Lab Exercise 7– Terraform Variables with Command Line Arguments

Objective:

Learn how to pass values to Terraform variables using command line arguments.

Prerequisites:

- Terraform installed on your machine.
- Basic knowledge of Terraform variables.

Steps:

1. Create a Terraform Directory:

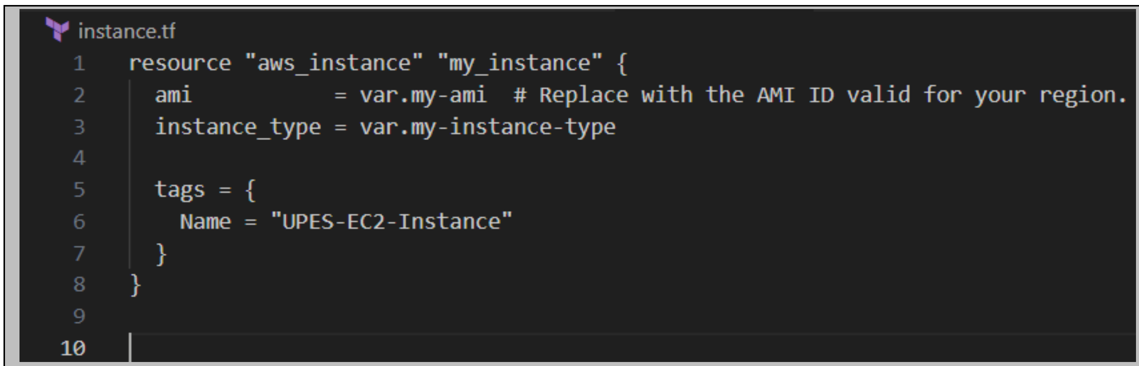
```
mkdir terraform-cli-variables

cd terraform-cli-variables
```

2. Create Terraform Configuration Files:

- Create a file named main.tf:

# instance.tf



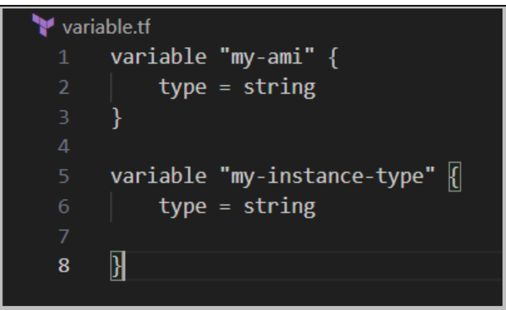
```
resource "aws_instance" "my_instance" {
  ami           = var.my-ami  # Replace with the AMI ID valid for your region.
  instance_type = var.my-instance-type

  tags = {
    Name = "UPES-EC2-Instance"
  }
}

```

- Create a file named variables.tf:

# variables.tf



variable.tf

```
variable "my-ami" {
    type = string
}

variable "my-instance-type" {
    type = string

}
```

