Experiment no. 03

Aim: To construct, change, and destroy AWS using Terraform.

LOs: LO1, LO3

Theory:

Managing AWS Infrastructure with Terraform

This guide will walk you through the practical steps of using Terraform to **construct, change, and destroy** infrastructure in Amazon Web Services (AWS). We will use a simple EC2 instance as our example resource.

Prerequisites

Before you begin, ensure you have the following set up:

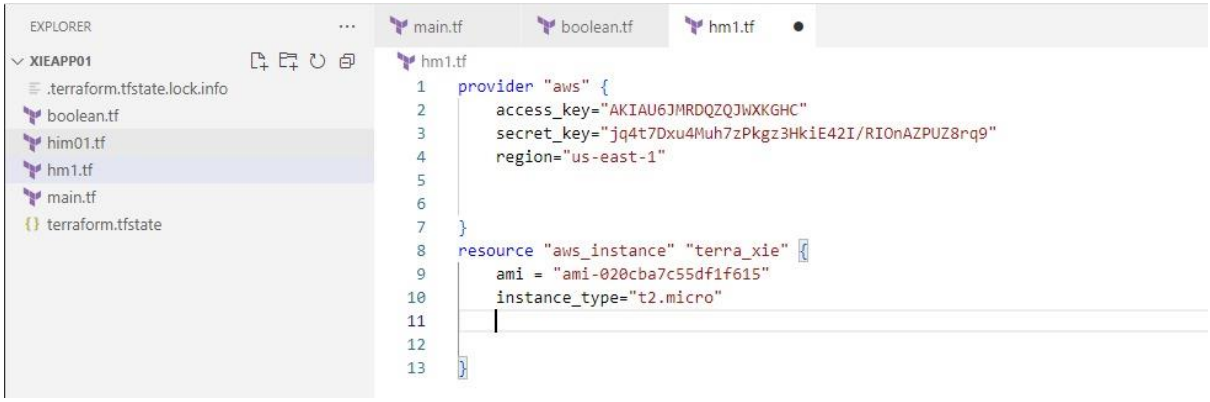
1. **An AWS Account:** You need an active AWS account.
2. **Terraform Installed:** You must have Terraform installed on your machine.
3. **AWS CLI Configured:** The easiest way to provide AWS credentials to Terraform is by installing the AWS CLI and configuring it with your access keys by running aws configure. Terraform will automatically use these credentials.

Step 1: Constructing Infrastructure (Create)

First, we'll define a simple configuration to create an AWS EC2 instance.

1. **Create a Project Directory:** Create a new folder for your Terraform project (e.g., terraform-aws-demo).
2. **Create a Configuration File:** Inside this folder, create a file named main.tf and add the following code. This code tells Terraform to use the AWS provider and defines a single EC2 instance.
3. Check that not any instance is running in EC2
4. Create an IAM user with programmatic password
5. Create access key and secret key for command line interface

6. Write a terraform program



7. **Initialize Terraform:** Open a terminal in your project directory and run:

8. terraform init

```
C:\xieapp01>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.5.0...
- Installed hashicorp/aws v6.5.0 (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.
```

This will download the AWS provider plugin.

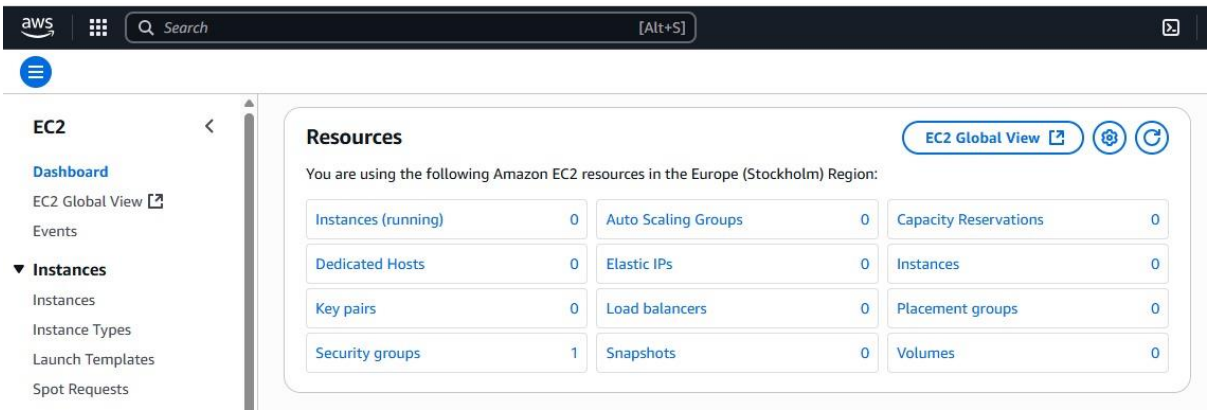
9. **Plan the Changes:** Run the plan command to see what Terraform will do:

10. terraform plan

The output will show that **1 resource will be added**.

11. **Apply the Changes:** To create the EC2 instance, run:

12. check the instance on EC2 before running the command terraform apply



13. terraform apply

Terraform will show you the plan again and ask for confirmation. Type yes and press Enter. After a few moments, your EC2 instance will be up and running in your AWS account!

Step 2: Modifying Infrastructure (Change)

Infrastructure is rarely static. Let's modify our instance. For example, we'll change its type from t2.micro to t2.small.

