Terraform Practice Exercises for DevOps Engineers

Exercise 1: Provision an EC2 Instance on AWS

Question:
Provision an EC2 instance using Terraform on AWS. Define the AMI, instance type, and key pair as variables.

Answer:

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

# Define variables
variable "ami" {
 default = "ami-0c55b159cbfafe1f0"
}

variable "instance_type" {
 default = "t2.micro"
}

variable "key_name" {
 default = "my-key"
}

# EC2 instance resource
resource "aws_instance" "example" {
 ami           = var.ami
 instance_type = var.instance_type
 key_name      = var.key_name
}
```

```
# Output the public IP
output "instance_ip" {
 value = aws_instance.example.public_ip
}
```

Steps to follow:
1. Initialize Terraform: `terraform init`
2. Validate: `terraform validate`
3. Apply: `terraform apply`
4. Check the output for the EC2 instance's public IP.

Exercise 2: Create a VPC with Subnets

Question:
Using Terraform, create a VPC with two subnets (public and private). Ensure that the public subnet has access to the internet via an internet gateway.

Answer:

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

# VPC
resource "aws_vpc" "example_vpc" {
 cidr_block = "10.0.0.0/16"
}

# Internet Gateway
resource "aws_internet_gateway" "example_igw" {
 vpc_id = aws_vpc.example_vpc.id
}

# Public Subnet
resource "aws_subnet" "public_subnet" {
 vpc_id            = aws_vpc.example_vpc.id
 cidr_block        = "10.0.1.0/24"
 map_public_ip_on_launch = true
```

```
}

# Private Subnet
resource "aws_subnet" "private_subnet" {
 vpc_id            = aws_vpc.example_vpc.id
 cidr_block        = "10.0.2.0/24"
}

# Route Table for Public Subnet
resource "aws_route_table" "public_route_table" {
 vpc_id = aws_vpc.example_vpc.id
}

resource "aws_route" "public_route" {
 route_table_id         = aws_route_table.public_route_table.id
 destination_cidr_block = "0.0.0.0/0"
 gateway_id             = aws_internet_gateway.example_igw.id
}

# Associate Route Table with Public Subnet
resource "aws_route_table_association" "public_association" {
 subnet_id     = aws_subnet.public_subnet.id
 route_table_id = aws_route_table.public_route_table.id
}
```

Steps to follow
1. Initialize Terraform: `terraform init`
2. Apply the configuration: `terraform apply`
3. Verify the VPC and subnet configuration in the AWS console.

Exercise 3: S3 Bucket with Versioning and Encryption

Question:
Create an S3 bucket with versioning and server-side encryption enabled using Terraform.

Answer:

```
# S3 bucket resource
resource "aws_s3_bucket" "example_bucket" {
 bucket = "my-example-bucket"
 acl    = "private"
```

```
 versioning {
   enabled = true
 }

 server_side_encryption_configuration {
   rule {
   apply_server_side_encryption_by_default {
       sse_algorithm = "AES256"
   }
   }
 }
}

# Output the bucket name
output "bucket_name" {
 value = aws_s3_bucket.example_bucket.bucket
}
```

Steps to follow:
1. Initialize Terraform: `terraform init`
2. Apply: `terraform apply`
3. Check the AWS console for the S3 bucket, versioning, and encryption settings.

Exercise 4: Deploy a Simple Web Server on EC2

Question:
Deploy a simple web server on an EC2 instance. Use user data to automatically install and run a web server.

Answer:

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

# EC2 instance
resource "aws_instance" "web_server" {
 ami           = "ami-0c55b159cbfafe1f0"
 instance_type = "t2.micro"
 key_name      = "my-key"
```

```
 user_data = <<-EOF
   #!/bin/bash
   sudo yum update -y
   sudo yum install -y httpd
   sudo systemctl start httpd
   sudo systemctl enable httpd
 EOF

 tags = {
   Name = "WebServer"
 }
}

# Output the public IP
output "web_server_ip" {
 value = aws_instance.web_server.public_ip
}
```

Steps to follow:
1. Initialize Terraform: `terraform init`
2. Apply: `terraform apply`
3. Use the public IP of the instance to access the web server via a browser.

Exercise 5: Use Terraform to Manage IAM Roles and Policies

Question:
Create an IAM role for EC2 with a policy that allows access to S3.

Answer:

```
# IAM Role
resource "aws_iam_role" "ec2_role" {
 name = "ec2_role"

 assume_role_policy = jsonencode({
   Version = "2012-10-17"
   Statement = [
   {
       Action = "sts:AssumeRole"
       Effect = "Allow"
```

```
        Principal = {
        Service = "ec2.amazonaws.com"
        }
    }
    ]
 })
}

# IAM Policy
resource "aws_iam_policy" "s3_policy" {
 name = "s3_access_policy"

 policy = jsonencode({
   Version = "2012-10-17"
   Statement = [
   {
        Action = [
        "s3:ListBucket",
        "s3:GetObject"
        ]
        Effect = "Allow"
        Resource = [
        "arn:aws:s3:::my-bucket",
        "arn:aws:s3:::my-bucket/*"
        ]
    }
    ]
 })
}

# Attach Policy to Role
resource "aws_iam_role_policy_attachment" "attach_s3_policy" {
 role       = aws_iam_role.ec2_role.name
 policy_arn = aws_iam_policy.s3_policy.arn
}
```

Steps to follow:
1. Initialize Terraform: `terraform init`
2. Apply: `terraform apply`
3. Verify the IAM role and policy in the AWS console.

Exercise 6: Create an RDS MySQL Instance

Question:
Provision an Amazon RDS MySQL instance using Terraform with an allocated storage size of 20 GB and a db instance class of `db.t2.micro`.

Answer:

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

resource "aws_db_instance" "example_db" {
 allocated_storage = 20
 engine            = "mysql"
 engine_version    = "5.7"
 instance_class    = "db.t2.micro"
 name              = "exampledb"
 username          = "admin"
 password          = "password"
 parameter_group_name = "default.mysql5.7"
 skip_final_snapshot  = true
}

output "db_endpoint" {
 value = aws_db_instance.example_db.endpoint
}
```

Steps to follow:
1. Initialize Terraform: `terraform init`
2. Apply: `terraform apply`
3. Retrieve the DB endpoint from the output.