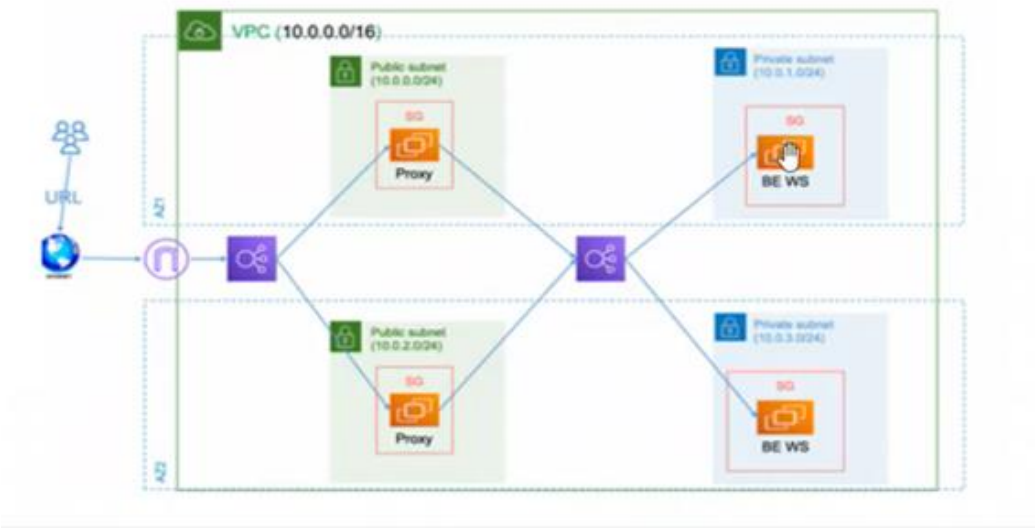
Project Overview

This Terraform project is designed to create an infrastructure with multiple EC2 instances and two load balancers, as shown in the provided diagram. The infrastructure consists of:

1. A Virtual Private Cloud (VPC) with public and private subnets.
2. EC2 instances configured as web servers in the private subnets.
3. Two load balancers:
    - The first one is a public load balancer that forwards traffic to the proxy server.
    - The second one is a private load balancer that forwards traffic to the backend EC2 web servers.

The project uses AWS services, including EC2, Load Balancers and S3 for storing Terraform state files.

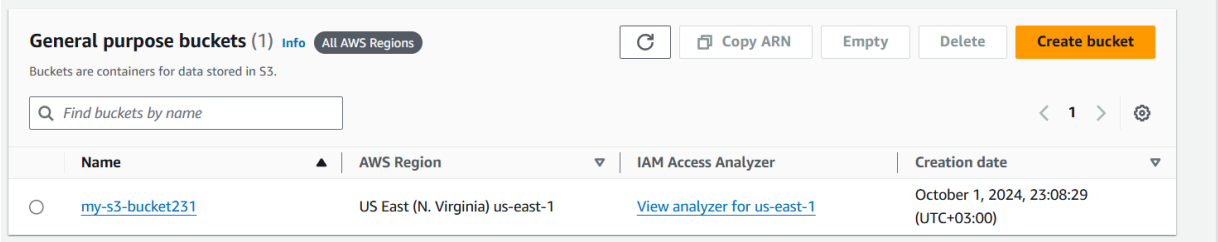


Implementation Details

1. **Remote State Configuration**: Configure the backend to store the state file remotely using S3:

remote_backend.tf > ...

```hcl
data "aws_s3_bucket" "existing_bucket" {
  bucket =  "my-s3-bucket231"
}
resource "aws_s3_bucket" "terraform_state" {
  count  = length(data.aws_s3_bucket.existing_bucket.id) == 0 ? 1 : 0
  bucket = "my-s3-bucket231"
  force_destroy = true
  lifecycle {
    prevent_destroy = true
  }
  tags = {
    Name = "Terraform State Bucket"
  }
}
resource "aws_s3_bucket_versioning" "enable" {
  count = length(data.aws_s3_bucket.existing_bucket.id) == 0 ? 1 : 0
  bucket = aws_s3_bucket.terraform_state[0].id
  versioning_configuration {
    status = "Enabled"
  }
}
resource "aws_dynamodb_table" "terraform_locks" {
  name         = "dynamodb-locks"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }

  lifecycle {
    ignore_changes = [name]
  }

  tags = {
    Name = "Terraform Lock Table"
  }
}

terraform {
  backend "s3" {
    bucket         =  "my-s3-bucket231"
    key            = "terraform.tfstate"
    region         = "us-east-1"
    dynamodb_table = "NM-locks"
    encrypt        = true
  }
}
```



