

LAB EXERCISE 4 –PROVISIONING AN EC2 INSTANCE ON AWS

PREREQUISITES: TERRAFORM INSTALLED: MAKE SURE YOU HAVE TERRAFORM INSTALLED ON YOUR MACHINE. FOLLOW THE OFFICIAL INSTALLATION GUIDE IF NEEDED.

AWS CREDENTIALS: ENSURE YOU HAVE AWS CREDENTIALS (ACCESS KEY ID AND SECRET ACCESS KEY) CONFIGURED. YOU CAN SET THEM UP USING THE AWS CLI OR BY SETTING ENVIRONMENT VARIABLES.

EXERCISE STEPS:

STEP 1: CREATE A NEW DIRECTORY:

CREATE A NEW DIRECTORY FOR YOUR TERRAFORM CONFIGURATION:

“TERRAFORM-DEMO”

STEP 2: CREATE TERRAFORM CONFIGURATION FILE (MAIN.TF):

CREATE A FILE NAMED MAIN.TF WITH THE FOLLOWING CONTENT:

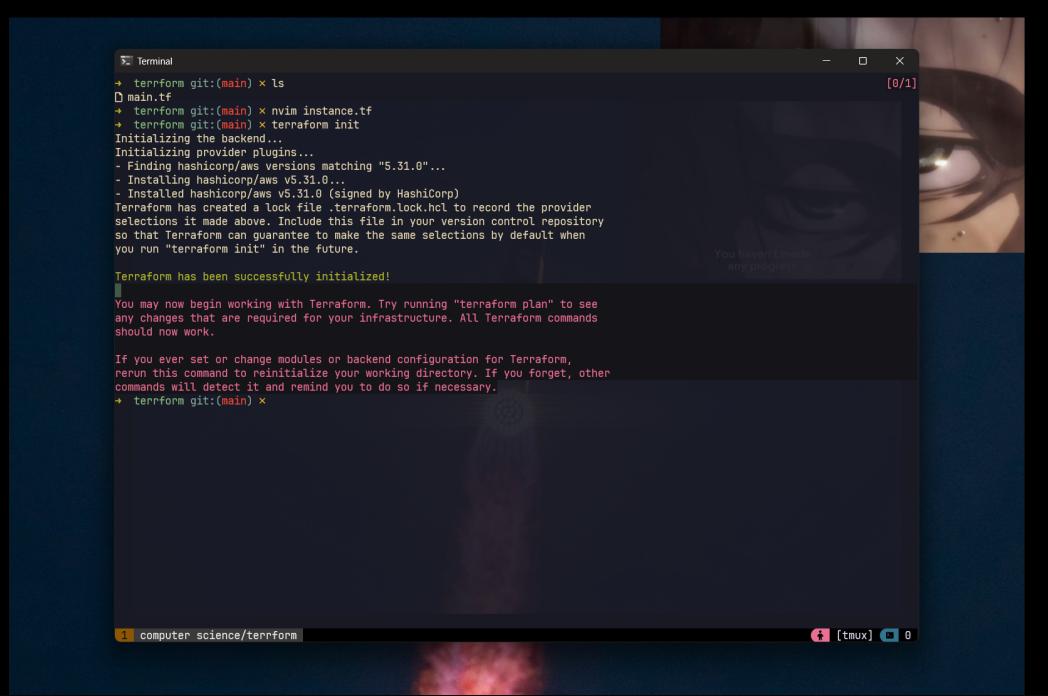
```
TERRAFORM {  
    REQUIRED_PROVIDERS {  
        AWS = {  
            SOURCE = "HASHICORP/AWS"  
            VERSION = "5.31.0"  
        }  
    }  
}  
PROVIDER "AWS" {  
    REGION      = "AP-SOUTH-1"  
    ACCESS_KEY  = "YOUR IAM ACCESS KEY"  
    SECRET_KEY  = "YOUR SECRET ACCESS KEY"  
}
```

THIS SCRIPT DEFINES AN AWS PROVIDER AND PROVISIONS AN EC2 INSTANCE.

STEP 3: INITIALIZE TERRAFORM:

RUN THE FOLLOWING COMMAND TO INITIALIZE YOUR TERRAFORM WORKING DIRECTORY:

TERRAFORM INIT



```
Terminal
→ terraform git:(main) × ls
└── main.tf
→ terraform git:(main) × nvim instance.tf
→ terraform git:(main) × terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.31.0"...
- Installing hashicorp/aws v5.31.0...
- Installed hashicorp/aws v5.31.0 (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.

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.
→ terraform git:(main) ×
```

STEP 4: CREATE TERRAFORM CONFIGURATION FILE FOR EC2 INSTANCE (INSTANCE.TF):

