Goal of the Exercise

The goal of this exercise is to test your understanding of a Cloud Compute deployment with Terraform.

You are tasked with creating a codebase for a microservice deployment on a cloud environment, using Terraform. Since our focus is not the application itself, the service will be a simple web server that returns a static page.Requirements

Service Details

- **The service will be a simple web server that returns a static page.**
- **You can use apache2 or nginx to serve a simple HTML file.**
- **The service should be containerized and stored in a Docker registry.**
- **The service should be deployed on a single VM instance as a docker container.**
- **The image should be pulled from the registry specified earlier.**
- **The service should not be accessible from the internet but only through a Load Balancer, except for SSH access from specific IP addresses.**

Deliverables

You are to provide the following as part of your solution:

1. **Dockerfile to build the image.**

2. **Terraform codebase to deploy the following:**
    - **Docker registry**
    - **VPC and all required networking resources**
    - **EC2 instance**
    - **Load Balancer**

Notes

- **Since a CI/CD pipeline is not part of this exercise, we will work around that with the help of Terraform.**
- **To push the image to the Docker registry, build the image locally before running the Terraform code.**
- **Once the image registry is created, use Terraform to push the local image to the registry.**
- **Similarly, use Terraform to pull the image from the registry and run it on the VM instance once it is created.**
- **For SSH access:**
    - **Use a locally created key.**
    - **Add it to the cloud provider key storage.**
    - **Attach it to the instance.**
- **Use Terraform best practices for file structure and code organization.**

    - **Use variables, locals, modules, etc. when appropriate.**
- **Include all tfvars, scripts, and other helper files used to deploy the application.**
- **You may use any cloud provider of your choosing, although AWS is preferred.**
- **Use a free tier account or a trial account to deploy the resources.**

Login to your ec2-server

- mkdir app
- cd app
- vi Dockerfile

```
[root@ip-172-31-44-97 ~]# mkdir app
[root@ip-172-31-44-97 ~]# ls
app
[root@ip-172-31-44-97 ~]# vi Dockerfile
[root@ip-172-31-44-97 ~]#
```

```
[root@ip-172-31-44-97 app]# ls
Dockerfile
```

give this script in the Dockerfile.

FROM nginx:stable-alpine

LABEL maintainer=Mujaheed

 # Remove default nginx content and add our page

RUN rm -rf /usr/share/nginx/html/*

COPY index.html /usr/share/nginx/html/index.html

 # Expose port 80

EXPOSE 80

 # Use default nginx entrypoint

CMD ["nginx", "-g", "daemon off;"]

```
# Simple nginx image serving a static HTML page
FROM nginx:stable-alpine
LABEL maintainer=Mujaheed

# Remove default nginx content and add our page
RUN rm -rf /usr/share/nginx/html/*
COPY index.html /usr/share/nginx/html/index.html

# Expose port 80
EXPOSE 80

# Use default nginx entrypoint
CMD ["nginx", "-g", "daemon off;"]
```

