1.Creating Basic EC2 Instance by using terraform and hiding Aws Access and Secret Keys:

Main.tf

```
resource "aws_instance" "provider" {
    ami = "ami-0c7217cdde317cfec"
    instance_type = "t2.micro"
    key_name = "Venkat"
    tags = {
        Name = "Terraform-EC2"
    }

}
```

Provider.tf

```
provider "aws" {
    region = "us-east-1"
    profile = "Venkat"

}
```

2.Create Basic EC2 instance and assigns values in variable.tf file

Main.tf

```
#creating Ec2 Instance and by assigning values in variables
resource "aws_instance" "myc2" {
    ami = var.ami
    instance_type = var.instance_type
    key_name = var.key_name
    count = 4
    tags = {
        Name = "Venkat"
    }
}
```

Provider.tf

```
terraform {
```

```
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region = "var.region"
}
```

Variable.tf

```
variable "region" {
    default = "us-east-1"
}
variable "ami" {
    default = ""
}
variable "instance_type" {
    default = "t2.micro"
}
variable "key_name" {
    default = ""
}
```

3.Terraform Modules:

--------→ create Vcube1 directory in it you have to create

Main.tf

```
terraform {
  required_version = ">=0.12"
}

resource "aws_instance" "ec2_example" {

  ami                    = "ami-09d8b83b58eabf58b"
  instance_type          = "t3.micro"
  key_name               = "default"
  vpc_security_group_ids = [aws_security_group.main.id]
  user_data              = <<-EOF
            #!/bin/bash
            sudo su
```

```
            yum update -y
            yum install -y httpd
            cd /var/www/html
            wget
https://github.com/azeezsalu/techmax/archive/refs/heads/main.zip
            unzip main.zip
            cp -r techmax-main/* /var/www/html/
            rm -rf techmax-main main.zip
            systemctl enable httpd
            systemctl start httpd
      EOF
}

resource "aws_security_group" "main" {
  name        = "EC2-webserver-SG-1"
  description = "Webserver for EC2 Instances"

  ingress {
    from_port   = 80
    protocol    = "TCP"
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 22
    protocol    = "TCP"
    to_port     = 22
    cidr_blocks = ["0.0.0.0/0"]
  }

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

Output.tf

```
output "public_ip" {
  value = aws_instance.ec2_example.public_ip
}
```

---------→ create another directory Vcube2

Main.tf

```
terraform {
  required_version = ">=0.12"
}
resource "aws_instance" "ec2_example" {
  ami                    = "ami-09d8b83b58eabf58b"
  instance_type          = "t3.micro"
  key_name               = ""
  vpc_security_group_ids = [aws_security_group.main.id]

  user_data = <<-EOF
      #!/bin/bash
      sudo su
      yum update -y
      amazon-linux-extras install nginx1 -y
      systemctl enable nginx
      systemctl start nginx
      systemctl status nginx
      sudo echo <!DOCTYPE html> <html> <head> <meta name="viewport"
content="width=device-width, initial-scale=1"> <title>youtube Allow
Fullscreen</title> </head> <body> <!--Need Internet Connection--> <!--
Fullscreen allow--> <iframe width="420" height="315"
src="https://www.vcube.com/embed/OK7fy40Ai6A" allowfullscreen></iframe>
</body> </html>" > /usr/share/nginx/html/index.html
      systemctl restart nginx
      EOF
}

resource "aws_security_group" "main" {
  name        = "EC2-webserver-SG-2"
  description = "Webserver for EC2 Instances"

  ingress {
    from_port   = 8080
    protocol    = "TCP"
    to_port     = 8080
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 80
    protocol    = "TCP"
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
  }
```

```
  ingress {
    from_port   = 22
    protocol    = "TCP"
    to_port     = 22
    cidr_blocks = ["0.0.0.0/0"]
  }

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

output.tf

```
output "public_dns" {
  value = aws_instance.ec2_example.public_dns
}
```

After that you have to create main.tf and variable.tf outside the module

Main.tf

```
provider "aws" {
  region  = var.region
  profile = var.profile
}

module "webserver-1" {
  source = ".//module-1"
}

module "webserver-2" {
  source = ".//module-2"
}
```

