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

```
resource "aws_instance" "myinstance-1" {
   ami = var.myami
   instance_type = var.my_instance_type
   count = var.mycount
   tags = {
     Name= "My Instance"
   }
}
```

lab-6 > main.tf > ...

```
resource "aws_instance" "myinstance-1" {
ami = var.myami
instance_type = var.my_instance_type
count = var.mycount
tags = {
Name= "My Instance"
}
}
```

3. Define Variables:

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

 # variables.tf

```
variable "myami" {
 type = string
 default = "ami-08718895af4dfa033"
}

variable "mycount" {

 type = number
 default = 5
}

variable "my_instance_type" {
  type = string
  default = "t2.micro"
```

}

lab-6 > variables.tf > variable "mycount"

```
variable "myami" {
type = string
default = "ami-08718895af4dfa033"
}
variable "mycount" {
type = number
default = 5
}
variable "my_instance_type" {
type = string
default = "t2.micro"
}
```

4. Initialize and Apply:

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

terraform init

terraform plan

terraform apply -auto-approve

```
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

```
      + network_interface (known after apply)

      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

Plan: 5 to add, 0 to change, 0 to destroy.
```

```
dhruvchaubey@Mac lab-6 % terraform apply -auto-approve

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

```
aws_instance.myinstance-1[1]: Creation complete after 13s [id=i-00ceab7049be23d1
5]
aws_instance.myinstance-1[3]: Creation complete after 13s [id=i-049abc9e790f2fc0
f]
aws_instance.myinstance-1[0]: Creation complete after 13s [id=i-0499c282b608366
3]
aws_instance.myinstance-1[4]: Creation complete after 13s [id=i-0073763fa57346e
a]

Apply complete! Resources: 5 added, 0 changed, 0 destroyed.
```

Observe how the region changes based on the variable override.

5. Clean Up:

After testing, you can clean up resources.

```
terraform destroy
```

```
elapsed]
aws_instance.myinstance-1[4]: Destruction complete after 1m41s

Destroy complete! Resources: 5 destroyed.
dhruvchaubey@Mac lab-6 %
```

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.