2. **VPC**: Create a VPC using a custom VPC module.

```hcl
resource "aws_vpc" "vpc" {
  cidr_block = var.vpc_cidr
  tags = {
    Name = "VPC"
  }
}

resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.vpc.id
  tags = {
    Name = "IGW"
  }
}

resource "aws_eip" "eip" {
  domain   = "vpc"
}

resource "aws_nat_gateway" "nat" {
  allocation_id = aws_eip.eip.id
  subnet_id     = var.nat_subnet_id
  tags = {
    Name = "NAT"
  }
  depends_on = [aws_internet_gateway.igw]
}
```

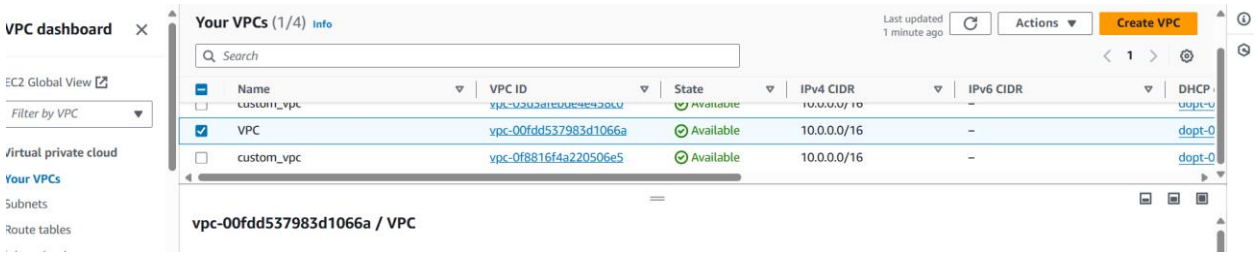
modules > VPC > outputs.tf > output "vpc_id"

```hcl
output "vpc_id" {
    value = aws_vpc.vpc.id
}

output "igw_id" {
    value = aws_internet_gateway.igw.id
}

output "nat_id" {
    value = aws_nat_gateway.nat.id
}
```

main.tf ./ | main.tf .../VPC | outputs.tf | variables.tf

modules > VPC > variables.tf > variable "vpc_cidr"

```hcl
variable "vpc_cidr" {
    description = "VPC CIDR"
    type = string
    }
variable "nat_subnet_id"{
    description = "The subnet ID of the public subnet in which to place the gateway"
    type = string
}
```



3. **EC2 Instances**:

- Create EC2 instances using a custom EC2 module.
- Use a `data` source to get the AMI ID for EC2:
- Create the security group
- Create public, private instance
- Use remote-exec provisioners to install Apache server

```hcl
data "aws_ami" "ami_id" {
  most_recent = true
  owners      = ["amazon"]
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
  }
resource "aws_security_group" "sg" {
  vpc_id = var.sg_vpc_id
  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  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 = "sg"
  }
```

```hcl
resource "aws_instance" "pub-ec2" {
  count                       = length(var.ec2_public_subnet_id)
  ami                         = data.aws_ami.ami_id.id
  instance_type               = "t2.micro"
  subnet_id                   = var.ec2_public_subnet_id[count.index]
  security_groups             = [aws_security_group.sg.id]
  associate_public_ip_address = true
  key_name                    = var.key_pair_name

  tags = {
    Name = "public_ec2_${count.index}"
  }

 provisioner "remote-exec" {
  inline = [
    "set -e",
    "sleep 10",
    "sudo yum update -y",
    "sudo yum install -y httpd",
    "sudo systemctl start httpd",
    "sudo systemctl enable httpd",
    <<-EOT
    echo '<html><body><h1>Welcome to Public Ahmed Negm EC2 Instance ${count.index}</h1>
    </body></html>' | sudo tee /var/www/html/index.html
    EOT
    ]
  connection {
    type        = "ssh"
    host        = self.public_ip
    user        = "ec2-user"
    private_key = file("~/Downloads/Nehmkey.pem")
    timeout     = "5m"
  }
}
```

```
  provisioner "local-exec" {
    when        = create
    on_failure  = continue
    command = "echo public-ip-${count.index} : ${self.public_ip} >> all-ips.txt"
  }
}


resource "aws_instance" "priv-ec2" {
  count                       = length(var.ec2_private_subnet_id)
  ami                         = data.aws_ami.ami_id.id
  instance_type               = "t2.micro"
  subnet_id                   = var.ec2_private_subnet_id[count.index]
  security_groups             = [aws_security_group.sg.id]
  associate_public_ip_address = false
  lifecycle {
    create_before_destroy = true
  }
  tags = {
    Name = "private_ec2_${count.index}"
  }


  user_data = <<EOF
            #!/bin/bash
            sleep 10
            sudo yum update -y
            sleep 10
            sudo yum install -y httpd
            sleep 10
            sudo systemctl start httpd
            sudo systemctl enable httpd
            echo "<html><body><h1>${var.ec2_html[count.index]}</h1>
            <p>welcome to Priv${count.index} </p>
            </body></html>" | sudo tee /var/www/html/index.html
            sudo systemctl restart httpd
            EOF


  provisioner "local-exec" {
    when        = create
    on_failure  = continue
    command = "echo private-ip-${count.index} : ${self.private_ip} >> all-ips.txt"
  }
}
```

modules > ec2 > outputs.tf > output "pub-ips"

```
output "public_ec2_id" {
  value = aws_instance.pub-ec2[*].id
}
output "private_ec2_id" {
  value = aws_instance.priv-ec2[*].id
}
output "security_group_id" {
    value = aws_security_group.sg.id
}
output "pub-ips" {
  value =  aws_instance.pub-ec2[*].public_ip

}
```

main.tf | variables.tf

modules > ec2 > variables.tf > variable "key_pair_name" > default

```
variable "sg_vpc_id"{
  type = string
}
variable "priv_lb_dns" {}
variable "ec2_public_subnet_id" {
  type = list
}
variable "ec2_private_subnet_id" {
  type = list
}
variable "ec2_html" {
  type    = list(string)
  default = [
   "Welcome to Private EC2 Instance 1",
   "Welcome to Private EC2 Instance 2"
  ]
}
variable "key_pair_name" {
  description = "Name of the EC2 Key Pair"
  type        = string
  default = "Nehmkey"
}
```

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv
private_ec2_0	i-0c130959bf945a528	Running	t2.micro	2/2 checks passec	View alarms	us-east-1a	–
public_ec2_0	i-0e4ac25571f326454	Running	t2.micro	2/2 checks passec	View alarms	us-east-1a	–
public_ec2_1	i-05e91c666d669f878	Running	t2.micro	2/2 checks passec	View alarms	us-east-1b	–
private_ec2_1	i-0977fcca8853fbb83	Running	t2.micro	2/2 checks passec	View alarms	us-east-1b	–

