TERRAFORM PROJECT

Yahya Abd El-Azim

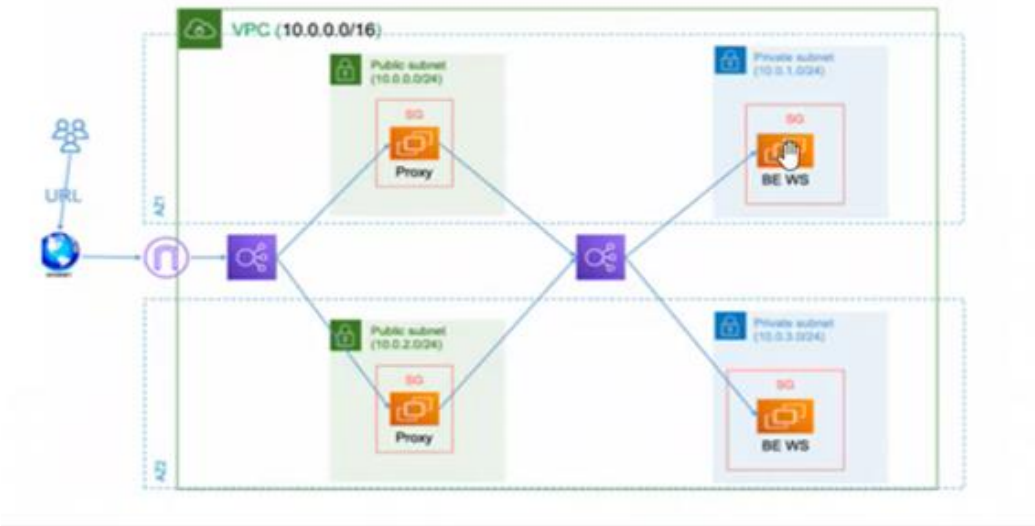
ENG: AYA NABIL

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. A proxy server in the public subnets that routes traffic to the private web servers.
4. 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.

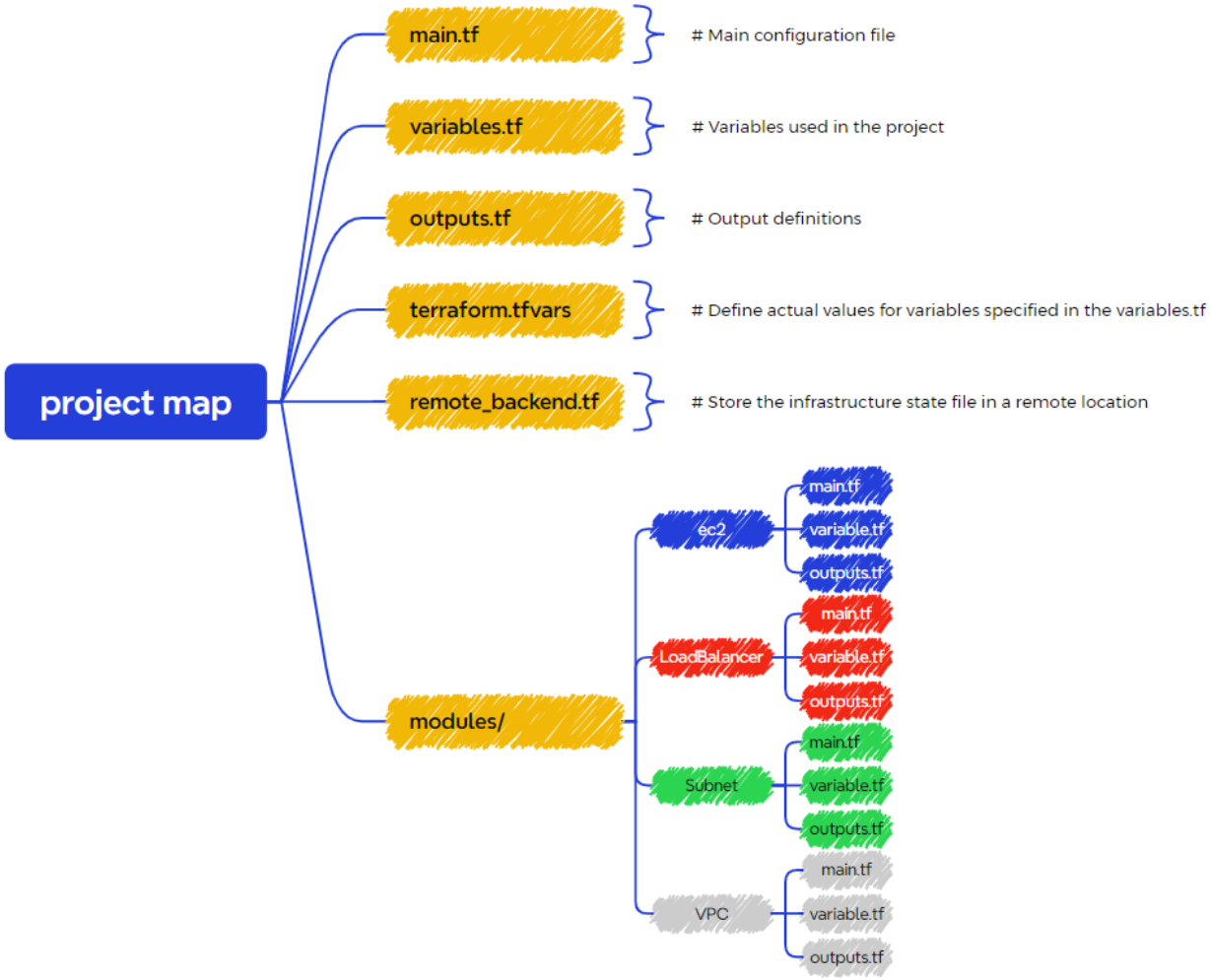
The project uses AWS services, including EC2, Elastic Load Balancers (ELB), and S3 for storing Terraform state files.



