Building APPLICATION LOAD BALANCER with Terraform

Elastic Load Balancing automatically distributes your incoming traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. It monitors the health of its registered targets, and routes traffic only to the healthy targets. Elastic Load Balancing scales your load balancer as your incoming traffic changes over time. It can automatically scale to the vast majority of workloads.

Elastic Load Balancing supports the following load balancers: Application Load Balancers, Network Load Balancers, and Gateway Load Balancers.

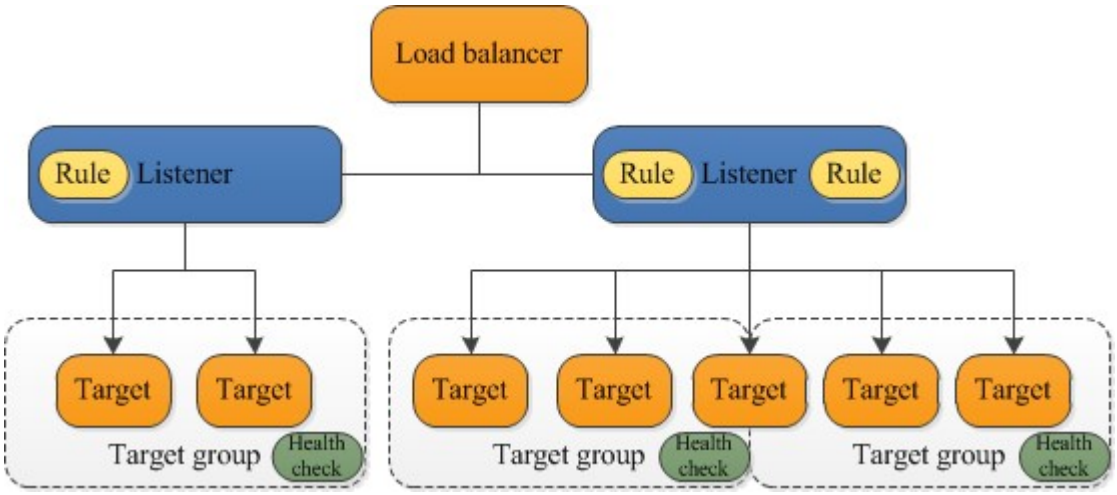
Application Load Balancer components

A load balancer serves as the single point of contact for clients. The load balancer distributes incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones. This increases the availability of your application. You add one or more listeners to your load balancer.

A listener checks for connection requests from clients, using the protocol and port that you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets. Each rule consists of a priority, one or more actions, and one or more conditions. When the conditions for a rule are met, then its actions are performed. You must define a default rule for each listener, and you can optionally define additional rules.

Each target group routes requests to one or more registered targets, such as EC2 instances, using the protocol and port number that you specify. You can register a target with multiple target groups. You can configure health checks on a per target group basis. Health checks are performed on all targets registered to a target group that is specified in a listener rule for your load balancer.

The following diagram illustrates the basic components. Notice that each listener contains a default rule, and one listener contains another rule that routes requests to a different target group. One target is registered with two target groups.



Terraform

Variable.tf

```
variable "access_key" {
   type = string
   description = "accesskey"
   default= "AKIA6LFFGBWBQ7YNMV2"
}

variable "region" {
   default = "us-east-1"
}

variable "secret_key" {
   type = string
   description = "secretkey"
   default= "ewjax30Ltck8MOJ28dUDdZKYmooUaDVPgQ19I3"
}
```

vpc.tf

```
resource "aws_vpc" "vpc" {
  cidr_block = "192.178.0.0/16"

  tags = {
     Name = "VPC"
    }
```

```
}
#PUBLIC SUBNET
resource "aws_subnet" "public" {
 vpc_id     = aws_vpc.vpc.id
 map_public_ip_on_launch = true
 cidr_block = "192.178.1.0/24"
 availability_zone = "us-east-1a"


 tags = {
  Name = "public"
 }
}

resource "aws_subnet" "public2" {
 vpc_id     = aws_vpc.vpc.id
 map_public_ip_on_launch = true
 cidr_block = "192.178.2.0/24"
 availability_zone = "us-east-1b"

 tags = {
  Name = "public2"
 }
}

# internet gateway for public subnet

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

 tags = {
  Name = "internet-gateway-vpc"
 }
}

#root table and assosiation with subnet

resource "aws_route_table" "route-public" {
 vpc_id = aws_vpc.vpc.id

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

 tags = {
```

```
   Name = "public-route-table"
  }
}

resource "aws_route_table_association" "public" {
  subnet_id      = aws_subnet.public.id
  route_table_id = aws_route_table.route-public.id
}
```

