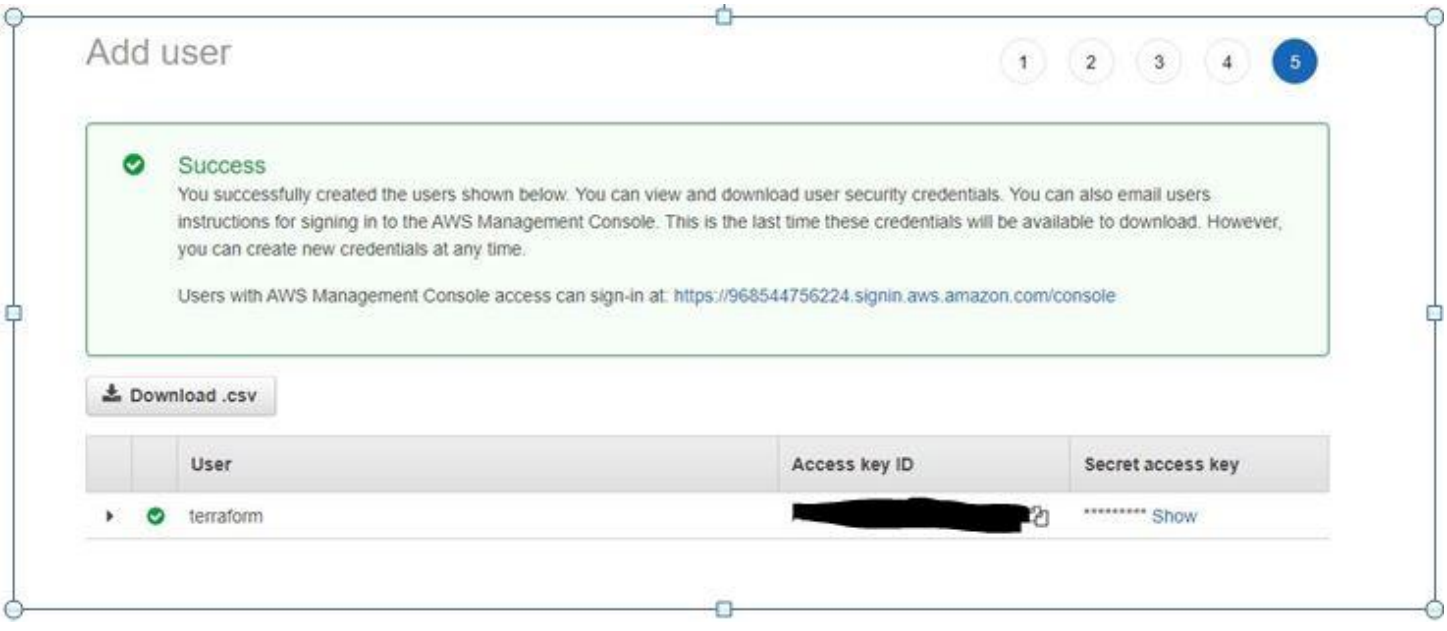


Step 3.

Add IAM Role and get access key ID and secret access key (I had demonstrated how to create an IAM Role is a previous post)

User: terraform

Provide AdministratorAccess



Step 4.

Create an EC2 instance – edit main.tf

To browse providers supported by Terraform go to: Browse Providers | Terraform Registry

Here you will see AWS, Azure, Google Cloud Platform, Kubernetes, Alibaba, and Oracle Cloud Infrastructure.

For the purposes of this project select on AWS then click on Documentation. Here you will see a resource page and AWS document for AWS services.

