

## 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.

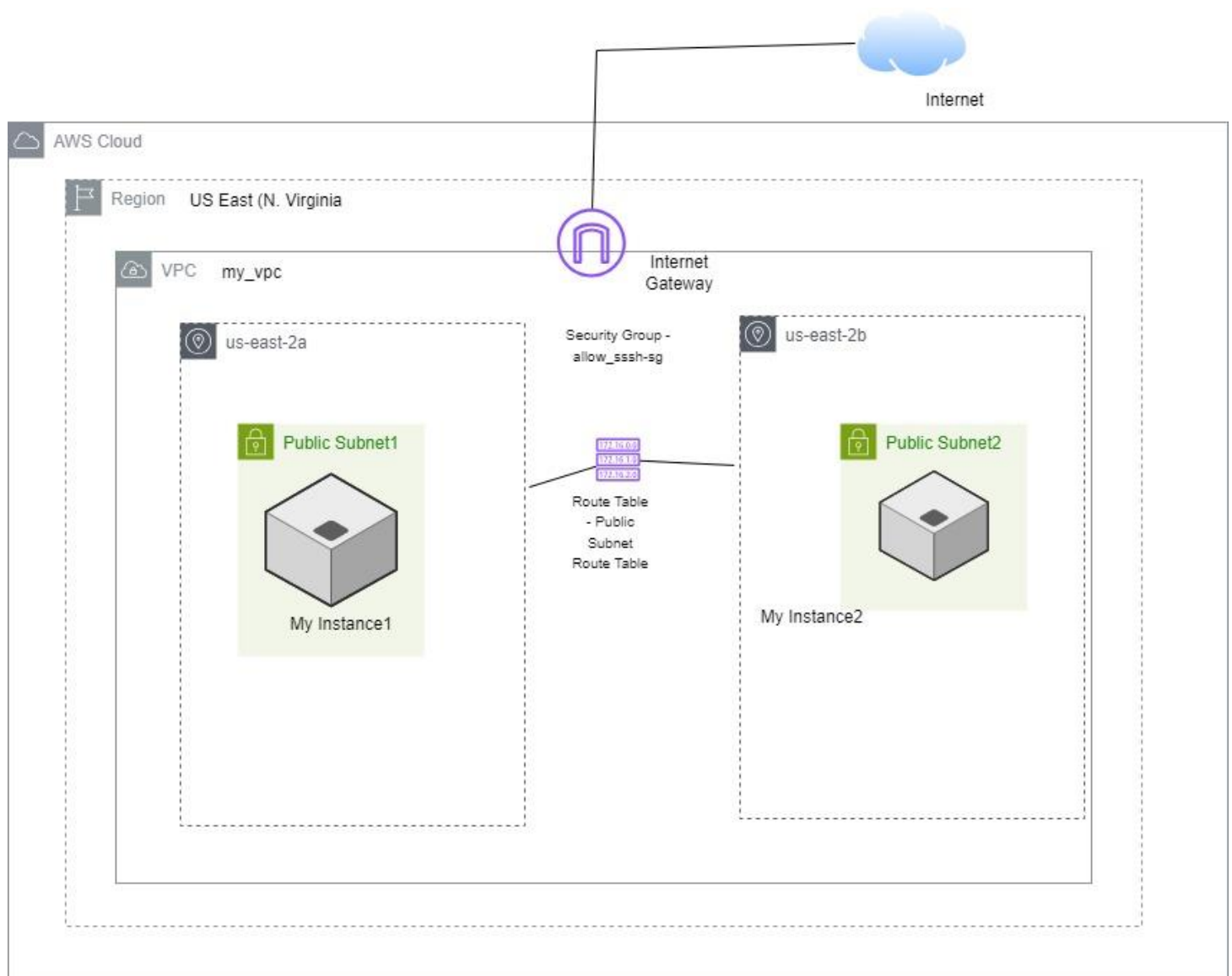
### The company Requirements:

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](#)

# Download Terraform

macOS

Windows

**Linux**

FreeBSD

OpenBSD

Solaris

PACKAGE MANAGER

Ubuntu/DebianCentOS/RHELFedoraAmazon LinuxHomebrew

```
$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -  
$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"  
$ sudo apt-get update && sudo apt-get install terraform
```

[View Tutorials at HashiCorp Learn](#)

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"**

**}**

```
ubuntu@ip-172-31-38-211:~$ mkdir casestudy_terraform
ubuntu@ip-172-31-38-211:~$ cd casestudy_terraform
ubuntu@ip-172-31-38-211:~/casestudy_terraform$ ls
ubuntu@ip-172-31-38-211:~/casestudy_terraform$ sudo nano main.tf
ubuntu@ip-172-31-38-211:~/casestudy_terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v4.20.1...
- Installed hashicorp/aws v4.20.1 (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.

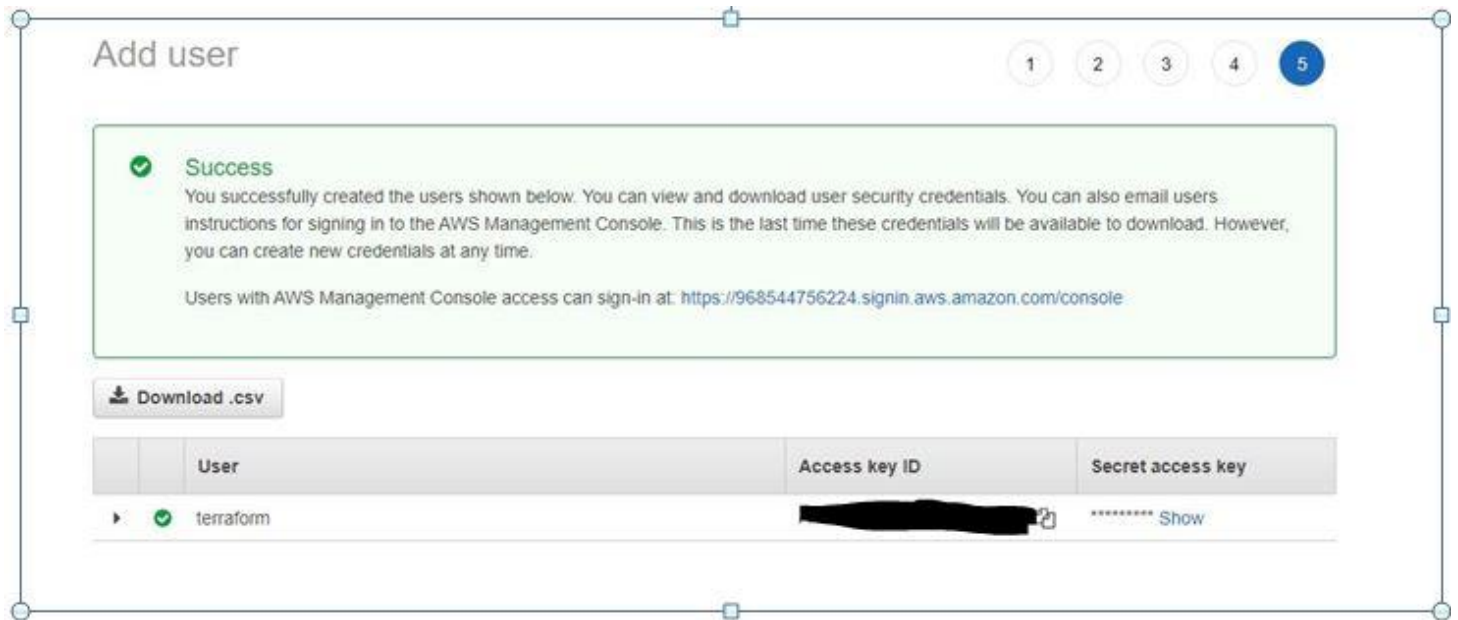
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
ubuntu@ip-172-31-38-211:~/casestudy_terraform$
```