Security Group.tf

```
resource "aws_security_group" "sg_vpc" {
  name        = "allow_SSH"
  description = "Allow SSH inbound traffic"
  vpc_id      = aws_vpc.vpc.id

  ingress {
    # SSH Port 22 allowed from any IP
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

    ingress {
      # SSH Port 22 allowed from any IP
      from_port   = 80
      to_port     = 80
      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"]
  }
}
```

ec2.tf

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

```
  access_key = var.access_key
  secret_key = var.secret_key
}
resource "aws_instance" "ec2_1" {
 ami = "ami-05fa00d4c63e32376"
 instance_type = "t2.micro"
 subnet_id = aws_subnet.public.id
 # Security group assign to instance
 vpc_security_group_ids = [aws_security_group.sg_vpc.id]

 # key name
 key_name = "08-09-2022"

 user_data = <<EOF
                #! /bin/bash
     sudo yum update -y
                sudo yum install -y httpd.x86_64
                sudo service httpd start
                sudo service httpd enable
                echo "<h1>Machine 1 </h1> <h2> Deployed  by Arun via Terraform </h2>" | sudo
tee /var/www/html/index.html
          EOF

 tags = {
  Name = "first Ec2 "
 }
}
resource "aws_instance" "ec2_2" {
 ami = "ami-05fa00d4c63e32376"
 instance_type = "t2.micro"
 subnet_id = aws_subnet.public.id
 # Security group assign to instance
 vpc_security_group_ids = [aws_security_group.sg_vpc.id]

 # key name
 key_name = "08-09-2022"

 user_data = <<EOF
                #! /bin/bash
     sudo yum update -y
                sudo yum install -y httpd.x86_64
                sudo service httpd start
                sudo service httpd enable
                echo "<h1>Machine 2 </h1> <h2> Deployed  by Arun via Terraform </h2>" | sudo
tee /var/www/html/index.html
          EOF
```

```
  tags = {
    Name = "second Ec2 "
  }
}
```

Main.tf

```
#ALB SG
resource "aws_security_group" "alb_sg" {
  name        = "ALB - SG"
  vpc_id      = aws_vpc.vpc.id

  ingress {
     # SSH Port 22 allowed from any IP
     from_port   = 80
     to_port     = 80
     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"]
  }
}
# ------------ Create AWS ALB -----------

resource "aws_lb" "alb" {
  name               = "alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb_sg.id]
  subnets            = [aws_subnet.public.id, aws_subnet.public2.id]

  enable_deletion_protection = false


  tags = {
    Environment = "load balancer"
  }
}
```

```
resource "aws_alb_target_group" "group" {
  name     = "alb-target"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.vpc.id
  stickiness {
    type = "lb_cookie"
  }
  # Alter the destination of the health check to be the login page.
  health_check {
    path = "/login"
    port = 80
  }
}
resource "aws_lb_target_group_attachment" "test" {
  target_group_arn = aws_alb_target_group.group.arn
  target_id        = aws_instance.ec2_1.id
  port             = 80
}
resource "aws_lb_target_group_attachment" "test2" {
  target_group_arn = aws_alb_target_group.group.arn
  target_id        = aws_instance.ec2_2.id
  port             = 80
}
# An example of a Listener
resource "aws_alb_listener" "my-alb-listener" {
  default_action {
    target_group_arn = aws_alb_target_group.group.arn
    type = "forward"
  }
  load_balancer_arn = aws_lb.alb.arn
  port = 80
  protocol = "HTTP"
}
```

Output.tf

```
output "instance_public_ip" {
  description = "Public IP address of the EC2 instance"
  value       = aws_instance.ec2_1.public_ip
}

output "instance_public_ip2" {
  description = "Public IP address of the 2nd EC2 instance"
  value       = aws_instance.ec2_2.public_ip
}
```

