

---

## 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"
```

}

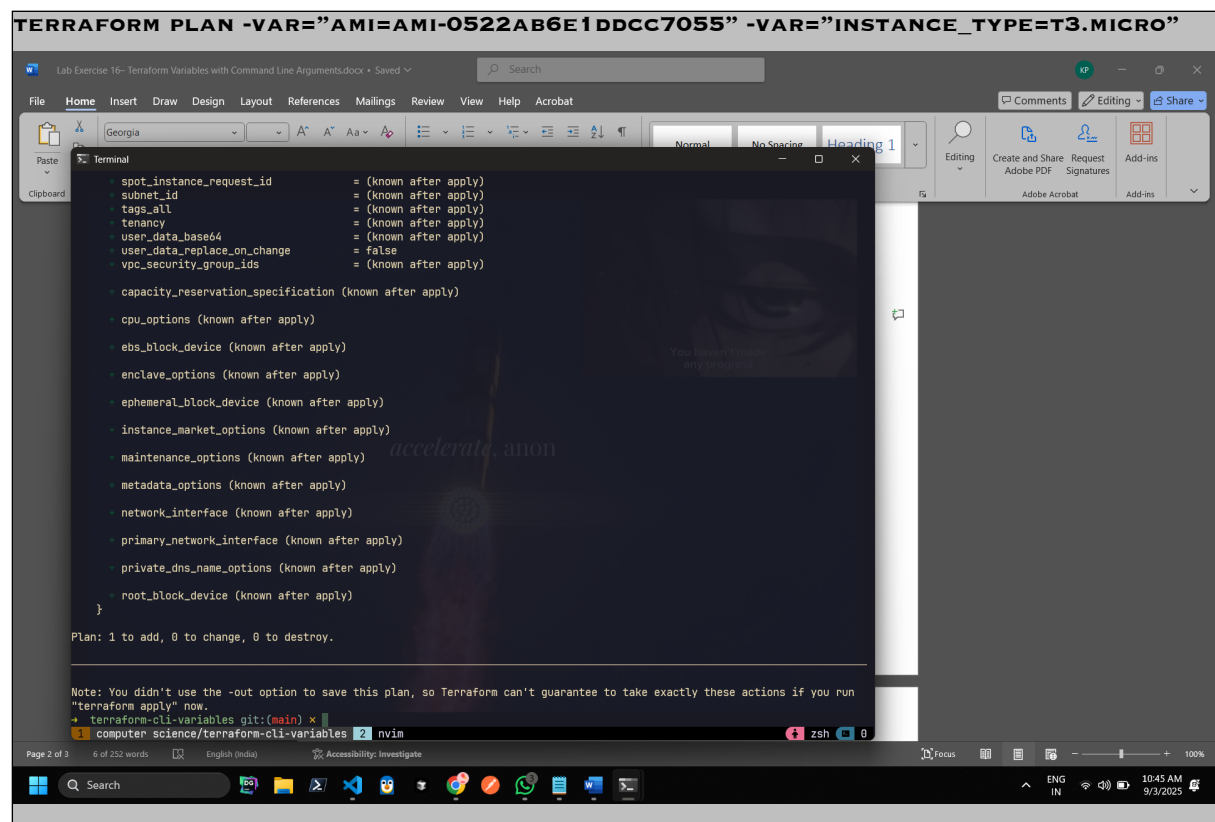
### 3. USE COMMAND LINE ARGUMENTS:

- 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"
```



```
spot_instance_request_id = (known after apply)
subnet_id                = (known after apply)
tags_all                 = (known after apply)
tenancy                  = (known after apply)
user_data_base64         = (known after apply)
user_data_replace_on_change = false
vpc_security_group_ids   = (known after apply)

capacity_reservation_specification (known after apply)

cpu_options (known after apply)

ebs_block_device (known after apply)

enclave_options (known after apply)

ephemeral_block_device (known after apply)

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)
}

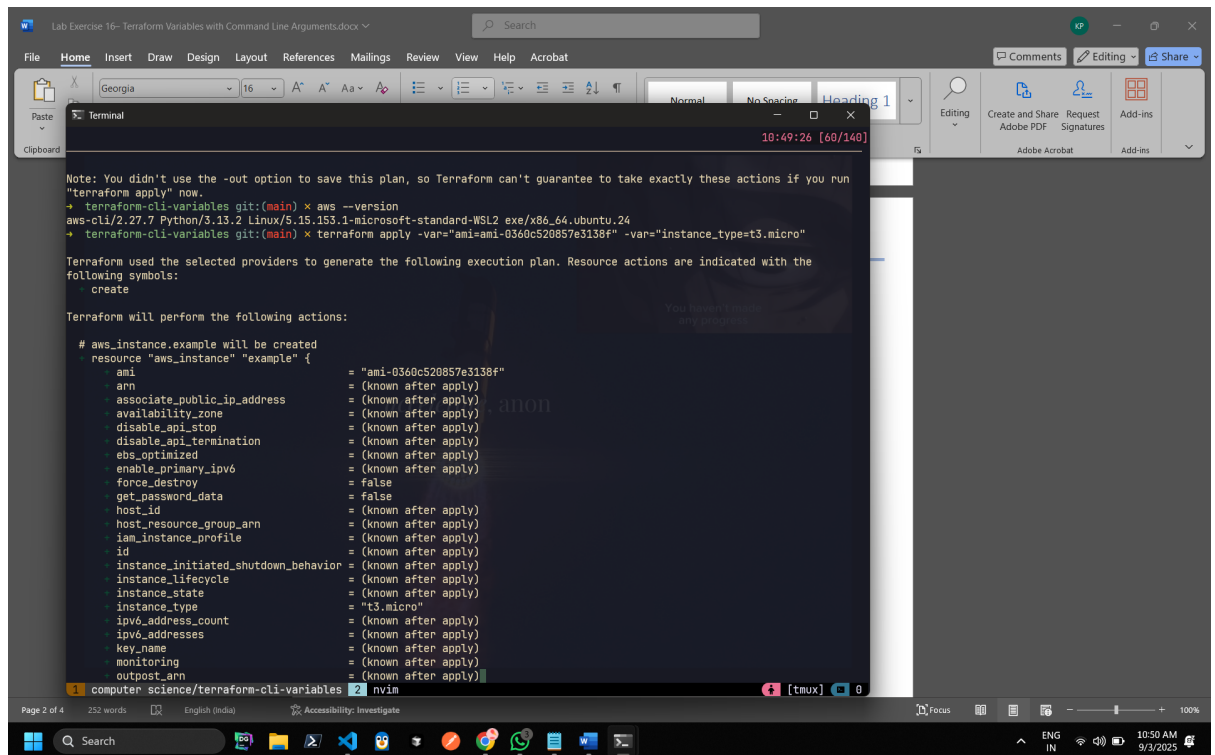
Plan: 1 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee it takes exactly these actions if you run "terraform apply" now.
→ terraform-cli-variables git:(main) x
1 computer science/terraform-cli-variables 2 nvim
```