In the app directory give index.html file

- vi index.html

give this script

<!doctype html>

<html>

<head>

<meta charset="utf-8"/>

<title>Terraform Microservice</title>

</head>

<body>

<h1>Microservice served from Docker container</h1>

** Deployed via Terraform on EC2 and fronted by internal ALB. **

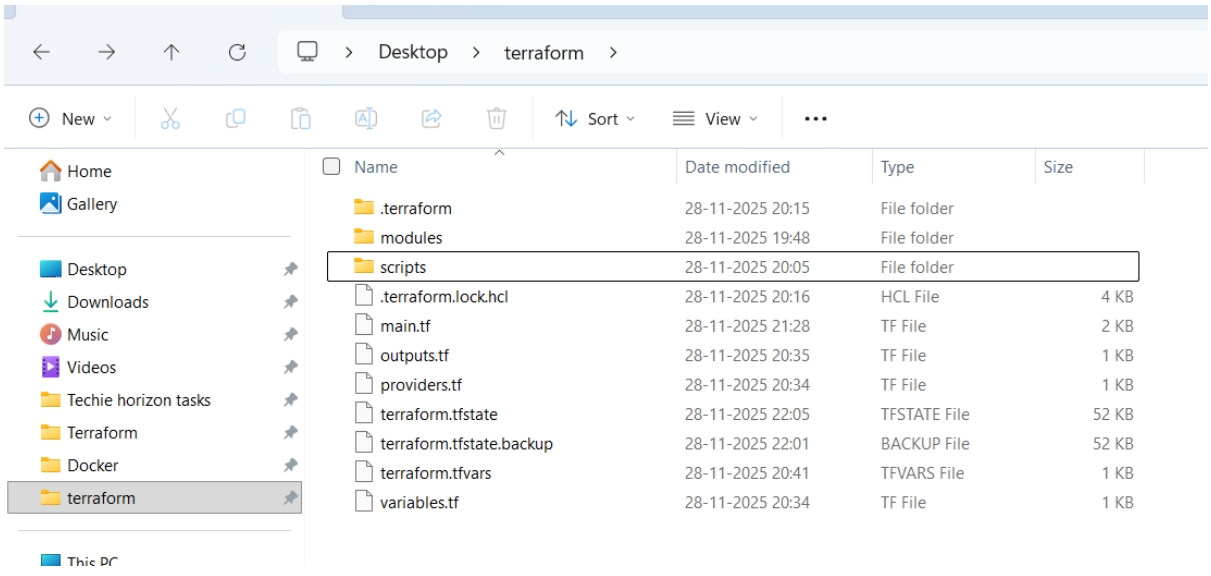
</body>

</html>

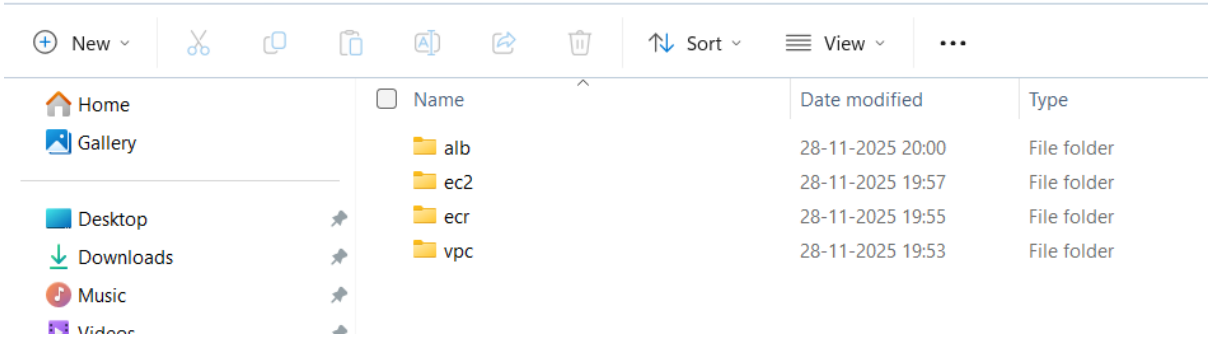
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Terraform Microservice</title>
  </head>
  <body>
    <h1>Microservice served from Docker container</h1>
    <p>Deployed via Terraform on EC2 and fronted by internal ALB.</p>
  </body>
</html>
```

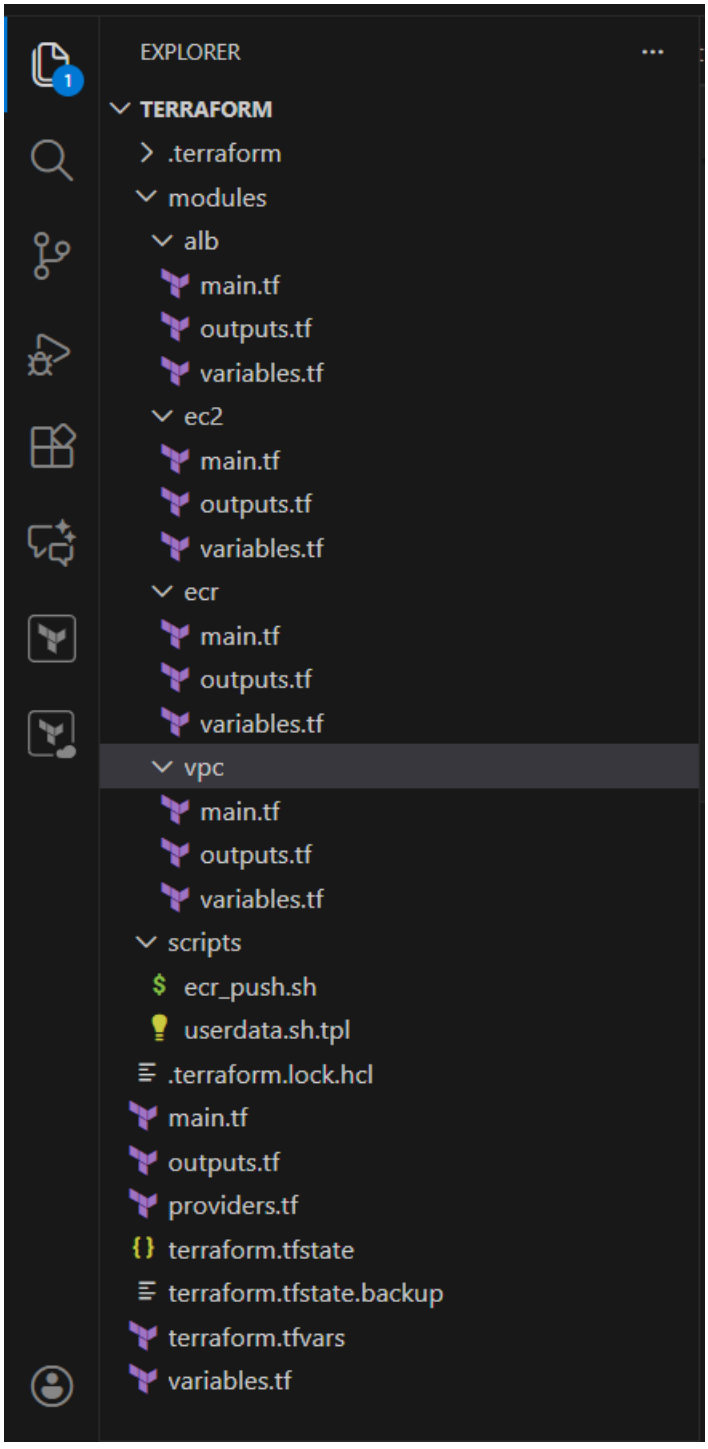
Open vscode make the directories in this format

Your root location like this.



Modules should be like this


New
Sort
View
Home
Gallery
Desktop
Downloads
Music
Name
Date modified
Type
alb
28-11-2025 20:00
File folder
ec2
28-11-2025 19:57
File folder
ecr
28-11-2025 19:55
File folder
vpc
28-11-2025 19:53
File folder


EXPLORER
TERRAFORM
.terraform
modules
alb
main.tf
outputs.tf
variables.tf
ec2
main.tf
outputs.tf
variables.tf
ecr
main.tf
outputs.tf
variables.tf
vpc
main.tf
outputs.tf
variables.tf
scripts
ecr_push.sh
userdata.sh.tpl
.terraform.lock.hcl
main.tf
outputs.tf
providers.tf
terraform.tfstate
terraform.tfstate.backup
terraform.tfvars
variables.tf

modules/alb

main.tf