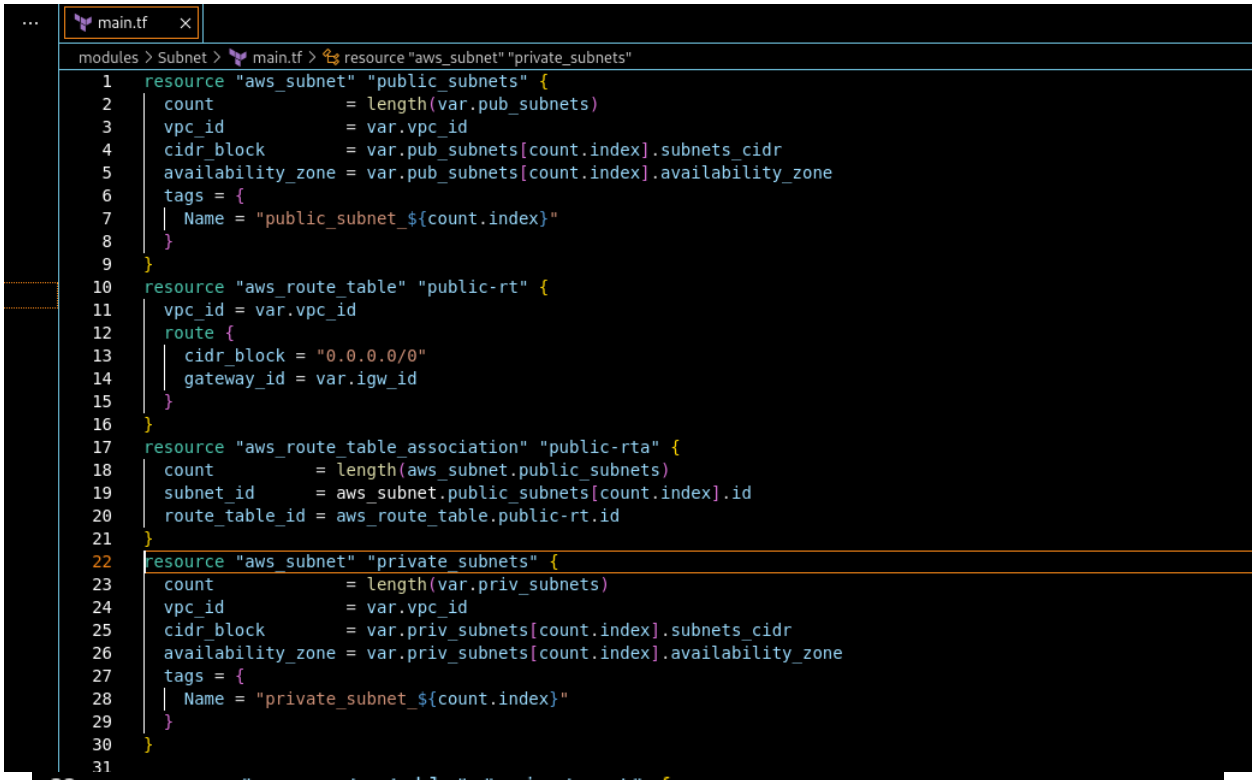
i-0c130959bf945a528 (private_ec2_0)

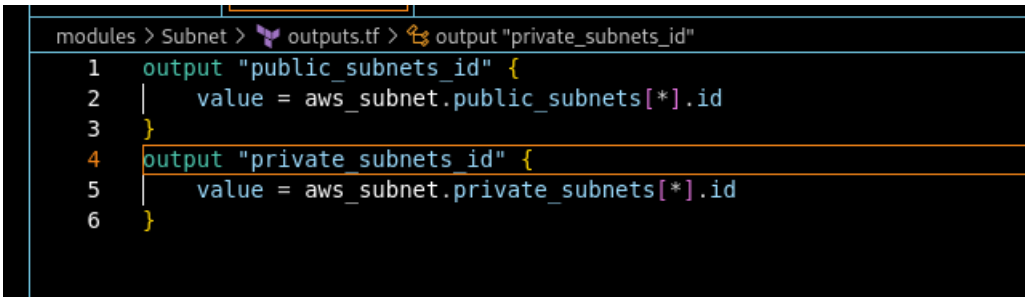
Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