- 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.



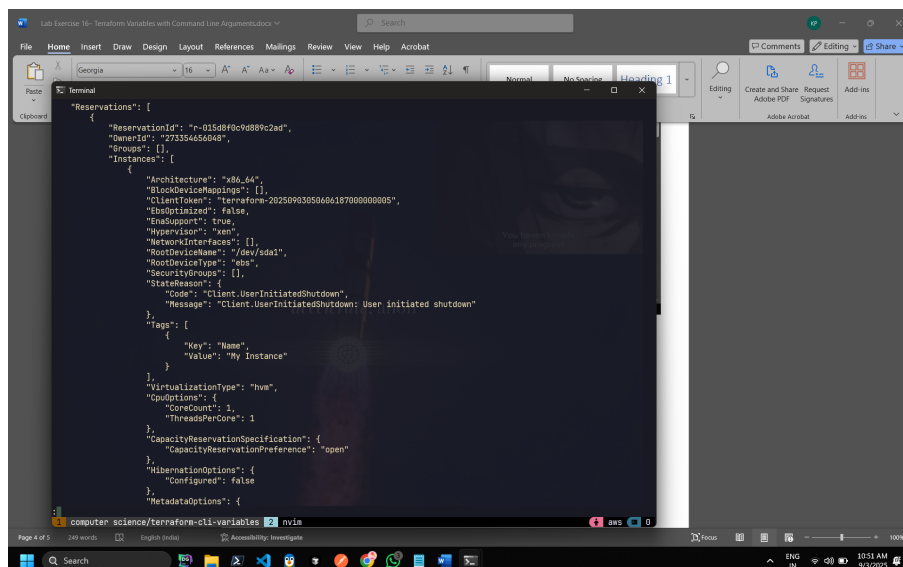
The screenshot shows a Windows terminal window with a dark background. The terminal output displays the Terraform apply process. It starts with a note about the -out option, followed by the command `terraform apply -var="ami=ami-0360c520857e3138f" -var="instance_type=t3.micro"`. The output shows the execution plan for creating an `aws_instance` resource named "example". The plan lists various attributes like `ami`, `arn`, `availability_zone`, etc., and their values. The terminal window has a title bar "Lab Exercise 16- Terraform Variables with Command Line Arguments.docx" and a menu bar with options like File, Home, Insert, Draw, Design, Layout, References, Mailings, Review, View, Help, Acrobat. The status bar at the bottom shows "Page 2 of 4", "252 words", "English (India)", and "Accessibility: Investigate".

```
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
→ terraform-cli-variables git:(main) x aws --version
aws-cli/2.27.7 Python/3.13.2 Linux/5.15.153.1-microsoft-standard-WSL2 exe/x86_64.ubuntu.24
→ terraform-cli-variables git:(main) x terraform apply -var="ami=ami-0360c520857e3138f" -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-0360c520857e3138f"
  arn           = (known after apply)
  associate_public_ip_address = (known after apply)
  availability_zone = (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)
  force_destroy   = false
  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)
  key_name        = (known after apply)
  monitoring      = (known after apply)
  outpost_arn     = (known after apply)
}
```