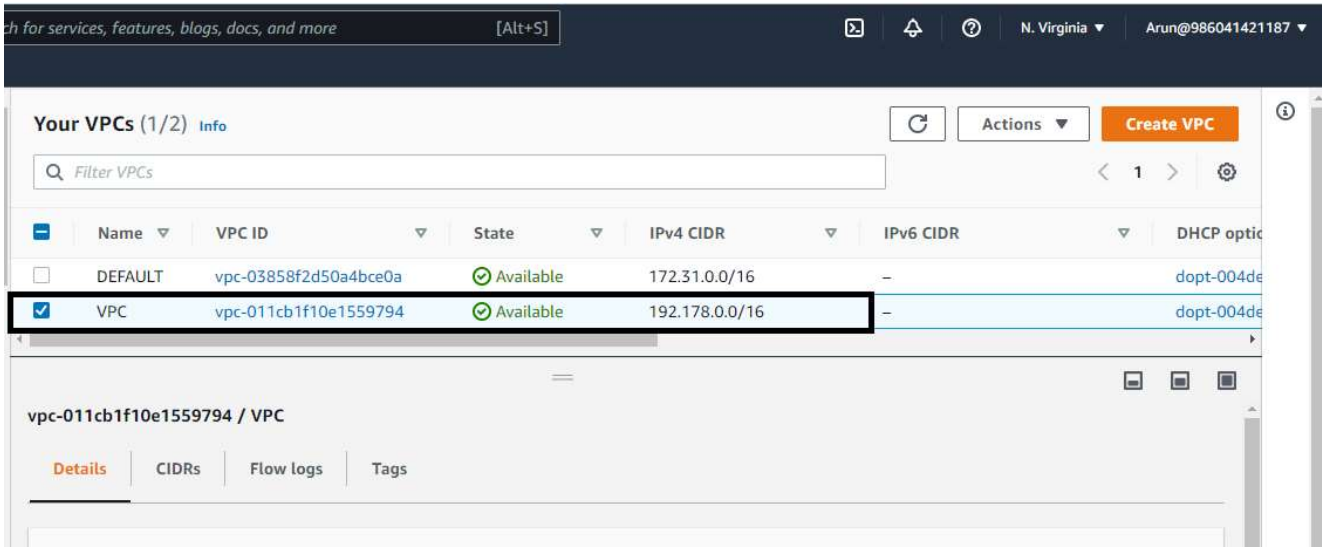
```
output "LB_DNS"{
  description = "DNS of ALB"
  value       = aws_lb.alb.dns_name
}
```

Screenshots Terminal output

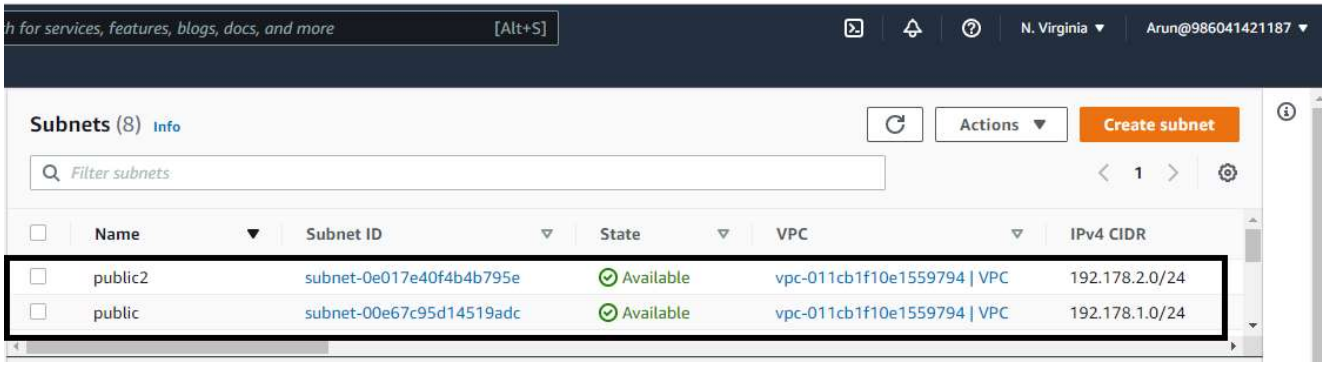
```
Apply complete! Resources: 15 added, 0 changed, 0 destroyed.

Outputs:

LB_DNS = "alb-1421878523.us-east-1.elb.amazonaws.com"
instance_public_ip = "54.175.175.206"
instance_public_ip2 = "3.87.91.140"
```

