Terraform Part-3

This guide provides a step-by-step explanation of creating a VPC with public and private subnets, an Internet Gateway, NAT Gateway, Security Groups, and EC2 instances using Terraform. The infrastructure will be set up using AWS, with Terraform state stored in an S3 bucket and state locking handled by a DynamoDB table.

---

1. Overview of Files

1. **main.tf**: Defines all the infrastructure resources (VPC, Subnets, Internet Gateway, NAT Gateway, Security Groups, EC2 instances).
2. **variables.tf**: Defines the variables to parameterize the Terraform configuration.
3. **outputs.tf**: Defines the outputs that will provide key information after the infrastructure is created (like VPC ID, Subnet IDs, and EC2 instance public IPs).
4. **terraform.tfvars**: Contains the actual values for the variables defined in variables.tf, allowing customization for different environments.

---

2. Detailed Explanation of the Configuration

2.1. main.tf – Core Resource Configuration

This file contains the main Terraform code that defines the infrastructure.

Provider Configuration

The provider block tells Terraform which cloud provider to use (in this case, AWS) and which region to work in.

```
provider "aws" {
  region = var.aws_region  # The region is dynamically passed via a variable
}
```

- **region**: This is the AWS region where the resources will be deployed. We are using a variable (var.aws_region) to define it.

Backend Configuration for Storing Terraform State

Terraform state files keep track of your infrastructure. By storing it remotely in an S3 bucket, multiple team members can safely collaborate. A DynamoDB table is used for state locking to prevent concurrent Terraform runs from conflicting with each other.

```
terraform {
  backend "s3" {
    bucket         = "dev-state123"  # Name of the S3 bucket for storing the state
    key            = "global/s3/terraform.tfstate"  # Path within the bucket to store the state file
    region         = "ap-south-1"  # AWS region where the S3 bucket and DynamoDB table exist
    dynamodb_table = "terraform-state-lock"  # DynamoDB table for state locking
    encrypt        = true  # Encrypts the state file
  }
```

```
}
```

- **bucket**: The S3 bucket that will store the Terraform state.
- **key**: Path within the bucket where the state file will be stored.
- **dynamodb_table**: This is the DynamoDB table used to handle state locking.
- **encrypt**: Ensures the state file is encrypted for security.

VPC (Virtual Private Cloud)

The VPC is the core network environment where all other resources will be deployed. Here, the VPC is created with a specific CIDR block.

```
resource "aws_vpc" "main_vpc" {
  cidr_block = var.vpc_cidr_block  # The CIDR block defines the IP range of the VPC
  enable_dns_support   = true  # Enables DNS support in the VPC
  enable_dns_hostnames = true  # Allows instances in the VPC to have DNS hostnames

  tags = {
    Name = "Main-VPC"  # Tags make it easier to identify resources
  }
}
```

- **cidr_block**: Defines the IP address range for the VPC, passed as a variable (var.vpc_cidr_block).
- **enable_dns_support**: Allows DNS resolution within the VPC.
- **enable_dns_hostnames**: Allows instances launched within the VPC to be assigned DNS hostnames.

Public Subnet

This subnet will host resources that need to be publicly accessible, like EC2 instances. The key feature of this subnet is that instances launched here will automatically be assigned public IP addresses.

```
resource "aws_subnet" "public_subnet" {
  vpc_id                  = aws_vpc.main_vpc.id  # Associates the subnet with the VPC
  cidr_block              = var.public_subnet_cidr  # IP range for this subnet
  availability_zone       = "${var.aws_region}a"  # Availability zone (part of the region)
  map_public_ip_on_launch = true  # Automatically assigns public IPs to instances

  tags = {
    Name = "Public-Subnet"
  }
}
```

- **vpc_id**: Associates the subnet with the VPC created earlier.
- **cidr_block**: Defines the IP range for the subnet, passed via a variable.
- **availability_zone**: The specific availability zone within the region (e.g., ap-south-1a).
- **map_public_ip_on_launch**: Ensures that instances in this subnet are automatically assigned public IPs, making them accessible from the internet.