Variable.tf

```
variable "region" {
  default = "ap-south-2"
}
```

```
variable "profile" {
  default = ""
}
```

terraform init /plan/apply

Copy Public_IP and paste in Browser you see the result

4.Create S3 bucket and upload Multiple files in S3 bucket

```
#Create S3 Bucket & Upload files using terraform
provider "aws" {
    region = "us-east-1"
}

resource "aws_s3_bucket" "mybucket" {
  bucket = "my-bucket-31-1-2024"

  tags = {
    Environment = "dev"
  }
}

resource "aws_s3_object" "einstein" {
  bucket = "my-bucket-31-1-2024"
  key    = "einstein.jpg"
  source = "C:\\Users\\singa\\OneDrive\\Pictures\\einstein.jpg"
}
resource "aws_s3_bucket_versioning" "versioning" {
  bucket = "my-bucket-31-1-2024"
  versioning_configuration {
    status = "Enabled"
  }
}
resource "null_resource" "multiple-files-uploding" {
  provisioner "local-exec" {
    command = "aws s3 sync D:\\my-folder s3://my-bucket-31-1-2024"
  }
}
```

5.Create VPC and Load Balancer using terraform:

Main.tf

```
resource "aws_vpc" "myvpc" {
```

```
  cidr_block              = var.vpc_cidr
  instance_tenancy        = "default"
  enable_dns_hostnames    = true

  tags      = {
    Name    = "myvpc"
  }
}

resource "aws_subnet" "sub1" {
  vpc_id                  = aws_vpc.myvpc.id
  cidr_block              = var.public_sub1_cidr
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true

  tags      = {
    Name    = "sub1"
  }
}

resource "aws_subnet" "sub2" {
  vpc_id                  = aws_vpc.myvpc.id
  cidr_block              = var.public_sub2_cidr
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = true

  tags      = {
    Name    = "sub2"
  }
}

resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.myvpc.id

  tags      = {
    Name    = "igw"
  }
}

resource "aws_route_table" "RT" {
  vpc_id = aws_vpc.myvpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }
  tags      = {
    Name    = "RT"
```

```
  }
}

resource "aws_route_table_association" "rta1" {
  subnet_id      = aws_subnet.sub1.id
  route_table_id = aws_route_table.RT.id
}

resource "aws_route_table_association" "rta2" {
  subnet_id      = aws_subnet.sub2.id
  route_table_id = aws_route_table.RT.id
}

resource "aws_security_group" "webSg" {
  name   = "web"
  vpc_id = aws_vpc.myvpc.id

  ingress {
    description = "HTTP from VPC"
    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 = "web-Sg"
  }
}

resource "aws_instance" "webserver1" {
  ami                    = "ami-0c7217cdde317cfec"
  instance_type          = "t2.micro"
  key_name               = "Venkat"
```

```
  vpc_security_group_ids = [aws_security_group.webSg.id]
  subnet_id              = aws_subnet.sub1.id
  user_data              = base64encode(file("user.sh"))
}

resource "aws_instance" "webserver2" {
  ami                    = "ami-0c7217cdde317cfec"
  instance_type          = "t2.micro"
  key_name               = "Venkat"
  vpc_security_group_ids = [aws_security_group.webSg.id]
  subnet_id              = aws_subnet.sub2.id
  user_data              = base64encode(file("user1.sh"))
}

#create a Load Balancer
resource "aws_lb" "myalb" {
  name               = "myalb"
  internal           = false
  load_balancer_type = "application"

  security_groups = [aws_security_group.webSg.id]
  subnets         = [aws_subnet.sub1.id, aws_subnet.sub2.id]

  tags = {
    Name = "myalb"
  }
}

resource "aws_lb_target_group" "tg" {
  name     = "myTG"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.myvpc.id

  health_check {
    path = "/"
    port = "traffic-port"
  }
}

resource "aws_lb_target_group_attachment" "attach1" {
  target_group_arn = aws_lb_target_group.tg.arn
  target_id        = aws_instance.webserver1.id
  port             = 80
}

resource "aws_lb_target_group_attachment" "attach2" {
  target_group_arn = aws_lb_target_group.tg.arn
```

```
  target_id        = aws_instance.webserver2.id
  port             = 80
}

resource "aws_lb_listener" "listener" {
  load_balancer_arn = aws_lb.myalb.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    target_group_arn = aws_lb_target_group.tg.arn
    type             = "forward"
  }
}

output "loadbalancerdns" {
  value = aws_lb.myalb.dns_name
}
```

