Terraform Part-2

A **Basic setup** to create a single EC2 instance, and an **advanced setup** to create multiple EC2 instances with state management stored in S3 and DynamoDB.

The documentation will also explain how to parameterize the configuration using variables for easier reuse.

---

1. Prerequisites

Before you begin, ensure the following:

- **AWS Account:** You must have an AWS account with access to create EC2 instances, VPCs, and other related resources.
- **Terraform Installed:** Terraform should be installed on your local machine. You can verify this by running:

```
terraform -v
```

If not installed, follow the instructions here to install Terraform.

- **AWS CLI Configured:** The AWS CLI should be configured with your credentials. You can set it up using the following command:

```
aws configure
```

Make sure you have a VPC with at least one public subnet, a security group, and an SSH key pair available in the region you wish to deploy the EC2 instance.

---

2. Basic Setup: Single EC2 Instance

This section explains how to create a basic configuration for a single EC2 instance.

Step 1: Create main.tf

The main.tf file contains the core configuration that defines resources like EC2 instances. Follow these steps:

1. **Define the Provider**: The provider block specifies which cloud provider (in this case, AWS) you are using and in which region you want to deploy your resources.
2. **Define the Resource**: The resource block creates an EC2 instance, and you can customize its properties like the Amazon Machine Image (AMI), instance type, and other configurations.
3. **Add Outputs**: Outputs allow you to access useful information, like the public IP and instance ID, after deployment.

Example of main.tf:

```
provider "aws" {
  region = "ap-south-1"  # Specify the AWS region
}

# EC2 instance resource
resource "aws_instance" "test" {
  ami           = "ami-0522ab6e1ddcc7055"    # The provided AMI (Amazon Machine Image)
  instance_type = "t2.medium"               # EC2 instance type
  key_name      = "DevOps-Shack"            # Key pair for SSH access

  # Security group and subnet for the EC2 instance
  vpc_security_group_ids = ["sg-04c97b7b938f67045"]  # Security Group ID from your VPC
  subnet_id              = "subnet-0ead1fda61faa7009"  # Subnet ID from your VPC

  # Storage configuration
  root_block_device {
    volume_size = 20      # Size of the root volume in GiB
    volume_type = "gp2"    # General Purpose SSD (gp2)
  }

  # Tags (Optional, but useful for identifying resources)
  tags = {
    Name = "DevOps-Shack-Instance"
  }
}

# Output the public IP address of the instance
output "instance_public_ip" {
  value = aws_instance.test.public_ip
}

# Output the instance ID
output "instance_id" {
  value = aws_instance.test.id
}
```

Explanation:

- **Provider Block**: Specifies AWS as the provider and the ap-south-1 region (you can change this based on your needs).
- **Resource Block**: Defines an EC2 instance using an AMI, an instance type (t2.medium), and attaches a security group and subnet from your VPC.
- **Storage Configuration**: The root block device specifies the size and type of the EBS volume.
- **Tags**: Tags are used to label your resources (optional but recommended).
- **Outputs**: These provide details about the instance, such as the public IP and instance ID, after deployment.

Step 2: Initialize Terraform

Before applying the configuration, you need to initialize Terraform to download the necessary provider plugins.

```
terraform init
```

This command downloads the AWS provider plugin and initializes the working directory.

Step 3: Apply the Terraform Configuration

To create the EC2 instance, apply the Terraform configuration. Terraform will prompt you to confirm the changes before proceeding.

```
terraform apply
```

Review the changes Terraform will make, and if everything looks good, type yes to approve.

Step 4: Verify the EC2 Instance

Once the configuration is applied, Terraform will output the public IP and instance ID of the EC2 instance. You can log into the instance using the following SSH command:

```
ssh -i /path/to/your/key.pem ubuntu@<instance_public_ip>
```

---

3. Advanced Setup: Multiple EC2 Instances with State Management

In this section, we will create multiple EC2 instances and configure state management using an S3 bucket and DynamoDB for locking.

