***Terraform command***

***Terraform init*** – it will download and install required packages for your project

***Terraform plan -*** it will provide the plan and the details of package required and used in your current project.

***Terraform apply*** - its like similar to apply but it will first check the packages and then run your command.

***Terraform destroy*** – It is used to destroy our instance, if we use terraform destroy it will delete all instances, so we need to use it carefully.

***Terraform destroy –target aws\_route\_table\_association*** – To destroy particular target resource.

***Terraform state list*** – I will list the different type of command list

***Terraform state show (name of command used )*** - it will show details of the command

* Terraform is written in Harshicorp Configuration language which has **.tf** extension.
* Terraform code –

provider "aws" {

  region     = "us-west-1"

  access\_key = "AKIA55CNGTPDGJDZWLSD"

  secret\_key = "jm7I3UJArqGqhrAKQ8LKKDXcflbkXtuDImecvRpn"

}

resource "aws\_instance" "first\_server" {

  ami           = "ami-07b068f843ec78e72"

  instance\_type = "t2.micro"

}

resource "<provider>\_<resource\_type>" "name"{

  config options............

  key = "value"

  key2 = "value2"

}

* In provider setup we need to provide provider name and the region and also account details like access key and secret key .
* Since these Access key and secret key are your private account details, so we don’t need to provide these credentials.
* In resource, aws is provider name and instance is resource type and first\_server is name .
* Now , for ami we need to go to aws ec2 and then in instance go to launch instance and there we can get various **ami(Amazon Machine Images)** where we can get the ami id and instance type .
* Now open the terminal or use visual studio terminal as it will open to your project location.
* After opening terminal we use terraform init in which it will download the required plugins .
* If we want to add another provider then we use , and it will download it

provider "azure"

* Then we will use terraform plan command which will drive through our code and provide what changes will take place or for quick check .
* So now run Terraform apply to run our code , it will ask for yes to run our code so type yes.
* Now we can check in aws that our server got build .

***For creating subnet and vpc using terraform***

resource "aws\_vpc" "first\_vpc" {

  cidr\_block = "10.0.0.0/16"

  tags = {

    Name = "My\_First\_vpc"

  }

}

resource "aws\_subnet" "first\_subnet" {

  vpc\_id     = aws\_vpc.first\_vpc.id

  cidr\_block = "10.0.1.0/24"

  tags = {

    Name = "My\_first\_subnet"

  }

}

In vpc id we had taken the id from above vpc , as the code will create the vpc and we can use the vpc in our subnet .So we had declared the provider and resource name along with the name of vpc with id like **vpc\_id = aws\_vpc.first\_vpc.id .**

**PROJECT**

Steps -:

1. Create vpc.
2. Create Internet Gateway
3. Create custom Route table
4. Create a Subnet
5. Associate subnet with Route Table
6. Create Security Group to allow port 22,80,443
7. Create a network Interface with an IP in the subnet that was created in step 4
8. Assign an Elastic IP to the network interface created in Step 7
9. Create Ubuntu server and install/enable apache2.
10. First we need to create a key-pair in aws ec2 so that we can connect to these devices
11. After giving name and file type to key the file will save in your local machine .

# 1.  Create vpc.

resource "aws\_vpc" "proj\_vpc" {

  cidr\_block = "10.0.0.0/16"

  tags = {

      Name = "my\_vpc"

  }

}

# 2.  Create Internet Gateway

resource "aws\_internet\_gateway" "proj\_gateway" {

  vpc\_id = aws\_vpc.proj\_vpc.id

  tags = {

    Name = "my\_gateway"

  }

}

# 3.  Create custom Route table

resource "aws\_route\_table" "proj\_route\_table" {

  vpc\_id = aws\_vpc.proj\_vpc.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.proj\_gateway.id

  }

  route {

    ipv6\_cidr\_block        = "::/0"

    gateway\_id             = aws\_internet\_gateway.proj\_gateway.id

  }

  tags = {

    Name = "my\_route\_table"

  }

}

# 4.  Create a Subnet

resource "aws\_subnet""proj\_subnet" {

    vpc\_id = aws\_vpc.proj\_vpc.id

    cidr\_block = "10.0.1.0/24"

    availability\_zone = "us-west-1b"

    tags = {

        Name = "proj\_subnet"

    }

}

# 5.  Associate subnet with Route Table

resource "aws\_route\_table\_association" "a" {

  subnet\_id      = aws\_subnet.proj\_subnet.id

  route\_table\_id = aws\_route\_table.proj\_route\_table.id

}

# 6.  Create Security Group to allow port 22,80,443

resource "aws\_security\_group" "allow\_web" {

  name        = "allow\_web\_traffic"

  description = "Allow web traffic"

  vpc\_id      = aws\_vpc.proj\_vpc.id

  ingress {

    description      = "HTTPS"

    from\_port        = 443

    to\_port          = 443

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

  }

    ingress {

    description      = "HTTP"

    from\_port        = 80

    to\_port          = 80

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

  }

    ingress {

    description      = "SSH"

    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\_web"

  }

}

# 7.  Create a network Interface with an IP in the subnet that was created in step 4

resource "aws\_network\_interface" "web\_server\_nic" {

  subnet\_id       = aws\_subnet.proj\_subnet.id

  private\_ips     = ["10.0.1.50"]

  security\_groups = [aws\_security\_group.allow\_web.id]

}

# 8.  Assign an Elastic IP to the network interface created in Step 7

resource "aws\_eip""one" {

  vpc                       = true

  network\_interface         = aws\_network\_interface.web\_server\_nic.id

  associate\_with\_private\_ip = "10.0.1.50"

  depends\_on = [aws\_internet\_gateway.proj\_gateway]

}

# 9.  Create Ubuntu server and install/enable apache2.

resource "aws\_instance""web\_instance"{

    ami = "ami-0d382e80be7ffdae5"

    instance\_type = "t2.micro"

    availability\_zone = "us-west-1b"

    key\_name = "Main\_key"

    network\_interface {

        device\_index = 0

        network\_interface\_id = aws\_network\_interface.web\_server\_nic.id

    }

    user\_data = <<-EOF

                #!/bin/bash

                sudo apt update -y

                sudo apt install apache2 -y

                sudo systemctl start apache2

                sudo bash -c 'echo your very first web server > /var/www/html/index.html'

                EOF

    tags ={

        Name = "web\_server"

    }

}

* To get the value in output we use output command

output "server\_public\_ip" {

   value = aws\_eip.one.public\_ip

 }

In value we provide the value of id for which we want to see the result in output

* To declare a variable in terraform we use variable command

variable "subnet\_profile" {

  type        = string

  # default     = ""

  description = "subnet value"

}

* And to use this variable we use like

**Cidr\_block = var.subnet\_profile**

* If we want to give value at runtime to a variable

***Terraform apply –var “subnet\_profile=10.0.1.0/50”***

* We can also create a ***.tfvars*** file where we can declare the value of our variable so that at the runtime it didn’t ask for that .
* Suppose there are so many files and if we want run particular variable file so we can use this command.

***Terraform apply –var-file examplr.tfvars***

* And if we want to declare default value then we can pass it in variable default.

variable "subnet\_profile" {

  type        = string

  default     = "10.0.1.0/50"

  description = "subnet value"

}