Experiment 6

Aim:To build, change, and destroy infrastructure on AWS, GCP, Microsoft Azure, or DigitalOcean using Terraform, a tool for automating the management of cloud resources.

Theory:

Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. It allows developers and system administrators to define, provision, and manage cloud infrastructure using a declarative configuration language called HashiCorp Configuration Language (HCL). Terraform supports multiple cloud providers, including AWS, GCP, Azure, and DigitalOcean, making it a versatile tool for managing cloud environments.

Key Concepts of Terraform:

1. Infrastructure as Code (IaC): Terraform enables defining cloud infrastructure through code, allowing for version control, reproducibility, and collaboration.

2. State Management: Terraform maintains the state of the infrastructure, allowing it to track and manage changes over time.

3. Resource Provisioning: It can provision various resources like virtual machines, storage, networking, and more across multiple cloud platforms.

4. Modular Approach: Infrastructure configurations can be broken into reusable modules, making them easy to manage and maintain.

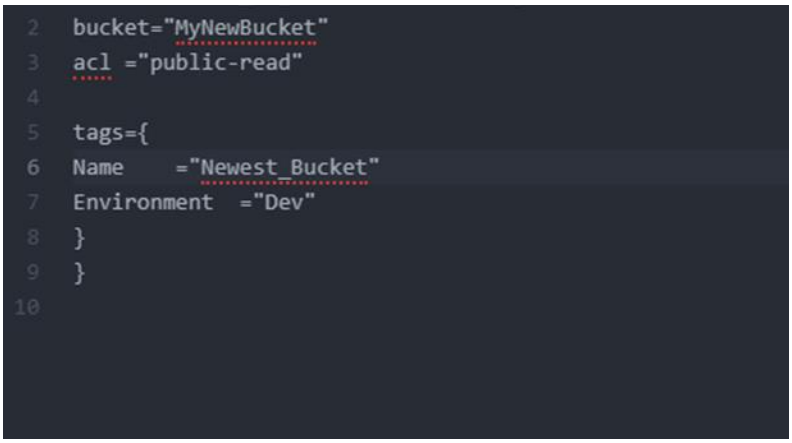
5. Lifecycle Management: Terraform provides commands to create (apply), update (plan), and destroy (destroy) resources, making it easy to manage the full lifecycle of infrastructure.

Benefits of Using Terraform:

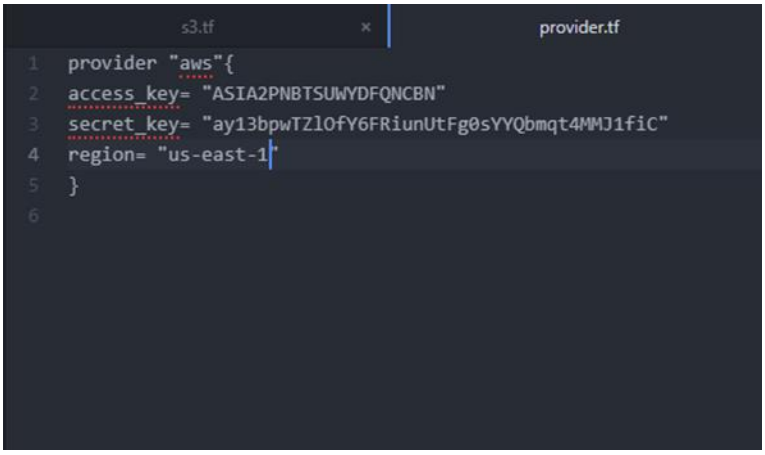
- Multi-Cloud Compatibility: Supports various cloud providers, offering flexibility and reducing vendor lock-in.
- Automation: Automates infrastructure provisioning, reducing manual efforts and minimizing errors.
- Version Control: Changes in infrastructure can be tracked, reviewed, and rolled back using version control systems.

- Scalability: Easily scales infrastructure up or down according to needs, optimizing resource usage and cost.

