Sai Pavan Phaneendra Vaddi
Saipavan2930@gmail.com
9515952930

Task: Create a Load balancer and attach it to two instances through terraform.

Prerequisites1

1. Installation of AWS Cli on server using below commands.
   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
   unzip awscliv2.zip
   sudo ./aws/install

2. Installation of Terraform on server using below commands.
   wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
   echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
   sudo apt update && sudo apt install terraform

3. Configure credentials using aws configure.
   - **AWS Access Key ID**: This is your AWS access key. You can generate one in your AWS Management Console under **IAM > Users > Security credentials**.
   - **AWS Secret Access Key**: This is the secret key associated with your access key. Keep it private and secure.
   - **Default region name**: Specify the AWS region where you want to create and manage resources. For example, us-east-1, eu-west-1, etc.
   - **Default output format**: Choose the output format (optional). Common formats are: json (default), text, table

Example:

$ aws configure

AWS Access Key ID [None]: XXXXXXXXXXXXX

AWS Secret Access Key [None]: XXXXXXXXXXXXXXXXXX

region name [None]: us-east-1

output format [None]: json

Step 1: Define Provider

Start by defining the AWS provider. The provider is the plugin that allows Terraform to interact with AWS.

Code:

```
provider "aws" {
  region = "us-east-1"  # Specify your preferred region
}
```

Step 2: Define Terraform block

In Terraform, the terraform block is used to configure settings related to Terraform itself. These settings can include backend configurations, required providers, and version constraints.

Code:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "5.69.0"
    }
  }
}
```

Step 3: Create Security Groups

We need security groups for both the EC2 instances and the load balancer to allow incoming and outgoing traffic.

Code:

```
# Security Group for EC2
resource "aws_security_group" "instance_sg" {
  name        = "instance-sg"
  description = "Allow traffic to EC2 instances"
  ingress {
```

```
    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"]
  }
}
#Security Group for Load Balancer
resource "aws_security_group" "lb_sg" {
  name        = "lb-sg"
  description = "Allow inbound traffic to Load Balancer"

  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"]
 }
}
```

Step 4: Create Two Instances

To create two instances, we will use the aws_instance resource in Terraform.

Code:

```
resource "aws_instance" "app" {
 count         = 2  # Create two instances
 ami           = "ami-0c55b159cbfafe1f0"
 instance_type = "t2.micro"
 security_groups = [aws_security_groups. instance_sg.name
 tags = {
          Name = "AppInstance-${count.index + 1}"
}
}
```

Step 5: Create the Target Group

A target group keeps track of the registered instances and their health status.

Code:

```
resource "aws_lb_target_group" "app_tg" {
 name     = "app-target-group"
```

```
  port     = 80
  protocol = "HTTP"
  vpc_id   = "vpc-abc12345"  # Replace with your VPC ID
}
```

Step 6: Attach Instances to Target Group

We will register the instances created in Step 2 with the target group.

Code:

```
resource "aws_lb_target_group_attachment" "app_tg_attachment" {
  count            = 2
  target_group_arn = aws_lb_target_group.app_tg.arn
  target_id        = aws_instance.app[count.index].id
  port             = 80
}
```

Step 4: Create the Load Balancer

Define the load balancer that will distribute traffic across the EC2 instances.

Code:

```
resource "aws_lb" "app_lb" {
  name               = "app-load-balancer"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.lb_sg.id]
  subnets            = ["subnet-abc12345", "subnet-def67890"]  # Use your subnets
}
```

Step 7: Create Load Balancer Listener

The listener defines how the load balancer routes incoming traffic to the target group.

Code:

```
resource "aws_lb_listener" "app_lb_listener" {
```

```
  load_balancer_arn = aws_lb.app_lb.arn
  port              = 80
  protocol          = "HTTP"

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

Step 8: Output the Load Balancer DNS

To easily access the load balancer after deployment, we can output the DNS name.

Code:

```
output "load_balancer_dns" {
  value = aws_lb.app_lb.dns_name
}
```

Step 9: Apply the Terraform Configuration using following commands.

terraform init – To initialize terraform

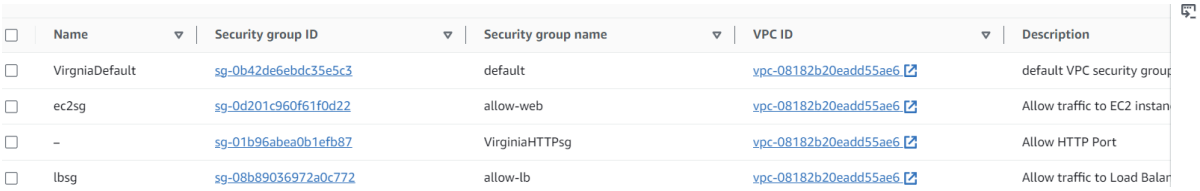
terraform validate - To validate the code

terraform plan - To plan the changes

terraform apply - To apply the Configuration:

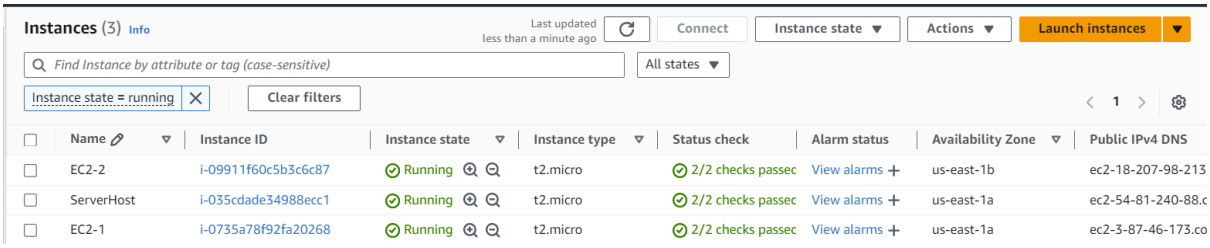
Outputs:

Security Groups:

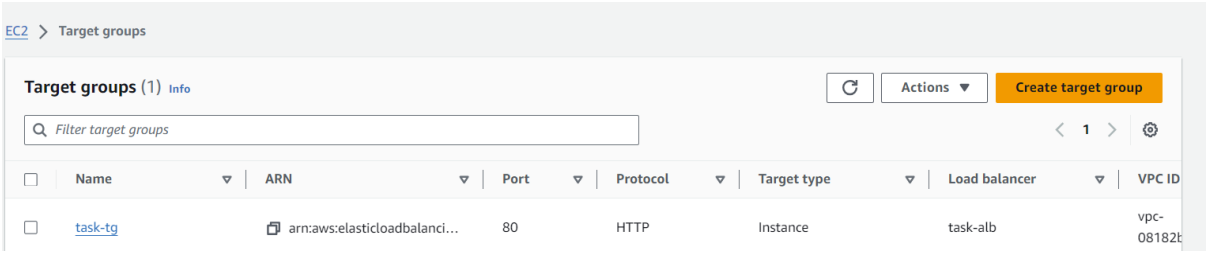


Name	Security group ID	Security group name	VPC ID	Description
VirgniaDefault	sg-0b42de6ebdc35e5c3	default	vpc-08182b20eadd55ae6	default VPC security group
ec2sg	sg-0d201c960f61f0d22	allow-web	vpc-08182b20eadd55ae6	Allow traffic to EC2 instan
–	sg-01b96abea0b1efb87	VirginiaHTTPsg	vpc-08182b20eadd55ae6	Allow HTTP Port
lbsg	sg-08b89036972a0c772	allow-lb	vpc-08182b20eadd55ae6	Allow traffic to Load Balar

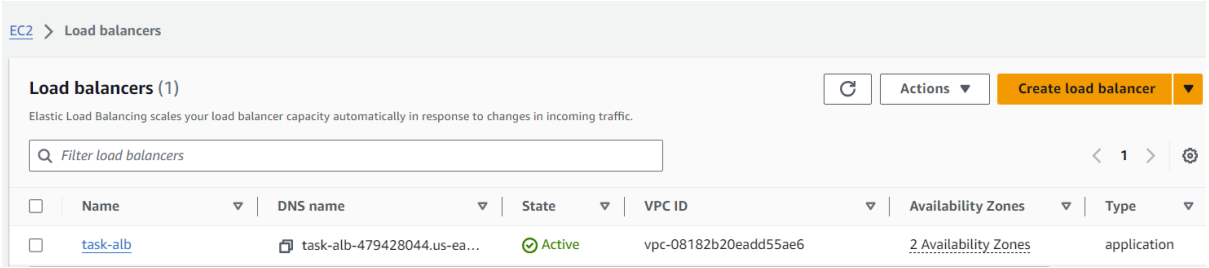
Instances:



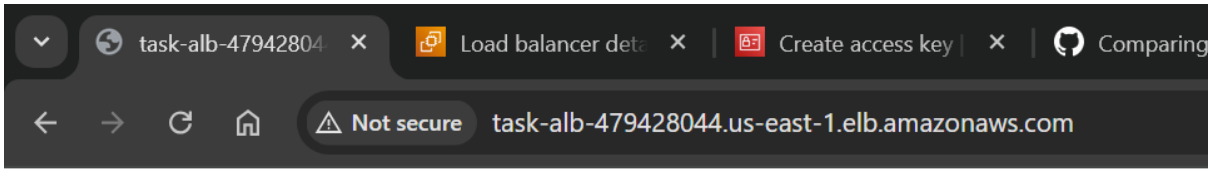
Target Groups:



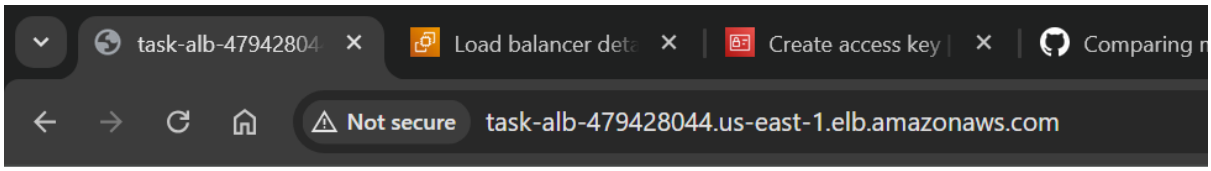
Load Balncer:



Server pages:



Welcome to WEBSERVER-1



Welcome to WEBSERVER-2