Prerequisites

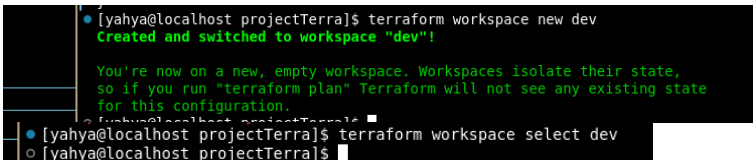
1. **Terraform** installed on your local machine.
2. **AWS CLI** configured with appropriate credentials.
3. An existing **S3 bucket** and **DynamoDB table** for remote state management.
4. **SSH key pair** for EC2 access.

Project Structure



Implementation Details

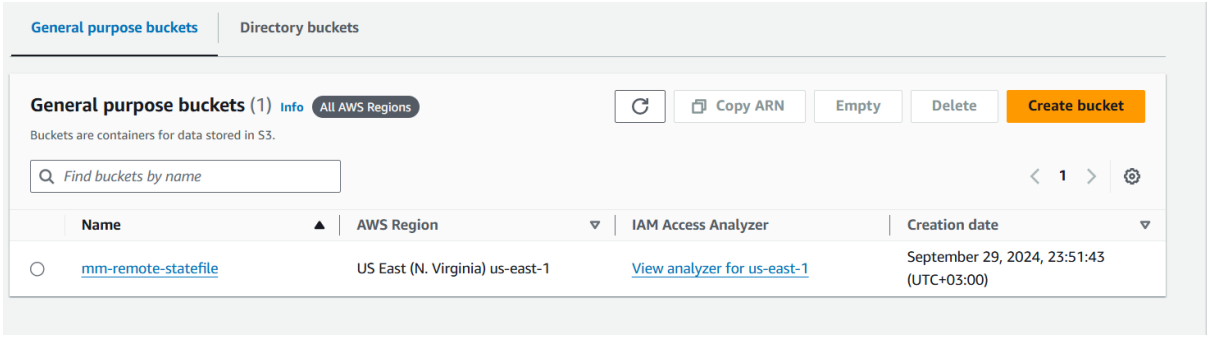
1. **Create a New Workspace**: Start by creating a new Terraform workspace called `dev`:



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

remote_backend.tf > resource "aws_dynamodb_table" "terraform_locks" > tags

```hcl
data "aws_s3_bucket" "existing_bucket" {
  bucket = "mm-remote-statefile"
}

resource "aws_s3_bucket" "terraform_state" {
  count  = length(data.aws_s3_bucket.existing_bucket.id) == 0 ? 1 : 0
  bucket = "mm-remote-statefile"
  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         = "NM-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         = "mm-remote-statefile"
    key            = "terraform.tfstate"
    region         = "us-east-1"
    dynamodb_table = "NM-locks"
    encrypt        = true
  }
}
```