Private Subnet

This subnet is for resources that should not be directly accessible from the internet. Instances in this subnet will route outbound traffic through the NAT Gateway.

```
resource "aws_subnet" "private_subnet" {
  vpc_id     = aws_vpc.main_vpc.id
  cidr_block = var.private_subnet_cidr  # IP range for the private subnet
  availability_zone = "${var.aws_region}a"

  tags = {
    Name = "Private-Subnet"
  }
}
```

- **cidr_block**: Defines the IP range for the private subnet, passed via a variable.

Internet Gateway

The Internet Gateway allows traffic between instances in the public subnet and the internet.

```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main_vpc.id  # Attaches the Internet Gateway to the VPC

  tags = {
    Name = "Main-IGW"
  }
}
```

- **vpc_id**: Associates the Internet Gateway with the VPC.

NAT Gateway

The NAT Gateway enables instances in the private subnet to connect to the internet, for example, to download updates or patches.

```
resource "aws_eip" "nat_eip" {
  vpc = true  # Indicates the Elastic IP is within the VPC
}

resource "aws_nat_gateway" "nat_gw" {
  allocation_id = aws_eip.nat_eip.id  # Uses the Elastic IP created earlier
  subnet_id     = aws_subnet.public_subnet.id  # Must be placed in a public subnet

  tags = {
    Name = "Main-NAT-GW"
  }
}
```

- **allocation_id**: The Elastic IP created for the NAT Gateway.
- **subnet_id**: The NAT Gateway must be deployed in a public subnet to allow instances in the private subnet to access the internet.

Route Tables

Route tables are needed to define how traffic flows within the VPC. Here we create one for public traffic and one for private traffic.

- **Public Route Table**: Routes all outbound traffic from the public subnet to the Internet Gateway.

```
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.main_vpc.id  # Associate with the VPC

  route {
    cidr_block = "0.0.0.0/0"  # All outbound traffic
    gateway_id = aws_internet_gateway.igw.id  # Route traffic through the Internet Gateway
  }

  tags = {
    Name = "Public-RouteTable"
  }
}
```

- **Private Route Table**: Routes outbound traffic from the private subnet through the NAT Gateway.

```
resource "aws_route_table" "private_rt" {
  vpc_id = aws_vpc.main_vpc.id  # Associate with the VPC

  route {
    cidr_block     = "0.0.0.0/0"  # All outbound traffic
    nat_gateway_id = aws_nat_gateway.nat_gw.id  # Route traffic through the NAT Gateway
  }

  tags = {
    Name = "Private-RouteTable"
  }
}
```

Associating Subnets with Route Tables

Each subnet needs to be explicitly associated with its respective route table.

- **Public Subnet Association**: Associates the public subnet with the public route table.

```
resource "aws_route_table_association" "public_assoc" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.public_rt.id
}
```

Private Subnet Association: Associates the private subnet with the private route table.

```
resource "aws_route_table_association" "private_assoc" {
  subnet_id      = aws_subnet.private_subnet.id
  route_table_id = aws_route_table.private_rt.id
}
```

Security Group for EC2 Instances

Security groups define the rules that control inbound and outbound traffic for EC2 instances. Here we allow:

- Inbound SSH (port 22)
- Inbound HTTP (port 80)

- Inbound HTTPS (port 443)

```
resource "aws_security_group" "public_sg" {
  vpc_id = aws_vpc.main_vpc.id

  # Allow SSH (Port 22)
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Allow SSH from anywhere
  }

  # Allow HTTP (Port 80)
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Allow HTTP from anywhere
  }

  # Allow HTTPS (Port 443)
  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Allow HTTPS from anywhere
  }

  # Allow all outbound traffic
  egress {
    from_port   = 0
```

```
  to_port     = 0
  protocol    = "-1"  # -1 means all protocols
  cidr_blocks = ["0.0.0.0/0"]
 }

 tags = {
  Name = "Public-SG"
 }
}
```

EC2 Instances in Public Subnet

We create multiple EC2 instances in the public subnet, each with a public IP address and SSH access using a key pair. The number of instances is determined by the vm_count variable.

```
resource "aws_instance" "public_instance" {
 count                  = var.vm_count  # Create multiple instances based on vm_count variable
 ami                    = "ami-0522ab6e1ddcc7055"  # Hardcoded Ubuntu 24.04 LTS AMI
 instance_type          = var.instance_type  # EC2 instance type (t2.micro by default)
 subnet_id              = aws_subnet.public_subnet.id  # Place the instances in the public subnet
 key_name               = var.key_name  # The SSH key pair to use
 security_groups        = [aws_security_group.public_sg.id]  # Associate with the security group
 associate_public_ip_address = true  # Assign a public IP to each instance

 tags = {
  Name = "Public-EC2-${count.index + 1}"  # Tag each instance with a unique name
 }
}
```

