Automated AWS VPC & EC2 Infrastructure Provisioning using Terraform(IaC)

1.0 Introduction

This document provides a comprehensive, step-by-step guide to provisioning a foundational Amazon Web Services (AWS) environment using HashiCorp Terraform. By leveraging the principles of Infrastructure as Code (IaC), we will automate the creation of a Virtual Private Cloud (VPC), associated networking components, and an EC2 instance. This approach ensures a repeatable, version-controlled, and secure infrastructure deployment process.

The guide moves beyond basic commands to incorporate industry best practices for security, scalability, and maintainability. Topics covered include secure credential management, state file handling, project structure, and the principle of least privilege. The objective is to build not just a functional environment, but a well-architected one suitable as a starting point for production workloads.

2.0 Prerequisites

Before proceeding, ensure the following prerequisites are met:

- **Terraform Installed:** Terraform (version 1.0.0 or later) must be installed on your local machine.

- **AWS Account:** An active AWS account with appropriate permissions to create VPC, EC2, and IAM resources.
- **AWS CLI Configured:** The AWS Command Line Interface (CLI) should be installed and configured with credentials. It is highly recommended to use IAM roles for authentication rather than long-lived user credentials, especially in production environments.

3.0 Foundational Setup

A well-structured project is crucial for long-term maintainability. Before writing resource definitions, we establish the provider configuration.

3.1 Provider file

`provider.tf` Contains the AWS provider configuration. This file tells Terraform which cloud provider to use and in which region to deploy the resources.

```
provider "aws" {
  region = "us-east-1" # Example region
}
```

4.0 Step-by-Step Infrastructure Provisioning

This section details the creation of each AWS resource as requested. The code snippets should be placed in a `main.tf` file.

4.1 Create a Virtual Private Cloud (VPC)

The VPC is the network foundation of your AWS environment, providing a logically isolated section of the AWS Cloud. Proper IP address planning is crucial for future growth and avoiding conflicts with other networks. We will create a VPC with the CIDR block `10.81.0.0/16`.

```
# main.tf

resource "aws_vpc" "proj_vpc" {
  cidr_block  = "10.81.0.0/16"
  tags = {
    Name = "proj-vpc"
  }
}
```

4.2 Create an Internet Gateway (IGW)

An Internet Gateway allows communication between your VPC and the internet. It is a horizontally scaled, redundant, and highly available VPC component.

```
resource "aws_internet_gateway" "proj_igw" {
  vpc_id = aws_vpc.proj-vpc.id

  tags = {
    Name = "proj-igw"
  }
}
```

4.3 Create a Custom Route Table

Route tables determine where network traffic from your subnets is directed. We create a custom route table and add a route to the Internet Gateway for public traffic (`0.0.0.0/0`). This makes any associated subnet a "public subnet."

```
resource "aws_route_table" "proj_rt" {
  vpc_id = aws_vpc.proj-vpc.id

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

  tags = {
    Name = "proj-rt"
  }
}
```

4.4 Create a Subnet

A subnet is a range of IP addresses in your VPC. Resources like EC2 instances are launched into subnets. For high availability, production applications should be deployed across subnets in multiple Availability Zones (AZs). This example creates a single public subnet.

Figure 1: A high-availability VPC architecture with public and private subnets in multiple Availability Zones. This design isolates workloads and provides fault tolerance.

```
resource "aws_subnet" "proj_subnet" {
  vpc_id                  = aws_vpc.proj-vpc.id
  cidr_block              = "10.81.1.0/24"
  tags = {
    Name = "proj-subnet"
  }
}
```

4.5 Associate Subnet with Route Table

To apply the routing rules defined in our custom route table, we must explicitly associate it with our subnet.

```
resource "aws_route_table_association" "proj-rt-assoc" {
  subnet_id      = aws_subnet.proj-subnet.id
  route_table_id = aws_route_table.proj-rt.id
}
```

4.6 Create a Security Group

Security Groups act as a stateful virtual firewall for your instances, controlling inbound and outbound traffic. The principle of least privilege dictates that you should only open ports that are absolutely necessary. For production, allowing traffic from `0.0.0.0/0` is highly discouraged. Instead, you should restrict the source to specific IP addresses (e.g., your office IP for SSH).