3. **VPC**: Create a VPC using a custom VPC module.
    - creating a VPC module in `modules/VPC/main.tf`:

```
# CREATE A VPC
resource "aws_vpc" "vpc" {
  cidr_block = var.vpc_cidr
  tags = {
    Name = "N-VPC"
  }
}

# CREATE AN INTERNET GATEWAY
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.vpc.id
  tags = {
    Name = "N-IGW"
  }
}

# CREATE ELASTIC IP
resource "aws_eip" "eip" {
  domain   = "vpc"
}

# CREATE NAT IN THE FIRST PUBLIC SUBNET
resource "aws_nat_gateway" "nat" {
  allocation_id = aws_eip.eip.id
  subnet_id     = var.nat_subnet_id
  tags = {
    Name = "N-NAT"
  }
  # To ensure proper ordering, i will add an explicit dependency on the Internet Gateway.
  depends_on = [aws_internet_gateway.igw]
}
```

main.tf | outputs.tf

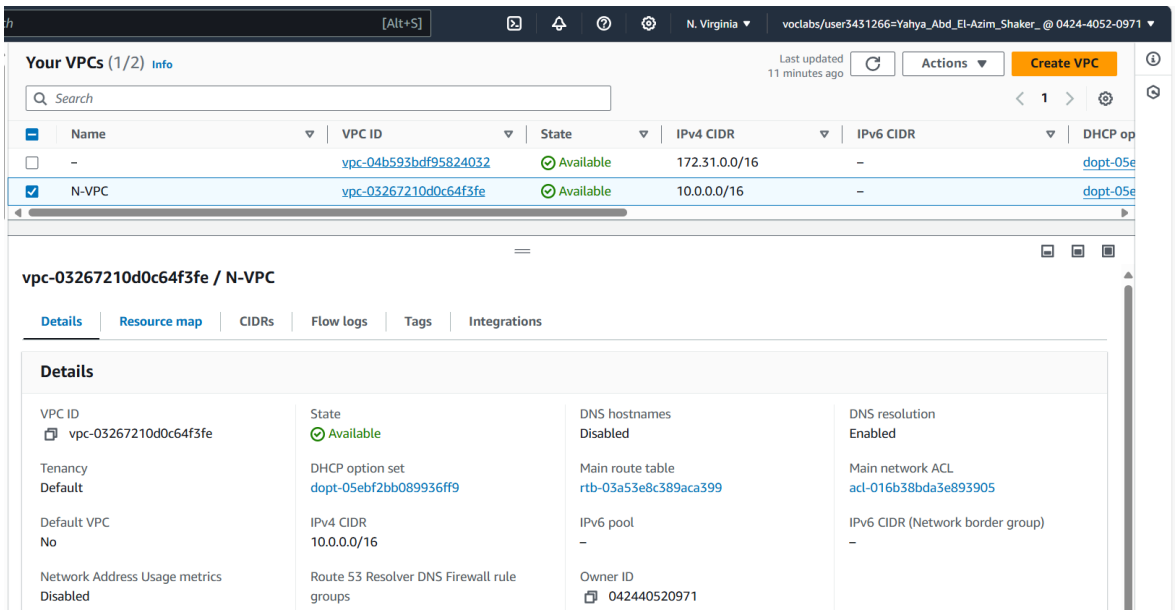
modules > VPC > outputs.tf > output "igw_id" > value

```
# OUTPUT TO RETRIVE VPC ID
output "vpc_id" {
    value = aws_vpc.vpc.id
}

# OUTPUT TO RETRIVE INTERNET GATEWAY ID
output "igw_id" {
    value = aws_internet_gateway.igw.id
}

#OUTPUT TO RETRIVE NAT ID
output "nat_id" {
    value = aws_nat_gateway.nat.id
}
```

main.tf | outputs.tf | variables.tf

modules > VPC > variables.tf > ...