```
resource "aws_security_group" "alb_sg" {
  name   = "${var.env}-alb-sg"
  vpc_id = var.vpc_id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["10.10.0.0/16"]
  }

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

  tags = { Name = "${var.env}-alb-sg" }
```

```
}

resource "aws_lb" "internal_alb" {
  internal           = false  # change from true to false
  load_balancer_type = "application"
  subnets            = var.subnet_ids
  security_groups    = [aws_security_group.alb_sg.id]
}

resource "aws_lb_target_group" "tg" {
  name       = "${var.env}-tg"
  port       = var.target_port
  protocol   = "HTTP"
  vpc_id     = var.vpc_id
  target_type = "instance"
}

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

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

resource "aws_lb_target_group_attachment" "attach" {
  target_group_arn = aws_lb_target_group.tg.arn
  target_id        = var.target_instance_id
  port             = var.target_port
}
```

Variables.tf:

```
variable "env" { type = string }
variable "vpc_id" { type = string }
variable "subnet_ids" { type = list(string) }
variable "target_instance_id" { type = string }
variable "target_port" { type = number }
```

modules/ec2

main.tf:

```
data "local_file" "pubkey" {
  filename = var.local_pub_key_path
}

resource "aws_key_pair" "deployer" {
  key_name   = var.key_name
  public_key = data.local_file.pubkey.content
}

resource "aws_security_group" "ec2_sg" {
  name   = "${var.env}-ec2-sg"
  vpc_id = var.vpc_id