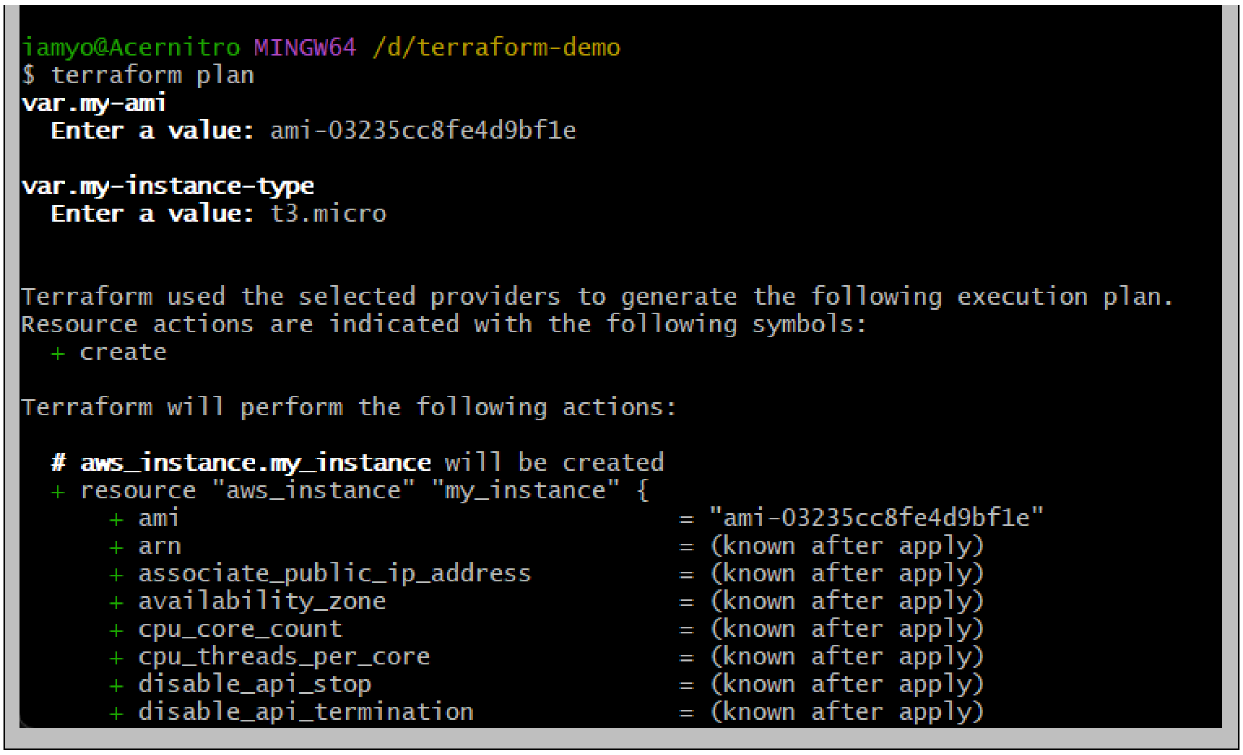
3. Use Command Line Arguments:

- Open a terminal and navigate to your Terraform project directory.
- Run the terraform init command:

```
terraform init
```

- Run the terraform apply command with command line arguments to set variable values:

```
terraform plan -var=”ami=ami-0522ab6e1ddcc7055” -var=”instance_type=t3.micro”
```



```
iamyo@Acernitro MINGW64 /d/terraform-demo
$ terraform plan
var.my-ami
  Enter a value: ami-03235cc8fe4d9bf1e

var.my-instance-type
  Enter a value: t3.micro


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)
```

- Adjust the values based on your preferences.