### **Step 3.**

Add IAM Role and get access key ID and secret access key (I had demonstrated how to create an IAM Role in 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 = " "
secret_key = " "
}

resource "aws_instance" "example" {

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

"Name" = "terraform-instance"

}

}

```

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

```

commands will detect it and remind you to do so if necessary.
ubuntu@ip-172-31-44-68:~/casestudy_terraform$ sudo nano main.tf
ubuntu@ip-172-31-44-68:~/casestudy_terraform$ sudo nano main.tf
ubuntu@ip-172-31-44-68:~/casestudy_terraform$ 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.example will be created
+ resource "aws_instance" "example" {
  + ami                    = "ami-0960ab670c8bb45f3"
  + 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)
  + 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               = (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)
  + 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_all               = (known after apply)
  + 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)

+ capacity_reservation_specification {
  + capacity_reservation_preference = (known after apply)

  + capacity_reservation_target {
    + capacity_reservation_id = (known after apply)
    + capacity_reservation_resource_group_arn = (known after apply)
  }
}

```

bscribing to the professional edition here: <https://mobaxterm.mobatek.net>

## 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

```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Still creating... [20s elapsed]
aws_instance.example: Creation complete after 21s [id=i-05d61869092c00ea0]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
ubuntu@ip-172-31-44-68:~/casestudy_terraform$
```

## 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.

<input type="checkbox"/>	-	vpc-2ab7c241	✔ Available	172.31.0.0/16	-	dopt-5ee2ad35	rtb-eabe8c81	a
<input type="checkbox"/>	My VPC	vpc-02645e9d55858844b	✔ Available	10.0.0.0/16	-	dopt-5ee2ad35	rtb-0fb0486d60a3e22c1	a

## 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.

Filter subnets						
<input type="checkbox"/>	Name	Subnet ID	State	VPC	IPv4 CIDR	
<input type="checkbox"/>	-	subnet-a958c5c2	Available	vpc-2ab7c241	172.31.0.0/20	
<input type="checkbox"/>	Public Subnet2	subnet-006f3b283d247667d	Available	vpc-02645e9d55858844b   M...	10.0.1.0/24	
<input type="checkbox"/>	-	subnet-f16c5ebd	Available	vpc-2ab7c241	172.31.32.0/20	
<input type="checkbox"/>	-	subnet-4d468d30	Available	vpc-2ab7c241	172.31.16.0/20	
<input type="checkbox"/>	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

<input type="checkbox"/>	Name	Internet Gateway ID	State	VPC	IPv4 CIDR
<input type="checkbox"/>	My VPC - Internet Gateway	igw-01a868e5719cedc31	Attached	vpc-02645e9d55858844b   My VPC	968544756224
<input type="checkbox"/>	-	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 associations	Route associations	Main	VPC	Owner ID
<input type="checkbox"/>	-	rtb-0fb0486d60a3e22c1	-	-	Yes	vpc-02645e9d55858844b   M...	968544756224
<input type="checkbox"/>	Public Subnet Route Table	rtb-0d65123a561fc0f9d	-	-	No	vpc-02645e9d55858844b   M...	968544756224
<input type="checkbox"/>	-	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.

Route tables (1/3) [Info](#)

Filter route tables

	Name	Route table ID	Explicit subnet associat...	Edge associations	Main	VPC	Owner ID
<input type="checkbox"/>	-	rtb-0fb0486d60a3e22c1	-	-	Yes	vpc-02645e9d55858844b   M...	968544756224
