Demo Project: Modularize Project

This project demonstrates how to divide Terraform resources into reusable modules to improve organization, reusability, and maintainability. We will create a shared project structure with a root module and child modules (e.g., `webserver` and `subnets`). The root module references the child modules, and module outputs are used to pass values between them.

Step 1: Set Up Your Project Structure

Note: Terraform also provides reusable modules at: https://registry.terraform.io/namespaces/terraform-aws-modules

1.1 Create a New Git Branch

Create a new branch from `feature/deploy-to-ec2-default-components` for modularization: `git checkout -b feature/modules`

1.2 Establish the Directory Structure

Organize your project as follows:

```
├── main.tf
├── variables.tf
├── outputs.tf
├── providers.tf
```

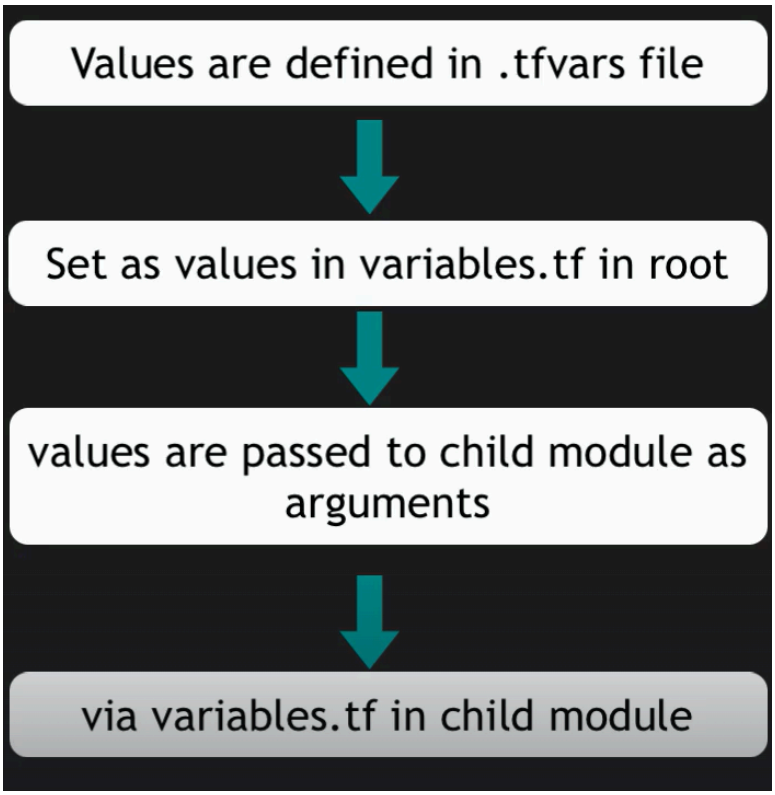
```
├── terraform.tfvars      # (Not committed; listed in .gitignore)
└── modules
    ├── webserver
    │   ├── main.tf
    │   ├── variables.tf
    │   ├── outputs.tf
    │   └── providers.tf
    └── subnets
        ├── main.tf
        ├── variables.tf
        ├── outputs.tf
        └── providers.tf
```

1.3 Create Module Folders and Files
Create directories and files for
the webserver and subnet modules:

```
mkdir -p modules/webserver modules/subnets
touch modules/webserver/main.tf modules/webserver/variables.tf module
s/webserver/outputs.tf modules/webserver/providers.tf
touch modules/subnets/main.tf modules/subnets/variables.tf modules/sub
nets/outputs.tf modules/subnets/providers.tf
```

Step 2: Develop the Child Modules

We will create resources in the child modules and reference them from the root module.