Best practice dictates separating the security group definition from its rules. This avoids destructive updates and allows rules to be managed independently, for example, by different modules.

Figure 2: Example of security group rules controlling traffic. Ingress rules for Subnet A allow SSH from a specific IP range, while rules for Subnet B allow SSH from Subnet A.

```
resource "aws_security_group" "proj_sg" {
 name        = "proj_sg"
 description = "Allow SSH, HTTP, HTTPS from anywhere"
 vpc_id      = aws_vpc.proj_vpc.id
 ingress {
 from_port   = 22
 to_port     = 22
 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   = 443
 to_port     = 443
 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 = "proj-sg"
 }
 }
```

4.7 Create a Network Interface (ENI)

A Network Interface can be created and attached to an EC2 instance. It allows you to have a persistent private IP address that can be moved between instances, which is useful for certain high-availability scenarios.

```
resource "aws_network_interface" "proj_nic" {
 subnet_id       = aws_subnet.proj_subnet.id
 private_ips     = ["10.81.1.10"]
 security_groups = [aws_security_group.proj_sg.id]
 tags = {
 Name = "proj-nic"
 }
 }
```

4.8 Assign an Elastic IP (EIP)

An Elastic IP is a static, public IPv4 address designed for dynamic cloud computing. By associating it with your network interface (and thus your instance), you provide a fixed public endpoint for your application.

```
 resource "aws_eip" "proj_eip" {
 vpc                 = true
 network_interface = aws_network_interface.proj_nic.id
 depends_on        = [aws_internet_gateway.proj_igw]
 tags = {
 Name = "proj-eip"
 }
 }
```

4.9 Create an EC2 Instance

Finally, we launch the EC2 instance. It is attached to the network interface created in the previous step. We use `user_data` to bootstrap a simple web server for demonstration purposes.

Security Best Practice: Avoid using password-based authentication. Use an EC2 Key Pair for initial access. More importantly, for application access to other AWS services, always assign an **IAM Role** to the instance profile instead of embedding access keys in your application or instance.

```
resource "aws_instance" "proj_ec2" {
 ami           = "ami-0c55b159cbfafe1f0" # Example Amazon
Linux 2 AMI
 instance_type = "t2.micro"
 user_data     = file("userdata.sh")
 network_interface {
 device_index         = 0
 network_interface_id = aws_network_interface.proj_nic.id
 }
 tags = {
 Name = "proj-ec2"
 }
 }

 #!/bin/bash
 yum update -y
 yum install -y httpd
 systemctl start httpd
 systemctl enable httpd
 echo "<h1>Hello from Terraform! This is my web application.
</h1>" > /var/www/html/index.html
```

5.0 Project Deployment and Validation

1. Initialize Terraform

Run terraform init to download the AWS provider plugin and initialize the working directory.

```
 terraform init
```

***2.* Review the Plan**

Run terraform plan to see a preview of the resources that Terraform will create. This is a crucial step to verify changes before applying them.

```
terraform plan
```

3. Apply the Configuration

If the plan looks correct, run terraform apply to create the resources. Terraform will prompt for confirmation.

```
terraform apply
```

---

4. Access the Web Application

Once the deployment is complete, Terraform will output the public IP of the EC2 instance. Open a web browser and navigate to http://<your-ec2-public-ip>You should see the "Hello from Terraform!" message.

5. Clean Up (Optional)

When you are finished, you can destroy all the created resources using terraform destroy to avoid incurring unnecessary costs. This will remove everything defined in your configuration.

```
terraform destroy
```