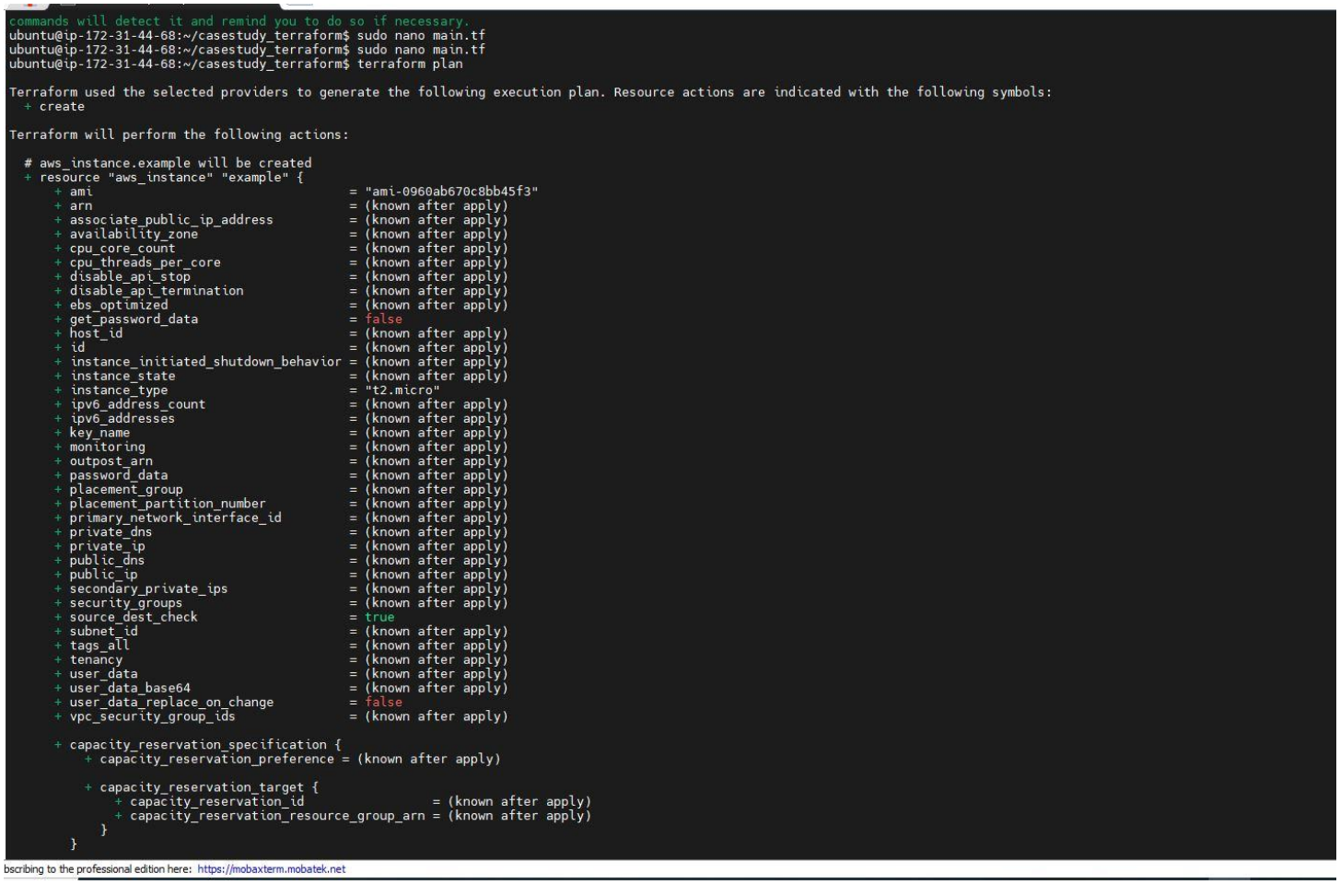
We are now creating a resource called an AWS EC2 instance. If you remember when creating a EC2 instance **manually** we need the following as inputs: Choose AMI (Amazon Machine Image), instance type, configure settings, add storage, add tags, and configure security group. That's a lot of work. Below I will show you how you can incorporate all of this into a terraform script.