2A: Subnets Module

1. Define the Subnet Resource in `modules/subnets/main.tf`:

```
resource "aws_subnet" "myapp-subnet-1" {
  vpc_id = var.vpc_id
  cidr_block = var.subnet_cidr_block
  availability_zone = var.avail_zone
  tags = {
    Name: "${var.env_prefix}-subnet-1" # Name tag for the subnet
  }
}
```

2. Declare Variables in `modules/subnets/variables.tf`:

```
variable subnet_cidr_block {}
variable avail_zone {}
variable env_prefix {}
variable vpc_id {}
variable default_route_table_id {}
```

3. Export the Subnet ID in `modules/subnets/outputs.tf`:

```
output "subnet" {
  value = aws_subnet.myapp-subnet-1
}
```

4. **(Optional)** `modules/subnets/providers.tf`**:** You can include provider configuration here if needed or rely on the root provider.

2B: Webserver Module

1. Define the EC2 Instance in `modules/webserver/main.tf`:

```
resource "aws_instance" "myapp-server" {
    ami = data.aws_ami.latest-amazon-linux-image.id
    instance_type = var.instance_type

    subnet_id = var.subnet_id
    vpc_security_group_ids = [aws_default_security_group.default-sg.id]
    availability_zone = var.avail_zone

    associate_public_ip_address = true
    key_name = aws_key_pair.ssh-key.key_name

    user_data = file("entry-script.sh")
```

```
    user_data_replace_on_change = true

    tags = {
        Name: "${var.env_prefix}-server"
    }
}
```

2. Declare Variables in `modules/webserver/variables.tf`:

```
variable vpc_id {}
variable my_ip {}
variable env_prefix {}
variable image_name {}
variable public_key_location {}
variable instance_type {}
variable subnet_id {}
variable avail_zone {}
```

3. Export the EC2 Instance in `modules/webserver/outputs.tf`:

```
output "instance" {
  value = aws_instance.myapp-server
}
```

4. **(Optional)** `modules/webserver/providers.tf`**:** Include provider configuration if necessary, or rely on the root provider.

Step 3: Update the Root Module to Use Child Modules

1. Reference the Child Modules in the Root `main.tf`:

```
module "myapp-subnet" {
    source = "./modules/subnets"
    subnet_cidr_block = var.subnet_cidr_block
    avail_zone = var.avail_zone
    env_prefix = var.env_prefix
    vpc_id = aws_vpc.myapp-vpc.id
    default_route_table_id = aws_vpc.myapp-vpc.default_route_table_id
}

module "myapp-server" {
  source = "./modules/webserver"
  vpc_id = aws_vpc.myapp-vpc.id
  my_ip = var.my_ip
  env_prefix = var.env_prefix
  image_name = var.image_name
  public_key_location = var.public_key_location
  instance_type = var.instance_type
  subnet_id = module.myapp-subnet.subnet.id
  avail_zone = var.avail_zone
}
```

2. Declare Variables in the Root `variables.tf`:

```
variable vpc_cidr_blocks {}
variable subnet_cidr_block {}
variable avail_zone {}
variable env_prefix {}
variable my_ip {}
variable instance_type {}
variable public_key_location {}
variable image_name {}
```

3. Export Outputs from Child Modules in `outputs.tf`:

```
output "ec-public_ip" {
   value = module.myapp-server.instance.public_ip
}
```

4. Configure the Provider in `providers.tf`:

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.20.1"
    }
  }
}
```

Step 4: Initialize, Plan, and Apply Your Terraform Configuration

1. **Initialize Terraform:** `terraform init`
2. **Preview Changes:** `terraform plan`
3. **Apply the Configuration:** `terraform apply -auto-approve`

Step 5: Clean Up

1. **Destroy the Infrastructure:** `terraform destroy -auto-approve`

Additional Notes

- **Module Outputs:** Child modules should expose critical resource attributes (e.g., subnet IDs, public IP addresses) via outputs. These outputs are then

used in the root module.

- **Project Structure Best Practices:** Maintain a clean separation between root configuration and modules. The root module orchestrates resources by calling child modules, improving reusability and maintainability.
- **Version Control:** Keep your `terraform.tfvars` file out of version control by adding it to your `.gitignore`.