Step 1: Write a Terraform Script in Atom for creating S3 Bucket on Amazon AWS

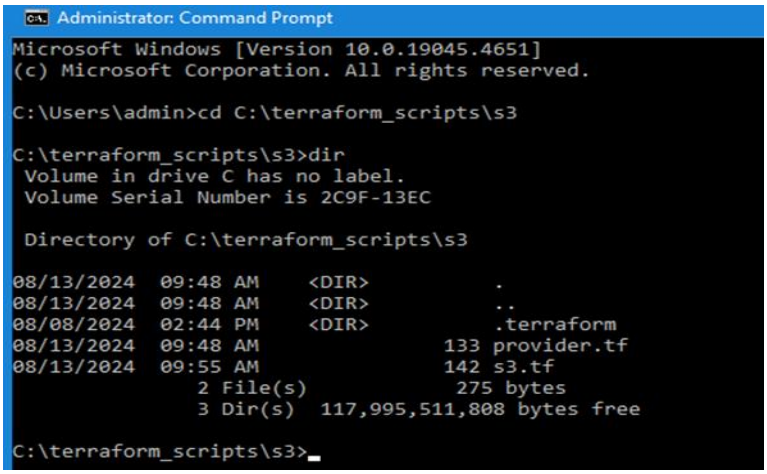


Create a new provider.tf file and write the following contents into it.

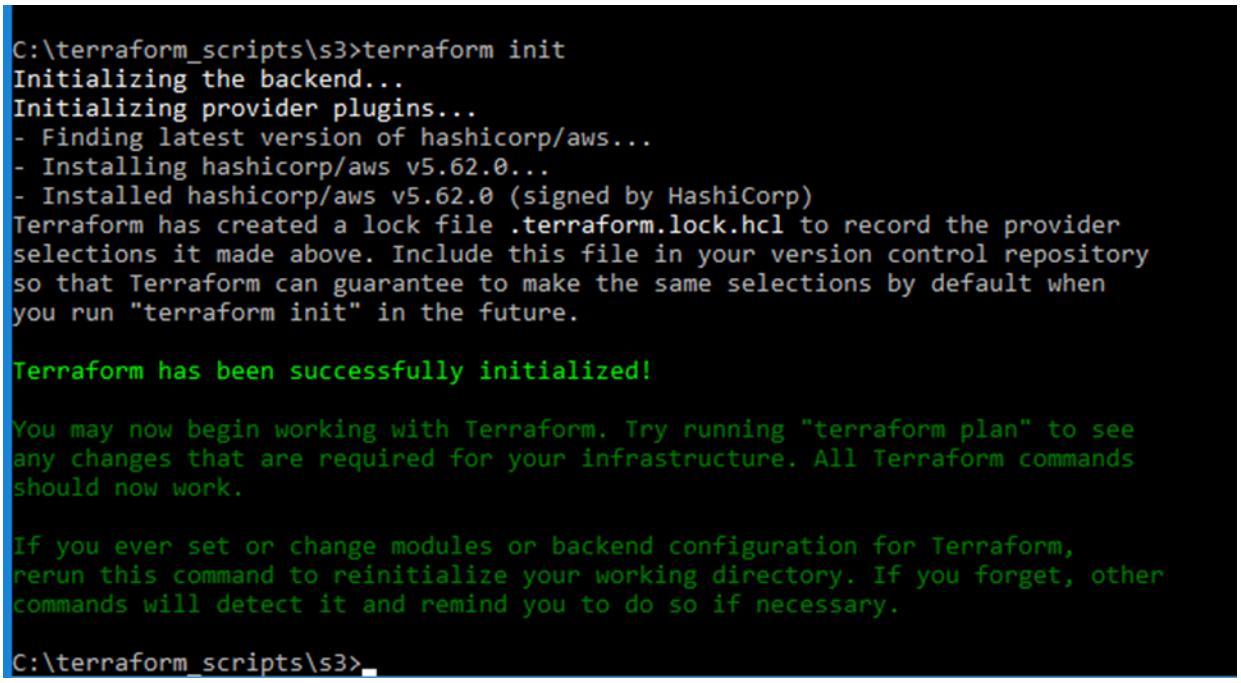


Save both the files in same directory Terraform_Scripts/S3

Step 2: Open Command Prompt and go to Terraform_Script\S3 directory where our .tf files are stored