Step 1: Create an S3 Bucket and DynamoDB Table

Before configuring Terraform, you need:

- **S3 Bucket**: To store the Terraform state.
- **DynamoDB Table**: To lock the state file and prevent conflicts during simultaneous Terraform runs.

You can create these manually in the AWS Management Console or using the AWS CLI.

Step 2: Create main.tf

This file is similar to the basic configuration but includes:

- **State Backend**: Configures Terraform to store the state file in an S3 bucket.
- **Count Parameter**: Allows you to create multiple EC2 instances with a single configuration.

Example of main.tf:

```
provider "aws" {
  region = "ap-south-1"
}

terraform {
  backend "s3" {
    bucket         = "dev-state123"  # S3 bucket to store the Terraform state
    key            = "global/s3/terraform.tfstate"  # Path to the state file in the bucket
    region         = "ap-south-1"    # AWS region
    dynamodb_table = "terraform-state-lock"  # DynamoDB table for state locking
    encrypt        = true           # Encrypt the state file in S3
  }
}

# Create multiple EC2 instances
resource "aws_instance" "test" {
  count         = 3                        # Create 3 instances
  ami           = "ami-0522ab6e1ddcc7055"        # AMI ID
  instance_type = "t2.medium"                  # Instance type
  key_name      = "DevOps-Shack"                # SSH key pair

  # Network settings
  vpc_security_group_ids = ["sg-04c97b7b938f67045"]  # Security group
  subnet_id              = "subnet-0ead1fda61faa7009"  # Subnet

  # Storage settings
  root_block_device {
    volume_size = 20
    volume_type = "gp2"
  }

  # Tags with unique identifiers
  tags = {
    Name = "DevOps-Shack-Instance-${count.index}"  # Unique tag for each instance
  }
}

# Output the public IPs of all instances
output "instance_public_ips" {
  value = aws_instance.test[*].public_ip
}

# Output the instance IDs of all instances
output "instance_ids" {
  value = aws_instance.test[*].id
}
```

Step 3: Initialize Terraform with Backend

Run the initialization command to set up Terraform to use the backend (S3 for state storage).

```
terraform init
```

Terraform will download the necessary backend configuration.

Step 4: Apply the Configuration

Apply the Terraform configuration to create multiple EC2 instances. As before, Terraform will prompt you to review the plan before execution.

```
terraform apply
```

Review the plan and type yes to confirm.

Step 5: Verify the Instances

Once Terraform completes the deployment, it will output the public IPs and instance IDs of all created instances.

You can SSH into any of the instances using:

```
ssh -i /path/to/your/key.pem ubuntu@<instance_public_ip>
```

---

4. Using Variables in Terraform

Variables allow you to make your configuration more flexible by avoiding hardcoded values.

Step 1: Create variables.tf

This file defines the variables used in the main.tf file. Here's an example:

Example of variables.tf:

```
variable "aws_region" {
  description = "The AWS region to deploy resources"
  default     = "ap-south-1"
}

variable "ami" {
  description = "AMI to use for the EC2 instances"
  default     = "ami-0522ab6e1ddcc7055"
}

variable "instance_type" {
  description = "Type of EC2 instance"
  default     = "t2.medium"
}

variable "key_name" {
  description = "Key pair for SSH access"
```

```
  default     = "DevOps-Shack"
}

variable "subnet_id" {
  description = "VPC subnet ID"
  default     = "subnet-0ead1fda61faa7009"
}

variable "security_group_ids" {
  description = "VPC security group IDs"
  type        = list(string)
  default     = ["sg-04c97b7b938f67045"]
}
```

In your main.tf, you can replace hardcoded values with variables like ${var.aws_region}.

Step 2: Apply the Configuration with Variables

Run the following command to apply the Terraform configuration using the variables:

```
terraform apply
```

---

5. Conclusion

This document provided a detailed explanation of setting up EC2 instances using Terraform in both basic and advanced configurations. With the use of variables, S3 state management, and DynamoDB, Terraform becomes a powerful tool for managing AWS infrastructure at scale.