  # SSH only from allowed CIDR
  ingress {
    description = "SSH from allowed"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = [var.allowed_ssh_cidr]
```

```
  }

  # Allow HTTP from VPC CIDR (ALB is internal; it will be in the same VPC)
  ingress {
    description = "HTTP from VPC"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["10.10.0.0/16"]
  }

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

  tags = { Name = "${var.env}-ec2-sg" }
}
```

```
data "aws_ami" "amzn2" {
  most_recent = true
  owners      = ["amazon"]
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}

resource "aws_iam_role" "ec2_role" {
  name = "${var.env}-ec2-role"
  assume_role_policy =
data.aws_iam_policy_document.ec2_assume.json
}

data "aws_iam_policy_document" "ec2_assume" {
  statement {
    actions = ["sts:AssumeRole"]
    principals {
      type = "Service"
```

```
      identifiers = ["ec2.amazonaws.com"]
    }
  }
}

resource "aws_iam_role_policy_attachment" "ecr_read" {
  role       = aws_iam_role.ec2_role.name
  policy_arn =
"arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryRead
Only"
}

resource "aws_iam_instance_profile" "ec2_profile" {
  name = "${var.env}-ec2-profile"
  role = aws_iam_role.ec2_role.name
}

resource "aws_instance" "app" {
  ami                  = data.aws_ami.amzn2.id
  instance_type        = var.instance_type
  subnet_id            = var.public_subnet_id
```

```
  associate_public_ip_address = true
  key_name                    = aws_key_pair.deployer.key_name
  vpc_security_group_ids = [aws_security_group.ec2_sg.id]
  iam_instance_profile   =
aws_iam_instance_profile.ec2_profile.name

  user_data =
templatefile("${path.module}/../../scripts/userdata.sh.tpl", {
    repository_url = var.image_repo_uri
    image_tag      = var.image_tag
    region         = var.aws_region
  })

  tags = { Name = "${var.env}-app-instance" }
}
```

Variables.tf:

```
variable "env" { type = string }
variable "vpc_id" { type = string }
variable "public_subnet_id" { type = string }
variable "private_subnet_id" { type = string }
```

```
variable "allowed_ssh_cidr" { type = string }
variable "key_name" { type = string }
variable "instance_type" { type = string }
variable "local_pub_key_path" { type = string }
variable "image_repo_uri" { type = string }
variable "image_tag" { type = string }
variable "aws_region" { type = string }
```

modules/ecr

main.tf:

```
resource "aws_ecr_repository" "repo" {
  name                 = var.name
  image_tag_mutability = "MUTABLE"

  tags = {
    Name = var.name
  }
}
```

Variables.tf:

```
variable "name" { type = string }
```

modules/vpc

main.tf:

```
resource "aws_vpc" "this" {
  cidr_block           = "10.10.0.0/16"
  enable_dns_support   = true
  enable_dns_hostnames = true
}

resource "aws_internet_gateway" "this" {
  vpc_id = aws_vpc.this.id
}

resource "aws_subnet" "public" {
  count = 2
  vpc_id = aws_vpc.this.id
  cidr_block = var.public_subnet_cidrs[count.index]
  availability_zone = var.availability_zones[count.index]
  map_public_ip_on_launch = true
}

resource "aws_subnet" "private" {
```

```
  count = 2
  vpc_id = aws_vpc.this.id
  cidr_block = var.private_subnet_cidrs[count.index]
  availability_zone = var.availability_zones[count.index]
}

resource "aws_route_table" "public" {
  vpc_id = aws_vpc.this.id

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

resource "aws_route_table_association" "public_assoc" {
  count          = 2
  subnet_id      = aws_subnet.public[count.index].id
  route_table_id = aws_route_table.public.id
}
```

Variables.tf:

```
variable "env" { type = string }
variable "region" { type = string }

variable "availability_zones" {
  type    = list(string)
  default = ["us-east-1a", "us-east-1b"]
}

variable "public_subnet_cidrs" {
  type    = list(string)
  default = ["10.10.1.0/24", "10.10.2.0/24"]
}

variable "private_subnet_cidrs" {
  type    = list(string)
  default = ["10.10.3.0/24", "10.10.4.0/24"]
}
```

Root

Main.tf:

```
module "vpc" {
  source = "./modules/vpc"
  env    = var.env
  region = var.aws_region
}

module "ecr" {
  source = "./modules/ecr"
  name   = "${var.env}-${var.image_name}"
}