So to do this edit main.tf and include the below HCL (HashiCorp Configuration Language) code by using the command:

sudo nano main.tf

```
provider "aws"{
region = "us-east-2"
access_key = "[redacted]"
secret_key = "[redacted]"
}

resource "aws_instance"  "example" {

 ami = "ami-0960ab670c8bb45f3"
 instance_type = "t2.micro"
  tags = {

    "Name" = "terraform-instance"

 }

}
```

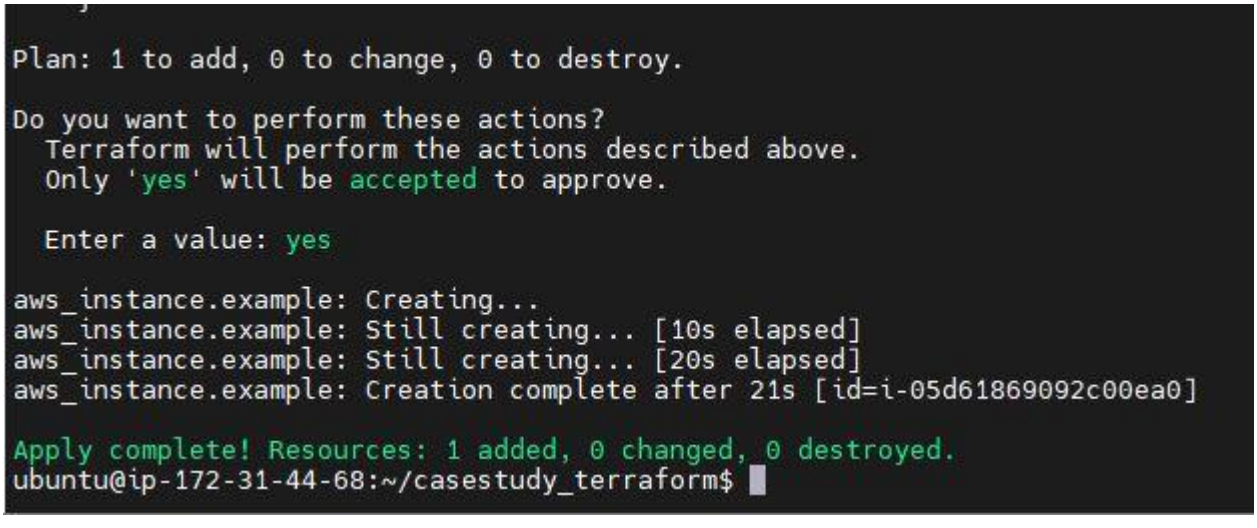
At command line run the following command: **terraform plan**



Step 5.

Then run the command:

terraform apply – this will apply all of your changes in the script and an AWS instance was successfully created



Step 6. Install Apache2 (install.sh) and include in main.tf file

Command: **sudo nano install.sh**

#!/bin/sh
sudo apt-get update
sudo apt-get -y install apache2

Command: **sudo nano main.tf** (Here you are editing your main.tf file to include your user_data (install.sh) that you wrote)

```
provider "aws"{
region = "us-east-2"
access_key = "[redacted]"
secret_key = "[redacted]"
}

resource "aws_instance"  "terraform-instance" {

 ami = "ami-0960ab670c8bb45f3"
 instance_type = "t2.micro"
 tags = {

  "Name" = "terraform-instance"
 }

 user_data = file("./install.sh")

}
```

Step 7.

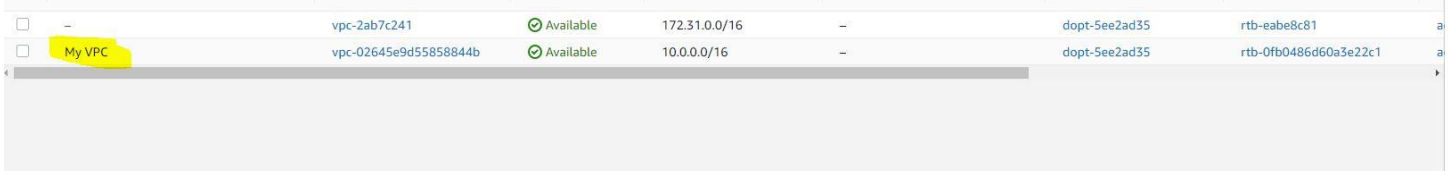
Now you will add:
Input: VPC to main.tf
Commands:
sudo nano main.tf

```
provider "aws"{
region = "us-east-2"
access_key = "[redacted]"
secret_key = "[redacted]"
}

# declare a VPC
resource "aws_vpc" "my_vpc" {
 cidr_block      = "10.0.0.0/16"
 enable_dns_hostnames = true

 tags = {
  Name = "My VPC"
 }
}
```

Run Commands:

terraform plan
terraform apply

Output: MyVPC has been created successfully.