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



4. **Subnets: create** public and private subnets using a custom **Subnet** module
    - Define subnets, route tables, and gateways as needed in `modules/Subnet/main.tf`

```hcl
resource "aws_subnet" "public_subnets" {
  count             = length(var.pub_subnets)
  vpc_id            = var.vpc_id
  cidr_block        = var.pub_subnets[count.index].subnets_cidr
  availability_zone = var.pub_subnets[count.index].availability_zone
  tags = {
    Name = "sprints_public_subnet_${count.index}"
  }
}

resource "aws_route_table" "public-rt" {
  vpc_id = var.vpc_id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = var.igw_id
  }
}

resource "aws_route_table_association" "public-rta" {
  count          = length(aws_subnet.public_subnets)
  subnet_id      = aws_subnet.public_subnets[count.index].id
  route_table_id = aws_route_table.public-rt.id
}

resource "aws_subnet" "private_subnets" {
  count             = length(var.priv_subnets)
  vpc_id            = var.vpc_id
  cidr_block        = var.priv_subnets[count.index].subnets_cidr
  availability_zone = var.priv_subnets[count.index].availability_zone
  tags = {
    Name = "sprints_private_subnet_${count.index}"
  }
}

resource "aws_route_table" "private-rt" {
  vpc_id = var.vpc_id
  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = var.nat_id
  }
}

resource "aws_route_table_association" "private-rta" {
  count          = length(aws_subnet.private_subnets)
  subnet_id      = aws_subnet.private_subnets[count.index].id
  route_table_id = aws_route_table.private-rt.id
}
```