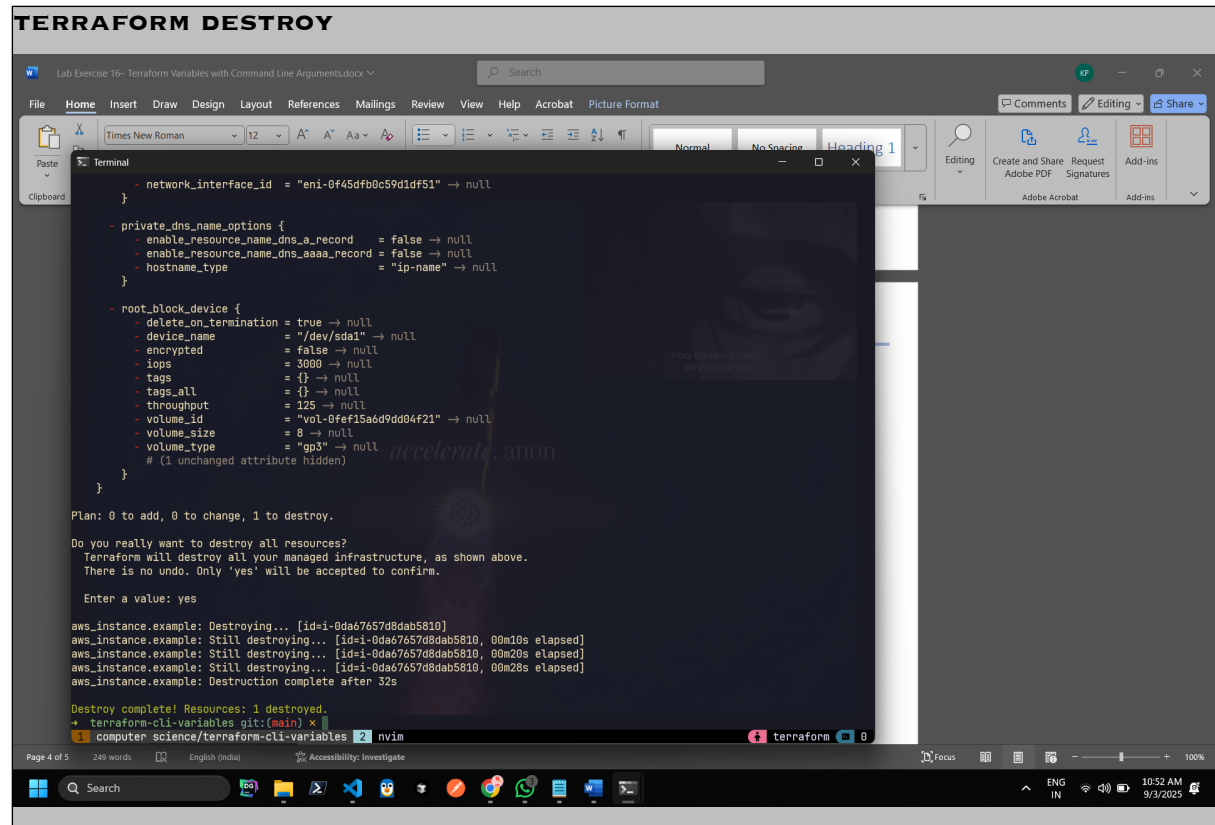


The screenshot shows a Windows terminal window with a dark background. The terminal output displays the AWS CLI command `aws ec2 describe-reservations` and its output. The output shows a list of reservations, including the `ReservationId`, `OwnerId`, `Groups`, and `Instances`. The `Instances` list includes details like `Architecture`, `BlockDeviceMappings`, `ClientToken`, `EbsOptimized`, `EnaSupport`, `Hypervisor`, `NetworkInterfaces`, `RootDeviceName`, `RootDeviceType`, `SecurityGroups`, `StateReason`, `Tags`, `VirtualizationType`, `CpuOptions`, `CapacityReservationSpecification`, `HibernationOptions`, and `MetadataOptions`. The terminal window has a title bar "Lab Exercise 16- Terraform Variables with Command Line Arguments.docx" and a menu bar with options like File, Home, Insert, Draw, Design, Layout, References, Mailings, Review, View, Help, Acrobat. The status bar at the bottom shows "Page 4 of 5", "249 words", "English (India)", and "Accessibility: Investigate".

```
"Reservations": [
  {
    "ReservationId": "r-015d9f0c9d89c2ad",
    "OwnerId": "27335465048",
    "Groups": [],
    "Instances": [
      {
        "Architecture": "x86_64",
        "BlockDeviceMappings": [],
        "ClientToken": "terraform-20250903050606187000000005",
        "EbsOptimized": false,
        "EnaSupport": true,
        "Hypervisor": "vm",
        "NetworkInterfaces": [],
        "RootDeviceName": "/dev/sda1",
        "RootDeviceType": "efs",
        "SecurityGroups": [],
        "StateReason": {
          "Code": "Client.UserInitiatedShutdown",
          "Message": "Client.UserInitiatedShutdown: User initiated shutdown"
        },
        "Tags": [
          {
            "Key": "Name",
            "Value": "My Instance"
          }
        ],
        "VirtualizationType": "hvm",
        "CpuOptions": {
          "CoreCount": 1,