Step 8.

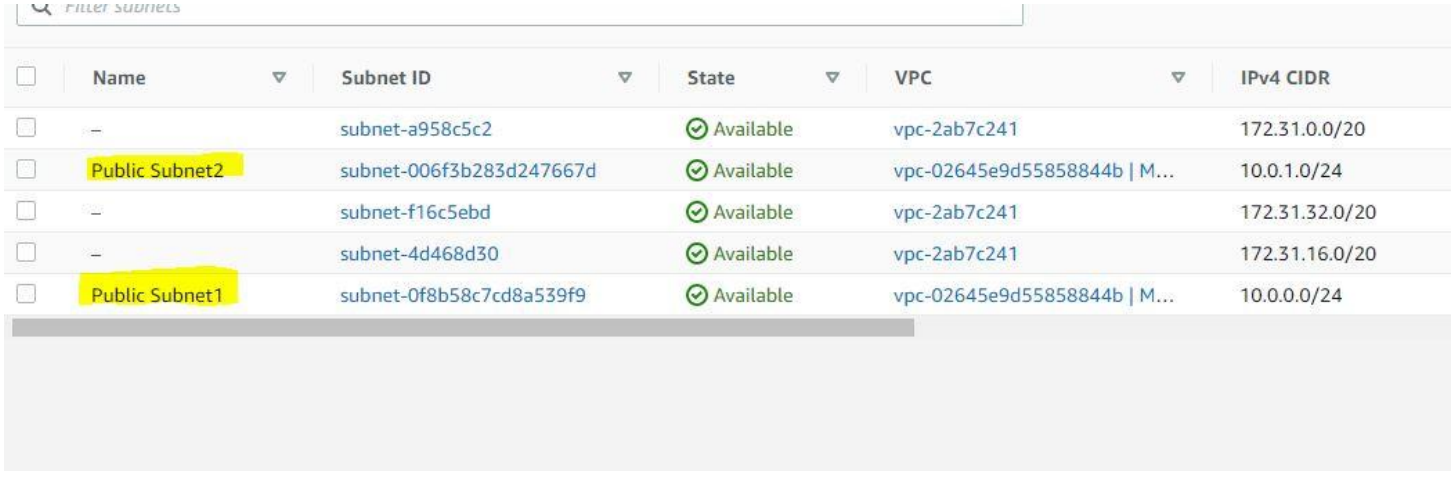
Now you will add:
Input: Public Subnets to main.tf file
Commands:
sudo nano main.tf

```
resource "aws_subnet" "public1" {
 vpc_id     = aws_vpc.my_vpc.id
 cidr_block = "10.0.0.0/24"
 availability_zone = "us-east-2a"

 tags = {
  Name = "Public Subnet1"
 }
}

resource "aws_subnet" "public2" {
 vpc_id     = aws_vpc.my_vpc.id
 cidr_block = "10.0.1.0/24"
 availability_zone = "us-east-2b"

 tags = {
  Name = "Public Subnet2"
 }
}
```

Run Commands:

terraform plan
terraform apply

Output: Two Public Subnets were created successfully.



Name	Subnet ID	State	VPC	IPv4 CIDR
–	subnet-a958c5c2	Available	vpc-2ab7c241	172.31.0.0/20
Public Subnet2	subnet-006f3b283d247667d	Available	vpc-02645e9d55858844b | M...	10.0.1.0/24
–	subnet-f16c5ebd	Available	vpc-2ab7c241	172.31.32.0/20
–	subnet-4d468d30	Available	vpc-2ab7c241	172.31.16.0/20
Public Subnet1	subnet-0f8b58c7cd8a539f9	Available	vpc-02645e9d55858844b | M...	10.0.0.0/24

Step 9.