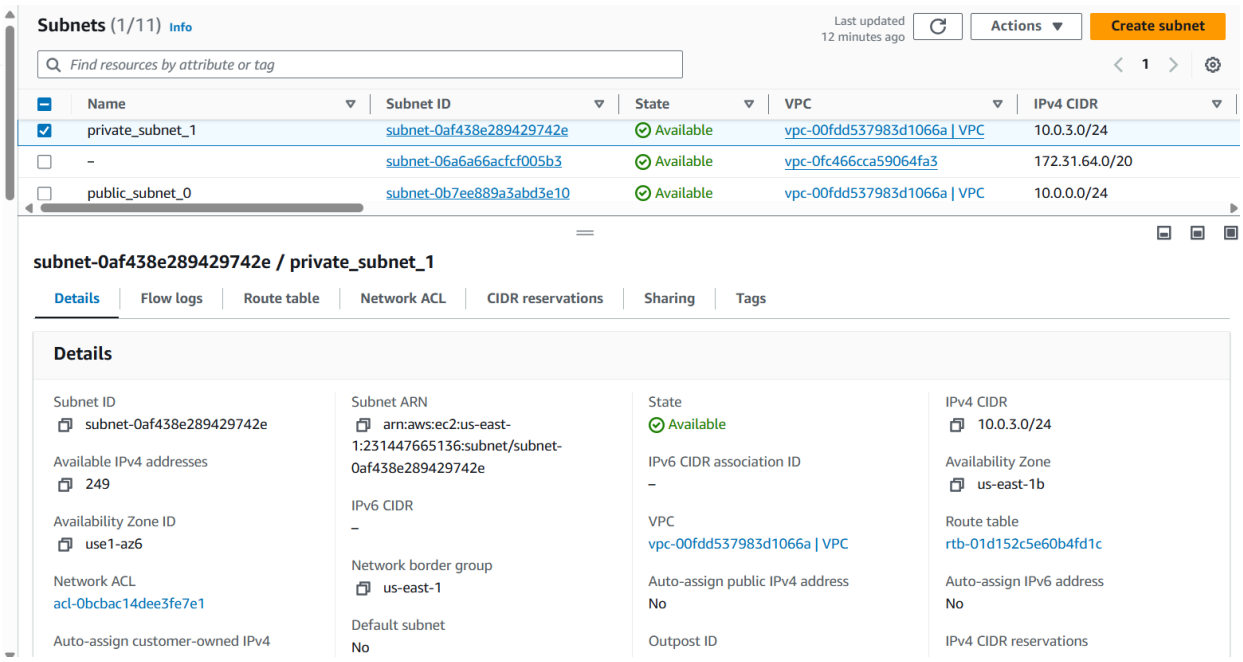
Instance summary Info

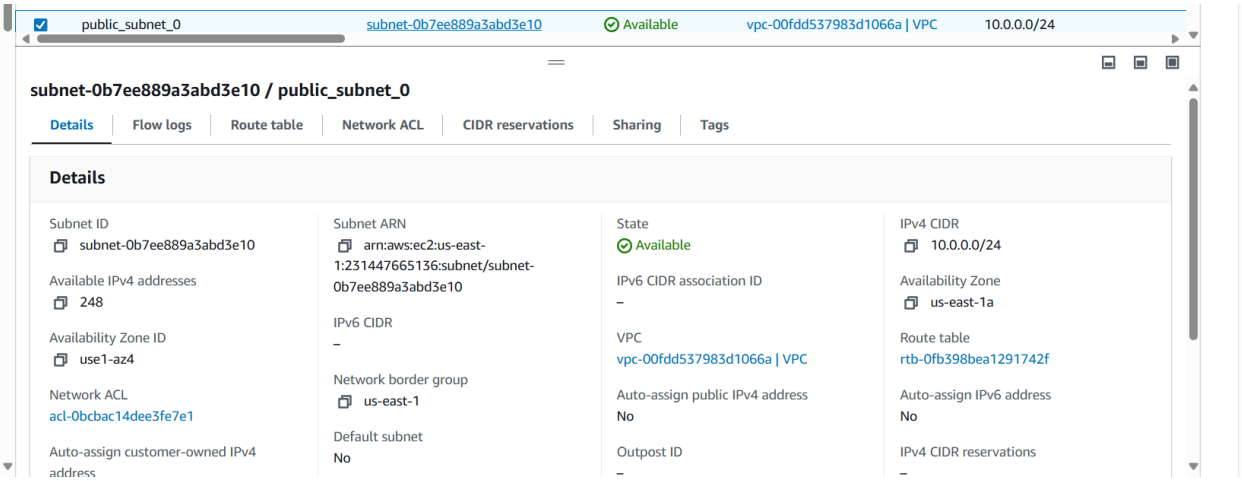
Instance ID i-0c130959bf945a528 (private_ec2_0)	Public IPv4 address –	Private IPv4 addresses 10.0.1.133
IPv6 address –	Instance state Running	Public IPv4 DNS –
Hostname type IP name: ip-10-0-1-133.ec2.internal	Private IP DNS name (IPv4 only) ip-10-0-1-133.ec2.internal	

4. **Subnets: create** public and private subnets using a custom **Subnet** module
    - Define subnets, route tables, and gateways