1. **Update the Configuration:** Edit your main.tf file and change the instance_type line:
2. **Plan the Changes:** Run terraform plan again.
3. terraform plan

```
C:\xieapp01>terraform plan

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.terra_xie will be created
  + resource "aws_instance" "terra_xie" {
      + ami                                  = "ami-020cba7c55df1f615"
      + 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)
      + 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"
```

The output will now show that **1 resource will be changed**. It will highlight the specific attribute that is being modified (instance_type).

4. **Apply the Changes:** Run terraform apply and confirm with yes.
5. terraform apply

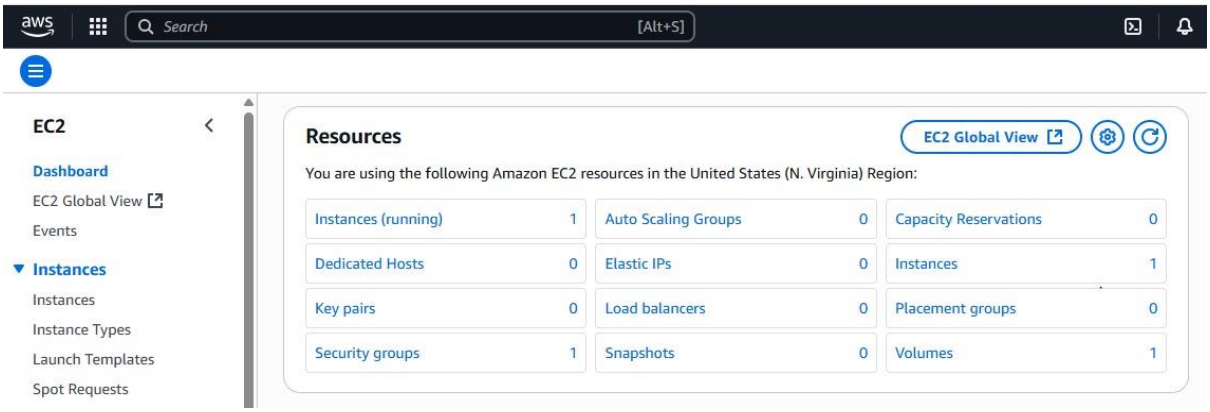
```
C:\xieapp01>terraform apply

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.terra_xie will be created
  + resource "aws_instance" "terra_xie" {
      + ami                                  = "ami-020cba7c55df1f615"
      + 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)
      + 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)
```

6. check terraform create an instance on EC2



Terraform will now connect to AWS and modify the existing EC2 instance to the new t2.small size.

Step 3: Destroying Infrastructure (Destroy)

Once you no longer need the infrastructure, you can easily tear it all down.

1. **Run the Destroy Command:** In your terminal, run the following command:
2. terraform destroy
3. **Confirm Destruction:** Terraform will show you a plan of all the resources that will be destroyed and will ask for a final confirmation. This action is irreversible.

```
C:\xieapp01>terraform destroy
aws_instance.terra_xie: Refreshing state... [id=i-0544e4159f9f631b1]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # aws_instance.terra_xie will be destroyed
  - resource "aws_instance" "terra_xie" {
      - ami                                  = "ami-020cba7c55df1f615" -> null
      - arn                                  = "arn:aws:ec2:us-east-1:339932683315:instance/i-0544e4159f9f631b1" -> null
      - associate_public_ip_address          = true -> null
      - availability_zone                    = "us-east-1d" -> null
      - disable_api_stop                     = false -> null
      - disable_api_termination              = false -> null
      - ebs_optimized                        = false -> null
      - get_password_data                    = false -> null
      - hibernation                          = false -> null
      - id                                   = "i-0544e4159f9f631b1" -> null
      - instance_initiated_shutdown_behavior = "stop" -> null
      - instance_state                       = "running" -> null
      - instance_type                        = "t3.micro" -> null
      - ipv6_address_count                   = 0 -> null
      - ipv6_addresses                       = [] -> null
      - monitoring                           = false -> null
      - placement_partition_number           = 0 -> null
      - primary_network_interface_id         = "eni-0aa00550236ca5113" -> null
      - private_dns                          = "ip-172-31-37-180.ec2.internal" -> null
      - private_ip                           = "172.31.37.180" -> null
      - public_dns                           = "ec2-3-89-134-40.compute-1.amazonaws.com" -> null
      - public_ip                            = "3.89.134.40" -> null
      - region                               = "us-east-1" -> null
      - secondary_private_ips                = [] -> null
      - security_groups                      = [
          - "default",
```

Type yes and press Enter.

Terraform will now delete the EC2 instance it created. Your AWS environment is back to the state it was in before you started.

This **create, change, and destroy** workflow is fundamental to managing infrastructure with Terraform, providing a safe, repeatable, and automated process.

Conclusion:

This guide demonstrates the practical power of Terraform's core workflow in a real-world scenario. By using simple, declarative configuration files, you were able to construct, modify, and ultimately destroy an AWS EC2 instance in a controlled and predictable manner. This process highlights the core benefit of Infrastructure as Code: it transforms complex manual tasks into a manageable, version-controlled, and automated workflow, giving you full lifecycle control over your cloud resources.

LOs: LO1, LO3 are achieved.

POs: PO1, PO2, PO3, PO5, PO8, PO10, PO12 are achieved.