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:

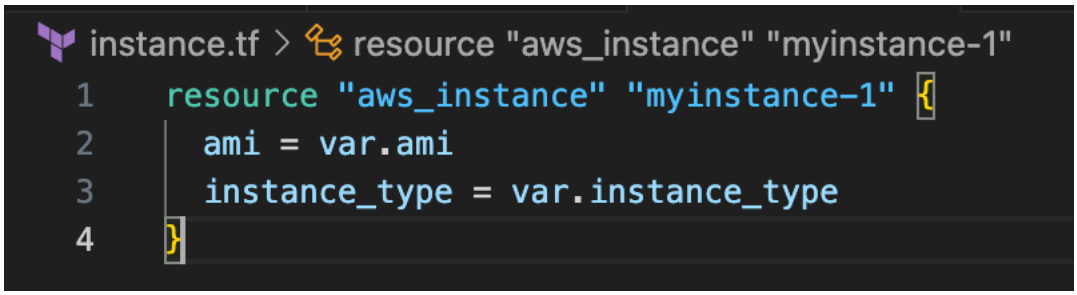
main.tf > provider "aws"

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.83.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  access_key = "AKIAQMEY6IA6U2E63M7N"
  secret_key = "6jQ41zqkpj8zLyl4jiV14AgeRIWD0FZ28qIY3aNa"
}
```

instance.tf

```
resource "aws_instance" "example" {
 ami           = var.ami
 instance_type = var.instance_type
}
```



- Create a file named variables.tf:

variables.tf

```
variable "ami" {
 description = "AMI ID"
 default     = " ami-08718895af4dfa033"
}

variable "instance_type" {
 description = "EC2 Instance Type"
 default     = "t2.micro"
}
```

3. Use Command Line Arguments:

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

terraform init

```
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.
palakgupta@Palaks-MacBook-Air terraform-cli-variables %
```

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

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

- Adjust the values based on your preferences.

```
Terraform will perform the following actions:

  # aws_instance.myinstance-1 will be created
  + resource "aws_instance" "myinstance-1" {
      + ami                                  = "ami-0df8c184d5f6ae949"
      + 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)
      + enable_primary_ipv6                  = (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)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t3.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
```

4. Test and Verify:

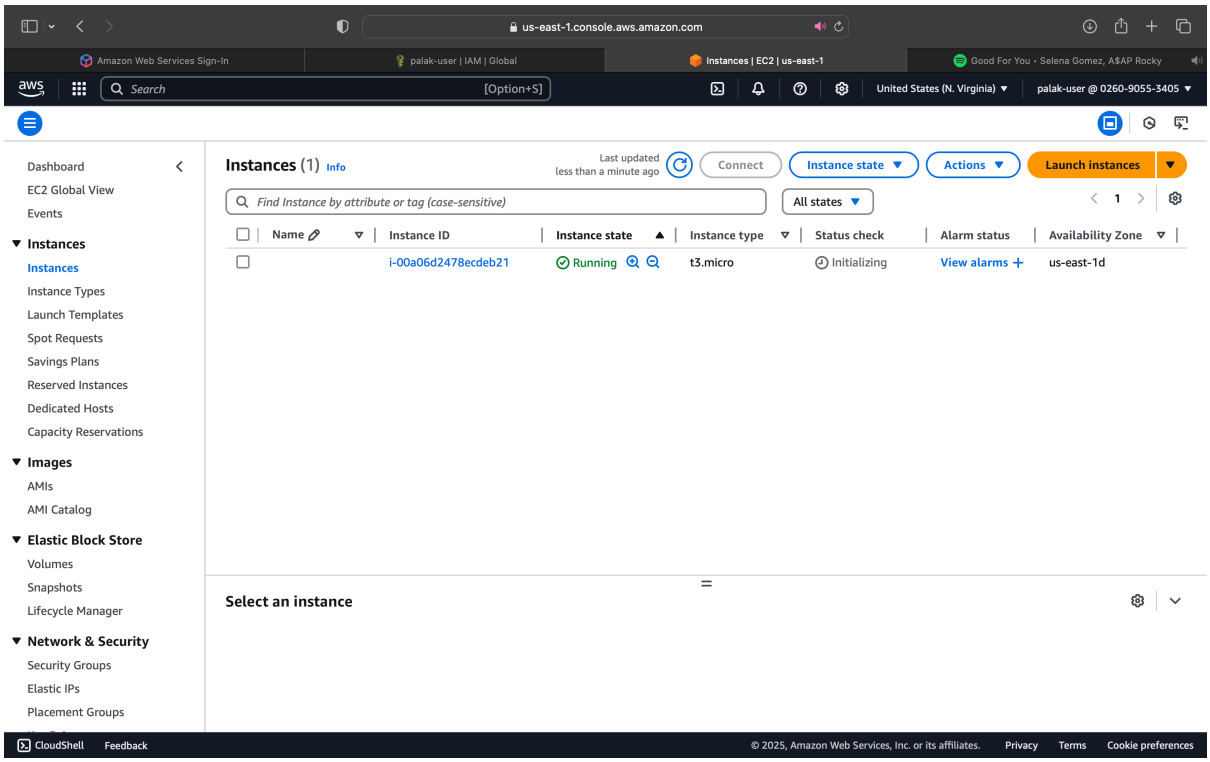
- Observe how the command line arguments dynamically set the variable values during the apply process.
- Access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified region.

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.myinstance-1: Creating...
aws_instance.myinstance-1: Still creating... [10s elapsed]
aws_instance.myinstance-1: Creation complete after 17s [id=i-00a06d2478ecdeb21]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
palakgupta@Palaks-MacBook-Air terraform-cli-variables %
```



5. Clean Up:

After testing, you can clean up resources:

```
terraform destroy
```

Confirm the destruction by typing yes.

```
Plan: 0 to add, 0 to change, 1 to destroy.
aws_instance.myinstance-1: Destroying... [id=i-00a06d2478ecdeb21]
aws_instance.myinstance-1: Still destroying... [id=i-00a06d2478ecdeb21, 10s elapsed]
aws_instance.myinstance-1: Still destroying... [id=i-00a06d2478ecdeb21, 20s elapsed]
aws_instance.myinstance-1: Still destroying... [id=i-00a06d2478ecdeb21, 30s elapsed]
aws_instance.myinstance-1: Still destroying... [id=i-00a06d2478ecdeb21, 40s elapsed]
aws_instance.myinstance-1: Still destroying... [id=i-00a06d2478ecdeb21, 50s elapsed]
aws_instance.myinstance-1: Destruction complete after 54s

Destroy complete! Resources: 1 destroyed.
palakgupta@Palaks-MacBook-Air terraform-cli-variables %
```

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.