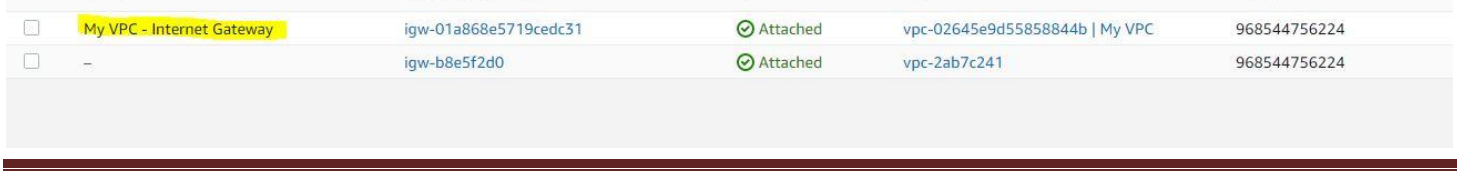
Now you will add:
Input: Add internet gateway
Commands:
sudo nano main.tf

```
resource "aws_internet_gateway" "my_vpc_igw" {
 vpc_id = aws_vpc.my_vpc.id

 tags = {
  Name = "My VPC - Internet Gateway"
 }
}
```

Run Commands:

terraform plan
terraform apply

Output: Internet Gateway was created successfully



My VPC - Internet Gateway	igw-01a868e5719cedc31	Attached	vpc-02645e9d55858844b | My VPC	968544756224
–	igw-b8e5f2d0	Attached	vpc-2ab7c241	968544756224

Step 10.

Now you will add:
Input: Add route table
Commands:
sudo nano main.tf

```
resource "aws_route_table" "my_vpc_us_east_2a_public" {
  vpc_id = aws_vpc.my_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.my_vpc_igw.id
  }

  tags = {
    Name = "Public Subnet Route Table"
  }
}
```

Run Commands:

terraform plan
terraform apply

Output: Public Subnet Route Table was created successfully

	Name	Route table ID	Explicit subnet associat...	Edge associations	Main	VPC	Owner ID
☐	–	rtb-0fb0486d60a3e22c1	–	–	Yes	vpc-02645e9d55858844b | M...	968544756224
☐	Public Subnet Route Table	rtb-0d65123a561fc0f9d	–	–	No	vpc-02645e9d55858844b | M...	968544756224
☐	–	rtb-eabe8c81	–	–	Yes	vpc-2ab7c241	968544756224

Step 11.

