

## Lab Exercise 11– Creating a VPC in Terraform

### Objective:

### Objective:

Learn how to use Terraform to create a basic Virtual Private Cloud (VPC) in AWS.

### Prerequisites:

- Terraform installed on your machine.
- AWS CLI configured with the necessary credentials.

### Steps:

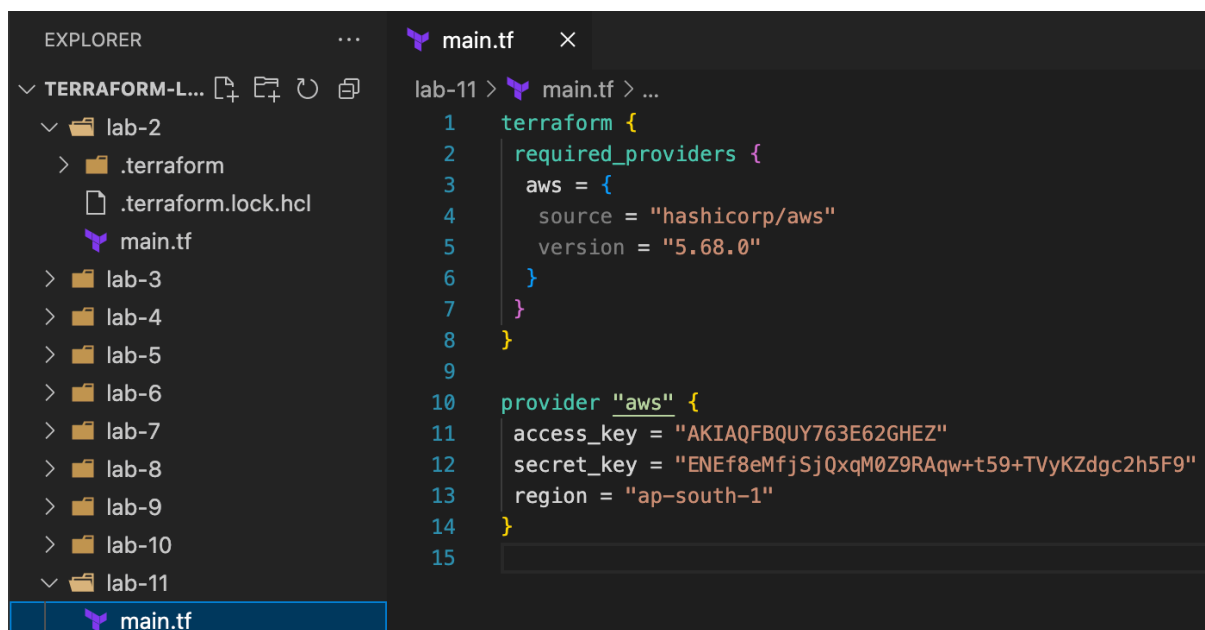
#### 1. Create a Terraform Directory:

```
mkdir lab-11
```

```
cd lab-11
```

```
[sai@Sais-Mac Terraform-Lab % mkdir lab-11  
[sai@Sais-Mac Terraform-Lab % cd lab-11  
sai@Sais-Mac lab-11 %
```

- Create Terraform Configuration Files:
- Create a file named main.tf:



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane displays a file tree with a directory named 'TERRAFORM-L...' containing subdirectories 'lab-2' through 'lab-11'. The 'lab-11' directory is selected, showing its contents: '.terraform', '.terraform.lock.hcl', and 'main.tf'. The 'main.tf' file is open in the editor. The editor shows the following Terraform configuration:

```
1 terraform {  
2   required_providers {  
3     aws = {  
4       source = "hashicorp/aws"  
5       version = "5.68.0"  
6     }  
7   }  
8 }  
9  
10 provider "aws" {  
11   access_key = "AKIAQFBQUY763E62GHEZ"  
12   secret_key = "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"  
13   region = "ap-south-1"  
14 }  
15
```