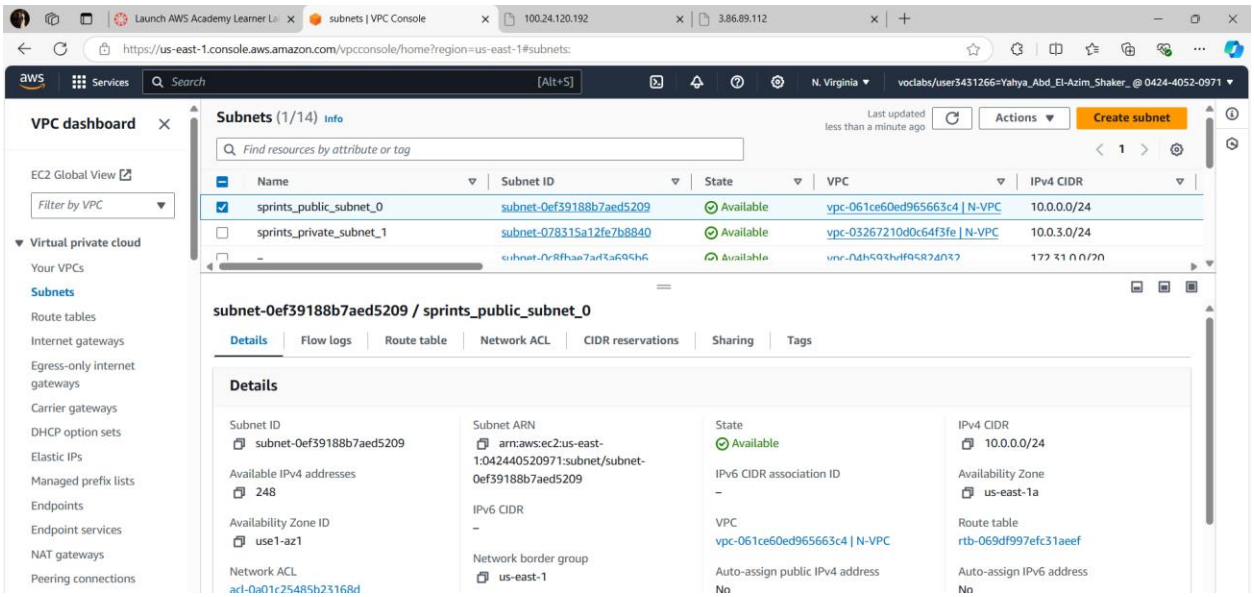
main.tf | outputs.tf

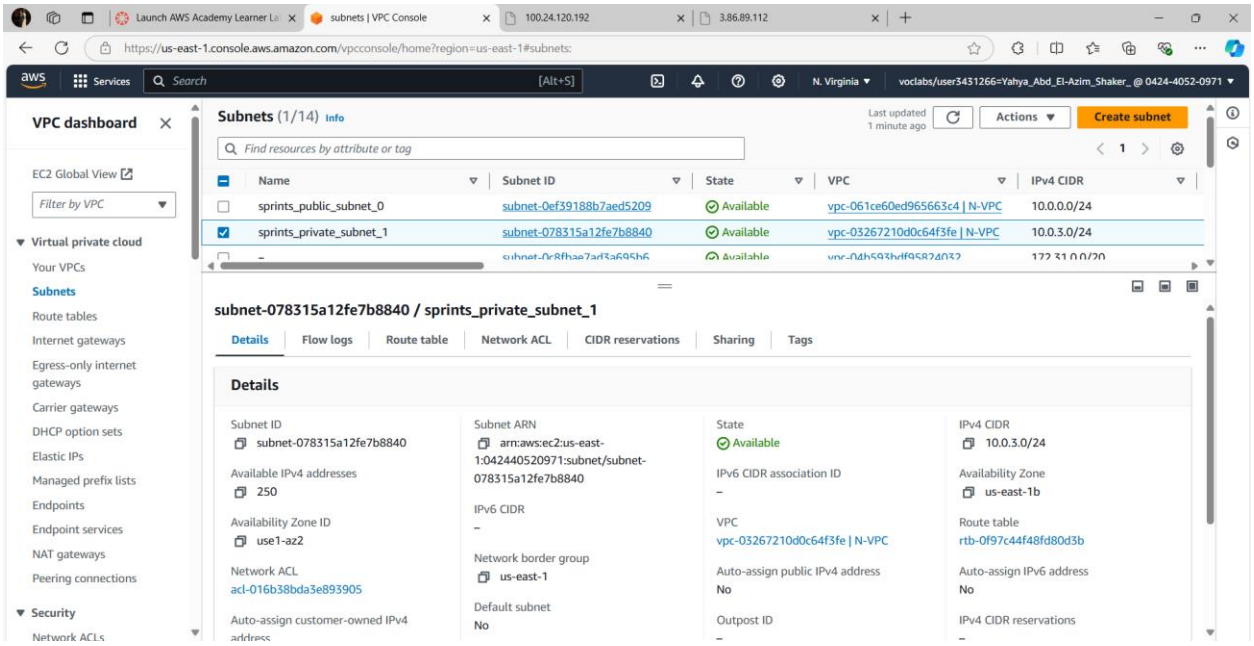
modules > Subnet > outputs.tf > output "private_subnets_id"

```hcl
output "public_subnets_id" {
    value = aws_subnet.public_subnets[*].id
}

output "private_subnets_id" {
    value = aws_subnet.private_subnets[*].id
}
```

modules > Subnet > variables.tf > variable "vpc_id"

```hcl
variable "vpc_id"{
    description = "ID of the vpc in where the subnets will be"
    type        = string
}

variable "pub_subnets"{
  description = "public subnets info"
  type        = list(object({
    subnets_cidr      = string
    availability_zone = string
  }))
}

variable "igw_id"{
    description = "ID of the IGW to be used in route table for public subnets"
    type = string
}

variable "priv_subnets"{
description = "private subnets info"
  type = list(object({
    subnets_cidr = string
    availability_zone = string
  }))
}

variable "nat_id"{
    description = "ID of the NAT to be used in route table for private subnets"
    type = string
}
```


Launch AWS Academy Learner La
subnets | VPC Console
100.24.120.192
3.86.89.112
https://us-east-1.console.aws.amazon.com/vpcconsole/home?region=us-east-1#subnets:
Services
Search
[Alt+S]
N. Virginia
voclabs/user3431266=Yahya_Abd_El-Azim_Shaker_ @ 0424-4052-0971
VPC dashboard
EC2 Global View
Filter by VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways
Peering connections
Subnets (1/14) Info
Last updated less than a minute ago
Actions
Create subnet
Find resources by attribute or tag
Name
Subnet ID
State
VPC
IPv4 CIDR
sprints_public_subnet_0
subnet-0ef39188b7aed5209
Available
vpc-061ce60ed965663c4 | N-VPC
10.0.0.0/24
sprints_private_subnet_1
subnet-078315a12fe7b8840
Available
vpc-03267210d0c64f3fe | N-VPC
10.0.3.0/24
subnet-0ef39188b7aed5209 / sprints_public_subnet_0
Details
Flow logs
Route table
Network ACL
CIDR reservations
Sharing
Tags
Details
Subnet ID
subnet-0ef39188b7aed5209
Subnet ARN
arn:aws:ec2:us-east-1:042440520971:subnet/subnet-0ef39188b7aed5209
State
Available
IPv4 CIDR
10.0.0.0/24
Available IPv4 addresses
248
IPv6 CIDR association ID
–
Availability Zone
us-east-1a
Availability Zone ID
use1-az1
IPv6 CIDR
–
VPC
vpc-061ce60ed965663c4 | N-VPC
Route table
rtb-069df997efc31aeef
Network ACL
Network border group
us-east-1
Auto-assign public IPv4 address
No
Auto-assign IPv6 address
No