Variable.tf

```
variable "vpc_cidr" {
  description = "vpc_cidr"
  type        = string
  default     = "10.0.0.0/16"
}
variable "public_sub1_cidr" {
  description = "public sub1"
  type        = string
  default     = "10.0.1.0/24"
}

variable "public_sub2_cidr" {
  description = "public sub2"
  type        = string
  default     = "10.0.2.0/24"
}
```

Provider.tf

```
provider "aws" {
    region = "us-east-1"
    profile = "Venkat"

}
```

User.sh

```
#! /bin/bash
sudo -i
sudo apt-get update
sudo apt-get install -y apache2
sudo systemctl start apache2
sudo systemctl enable apache2
echo "The page was created by the user Vcube" | sudo tee
/var/www/html/index.html
```

User1.sh

```
#! /bin/bash
sudo -i
sudo apt-get update
sudo apt-get install -y apache2
sudo systemctl start apache2
sudo systemctl enable apache2
echo "The page was created by the user Venkatesh" | sudo tee
/var/www/html/index.html
```

6.3-Tier VPC by using Terraform:

Main.tf

```
# create vpc
resource "aws_vpc" "My_Vpc" {
  cidr_block              = var.vpc_cidr
  instance_tenancy        = "default"
  enable_dns_hostnames    = true

  tags      = {
    Name    = "My_Vpc"
  }
}

# create internet gateway and attach it to vpc
resource "aws_internet_gateway" "My_IGT" {
  vpc_id    = aws_vpc.My_Vpc.id

  tags      = {
    Name    = "My_IGT"
  }
}
```

```
# create public subnet1
resource "aws_subnet" "public_subnet1" {
  vpc_id                  = aws_vpc.My_Vpc.id
  cidr_block              = var.public_subnet1_cidr
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true

  tags      = {
    Name    = "public_subnet1"
  }
}

# create public subnet2
resource "aws_subnet" "public_subnet2" {
  vpc_id                  = aws_vpc.My_Vpc.id
  cidr_block              = var.public_subnet2_cidr
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = true

  tags      = {
    Name    = "public_subnet2"
  }
}

# create route table and add public route
resource "aws_route_table" "public_route_table" {
  vpc_id       = aws_vpc.My_Vpc.id

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

  tags       = {
    Name     = "public_route_table"
  }
}

# associate public subnet1 to "public route table"
# terraform aws associate subnet with route table
resource "aws_route_table_association"
"public_subnet1_route_table_association" {
  subnet_id           = aws_subnet.public_subnet1.id
  route_table_id      = aws_route_table.public_route_table.id
}

# associate public subnet2 to "public route table"
```

```
# terraform aws associate subnet with route table
resource "aws_route_table_association"
"public_subnet_2_route_table_association" {
  subnet_id           = aws_subnet.public_subnet2.id
  route_table_id      = aws_route_table.public_route_table.id
}

# create private app subnet1
# terraform aws create subnet
resource "aws_subnet" "private_app_subnet1" {
  vpc_id                  = aws_vpc.My_Vpc.id
  cidr_block              = var.private_app_subnet1_cidr
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch  = false

  tags      = {
    Name    = "private_app_subnet1"
  }
}

# create private app subnet2
# terraform aws create subnet
resource "aws_subnet" "private_app_subnet2" {
  vpc_id                  = aws_vpc.My_Vpc.id
  cidr_block              = var.private_app_subnet2_cidr
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch  = false

  tags      = {
    Name    = "private_app_subnet2"
  }
}

# create private data subnet1
# terraform aws create subnet
resource "aws_subnet" "private_data_subnet1" {
  vpc_id                  = aws_vpc.My_Vpc.id
  cidr_block              = var.private_data_subnet1_cidr
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch  = false

  tags      = {
    Name    = "private_data_subnet1"
  }
}

# create private data subnet2
# terraform aws create subnet
```

```
resource "aws_subnet" "private_data_subnet2" {
  vpc_id                  = aws_vpc.My_Vpc.id
  cidr_block              = var.private_data_subnet2_cidr
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = false

  tags      = {
    Name    = "private_data_subnet2"
  }
}
```