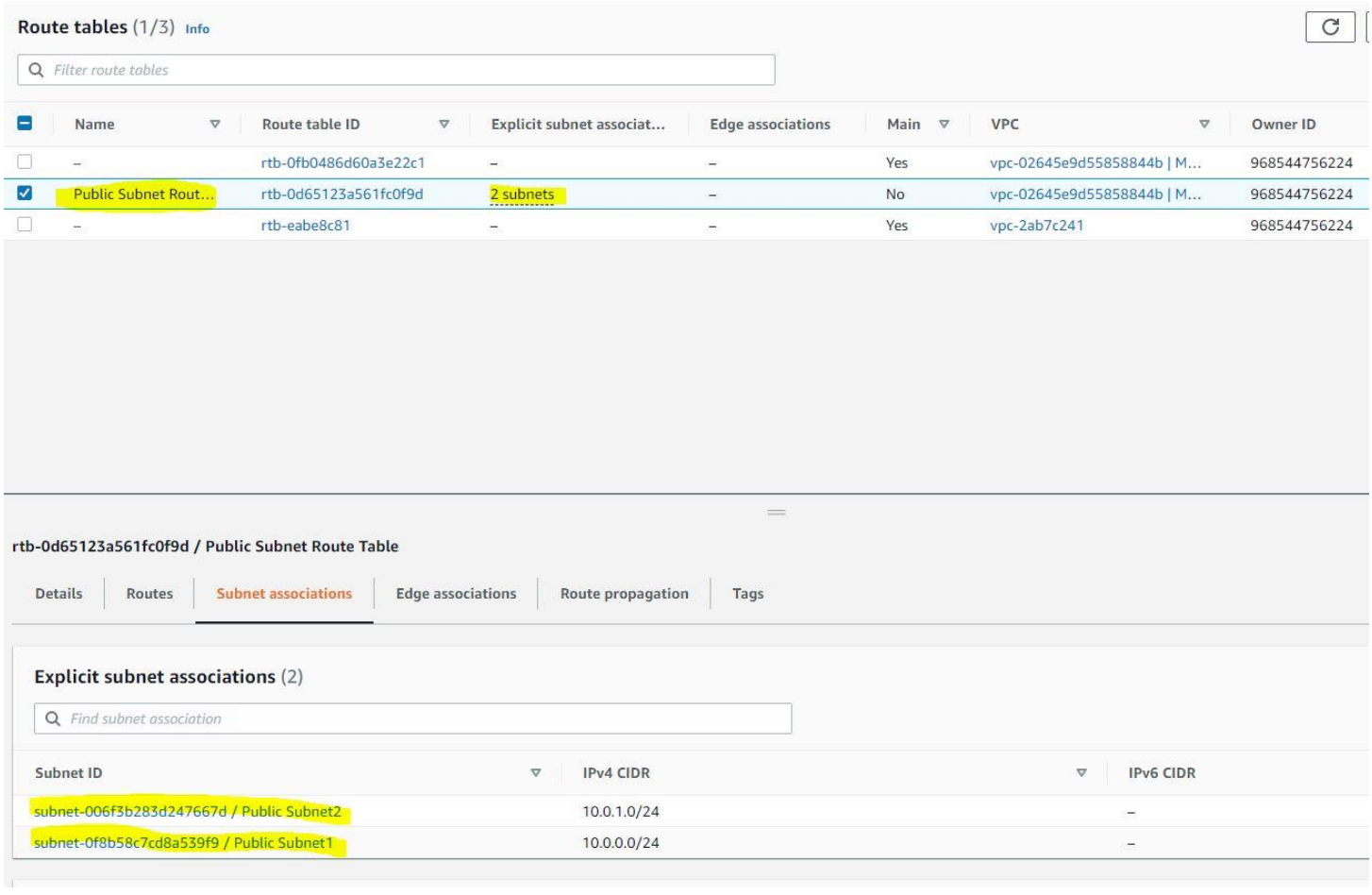
Now you will add:
Input: Subnet Associations
Commands:
sudo nano main.tf

```
resource "aws_route_table_association" "my_vpc_us_east_2a_public1" {
   subnet_id = aws_subnet.public1.id
   route_table_id = aws_route_table.my_vpc_us_east_2a_public.id
}

resource "aws_route_table_association" "my_vpc_us_east_2a_public2" {
   subnet_id = aws_subnet.public2.id
   route_table_id = aws_route_table.my_vpc_us_east_2a_public.id
}
```

Run Commands:

terraform plan
terraform apply

Output: Subnet Associations were created successfully and associations were made with subnets.



Step 12.

Now you will add:
Input: Attach security groups
Commands:
sudo nano main.tf

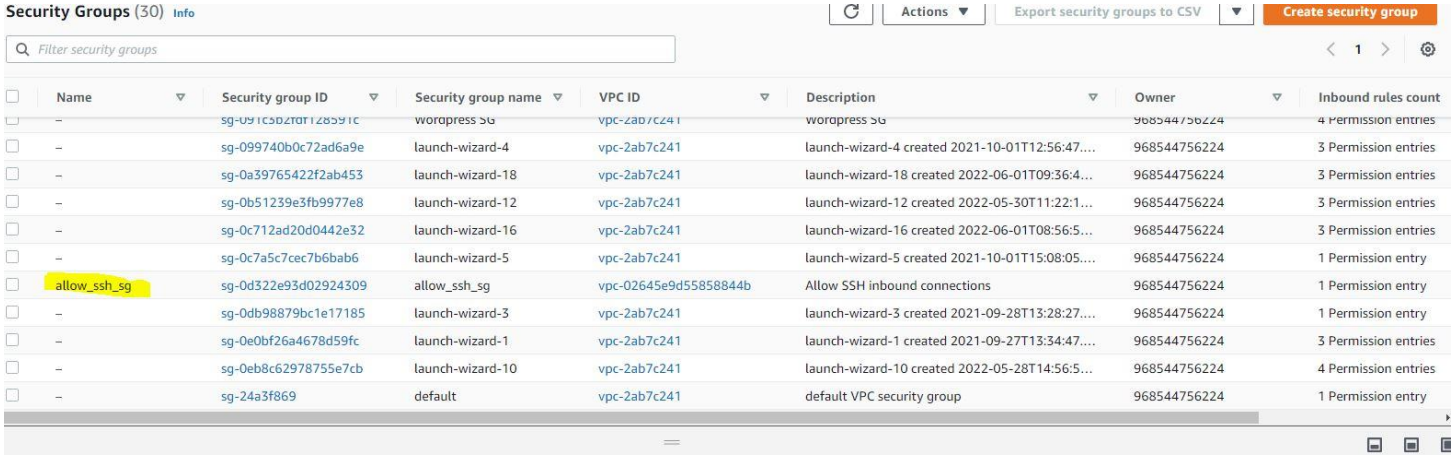
```
resource "aws_security_group" "allow_ssh" {
 name        = "allow_ssh_sg"
 description = "Allow SSH inbound connections"
 vpc_id = aws_vpc.my_vpc.id

 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 = "allow_ssh_sg"
 }
}
```