<input checked="" type="checkbox"/>	Public Subnet Rout...	rtb-0d65123a561fc0f9d	2 subnets	-	No	vpc-02645e9d55858844b   M...	968544756224
<input type="checkbox"/>	-	rtb-eabe8c81	-	-	Yes	vpc-2ab7c241	968544756224

rtb-0d65123a561fc0f9d / Public Subnet Route Table

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

Explicit subnet associations (2)

Find subnet association

Subnet ID	IPv4 CIDR	IPv6 CIDR
subnet-006f3b283d247667d / Public Subnet2	10.0.1.0/24	-
subnet-0f8b58c7cd8a539f9 / Public Subnet1	10.0.0.0/24	-

## 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

Security Groups (30) <a href="#">Info</a>								
<input type="text" value="Filter security groups"/> <span>Actions</span> <span>Export security groups to CSV</span> <span>Create security group</span>								
<input type="checkbox"/>	Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count	
<input type="checkbox"/>	–	sg-0911c3b2f0112b591c	woropress sg	vpc-2ab7c241	woropress sg	968544756224	4 Permission entries	
<input type="checkbox"/>	–	sg-099740b0c72ad6a9e	launch-wizard-4	vpc-2ab7c241	launch-wizard-4 created 2021-10-01T12:56:47...	968544756224	3 Permission entries	
<input type="checkbox"/>	–	sg-0a39765422f2ab453	launch-wizard-18	vpc-2ab7c241	launch-wizard-18 created 2022-06-01T09:36:4...	968544756224	3 Permission entries	
<input type="checkbox"/>	–	sg-0b51239e3fb9977e8	launch-wizard-12	vpc-2ab7c241	launch-wizard-12 created 2022-05-30T11:22:1...	968544756224	3 Permission entries	
<input type="checkbox"/>	–	sg-0c712ad20d0442e32	launch-wizard-16	vpc-2ab7c241	launch-wizard-16 created 2022-06-01T08:56:5...	968544756224	3 Permission entries	
<input type="checkbox"/>	–	sg-0c7a5c7cec7b6bab6	launch-wizard-5	vpc-2ab7c241	launch-wizard-5 created 2021-10-01T15:08:05...	968544756224	1 Permission entry	
<input type="checkbox"/>	allow_ssh_sg	sg-0d322e93d02924309	allow_ssh_sg	vpc-02645e9d55858844b	Allow SSH inbound connections	968544756224	1 Permission entry	
<input type="checkbox"/>	–	sg-0db98879bc1e17185	launch-wizard-3	vpc-2ab7c241	launch-wizard-3 created 2021-09-28T13:28:27...	968544756224	1 Permission entry	
<input type="checkbox"/>	–	sg-0e0bf26a4678d59fc	launch-wizard-1	vpc-2ab7c241	launch-wizard-1 created 2021-09-27T13:34:47...	968544756224	3 Permission entries	
<input type="checkbox"/>	–	sg-0eb8c62978755e7cb	launch-wizard-10	vpc-2ab7c241	launch-wizard-10 created 2022-05-28T14:56:5...	968544756224	4 Permission entries	
<input type="checkbox"/>	–	sg-24a3f869	default	vpc-2ab7c241	default VPC security group	968544756224	1 Permission entry	

## 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

Instances (3) Info

Search

Instance state = running

Clear filters

Refresh

Connect

Instance state

Actions

Launch Instances


< 1 >

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elas
<input type="checkbox"/>	My Instance1	i-05a560c906256b392	Running	t2.micro	2/2 checks passed	No alarms	us-east-2a	ec2-3-145-16-20.us-eas...	3.145.16.20	-
<input type="checkbox"/>	My Instance2	i-04edee8770945da18	Running	t2.micro	Initializing	No alarms	us-east-2b	ec2-18-218-223-94.us-...	18.218.223.94	-
<input type="checkbox"/>	terraform	i-0f82b130a6293c0f9	Running	t2.micro	2/2 checks passed	No alarms	us-east-2c	ec2-3-15-43-237.us-eas...	3.15.43.237	-

## 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.

re | 3.15.43.237



# Apache2 Ubuntu 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.

## 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.