          "ThreadsPerCore": 1
        },
        "CapacityReservationSpecification": {
          "CapacityReservationPreference": "open"
        },
        "HibernationOptions": {
          "Configured": false
        },
        "MetadataOptions": {}
      }
    ]
  }
]
```

## 5.CLEAN UP:

AFTER TESTING, YOU CAN CLEAN UP RESOURCES:



The screenshot shows a Windows terminal window with the title "TERRAFORM DESTROY". The terminal output displays the Terraform destroy command being executed. It lists the resources to be destroyed, including a network interface and a root block device. The output shows the plan, confirmation prompts, and the destruction progress for the resources. The terminal window is overlaid on a document titled "Lab Exercise 16- Terraform Variables with Command Line Arguments.docx".

```
network_interface_id = "eni-0f45dfb0c59d1df51" → null
}

private_dns_name_options {
  enable_resource_name_dns_a_record = false → null
  enable_resource_name_dns_aaaa_record = false → null
  hostname_type = "ip-name" → null
}

root_block_device {
  delete_on_termination = true → null
  device_name = "/dev/sda1" → null
  encrypted = false → null
  iops = 3000 → null
  tags = {} → null
  tags_all = {} → null
  throughput = 125 → null
  volume_id = "vol-0fe1f15a6d9dd04f21" → null
  volume_size = 8 → 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-0da67657d8dab5810]
aws_instance.example: Still destroying... [id=i-0da67657d8dab5810, 00m10s elapsed]
aws_instance.example: Still destroying... [id=i-0da67657d8dab5810, 00m20s elapsed]
aws_instance.example: Still destroying... [id=i-0da67657d8dab5810, 00m28s elapsed]
aws_instance.example: Destruction complete after 32s

Destroy complete! Resources: 1 destroyed.
→ terraform-cli-variables git:(main) x
1 computer science/terraform-cli-variables 2 nvim
```

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.