Step 3: Execute Terraform Init command to initialize the resources

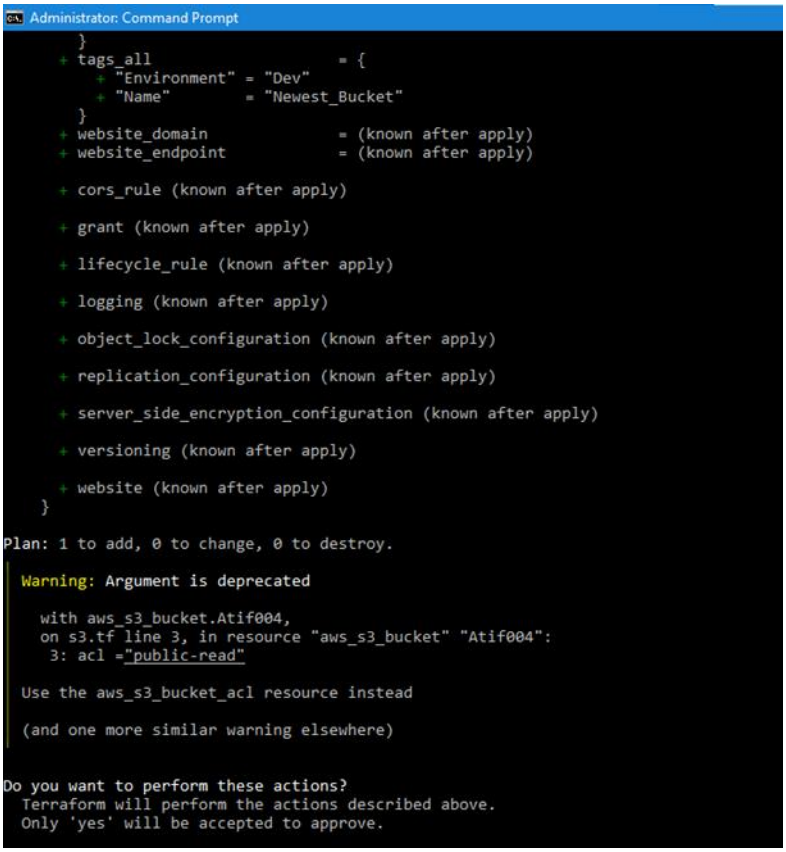


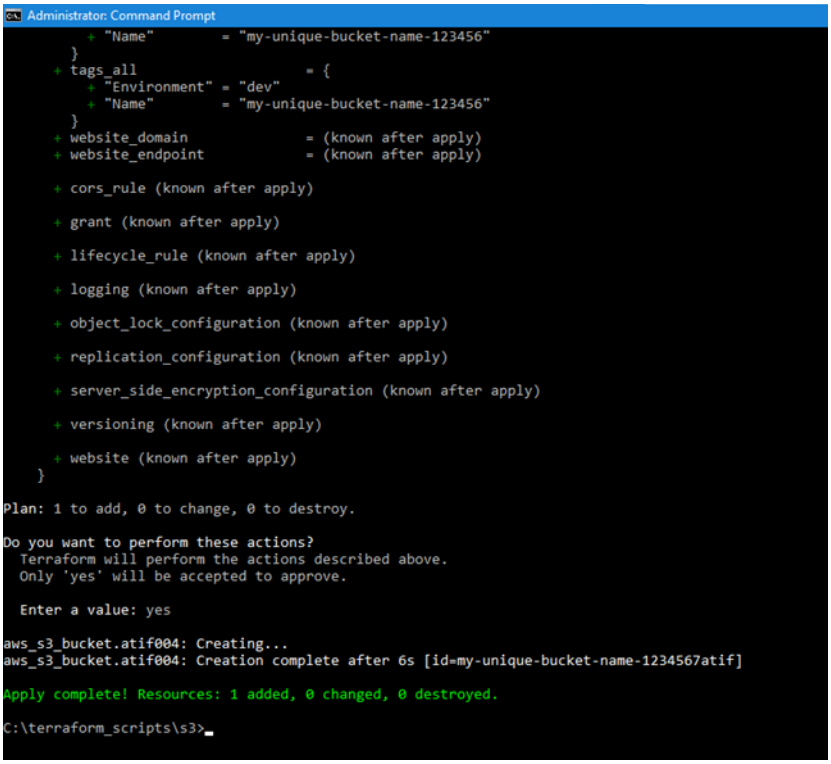
Step 4: Execute Terraform plan to see the available resources

```
C:\terraform_scripts\s3>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_s3_bucket.Atif004 will be created
  + resource "aws_s3_bucket" "Atif004" {
      + acceleration_status         = (known after apply)
      + acl                         = "public-read"
      + arn                         = (known after apply)
      + bucket                      = "MyNewBucket"
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Environment" = "Dev"
          + "Name"        = "Newest_Bucket"
        }
      + tags_all                    = {
          + "Environment" = "Dev"
          + "Name"        = "Newest_Bucket"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + cors_rule (known after apply)

      + grant (known after apply)

      + lifecycle_rule (known after apply)

      + logging (known after apply)
```