Run Commands:

terraform plan
terraform apply

Output: Security Groups were attached successfully



Step 13.

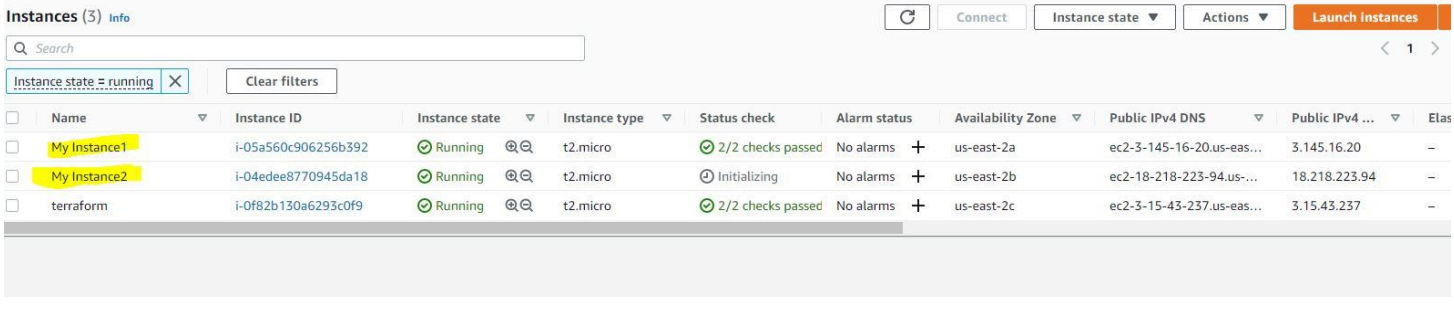
Now you will add:
Input: Add instances
Commands:
sudo nano main.tf

```
resource "aws_instance" "my_instance1" {
  ami           = "ami-0960ab670c8bb45f3"
  instance_type = "t2.micro"
  key_name = "Assign1"
  vpc_security_group_ids = [ aws_security_group.allow_ssh.id ]
  subnet_id = aws_subnet.public1.id
  associate_public_ip_address = true

  tags = {
    Name = "My Instance1"
  }
}

resource "aws_instance" "my_instance2" {
  ami           = "ami-0960ab670c8bb45f3"
  instance_type = "t2.micro"
  key_name = "Assign1"
  vpc_security_group_ids = [ aws_security_group.allow_ssh.id ]
  subnet_id = aws_subnet.public2.id
  associate_public_ip_address = true

  tags = {
    Name = "My Instance2"
  }
}
```

Run Commands:

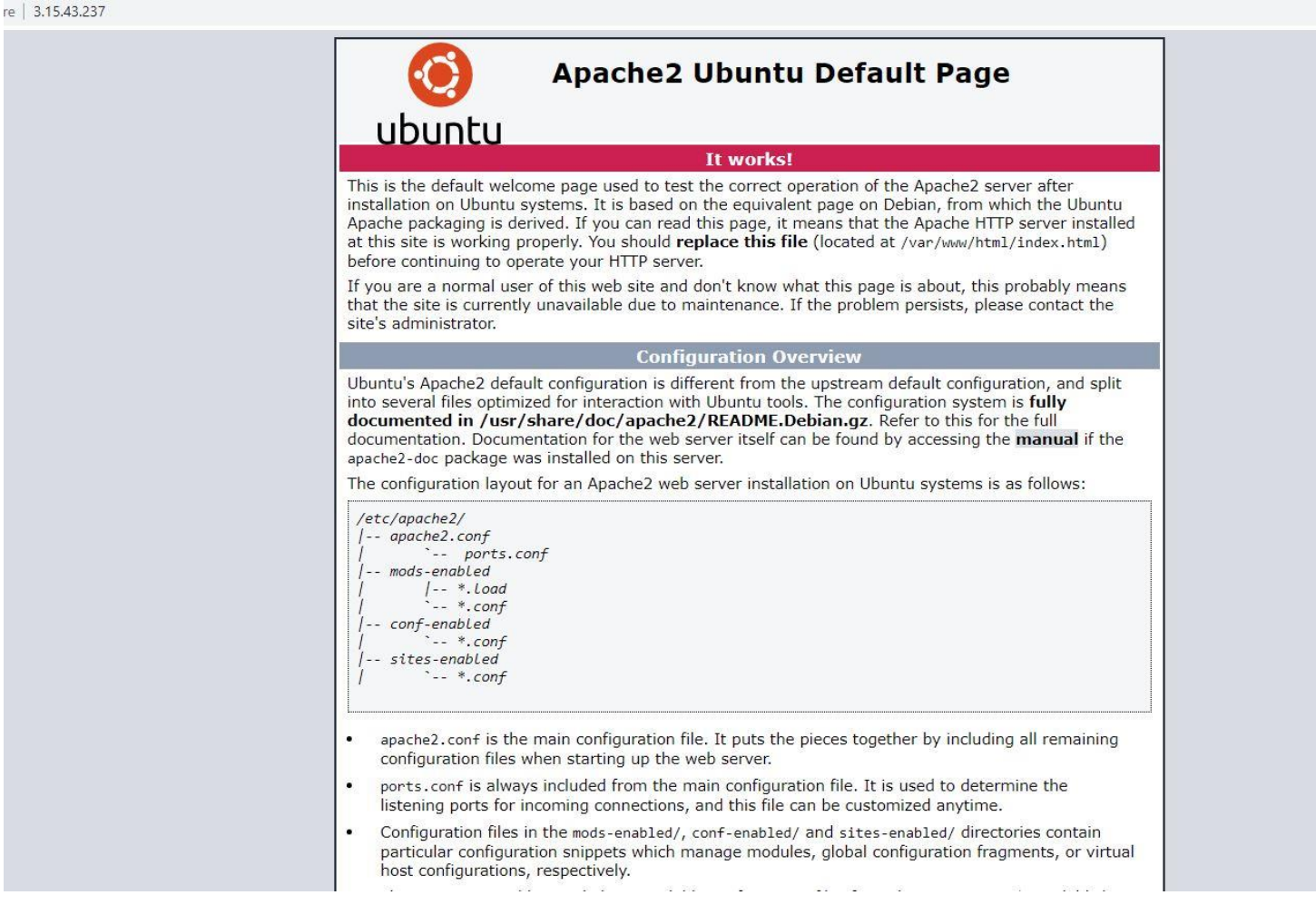
terraform plan
terraform apply

Output: The two (2) Instances were created successfully



Step 14.

Now validate that your site is up and running correctly by using the Public IP address (3.25.43.237) of your Bastion Host, EC2 instance, terraform.



Step 15.

Now instead of deleting all of your resources by hand, use the below command in the command line and all resources will be deleted.

Command:

terraform destroy

Project Conclusion:

Project was completed successfully, deployed at a much faster pace with one script versus creating all the resources manually, and all resources were deleted successfully as expected.

Note: I could, as well, have created a variables file to pass the access keys and the secret keys. This is a preferred method and a best practice, but I wanted to illustrate how the keys were being used in the code.