## # vpc.tf

```
resource "aws_vpc" "gfg-vpc" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "gfg-subnet" {
  vpc_id    = aws_vpc.gfg-vpc.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "gfg-subnet"
  }
}

resource "aws_internet_gateway" "gfg-gw" {
  vpc_id = aws_vpc.gfg-vpc.id

  tags = {
    Name = "gfg-IG"
  }
}

resource "aws_route_table" "gfg-rt" {
  vpc_id = aws_vpc.gfg-vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.gfg-gw.id
  }
}
```

```
tags = {
  Name = "GFG-Route-Table"
}
}

resource "aws_route_table_association" "gfg-rta" {
  subnet_id    = aws_subnet.gfg-subnet.id
  route_table_id = aws_route_table.gfg-rt.id
}

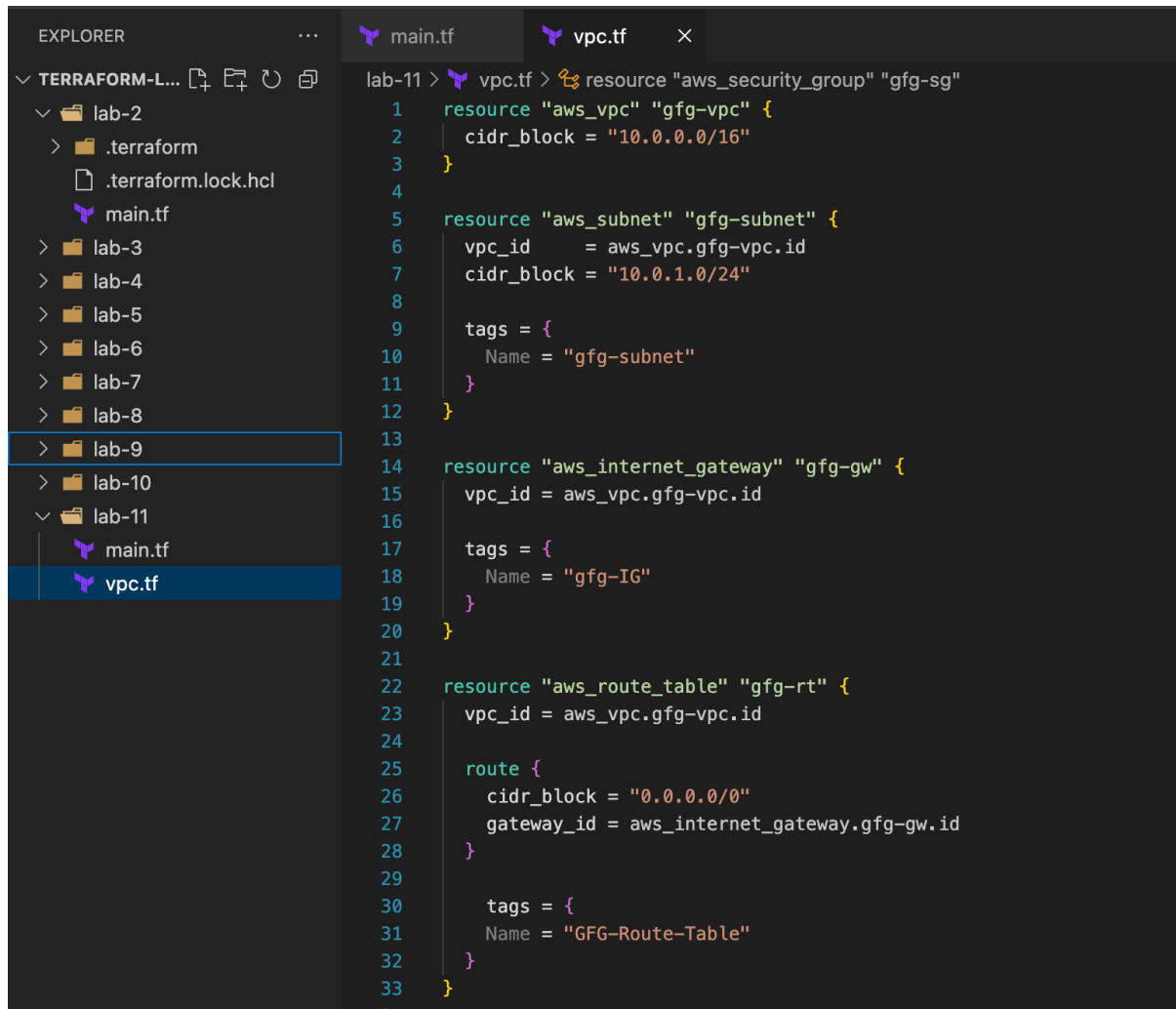
resource "aws_security_group" "gfg-sg" {
  name      = "my-gfg-sg"
  vpc_id    = aws_vpc.gfg-vpc.id

  ingress {
    description      = "TLS from VPC"
    from_port        = 20
    to_port          = 20
    protocol          = "tcp"
    cidr_blocks       = ["0.0.0.0/0"]
    ipv6_cidr_blocks = [":::/0"]
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol          = "-1"
    cidr_blocks       = ["0.0.0.0/0"]
    ipv6_cidr_blocks = [":::/0"]
  }

  tags = {
```

```
Name = "my-gfg-sg"
}
}
```



```
lab-11 > vpc.tf > resource "aws_security_group" "gfg-sg"
1 resource "aws_vpc" "gfg-vpc" {
2   cidr_block = "10.0.0.0/16"
3 }
4
5 resource "aws_subnet" "gfg-subnet" {
6   vpc_id      = aws_vpc.gfg-vpc.id
7   cidr_block  = "10.0.1.0/24"
8
9   tags = {
10     Name = "gfg-subnet"
11   }
12 }
13
14 resource "aws_internet_gateway" "gfg-gw" {
15   vpc_id = aws_vpc.gfg-vpc.id
16
17   tags = {
18     Name = "gfg-IG"
19   }
20 }
21
22 resource "aws_route_table" "gfg-rt" {
23   vpc_id = aws_vpc.gfg-vpc.id
24
25   route {
26     cidr_block = "0.0.0.0/0"
27     gateway_id = aws_internet_gateway.gfg-gw.id
28   }
29
30   tags = {
31     Name = "GFG-Route-Table"
32   }
33 }
```