Launch AWS Academy Learner La
subnets | VPC Console
100.24.120.192
3.86.89.112
https://us-east-1.console.aws.amazon.com/vpcconsole/home?region=us-east-1#subnets:
Services
Search
[Alt+S]
N. Virginia
voclabs/user3431266=Yahya_Abd_El-Azim_Shaker_ @ 0424-4052-0971
VPC dashboard
EC2 Global View
Filter by VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways
Peering connections
Security
Network ACLs
Subnets (1/14) Info
Last updated 1 minute ago
Actions
Create subnet
Find resources by attribute or tag
Name
Subnet ID
State
VPC
IPv4 CIDR
sprints_public_subnet_0
subnet-0ef39188b7aed5209
Available
vpc-061ce60ed965663c4 | N-VPC
10.0.0.0/24
sprints_private_subnet_1
subnet-078315a12fe7b8840
Available
vpc-03267210d0c64f3fe | N-VPC
10.0.3.0/24
subnet-078315a12fe7b8840 / sprints_private_subnet_1
Details
Flow logs
Route table
Network ACL
CIDR reservations
Sharing
Tags
Details
Subnet ID
subnet-078315a12fe7b8840
Subnet ARN
arn:aws:ec2:us-east-1:042440520971:subnet/subnet-078315a12fe7b8840
State
Available
IPv4 CIDR
10.0.3.0/24
Available IPv4 addresses
250
IPv6 CIDR association ID
–
Availability Zone
us-east-1b
Availability Zone ID
use1-az2
IPv6 CIDR
–
VPC
vpc-03267210d0c64f3fe | N-VPC
Route table
rtb-0f97c44f48fd80d3b
Network ACL
acl-016b38bda3e893905
Network border group
us-east-1
Auto-assign public IPv4 address
No
Auto-assign IPv6 address
No
Auto-assign customer-owned IPv4 address
Default subnet
No
Outpost ID
–
IPv4 CIDR reservations
–

5. **EC2 Instances**:
    - Create EC2 instances using a custom EC2 module.
    - Use a `data` source to get the AMI ID for EC2:

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

    - Create the security group

```
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"
  }
}

```

    - Create public instance

```
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}"
    }
```

    - Use remote-exec provisioners to install Apache server and configure the web servers:

```
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 yahya 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/kk.pem")
   timeout     = "5m"
 }
}
```

- Use local-exec

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

- Create private instance

```
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 to set up Apache and configure index.html
 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
```

```
 # Local provisioner to log the private IPs of created instancesyes

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

main.tf | outputs.tf

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

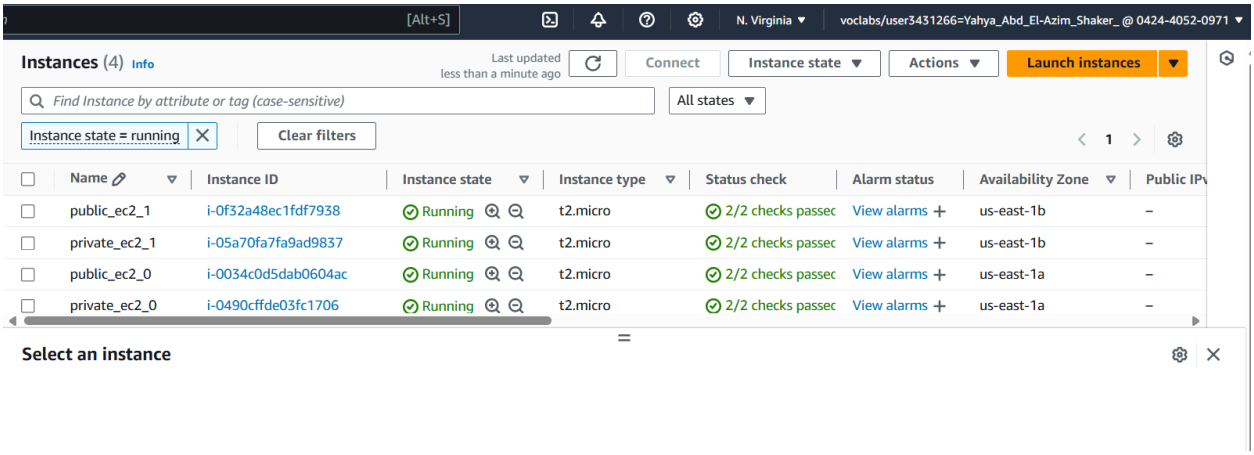
```
output "public_ec2_id" {
  value = aws_instance.pub-ec2[*].id
}

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 | outputs.tf | variables.tf

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

```
variable "ec2_html" {
  default = [
    "Welcome to Private EC2 Instance 2"
  ]
}
variable "key_pair_name" {
  description = "Name of the EC2 Key Pair"
  type        = string
  default = "kk"
}
```

The final result:



6. **Load Balancers**:
   - Create a public and private load balancer using a custom module.
   - Configure the target groups and listeners to forward traffic to the EC2 instances.