CREATE A FILE NAMED INSTNACE.TF WITH THE FOLLOWING CONTENT:

```
RESOURCE "AWS_INSTANCE" "MY-INSTANCE" {

    AMI = "AMI-03F4878755434977F"

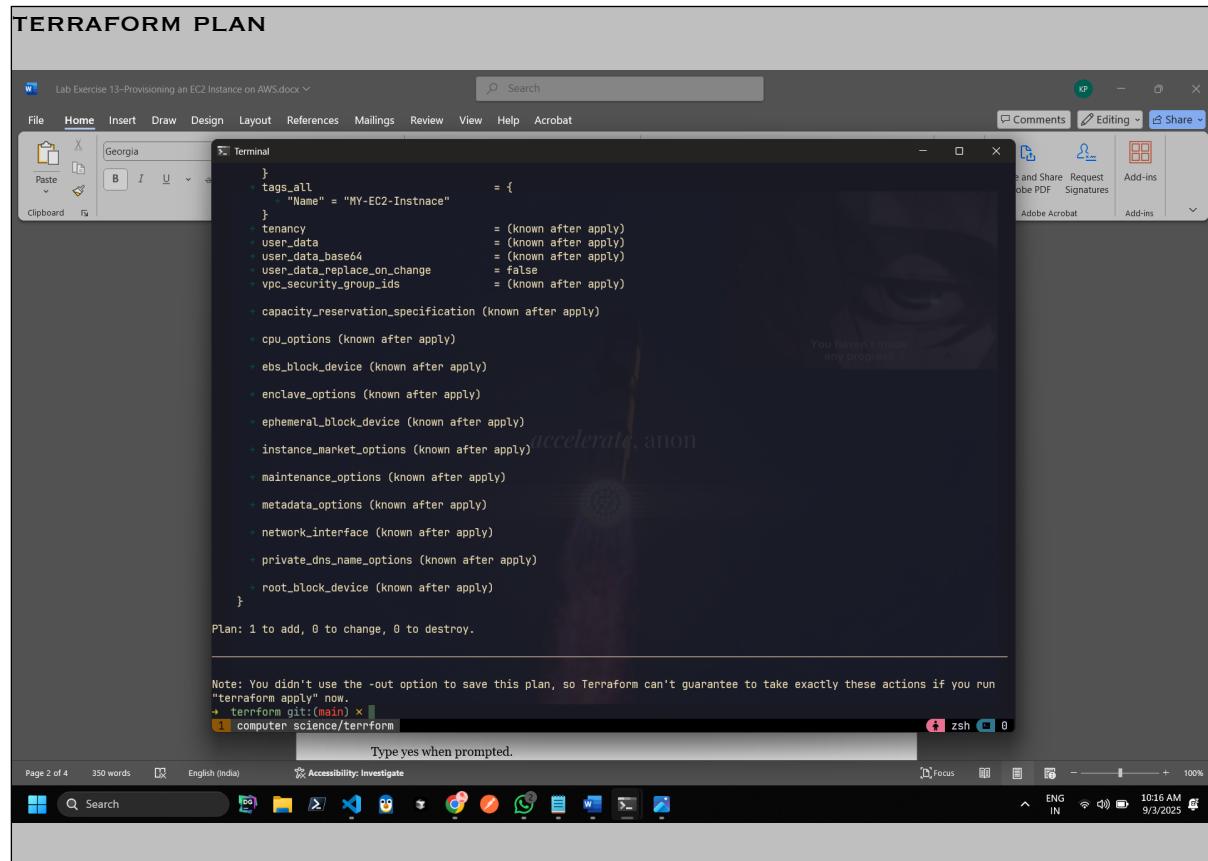
    INSTANCE_TYPE = "T2.MICRO"

    TAGS = {

        NAME = "MY-EC2-INSTNACE"
    }
}
```

STEP 5: REVIEW PLAN:

RUN THE FOLLOWING COMMAND TO SEE WHAT TERRAFORM WILL DO:



The screenshot shows a Microsoft Word document titled "Lab Exercise 13-Provisioning an EC2 Instance on AWS.docx". A terminal window is embedded within the document, displaying the following Terraform plan output:

```
tags_all = {
  "Name" = "MY-EC2-Instnace"
}
tenancy = (known after apply)
user_data = (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)
private_dns_name_options (known after apply)
root_block_device (known after apply)
}

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

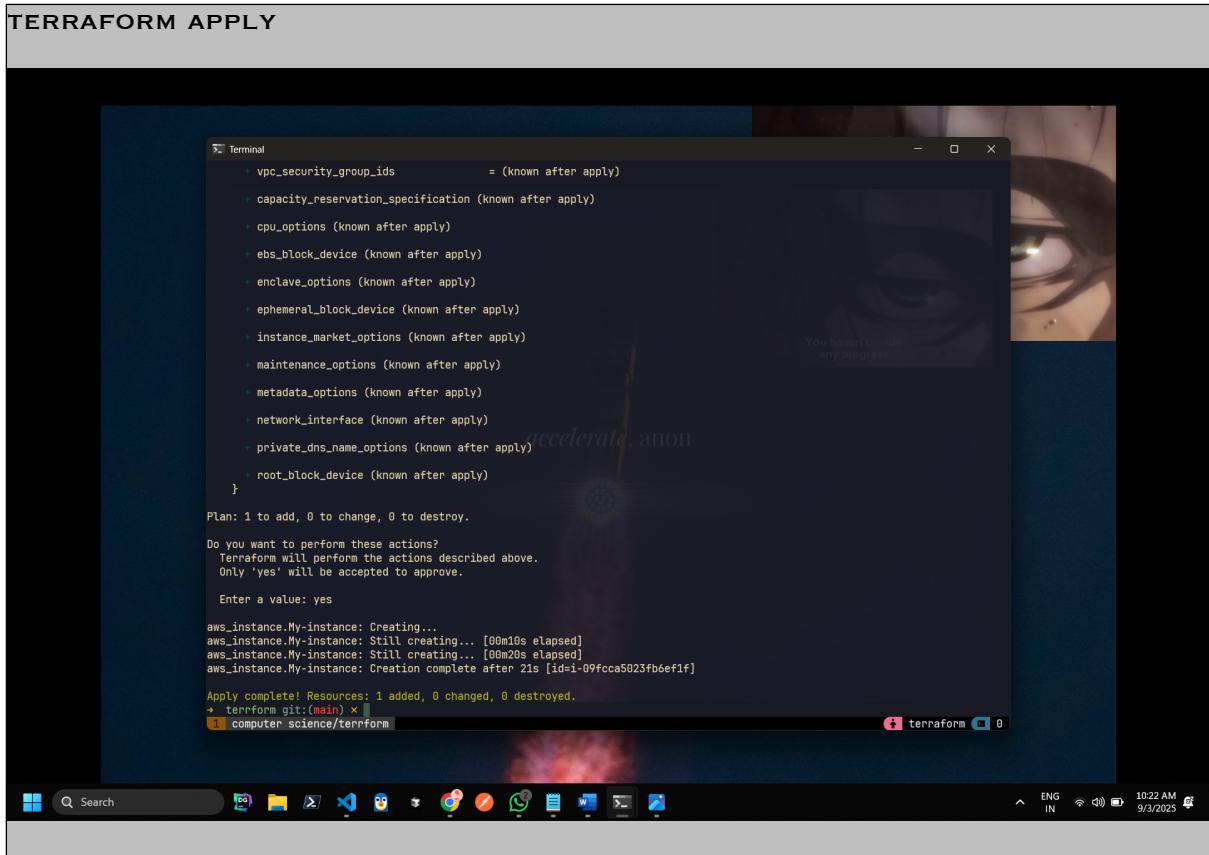
Note: You didn't use the `-out` option to save this plan, so Terraform can't guarantee to take exactly these actions if you run `"Terraform apply"` now.

The terminal window has a prompt: "Type yes when prompted." The status bar at the bottom of the terminal window shows "zsh 0".

REVIEW THE PLAN TO ENSURE IT ALIGNS WITH YOUR EXPECTATIONS.

STEP 6: APPLY CHANGES:

APPLY THE CHANGES TO CREATE THE AWS RESOURCES:



```
TERRAFORM APPLY

Terminal
  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)
  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.My-instance: Creating...
aws_instance.My-instance: Still creating... [00m10s elapsed]
aws_instance.My-instance: Still creating... [00m20s elapsed]
aws_instance.My-instance: Creation complete after 21s [id=i-09fcaca5023fb6ef1f]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
+ terraform git:(main) ✘ 1 computer science/terraform
```

TYPE YES WHEN PROMPTED.

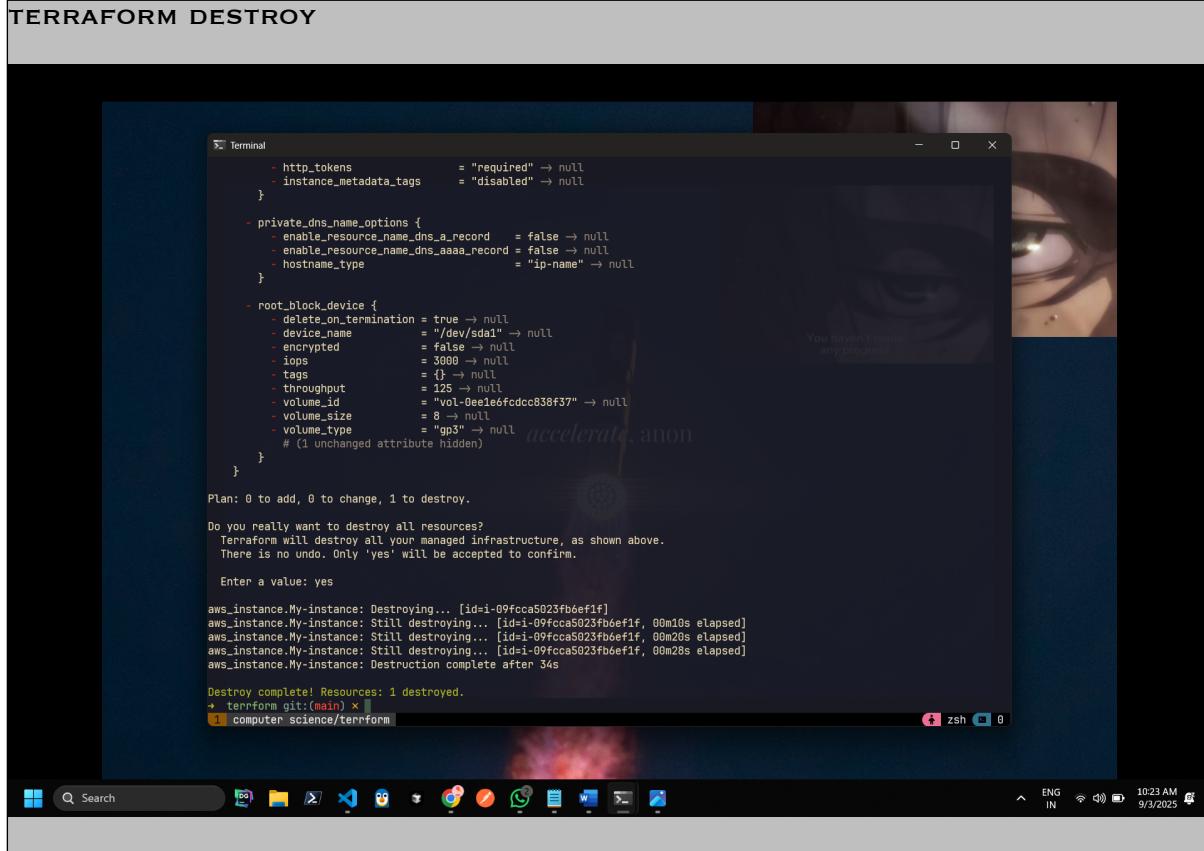
STEP 7: VERIFY RESOURCES:

AFTER THE TERRAFORM APPLY COMMAND COMPLETES, LOG IN TO YOUR AWS MANAGEMENT CONSOLE AND NAVIGATE TO THE EC2 DASHBOARD. VERIFY THAT THE EC2 INSTANCE HAS BEEN CREATED.

STEP 8: CLEANUP RESOURCES:

WHEN YOU ARE DONE EXPERIMENTING, RUN THE FOLLOWING COMMAND TO DESTROY THE CREATED RESOURCES:

TERRAFORM DESTROY



```
Terminal
  - http_tokens      = "required" → null
  - instance_metadata_tags = "disabled" → 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
  - throughput           = 125 → null
  - volume_id             = "vol-0ee1e6fc0cc838f37" → 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.My-instance: Destroying... [id=i-09fccaa5023fb6ef1f]
aws_instance.My-instance: Still destroying... [id=i-09fccaa5023fb6ef1f, 00m10s elapsed]
aws_instance.My-instance: Still destroying... [id=i-09fccaa5023fb6ef1f, 00m20s elapsed]
aws_instance.My-instance: Still destroying... [id=i-09fccaa5023fb6ef1f, 00m28s elapsed]
aws_instance.My-instance: Destruction complete after 34s

Destroy complete! Resources: 1 destroyed.
+ terraform git:(main) ✘
1 computer science/terraform
```

TYPE YES WHEN PROMPTED.

NOTES:

CUSTOMIZE THE INSTANCE.TF FILE TO PROVISION DIFFERENT AWS RESOURCES.

EXPLORE THE TERRAFORM AWS PROVIDER DOCUMENTATION FOR ADDITIONAL AWS RESOURCES AND CONFIGURATION OPTIONS.

ALWAYS BE CAUTIOUS WHEN RUNNING TERRAFORM DESTROY TO AVOID ACCIDENTAL RESOURCE DELETION.

THIS EXERCISE PROVIDES A BASIC INTRODUCTION TO USING TERRAFORM WITH THE AWS PROVIDER. FEEL FREE TO EXPLORE MORE COMPLEX TERRAFORM CONFIGURATIONS AND RESOURCES BASED ON YOUR NEEDS.