In this configuration, we define an AWS provider, a VPC with a specified CIDR block, and two subnets within the VPC.

## 2. Initialize and Apply:

- Run the following Terraform commands to initialize and apply the configuration:

```
terraform init
terraform apply
```

```
sai@Sais-Mac lab-11 % terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.68.0"...
- Installing hashicorp/aws v5.68.0...
- Installed hashicorp/aws v5.68.0 (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.

```
sai@Sais-Mac lab-11 % terraform apply

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:
```

Plan: 6 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

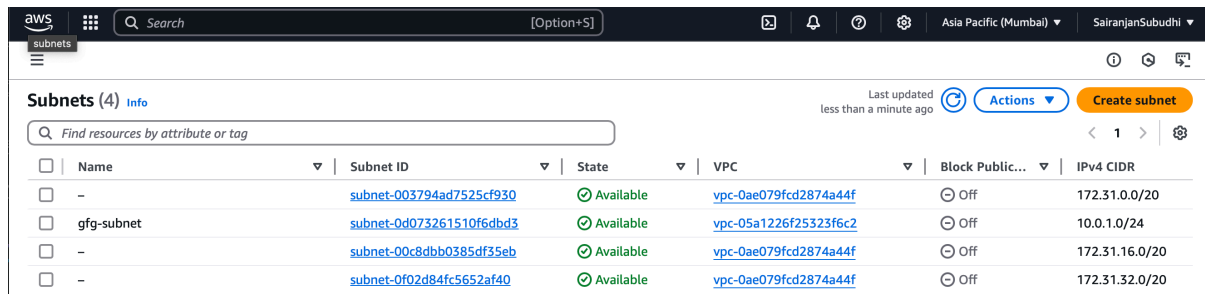
```
aws_vpc.gfg-vpc: Creating...
aws_vpc.gfg-vpc: Creation complete after 2s [id=vpc-05a1226f25323f6c2]
aws_internet_gateway.gfg-gw: Creating...
aws_subnet.gfg-subnet: Creating...
aws_security_group.gfg-sg: Creating...
aws_internet_gateway.gfg-gw: Creation complete after 0s [id=igw-0cda25b1c8dfa4c85]
aws_route_table.gfg-rt: Creating...
aws_subnet.gfg-subnet: Creation complete after 1s [id=subnet-0d073261510f6dbd3]
aws_route_table.gfg-rt: Creation complete after 1s [id=rtb-057881526b38f71ce]
aws_route_table_association.gfg-rta: Creating...
aws_route_table_association.gfg-rta: Creation complete after 0s [id=rtbassoc-07e103967e5f4d15b]
aws_security_group.gfg-sg: Creation complete after 3s [id=sg-0d11a17a0b40b163d]
```

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

- Terraform will prompt you to confirm the creation of the VPC and subnets. Type yes and press Enter.

### 3. Verify Resources in AWS Console:

- Log in to the AWS Management Console and navigate to the VPC service.
- Verify that the VPC and subnets with the specified names and settings have been created.



The screenshot shows the AWS Management Console 'Subnets' page. It displays a table with 4 subnets. The columns are Name, Subnet ID, State, VPC, Block Public..., and IPv4 CIDR. All subnets are in an 'Available' state. The first subnet is named 'gfg-subnet' and is associated with VPC 'vpc-05a1226f25323f6c2' and CIDR '10.0.1.0/24'. The other three subnets are unnamed and associated with VPC 'vpc-0ae079fcd2874a44f' and various CIDR blocks.

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
-	subnet-003794ad7525cf930	Available	vpc-0ae079fcd2874a44f	Off	172.31.0.0/20
gfg-subnet	subnet-0d073261510f6dbd3	Available	vpc-05a1226f25323f6c2	Off	10.0.1.0/24
-	subnet-00c8dbb0385df35eb	Available	vpc-0ae079fcd2874a44f	Off	172.31.16.0/20
-	subnet-0f02d84fc5652af40	Available	vpc-0ae079fcd2874a44f	Off	172.31.32.0/20

## 4. Update VPC Configuration:

- If you want to modify the VPC configuration, update the main.tf file with the desired changes.
- Rerun the terraform apply command to apply the changes:

### terraform apply

