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:

```
C:\Users\OM VATS>mkdir terraform-cli-variables

C:\Users\OM VATS>cd terraform-cli-variables
```

2. Create Terraform Configuration Files:

- Create a file named main.tf:

instance.tf

```
C:\Users\OM VATS\terraform-cli-variables>notepad main.tf
```

```
resource "aws_instance" "example" {
  ami           = var.ami
  instance_type = var.instance_type
}
provider "aws" {
  region = "us-east-1" # Replace with your preferred region
}
```

- Create a file named variables.tf:

variables.tf

```
C:\Users\OM VATS\terraform-cli-variables>notepad variables.tf
```

```
variable "ami" {
  description = "AMI ID"
  default     = "ami-002388442943a4626"
}

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:

```
C:\Users\OM VATS\terraform-cli-variables>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.84.0...
- Installed hashicorp/aws v5.84.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
```

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

```
C:\Users\OM VATS\terraform-cli-variables>terraform apply -var="ami=ami-002388442943a4626" -var="instance_type=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.example will be created
  + resource "aws_instance" "example" {
      + ami                                  = "ami-002388442943a4626"
      + 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)
```

- 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.

```
an: 1 to add, 0 to change, 0 to destroy.

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

Enter a value: yes

s_instance.example: Creating...
s_instance.example: Still creating... [10s elapsed]
s_instance.example: Creation complete after 18s [id=i-0625c7beb6fc139fe]

ply complete! Resources: 1 added, 0 changed, 0 destroyed.

\Users\OM VATS\terraform-cli-variables>aws ec2 describe-instances

  "Reservations": [
      {
          "ReservationId": "r-075ced90703fa46da",
          "OwnerId": "590183718293",
          "Groups": [],
          "Instances": [
              {
                  "Architecture": "x86_64",
                  "BlockDeviceMappings": [
                      {
                          "DeviceName": "/dev/xvda",
                          "Ebs": {
                              "AttachTime": "2025-01-23T08:42:22+00:00",
```

5. Clean Up:

After testing, you can clean up resources:

terraform destroy

Confirm the destruction by typing yes.

```
          - volume_size           = 105 -> null
          - volume_type           = "gp3" -> null
            # (1 unchanged attribute hidden)
        }
    }

Plan: 0 to add, 0 to change, 1 to destroy.

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_instance.example: Destroying... [id=i-0625c7beb6fc139fe]
aws_instance.example: Still destroying... [id=i-0625c7beb6fc139fe, 10s elapsed]
aws_instance.example: Still destroying... [id=i-0625c7beb6fc139fe, 20s elapsed]
aws_instance.example: Still destroying... [id=i-0625c7beb6fc139fe, 30s elapsed]
aws_instance.example: Destruction complete after 33s

Destroy complete! Resources: 1 destroyed.
```

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.