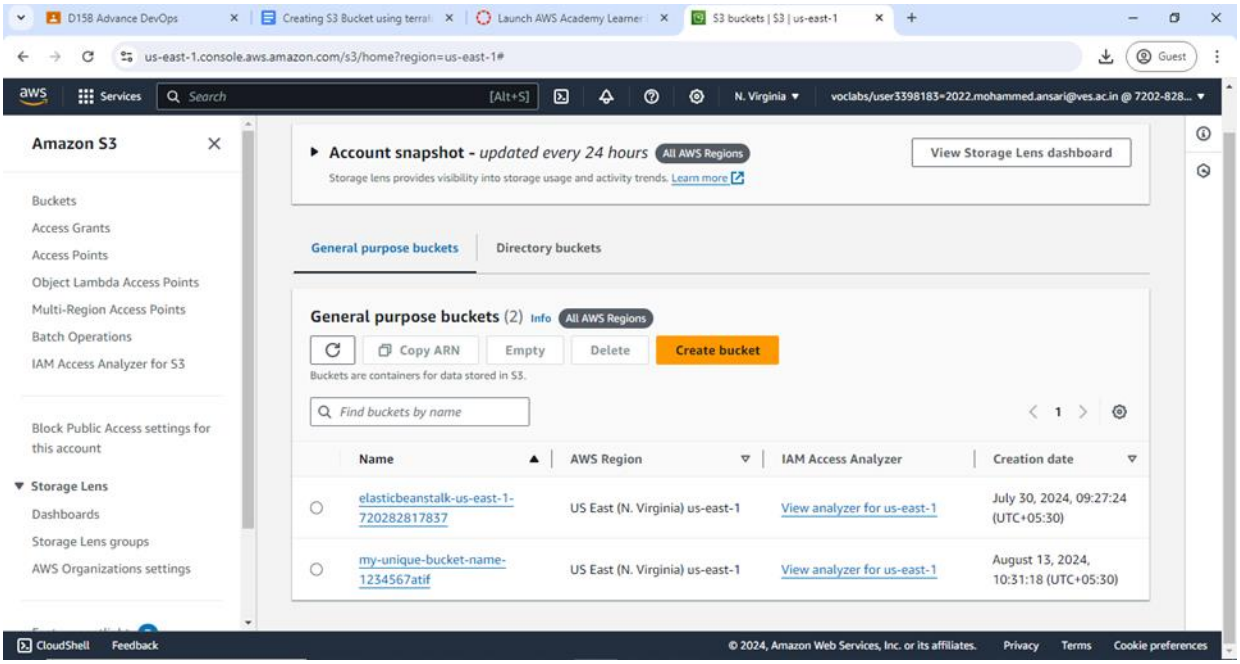
Step 5: Execute Terraform apply to apply the configuration, which will automatically create an S3 bucket based on our configuration.

```
Administrator: Command Prompt
      + logging (known after apply)

      + object_lock_configuration (known after apply)

      + replication_configuration (known after apply)

      + server_side_encryption_configuration (known after apply)

      + versioning (known after apply)

      + website (known after apply)
    }

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

  Warning: Argument is deprecated

    with aws_s3_bucket.Atif004,
    on s3.tf line 3, in resource "aws_s3_bucket" "Atif004":
     3: acl ="public-read"

  Use the aws_s3_bucket_acl resource instead

  (and one more similar warning elsewhere)

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.

C:\terraform_scripts\s3>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_s3_bucket.Atif004 will be created
  + resource "aws_s3_bucket" "Atif004" {
      + acceleration_status         = (known after apply)
      + acl                         = "public-read"
      + arn                         = (known after apply)
      + bucket                      = "MyNewBucket"
```





AWS S3bucket dashboard, Before Executing Apply command:

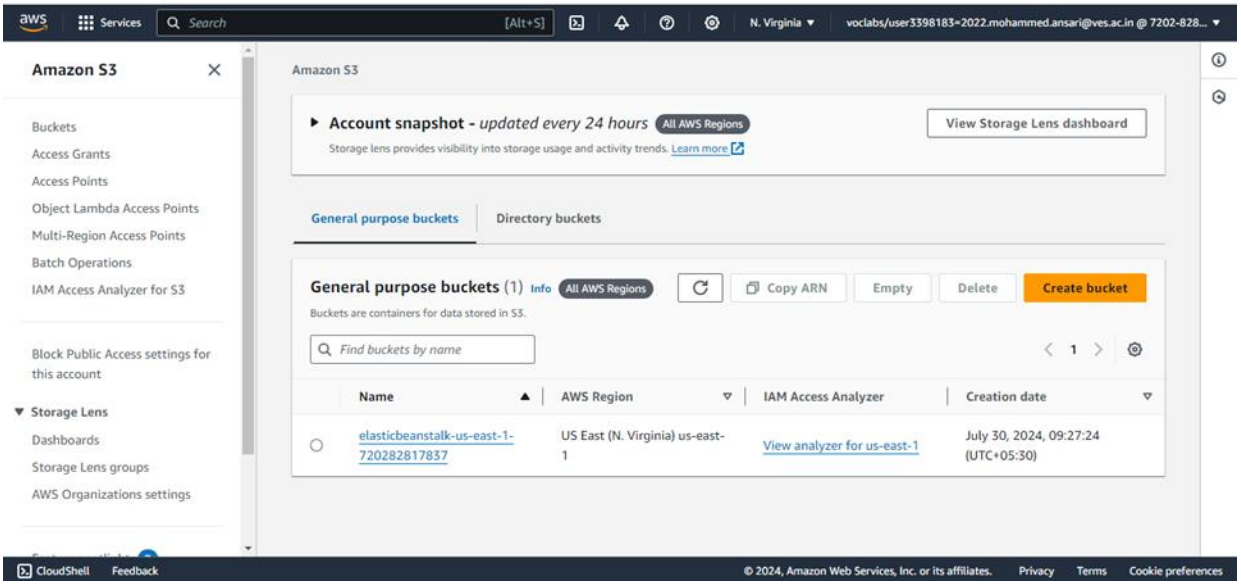


Step 6: Execute Terraform destroy to delete the configuration, which will automatically delete an EC2 instance

```
C:\terraform_scripts\s3>terraform destroy
aws_s3_bucket.atif004: Refreshing state... [id=my-unique-bucket-name-1234567atif]

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_s3_bucket.atif004 will be destroyed
  - resource "aws_s3_bucket" "atif004" {
      - arn                         = "arn:aws:s3:::my-unique-bucket-name-1234567atif" -> null
      - bucket                      = "my-unique-bucket-name-1234567atif" -> null
      - bucket_domain_name          = "my-unique-bucket-name-1234567atif.s3.amazonaws.com" -> null
      - bucket_regional_domain_name = "my-unique-bucket-name-1234567atif.s3.us-east-1.amazonaws.com" -> null
      - force_destroy               = false -> null
      - hosted_zone_id              = "Z3AQBSTGFYJSTF" -> null
      - id                          = "my-unique-bucket-name-1234567atif" -> null
      - object_lock_enabled         = false -> null
      - region                      = "us-east-1" -> null
      - request_payer               = "BucketOwner" -> null
      - tags                        = {
          - "Environment" = "dev"
```

```
        }
    }

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_s3_bucket.atif004: Destroying... [id=my-unique-bucket-name-1234567atif]
aws_s3_bucket.atif004: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.
```

AWS EC2 dashboard, After Executing Destroy step:



Conclusion:

Using Terraform to build, change, and destroy infrastructure across AWS, GCP, Microsoft Azure, and DigitalOcean showcases the power and flexibility of Infrastructure as Code (IaC). Terraform's ability to automate infrastructure provisioning, manage state, and support multiple cloud platforms streamlines operations, reduces human error, and enhances scalability. By employing Terraform, organizations can efficiently manage their cloud environments, enforce best practices, and achieve consistent infrastructure management across diverse cloud providers, ultimately accelerating their cloud adoption and operational agility.