

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" "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
```

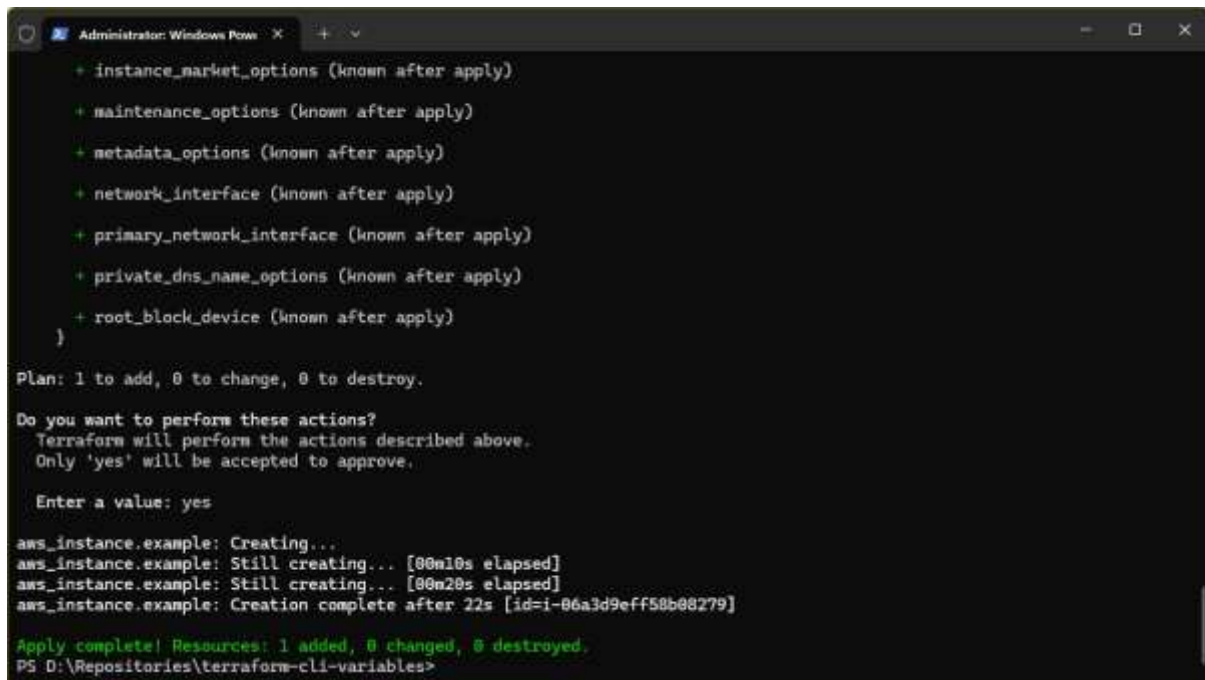
- Run the terraform apply command with command line arguments to set variable values:

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

- Adjust the values based on your preferences.

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.

A screenshot of a Windows PowerShell terminal window titled "Administrator: Windows PowerShell". The terminal displays the output of a Terraform apply command. It lists several resources to be added, including instance_market_options, maintenance_options, metadata_options, network_interface, primary_network_interface, private_dns_name_options, and root_block_device. A plan summary shows 1 resource to be added. The user is prompted to confirm the actions, and they respond with "yes". The terminal then shows the creation progress of an AWS instance, with status updates from "Creating..." to "Still creating..." and finally "Creation complete after 22s". The final output indicates that 1 resource was added and the apply was successful.

```
+ instance_market_options (known after apply)
+ maintenance_options (known after apply)
+ metadata_options (known after apply)
+ network_interface (known after apply)
+ primary_network_interface (known after apply)
+ private_dns_name_options (known after apply)
+ root_block_device (known after apply)
}

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

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.example: Creating...
aws_instance.example: Still creating... [00m10s elapsed]
aws_instance.example: Still creating... [00m20s elapsed]
aws_instance.example: Creation complete after 22s [id=i-06a3d9eff58b08279]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS D:\Repositories\terraform-cli-variables>
```

5. Clean Up:

After testing, you can clean up resources:

```
terraform destroy
```

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.