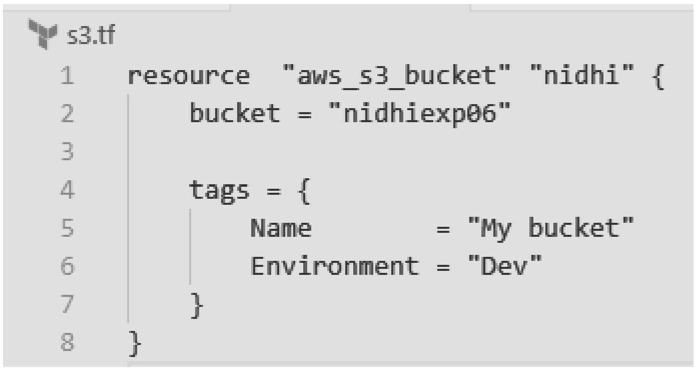
Name : Nidhi Rijhwani
Subject : Adv Devops Exp No 06

Aim: To Build, change, and destroy AWS / GCP /Microsoft Azure/ DigitalOcean infrastructure Using Terraform. (S3 bucket or Docker) fdp.

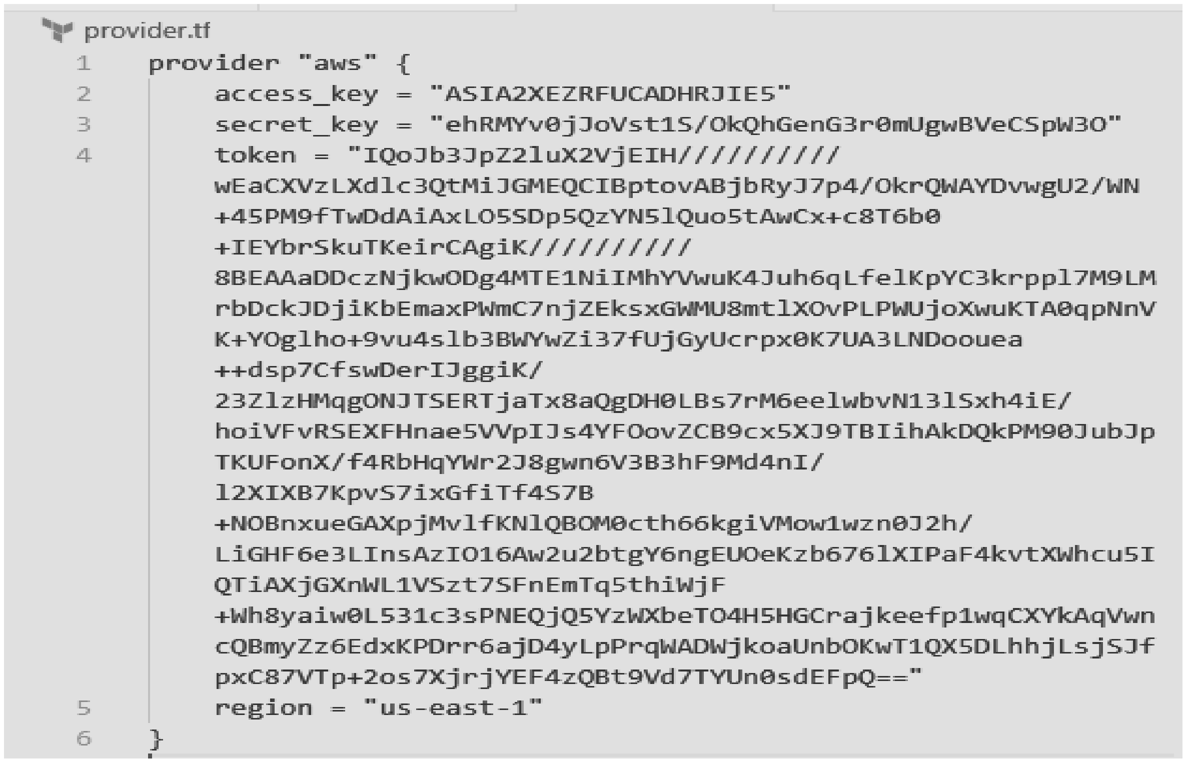
Theory:
Infrastructure as Code (IaC) is a practice where infrastructure is provisioned and managed using code rather than manual processes. Terraform is a widely-used IaC tool that allows users to define infrastructure resources in a declarative configuration language.

Key Concepts:
1. Providers: Terraform supports multiple cloud providers, allowing users to manage resources across AWS, GCP, Azure, and DigitalOcean. Each provider has its own set of APIs and resource types.
2. Resources: Resources are the fundamental components of a Terraform configuration. For example, an S3 bucket in AWS or a Docker container in a container orchestration platform like ECS.
3. State Management: Terraform maintains a state file that maps the configurations defined in code to the actual infrastructure. This state file is crucial for managing changes and ensuring the desired state is achieved.
4. Execution Plans: Before making changes, Terraform generates an execution plan that outlines the proposed actions, allowing users to review and approve changes before they are applied.
5. Modules: Terraform supports reusable code blocks called modules, which allow for organizing and sharing configurations, promoting best practices, and reducing duplication.