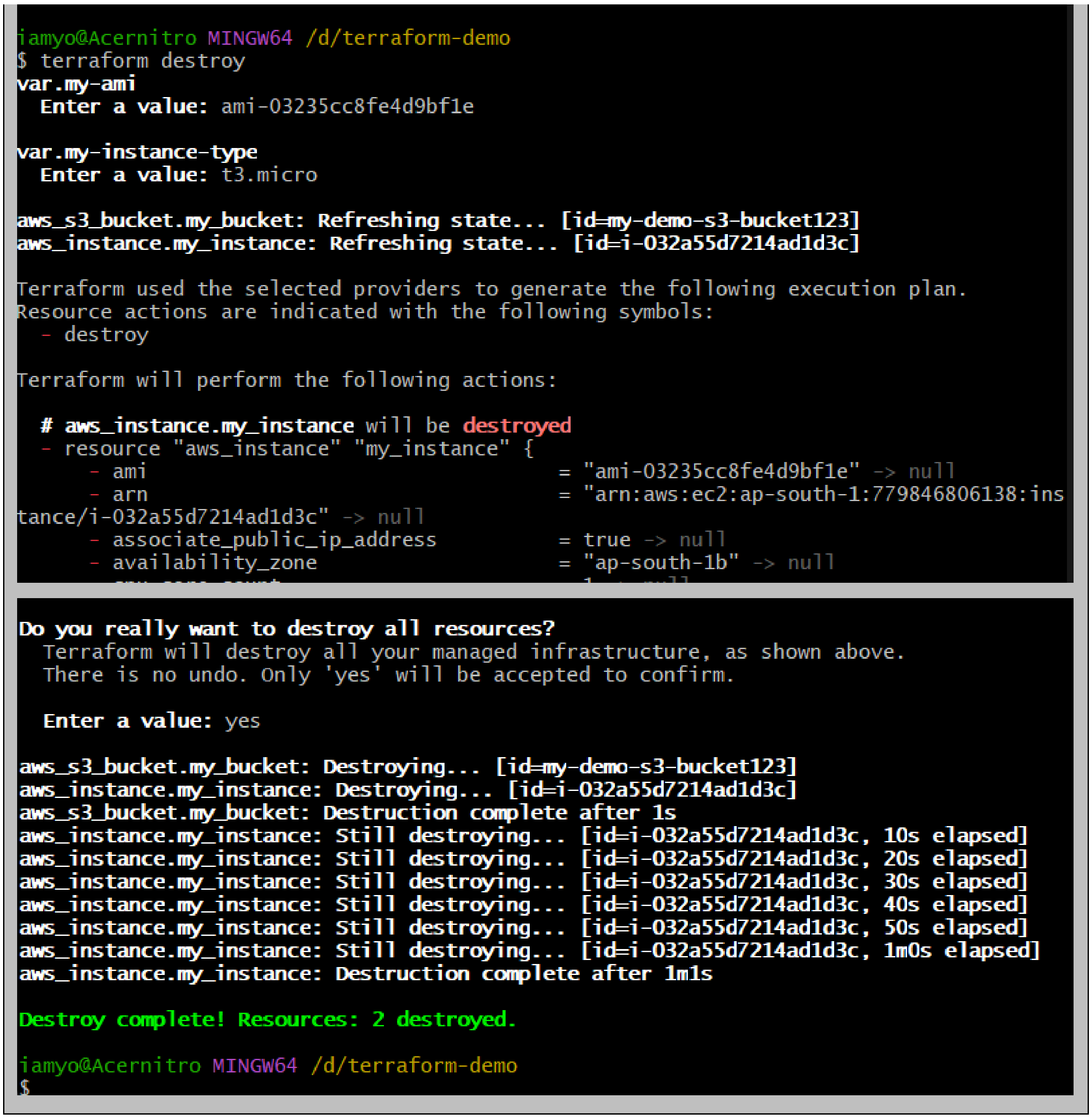
4. Test and Verify:

- Observe how the command line arguments dynamically set the variable values during the apply process.
- Access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified region.

5. Clean Up:

After testing, you can clean up resources:

```
terraform destroy
```



```
iamyo@Acernitro MINGW64 /d/terraform-demo
$ terraform destroy
var.my-ami
  Enter a value: ami-03235cc8fe4d9bf1e

var.my-instance-type
  Enter a value: t3.micro

aws_s3_bucket.my_bucket: Refreshing state... [id=my-demo-s3-bucket123]
aws_instance.my_instance: Refreshing state... [id=i-032a55d7214ad1d3c]

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

Terraform will perform the following actions:

  # aws_instance.my_instance will be destroyed
  - resource "aws_instance" "my_instance" {
      - ami                                  = "ami-03235cc8fe4d9bf1e" -> null
      - arn                                  = "arn:aws:ec2:ap-south-1:779846806138:ins
tance/i-032a55d7214ad1d3c" -> null
      - associate_public_ip_address          = true -> null
      - availability_zone                    = "ap-south-1b" -> null
```

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_s3_bucket.my_bucket: Destroying... [id=my-demo-s3-bucket123]
aws_instance.my_instance: Destroying... [id=i-032a55d7214ad1d3c]
aws_s3_bucket.my_bucket: Destruction complete after 1s
aws_instance.my_instance: Still destroying... [id=i-032a55d7214ad1d3c, 10s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a55d7214ad1d3c, 20s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a55d7214ad1d3c, 30s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a55d7214ad1d3c, 40s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a55d7214ad1d3c, 50s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a55d7214ad1d3c, 1m0s elapsed]
aws_instance.my_instance: Destruction complete after 1m1s

Destroy complete! Resources: 2 destroyed.

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

Confirm the destruction by typing yes.

6. Conclusion:

This lab exercise demonstrates how to use command line arguments to set variable values dynamically during the terraform apply process. It allows you to customize your Terraform deployments without modifying the configuration files directly. Experiment with different variable values and observe how command line arguments impact the infrastructure provisioning process.