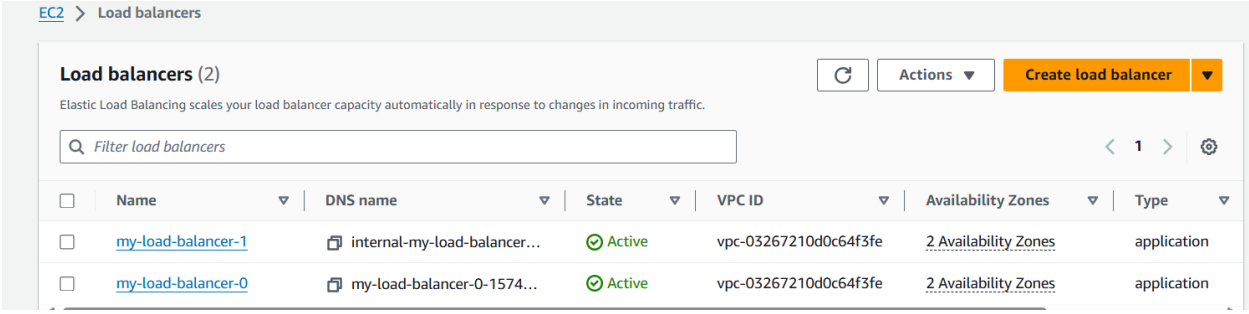
modules > LB > main.tf > ...

```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                       = "my-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
    }
}
```

modules > LB > outputs.tf > output "private_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
}
```

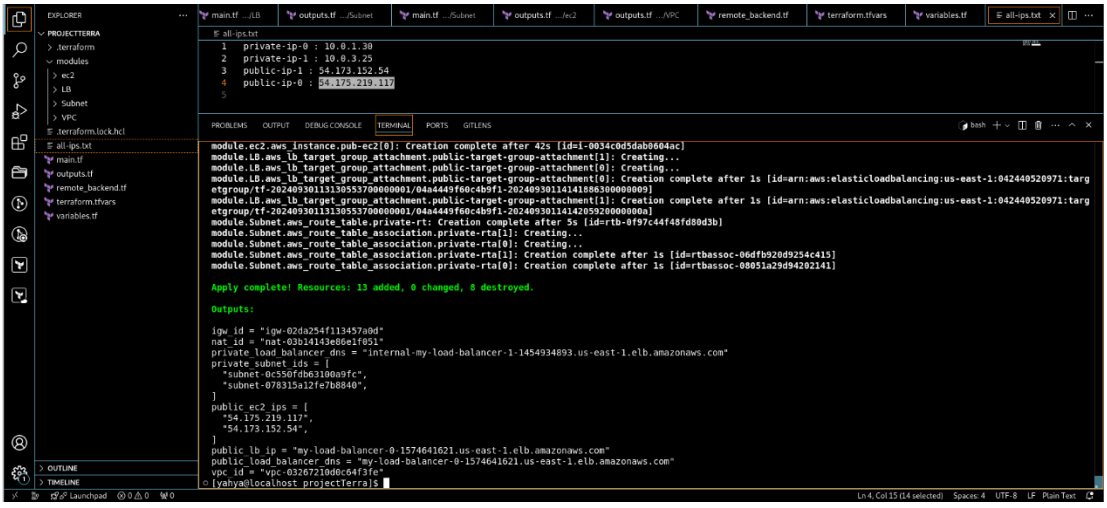
```hcl
variable "lb_vpc_id"{
    type = string
}
variable "pub_target_id"{
    type = list(string)
}
variable "priv_target_id"{
    type = list(string)
}
variable "lb_internal"{
    type = list(bool)
}
variable "lb_subnets"{
    type = list(list(string))
}
variable "lb_sg_id"{}
```



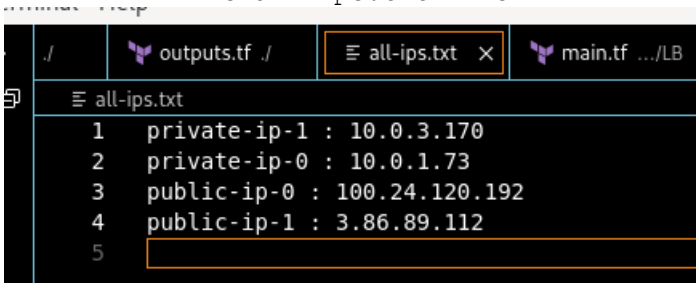
7. **Output Values**:
    - Define output values for the public IP addresses and DNS names of the load balancers.
    - Examples
        - **Public Load Balancer DNS**: The DNS name of the public load balancer.
        - **Private Load Balancer DNS**: The DNS name of the private load balancer.
        - **EC2 Public IPs**: The public IP addresses of EC2 instances created in the public subnets.