Subnets (1/11) Info
Last updated 12 minutes ago
Actions
Create subnet
Find resources by attribute or tag
1
Name
Subnet ID
State
VPC
IPv4 CIDR
private_subnet_1
subnet-0af438e289429742e
Available
vpc-00fdd537983d1066a | VPC
10.0.3.0/24
–
subnet-06a6a66acfcf005b3
Available
vpc-0fc466cca59064fa3
172.31.64.0/20
public_subnet_0
subnet-0b7ee889a3abd3e10
Available
vpc-00fdd537983d1066a | VPC
10.0.0.0/24
subnet-0af438e289429742e / private_subnet_1
Details
Flow logs
Route table
Network ACL
CIDR reservations
Sharing
Tags
Details
Subnet ID
subnet-0af438e289429742e
Available IPv4 addresses
249
Availability Zone ID
use1-az6
Network ACL
acl-0bcbac14dee3fe7e1
Auto-assign customer-owned IPv4
Subnet ARN
arn:aws:ec2:us-east-1:231447665136:subnet/subnet-0af438e289429742e
IPv6 CIDR
–
Network border group
us-east-1
Default subnet
No
State
Available
IPv6 CIDR association ID
–
VPC
vpc-00fdd537983d1066a | VPC
Auto-assign public IPv4 address
No
Outpost ID
IPv4 CIDR
10.0.3.0/24
Availability Zone
us-east-1b
Route table
rtb-01d152c5e60b4fd1c
Auto-assign IPv6 address
No
IPv4 CIDR reservations


public_subnet_0
subnet-0b7ee889a3abd3e10
Available
vpc-00fdd537983d1066a | VPC
10.0.0.0/24
subnet-0b7ee889a3abd3e10 / public_subnet_0
Details
Flow logs
Route table
Network ACL
CIDR reservations
Sharing
Tags
Details
Subnet ID
subnet-0b7ee889a3abd3e10
Available IPv4 addresses
248
Availability Zone ID
use1-az4
Network ACL
acl-0bcbac14dee3fe7e1
Auto-assign customer-owned IPv4 address
Subnet ARN
arn:aws:ec2:us-east-1:231447665136:subnet/subnet-0b7ee889a3abd3e10
IPv6 CIDR
–
Network border group
us-east-1
Default subnet
No
State
Available
IPv6 CIDR association ID
–
VPC
vpc-00fdd537983d1066a | VPC
Auto-assign public IPv4 address
No
Outpost ID
–
IPv4 CIDR
10.0.0.0/24
Availability Zone
us-east-1a
Route table
rtb-0fb398bea1291742f
Auto-assign IPv6 address
No
IPv4 CIDR reservations
–

5. **Load Balancers**:
    - Create a public and private load balancer using a custom module.

main.tf

modules > Load_Balancer > main.tf > resource "aws_lb" "load-balancer"

```hcl
resource "aws_lb_target_group" "tg" {
    count    = 2
    port     = 80
    protocol = "HTTP"
    vpc_id   = var.lb_vpc_id
}
resource "aws_lb_target_group_attachment" "public-target-group-attachment" {
    count            = length(var.pub_target_id)
    target_group_arn = aws_lb_target_group.tg[0].arn
    target_id        = var.pub_target_id[count.index]
    port             = 80
}
resource "aws_lb_target_group_attachment" "private-target-group-attachment" {
    count            = length(var.priv_target_id)
    target_group_arn = aws_lb_target_group.tg[1].arn
    target_id        = var.priv_target_id[count.index]
    port             = 80
}
resource "aws_lb" "load-balancer" {
    count                      = 2
    name                       = "load-balancer-${count.index}"
    internal                   = var.lb_internal[count.index]
    load_balancer_type         = "application"
    subnets                    = var.lb_subnets[count.index]
    security_groups            = [var.lb_sg_id]
}
resource "aws_lb_listener" "lb-listner" {
    count             = 2
    load_balancer_arn = aws_lb.load-balancer[count.index].id
    port              = "80"
    protocol          = "HTTP"
    default_action {
        type  = "forward"
        target_group_arn = aws_lb_target_group.tg[count.index].id
    }
}
```

main.tf | outputs.tf

modules > Load_Balancer > outputs.tf > output "public_load_balancer_dns"

```hcl
output "public_load_balancer_dns" {
  value = aws_lb.load-balancer[0].dns_name
}
output "private_load_balancer_dns" {
  value = aws_lb.load-balancer[1].dns_name
}
```

main.tf | variables.tf

modules > Load_Balancer > variables.tf > variable "lb_vpc_id"

```hcl
variable "lb_vpc_id"{
    type = string
}
variable "pub_target_id"{
    type = list(string)
}
variable "priv_target_id"{
    type = list(string)
}
variable "lb_internal"{
    type = list(bool)
}
variable "lb_subnets"{
    type = list(list(string))
}
variable "lb_sg_id"{}
```

