Lab Exercise 6– Terraform Variables

Objective:

Learn how to define and use variables in Terraform configuration.

Prerequisites:

- Install Terraform on your machine.

Steps:

1. Create a Terraform Directory:

- Create a new directory for your Terraform project.

```
mkdir terraform-variables

cd terraform-variables
```

2. Create a Terraform Configuration File:

- Create a file named main.tf within your project directory.

 # main.tf

main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "5.30.0"
    }
  }
}

provider "aws" {
  region     = "ap-south-1"
  access_key = "AKIA3LET55J5HKHNNZVY"
  secret_key = "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"
}

```

3. Define Variables:

- Open a new file named variables.tf. Define variables for region, ami, and instance_type.

 # variables.tf

instance.tf

```
resource "aws_instance" "my_instance" {
  ami           = var.my-ami  # Replace with the AMI ID valid for your region.
  instance_type = var.my-instance-type

  tags = {
    Name = "UPES-EC2-Instance"
  }
}
variable "my-ami" {
    type = string
    default = "ami-03235cc8fe4d9bf1e"
}

variable "my-instance-type" {
    type = string
    default = "t3.micro"

}

```

4. Initialize and Apply:

- Run the following Terraform commands to initialize and apply the configuration.

terraform init

terraform plan

```
iamyo@Acernitro MINGW64 /d/terraform-demo
$ terraform plan

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.my_instance will be created
  + resource "aws_instance" "my_instance" {
      + ami                                  = "ami-03235cc8fe4d9bf1e"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
```

terraform apply -auto-approve

```
iamyo@Acernitro MINGW64 /d/terraform-demo
$ terraform apply

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.my_instance will be created
  + resource "aws_instance" "my_instance" {
      + ami                                  = "ami-03235cc8fe4d9bf1e"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
```

Observe how the region changes based on the variable override.

5. Clean Up:

After testing, you can clean up resources.

terraform destroy

```
aws_s3_bucket.my_bucket: Destroying... [id=my-demo-s3-bucket123]
aws_instance.my_instance: Destroying... [id=i-0068d6118911be600]
aws_s3_bucket.my_bucket: Destruction complete after 0s
aws_instance.my_instance: Still destroying... [id=i-0068d6118911be600, 10s elaps
ed]
aws_instance.my_instance: Still destroying... [id=i-0068d6118911be600, 20s elaps
ed]
aws_instance.my_instance: Still destroying... [id=i-0068d6118911be600, 30s elaps
ed]
aws_instance.my_instance: Still destroying... [id=i-0068d6118911be600, 40s elaps
ed]
aws_instance.my_instance: Still destroying... [id=i-0068d6118911be600, 50s elaps
ed]
aws_instance.my_instance: Still destroying... [id=i-0068d6118911be600, 1m0s elap
sed]
aws_instance.my_instance: Still destroying... [id=i-0068d6118911be600, 1m10s ela
psed]
aws_instance.my_instance: Still destroying... [id=i-0068d6118911be600, 1m20s ela
psed]
aws_instance.my_instance: Destruction complete after 1m20s

Destroy complete! Resources: 2 destroyed.
```

Confirm the destruction by typing yes.

6. Conclusion:

This lab exercise introduces you to Terraform variables and demonstrates how to use them in your configurations. Experiment with different variable values and overrides to understand their impact on the infrastructure provisioning process.