AWS Terraform Lab for DevOps Engineers

Written by Zayan Ahmed | 4 min read

Prerequisites

Before starting, ensure you have the following:

- **AWS Account**: An active AWS account with IAM access to provision resources.
- **Terraform**: Installed on your local machine (version 1.x or higher).
- **AWS CLI**: Configured with credentials.
- **Basic Knowledge**: Familiarity with Terraform commands and AWS services.

Lab Overview

In this lab, we will:

1. Define a provider for AWS.
2. Create a VPC with public and private subnets.
3. Launch an EC2 instance within the public subnet.
4. Set up a security group allowing SSH access to the EC2 instance.
5. Output the public IP address of the EC2 instance.

Main Terraform Configuration

Save the following code as `main.tf` in your working directory.

```
# Configure the AWS Provider
provider "aws" {
 region = "us-east-1"
}

# Define VPC
resource "aws_vpc" "devops_vpc" {
 cidr_block = "10.0.0.0/16"
 tags = {
   Name = "DevOpsLab-VPC"
 }
}

# Create Public Subnet
resource "aws_subnet" "public_subnet" {
 vpc_id            = aws_vpc.devops_vpc.id
```

```hcl
 cidr_block        = "10.0.1.0/24"
 availability_zone = "us-east-1a"
 map_public_ip_on_launch = true
 tags = {
   Name = "DevOpsLab-Public-Subnet"
 }
}

# Create Private Subnet
resource "aws_subnet" "private_subnet" {
 vpc_id            = aws_vpc.devops_vpc.id
 cidr_block        = "10.0.2.0/24"
 availability_zone = "us-east-1b"
 tags = {
   Name = "DevOpsLab-Private-Subnet"
 }
}

# Create an Internet Gateway
resource "aws_internet_gateway" "igw" {
 vpc_id = aws_vpc.devops_vpc.id
 tags = {
   Name = "DevOpsLab-IGW"
 }
}

# Route table for public subnet
resource "aws_route_table" "public_route_table" {
 vpc_id = aws_vpc.devops_vpc.id
 route {
   cidr_block = "0.0.0.0/0"
   gateway_id = aws_internet_gateway.igw.id
 }
 tags = {
   Name = "DevOpsLab-Public-Route-Table"
 }
}

# Associate public subnet with route table
resource "aws_route_table_association" "public_association" {
 subnet_id      = aws_subnet.public_subnet.id
 route_table_id = aws_route_table.public_route_table.id
}
```

```
# Security Group allowing SSH access
resource "aws_security_group" "sg_ssh" {
 vpc_id = aws_vpc.devops_vpc.id
 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 = "DevOpsLab-SG"
 }
}

# Launch EC2 Instance
resource "aws_instance" "devops_instance" {
 ami                         = "ami-0c55b159cbfafe1f0"  # Amazon Linux 2 AMI
 instance_type               = "t2.micro"
 subnet_id                   = aws_subnet.public_subnet.id
 vpc_security_group_ids       = [aws_security_group.sg_ssh.id]
 associate_public_ip_address = true
 key_name                    = var.key_name  # Ensure you have an existing key pair

 tags = {
   Name = "DevOpsLab-Instance"
 }
}

# Output public IP address of EC2 instance
output "instance_public_ip" {
 value = aws_instance.devops_instance.public_ip
}
```

Explanation

1. **Provider Configuration**: The AWS provider block specifies the region for provisioning resources (`us-east-1` in this case).
2. **VPC Creation**: The `aws_vpc` resource defines a VPC with a CIDR block of `10.0.0.0/16`, which gives us a range of private IP addresses.
3. **Subnets**: Two subnets are created:
   - A public subnet with the CIDR block `10.0.1.0/24`.
   - A private subnet with the CIDR block `10.0.2.0/24`.
4. **Internet Gateway and Route Tables**: The internet gateway allows external traffic to reach the instances in the public subnet, while the route table ensures the public subnet is correctly routed to the internet.
5. **Security Group**: The security group allows SSH access to the EC2 instance from any IP (`0.0.0.0/0`). You can restrict this for security reasons.
6. **EC2 Instance**: A `t2.micro` instance is created using the Amazon Linux 2 AMI, and the public IP is associated automatically.

Instructions to Run the Lab

1. Initialize Terraform:

   ```
   terraform init
   ```

2. Validate the configuration:

   ```
   terraform validate
   ```

3. Apply the configuration:

   ```
   terraform apply
   ```

4. When prompted, type `yes` to confirm the deployment. After a few minutes, Terraform will create the infrastructure.

5. Check the output to see the public IP of your EC2 instance. You can SSH into the instance using the key pair specified in the `var.key_name` variable.

Conclusion

This lab demonstrated how to use Terraform to deploy an AWS infrastructure, including a VPC, subnets, an EC2 instance, and security groups. By automating this process with Terraform, DevOps engineers can efficiently manage infrastructure changes, scaling, and monitoring. The skills covered in this lab are foundational for any DevOps role that requires AWS and Terraform expertise.

Follow me on **LinkedIn** 😊