```
sai@Sais-Mac lab-11 % terraform apply
aws_vpc.gfg-vpc: Refreshing state... [id=vpc-05a1226f25323f6c2]
aws_internet_gateway.gfg-gw: Refreshing state... [id=igw-0cda25b1c8dfa4c85]
aws_subnet.gfg-subnet: Refreshing state... [id=subnet-0d073261510f6dbd3]
aws_security_group.gfg-sg: Refreshing state... [id=sg-0d11a17a0b40b163d]
aws_route_table.gfg-rt: Refreshing state... [id=rtb-057881526b38f71ce]
aws_route_table_association.gfg-rta: Refreshing state... [id=rtbassoc-07e103967e5f4d15b]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

## 5. Clean Up:

After testing, you can clean up the VPC and subnets:

### terraform destroy

```
sai@Sais-Mac lab-11 % terraform destroy
aws_vpc.gfg-vpc: Refreshing state... [id=vpc-05a1226f25323f6c2]
aws_internet_gateway.gfg-gw: Refreshing state... [id=igw-0cda25b1c8dfa4c85]
aws_subnet.gfg-subnet: Refreshing state... [id=subnet-0d073261510f6dbd3]
aws_security_group.gfg-sg: Refreshing state... [id=sg-0d11a17a0b40b163d]
aws_route_table.gfg-rt: Refreshing state... [id=rtb-057881526b38f71ce]
aws_route_table_association.gfg-rta: Refreshing state... [id=rtbassoc-07e103967e5f4d15b]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- destroy
```

Confirm the destruction by typing yes.

```
Plan: 0 to add, 0 to change, 6 to destroy.

Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

aws_route_table_association.gfg-rta: Destroying... [id=rtbassoc-07e103967e5f4d15b]
aws_security_group.gfg-sg: Destroying... [id=sg-0d11a17a0b40b163d]
aws_route_table_association.gfg-rta: Destruction complete after 0s
aws_subnet.gfg-subnet: Destroying... [id=subnet-0d073261510f6dbd3]
aws_route_table.gfg-rt: Destroying... [id=rtb-057881526b38f71ce]
aws_security_group.gfg-sg: Destruction complete after 1s
aws_subnet.gfg-subnet: Destruction complete after 1s
aws_route_table.gfg-rt: Destruction complete after 1s
aws_internet_gateway.gfg-gw: Destroying... [id=igw-0cda25b1c8dfa4c85]
aws_internet_gateway.gfg-gw: Destruction complete after 0s
aws_vpc.gfg-vpc: Destroying... [id=vpc-05a1226f25323f6c2]
aws_vpc.gfg-vpc: Destruction complete after 1s

Destroy complete! Resources: 6 destroyed.
```

## 6. Conclusion:

This lab exercise demonstrates how to create a basic Virtual Private Cloud (VPC) with subnets in AWS using Terraform. The example includes a simple VPC configuration with two subnets. Experiment with different CIDR blocks, settings, and additional AWS resources to customize your VPC.