Screenshots of Resources

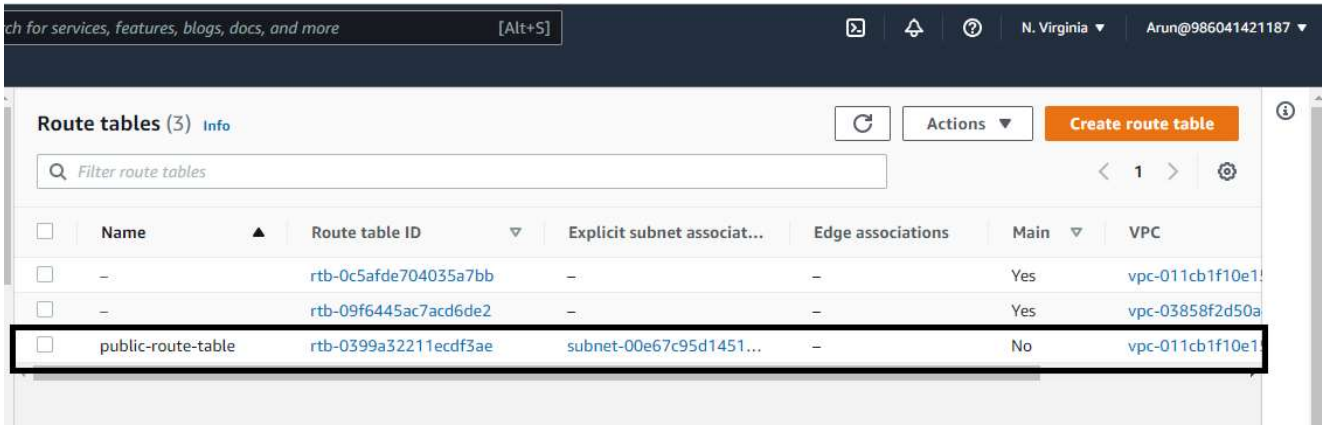
VPC



Subnets

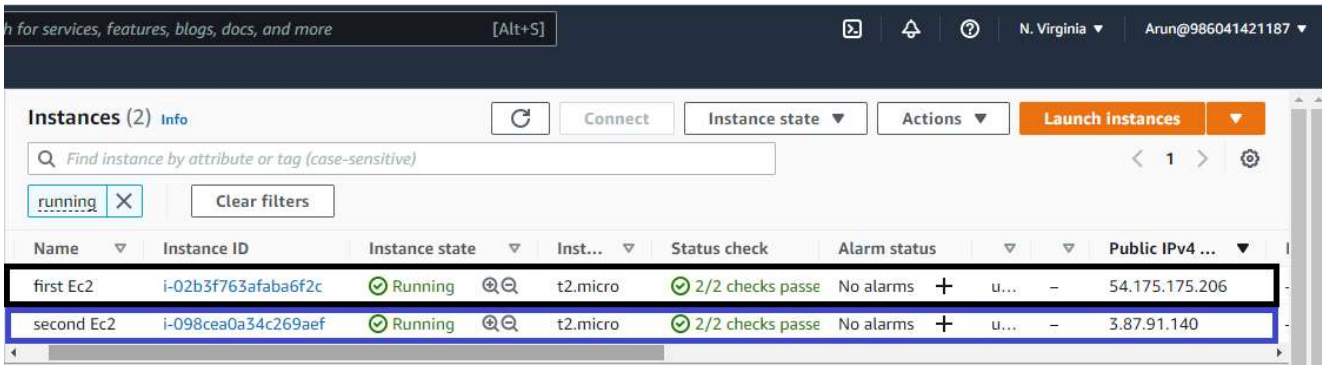


route table

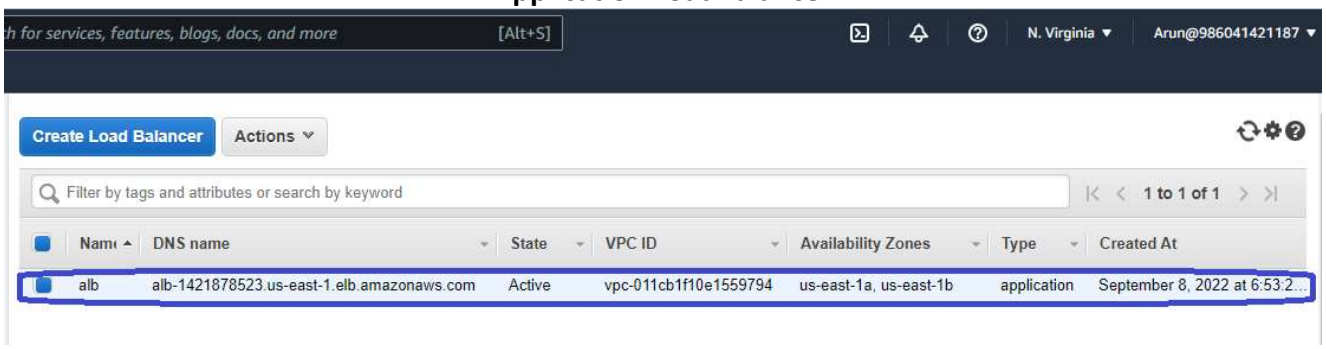


igw

EC2 instances



Application LoadBalancer



If we refresh again we get