Provider.tf

```
provider "aws" {
    region = "us-east-1"
    profile = "Venkat"

}
```

Variable.tf

```
# create vpc cidr blocks

variable "vpc_cidr" {
  description = "vpc_cidr"
  type        = string
  default     = "10.0.0.0/16"
}

variable "public_subnet1_cidr" {
  description = "public subnet1"
  type        = string
  default     = "10.0.1.0/24"
}

variable "public_subnet2_cidr" {
  description = "public subnet2"
  type        = string
  default     = "10.0.2.0/24"
}

variable "private_app_subnet1_cidr" {
  description = "private app subnet1"
  type        = string
  default     = "10.0.3.0/24"
}
```

```
variable "private_app_subnet2_cidr" {
  description = "private app subnet2"
  type        = string
  default     = "10.0.4.0/24"
}

variable "private_data_subnet1_cidr" {
  description = "private data subnet1"
  type        = string
  default     = "10.0.5.0/24"
}

variable "private_data_subnet2_cidr" {
  description = "private data subnet2"
  type        = string
  default     = "10.0.6.0/24"
}
```

7.How to store our terraform.tfstate file in our S3 bucket by using terraform Backend concept:

Method1:

Step1: First create directory mkdir terraform

Step2: Go to aws account create S3 Bucket

Main.tf

```
resource "aws_instance" "terraform_backend" {
    ami = "ami-0c7217cdde317cfec"
    availability_zone = "us-east-1a"
    instance_type = "t2.micro"
    key_name = "vscode"
    tags = {
        Name ="terraform_backend"
    }
}
```

Provider.tf

```
terraform {
  required_providers {
    aws = {
```

```
      source = "hashicorp/aws"
      version = "5.34.0"
    }
  }
}

provider "aws" {
  profile = "Venkat"
  region = "us-east-1"
}
```

Backend.tf

```
terraform {
  backend "s3" {
    bucket = "venkatesh-bucket-1-2-2024"
    key    = "C:\\Users\\singa\\OneDrive\\Desktop\\terraform
backend\\terraform.tfstate"
    region = "us-east-1"
  }
}
```

Method2:

Create directory in the same terraform with mkdir terra123

We can import our terraform.tfstate from our ec2 instance from desktop

Main.tf

```
resource "aws_instance" "myec2" {

}
```

Provider.tf

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.34.0"
    }
  }
}

provider "aws" {
  profile = "Venkat"
  region = "us-east-1"
}
```

terraform import aws_instance.myc2 EC2-id (i-0a37760…….)

Next remove terraformtf.state otherwise it won't import in next time it show error (Before importing this resource,please create its configuration in the root module)

Main.tf

```
resource "aws_s3_bucket" "my_bucket" {
  bucket = "venkatesh-bucket-1-2-2024"

  tags = {
    Name        = "my_bucket"
    Environment = "Dev"
  }
}
```

provider.tf

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.34.0"
    }
  }
}

provider "aws" {
  profile = "Venkat"
  region = "us-east-1"
}
```

terraform import aws_s3_bucket.my_bucket bucket Name(venkatesh-bucket-1-2-2024)

You will see the terraform.tfstate file importing from S3 Bucket

8.Create EC2 Instance without AMI ID:

Ami-resource.tf

```
# Get latest AMI ID for Amazon Linux2 OS
data "aws_ami" "amzlinux" {
  most_recent = true
  owners = [ "amazon" ]
  filter {
    name = "name"
    values = [ "amzn2-ami-hvm-*-gp2" ]
  }
  filter {
    name = "root-device-type"
    values = [ "ebs" ]
  }
  filter {
    name = "virtualization-type"
    values = [ "hvm" ]
  }
  filter {
    name = "architecture"
    values = [ "x86_64" ]
  }
}
```

EC2-instance.tf

```
resource "aws_instance" "my-ec2" {
  ami           = "data.aws_ami.amzlinux"
  instance_type = "t2.micro"
  key_name      = "vscode"
  tags = {
    "Name" = "my-ec2"
  }
}
```