# Create EC2 first (it doesn't require ALB outputs)
module "ec2" {
  source = "./modules/ec2"

  env              = var.env
  vpc_id           = module.vpc.vpc_id

  public_subnet_id   = module.vpc.public_subnet_ids[0]
  private_subnet_id  = module.vpc.private_subnet_ids[0]
```

```
  allowed_ssh_cidr   = var.allowed_ssh_cidr
  key_name           = var.key_name
  instance_type      = var.instance_type

  local_pub_key_path = var.local_pub_key_path

  image_repo_uri     = module.ecr.repository_url
  image_tag          = var.image_tag
  aws_region         = var.aws_region
}

# ALB uses EC2 instance id as target
module "alb" {
  source = "./modules/alb"

  env               = var.env
  vpc_id            = module.vpc.vpc_id
  subnet_ids        = module.vpc.private_subnet_ids

  target_instance_id = module.ec2.instance_id
  target_port        = 80
```

```
}

# local helper: build + push local image to ECR (requires Docker + aws cli locally)
resource "null_resource" "build_and_push" {
  depends_on = [module.ecr]

  provisioner "local-exec" {
    interpreter = ["C:/Program Files/Git/bin/bash.exe", "-c"]
    command     = "${path.module}/scripts/ecr_push.sh ${module.ecr.repository_url} ${var.image_name} ${var.image_tag} ${var.aws_region}"
  }
}
```

Providers.tf:

```
terraform {
  required_version = ">= 1.2.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
```

```
      version = ">= 4.0"
    }
    null = {
      source  = "hashicorp/null"
      version = ">= 3.0"
    }
  }
}

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

Variables.tf:

```
variable "aws_region" {
  type    = string
  default = "us-east-1"
}

variable "env" {
  type    = string
```

```
  default = "dev"
}

variable "allowed_ssh_cidr" {
  description = "CIDR allowed to SSH (replace with your IP e.g. 203.0.113.5/32)"
  type        = string
  default     = "0.0.0.0/0"
}

variable "local_pub_key_path" {
  type        = string
  description = "Path to the local public SSH key to upload"
  default     = "~/.ssh/id_rsa.pub"
}

variable "image_name" {
  description = "Local docker build context name"
  type        = string
  default     = "terraform-microservice"
}
```

```
variable "image_tag" {
  description = "Image tag"
  type        = string
  default     = "v1"
}

variable "instance_type" {
  type    = string
  default = "t3.micro"
}

variable "key_name" {
  description = "Name to give the imported key pair in AWS"
  type        = string
  default     = "tf-test-key"
}
```

Terraform.tfvars:

```
aws_region         = "us-east-1"
env                = "dev"
```

allowed_ssh_cidr = "192.168.1.0/24" # replace with your IP

image_name = "terraform-microservice"

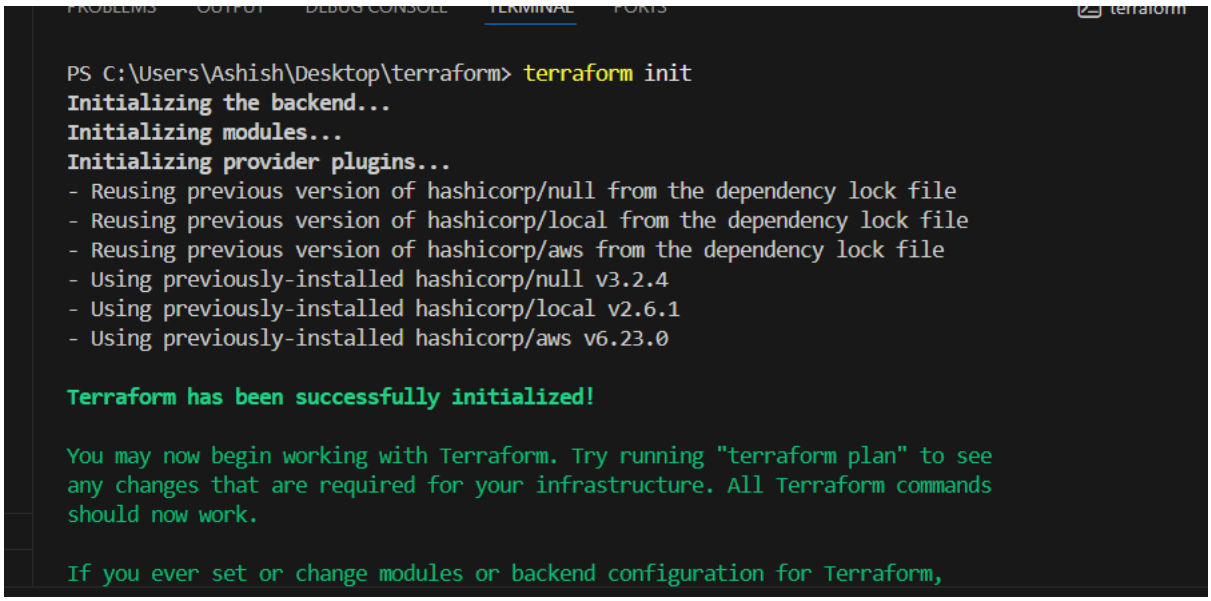
image_tag = "v1"