Load balancers (2)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

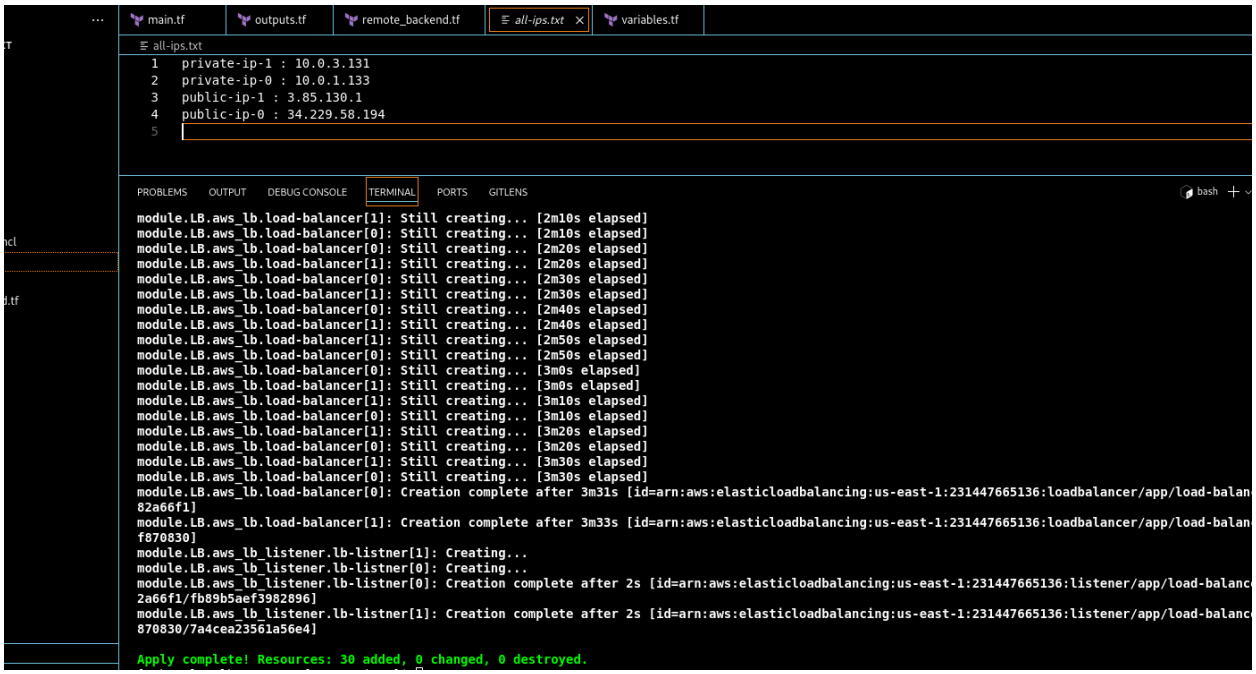
Actions | Create load balancer

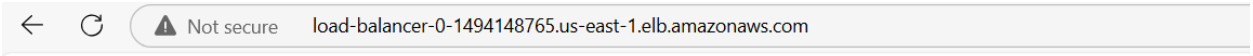
Filter load balancers

1

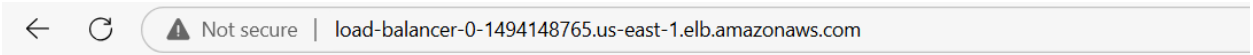
Name	DNS name	State	VPC ID	Availability Zones	Type
load-balancer-0	load-balancer-0-1494148...	Active	vpc-00fdd537983d10...	2 Availability Zones	application
load-balancer-1	internal-load-balancer-1-2...	Active	vpc-00fdd537983d10...	2 Availability Zones	application

The final Result:





Welcome to Public Ahmed Negm EC2 Instance 0



Welcome to Public Ahmed Negm EC2 Instance 1