```hcl
output "vpc_id" {
  description = "The ID of the VPC: "
  value       = module.VPC.vpc_id
}
output "igw_id" {
  description = "The ID of the Internet Gateway: "
  value       = module.VPC.igw_id
}
output "nat_id" {
  description = "The ID of the NAT Gateway: "
  value       = module.VPC.nat_id
}
output "public_load_balancer_dns" {
  description = "Public Load Balancer DNS name: "
  value       = module.LB.public_load_balancer_dns
}
output "private_load_balancer_dns" {
  description = "Private Load Balancer DNS name: "
  value       = module.LB.private_load_balancer_dns
}
output "public_ec2_ips" {
  description = "Public IPs of the EC2 instances: "
  value       = module.ec2.pub-ips
}
output "private_subnet_ids" {
  description = "Private subnet IDs: "
  value       = module.Subnet.private_subnets_id
}
output "current_workspace" {
  description = "The current Terraform workspace being used."
  value       = terraform.workspace
}

```

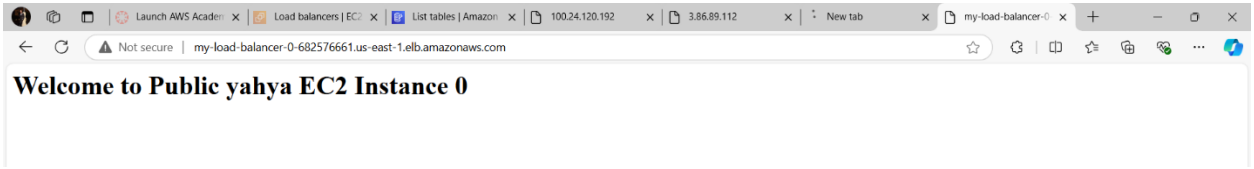
The final Result:

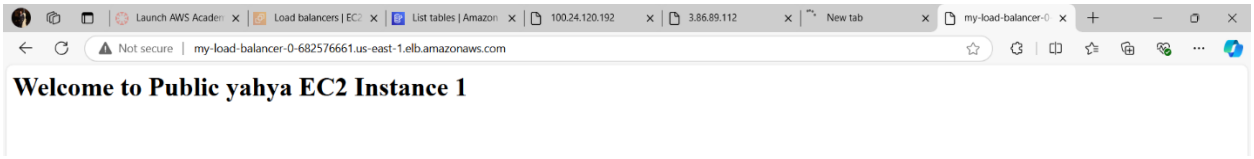


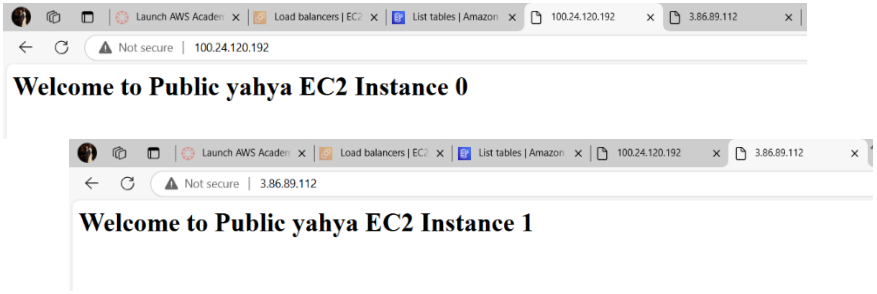
The all-ips.txt file



8. **Testing**:
    - Access the public DNS of the load balancer and verify it forwards traffic to the proxy.







Conclusion

This documentation outlines the steps to implement a complex AWS infrastructure using Terraform. The project is structured to separate concerns into modules, making it easier to manage and scale