instance_type = "t3.micro"

key_name = "tf-test-key"

MUST match variables.tf

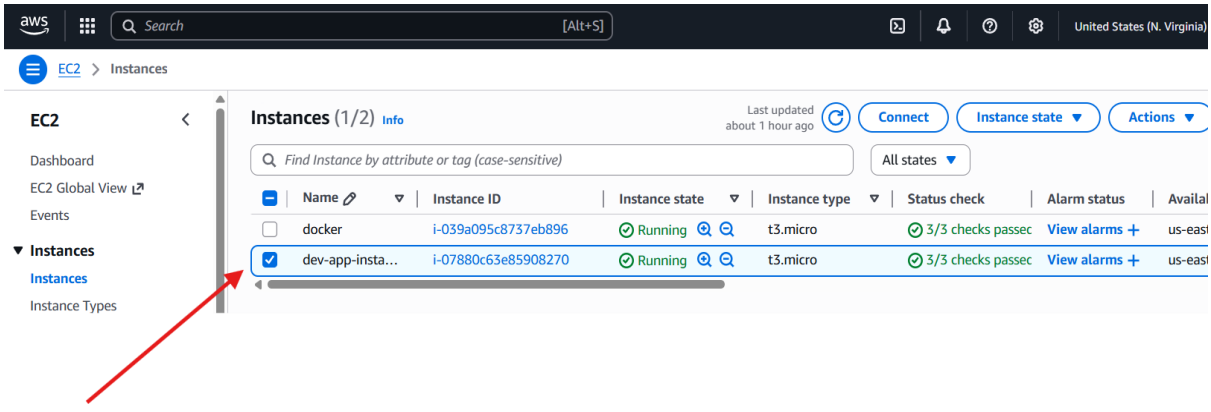
local_pub_key_path = "C:/Users/Ashish/.ssh/id_rsa.pub"