- **count**: Creates multiple instances based on the value of vm_count (which is 2 by default in terraform.tfvars).
- **ami**: Specifies the Amazon Machine Image (in this case, Ubuntu 24.04 LTS).
- **key_name**: Specifies the SSH key pair used to connect to the instances.

2.2. variables.tf – Parameterization

The variables.tf file defines variables that make the Terraform configuration more dynamic. Instead of hardcoding values like region, instance type, or CIDR blocks, we define them as variables.

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

variable "vpc_cidr_block" {
  description = "The CIDR block for the VPC"
  default     = "10.0.0.0/16"
}

variable "public_subnet_cidr" {
  description = "CIDR block for the public subnet"
  default     = "10.0.1.0/24"
}

variable "private_subnet_cidr" {
  description = "CIDR block for the private subnet"
  default     = "10.0.2.0/24"
}

variable "instance_type" {
  description = "EC2 instance type"
  default     = "t2.micro"
}

variable "key_name" {
  description = "The name of the SSH key pair to use for EC2 instances"
}

variable "vm_count" {
  description = "The number of EC2 instances to create"
  default     = 2  # Default is 2 EC2 instances
}
```

- **aws_region**: The AWS region where the resources will be deployed.
- **vpc_cidr_block**: The CIDR block for the VPC.
- **public_subnet_cidr**: The CIDR block for the public subnet.
- **private_subnet_cidr**: The CIDR block for the private subnet.
- **instance_type**: The type of EC2 instances to create (t2.micro by default).
- **key_name**: The name of the SSH key pair for EC2 instances.
- **vm_count**: The number of EC2 instances to create (2 by default).

2.3. outputs.tf – Output Configuration

The outputs.tf file specifies the key information you want Terraform to display after the infrastructure is created.

```
output "vpc_id" {
  value = aws_vpc.main_vpc.id
}

output "public_subnet_id" {
  value = aws_subnet.public_subnet.id
}

output "private_subnet_id" {
  value = aws_subnet.private_subnet.id
}

output "public_ec2_public_ip" {
  value = aws_instance.public_instance[*].public_ip  # Output public IPs of all instances
}
```

- **vpc_id**: Outputs the ID of the VPC.
- **public_subnet_id**: Outputs the ID of the public subnet.
- **private_subnet_id**: Outputs the ID of the private subnet.
- **public_ec2_public_ip**: Outputs the public IPs of all EC2 instances created.

2.4. terraform.tfvars – Values for Variables

The terraform.tfvars file contains the actual values for the variables defined in variables.tf. This allows you to quickly modify configurations for different environments.

```
vpc_cidr_block     = "10.0.0.0/16"

public_subnet_cidr = "10.0.1.0/24"

private_subnet_cidr = "10.0.2.0/24"

instance_type      = "t2.micro"

key_name           = "DevOps-Shack"

vm_count           = 2  # Create 2 EC2 instances
```

You can adjust these values depending on your environment's requirements. For example, you can modify vm_count to create more EC2 instances or change instance_type to launch larger instances.

---

3. Applying the Configuration

1. **Initialize Terraform**: Before applying any configuration, initialize Terraform to download the necessary provider plugins.

terraform init

2. **Apply the Configuration**: Use the following command to apply the configuration. Terraform will prompt you to confirm the changes before proceeding.

terraform apply -var-file="terraform.tfvars"

3. **Verify the Resources**: After the apply process is complete, Terraform will output information such as the public IPs of the EC2 instances, which you can use to SSH into them:

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

---

4. Conclusion

This document provides a detailed explanation of using Terraform to set up a complete AWS environment, including VPCs, subnets, route tables, gateways, security groups, and multiple EC2 instances. By using variables.tf and terraform.tfvars, the configuration is flexible and reusable for different environments.