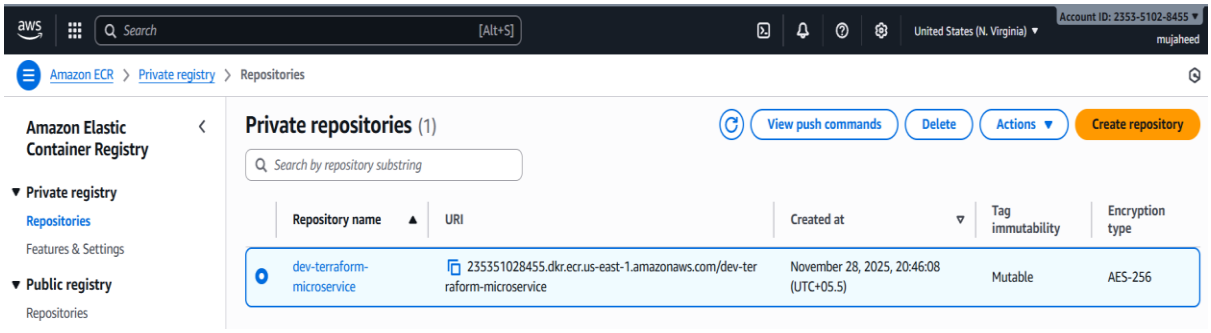
```
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\Ashish\Desktop\terraform> terraform apply
module.ec2.data.local_file.pubkey: Reading...
module.ec2.data.local_file.pubkey: Read complete after 0s [id=59057d101fb57f71baf8962ae87ba7f0
module.vpc.aws_route_table_association.a: Refreshing state... [id=rtbassoc-0cc6dbf7bab415303]
module.ecr.aws_ecr_repository.repo: Refreshing state... [id=dev-terraform-microservice]
module.ec2.aws_key_pair.deployer: Refreshing state... [id=tf-test-key]
module.ec2.data.aws_iam_policy_document.ec2_assume: Reading...
module.ec2.data.aws_ami.amzn2: Reading...
module.vpc.aws_vpc.this: Refreshing state... [id=vpc-09c82cb0e7833942a]
module.ec2.data.aws_iam_policy_document.ec2_assume: Read complete after 0s [id=1443847869]
module.ec2.aws_iam_role.ec2_role: Refreshing state... [id=dev-ec2-role]
null_resource.build_and_push: Refreshing state... [id=1174400565]
module.ec2.aws_iam_role_policy_attachment.ecr_read: Refreshing state... [id=dev-ec2-role/arn:aw
nEC2ContainerRegistryReadOnly]
```

```
module.alb.aws_lb.internal_alb: Still creating... [02m40s elapsed]
module.alb.aws_lb.internal_alb: Still creating... [02m50s elapsed]
module.alb.aws_lb.internal_alb: Still creating... [03m00s elapsed]
module.alb.aws_lb.internal_alb: Still creating... [03m10s elapsed]
module.alb.aws_lb.internal_alb: Creation complete after 3m11s [id=arn:aws:elasticloadbalancing:us-ea
loadbalancer/app/dev-internal-alb/bafada1a0b429dbc]
module.alb.aws_lb_listener.listener: Creating...
module.alb.aws_lb_listener.listener: Creation complete after 1s [id=arn:aws:elasticloadbalancing:us-
5:listener/app/dev-internal-alb/bafada1a0b429dbc/4d63ebf53621c27a]

Apply complete! Resources: 8 added, 4 changed, 3 destroyed.

Outputs:

alb_dns = "internal-dev-internal-alb-425609482.us-east-1.elb.amazonaws.com"
ec2_instance_id = "i-07880c63e85908270"
ecr_repository_url = "235351028455.dkr.ecr.us-east-1.amazonaws.com/dev-terraform-microservice"
```

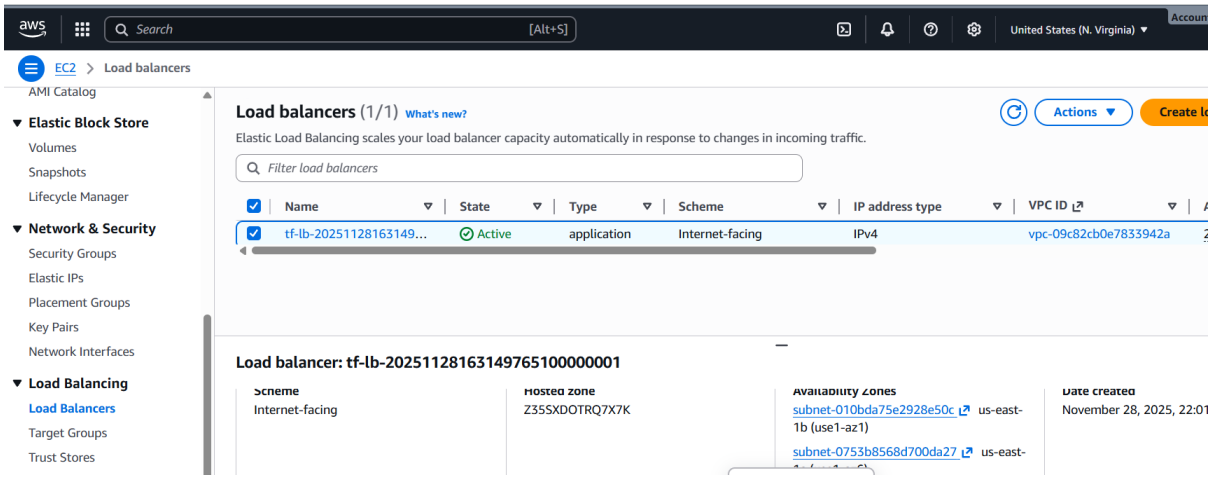
An ec2-instance has been created.



An image has been created in ecr with the terraform script.



An application loadbalancer has been created .



With the dnsname tf-lb-20251128163149765100000001-497167590.us-east-1.elb.amazonaws.com