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


s3.tf
```
resource  "aws_s3_bucket" "nidhi" {
    bucket = "nidhiexp06"

    tags = {
        Name        = "My bucket"
        Environment = "Dev"
    }
}
```

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

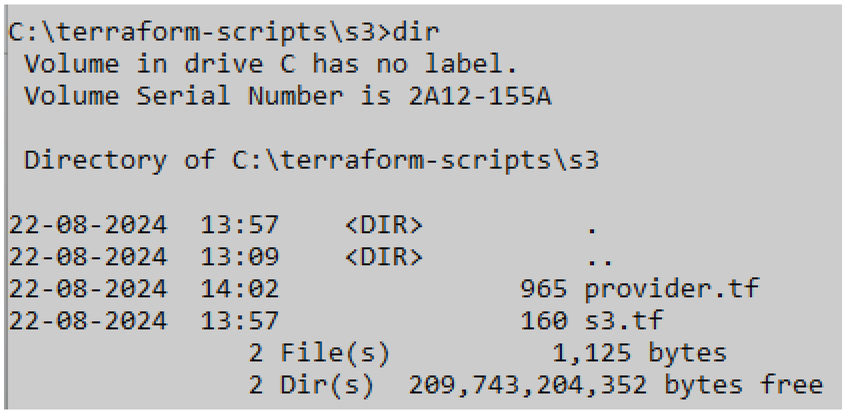


provider.tf

```
provider "aws" {
    access_key = "ASIA2XEZRFUCADHRJIE5"
    secret_key = "ehRMYv0jJoVst1S/OkQhGenG3r0mUgwBVeCSpW3O"
    token = "IQoJb3JpZ2luX2VjEIH//////////
    wEaCXVzLXdlc3QtMiJGMEQCIBptovABjbRyJ7p4/OkrQWAYDvwgU2/WN
    +45PM9fTwDdAiAxLO5SDp5QzYN5lQuo5tAwCx+c8T6b0
    +IEYbrSkuTKeirCAgiK//////////
    8BEAAaDDczNjkwODg4MTE1NiIMhYVwuK4Juh6qLfelKpYC3krppl7M9LM
    rbDckJDjiKbEmaxPWmC7njZEksxGWMU8mtlXOvPLPWUjoXwuKTA0qpNnV
    K+YOglho+9vu4slb3BWYwZi37fUjGyUcrpx0K7UA3LNDoouea
    ++dsp7CfswDerIJggiK/
    23ZlzHMqgONJTSERTjaTx8aQgDH0LBs7rM6eelwbvN13lSxh4iE/
    hoiVFvRSEXFHnae5VVpIJs4YFOovZCB9cx5XJ9TBIihAkDQkPM90JubJp
    TKUFonX/f4RbHqYWr2J8gwn6V3B3hF9Md4nI/
    l2XIXB7KpvS7ixGfiTf4S7B
    +NOBnxueGAXpjMvlfKNlQBOM0cth66kgiVMow1wzn0J2h/
    LiGHF6e3LInsAzIO16Aw2u2btgY6ngEUOeKzb676lXIPaF4kvtXWhcu5I
    QTiAXjGXnWL1VSzt7SFnEmTq5thiWjF
    +Wh8yaiw0L531c3sPNEQjQ5YzWXbeTO4H5HGCrajkeefp1wqCXYkAqVwn
    cQBmyZz6EdxKPDrr6ajD4yLpPrqWADWjkoaUnbOKwT1QX5DLhhjLsjSJf
    pxC87VTp+2os7XjrjYEF4zQBt9Vd7TYUn0sdEFpQ=="
    region = "us-east-1"
}
```

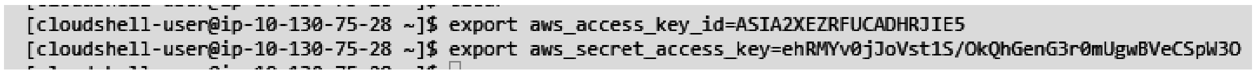
Save both the files in same directory Terraform_Scripts/S3

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

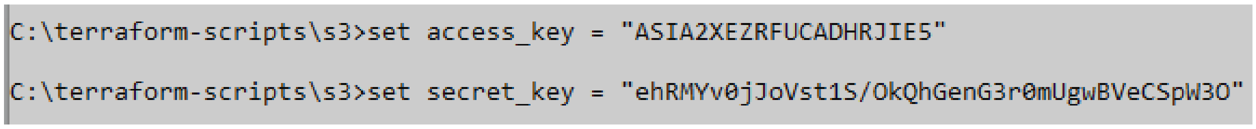


```
C:\terraform-scripts\s3>dir
 Volume in drive C has no label.
 Volume Serial Number is 2A12-155A

 Directory of C:\terraform-scripts\s3

22-08-2024  13:57    <DIR>          .
22-08-2024  13:09    <DIR>          ..
22-08-2024  14:02               965 provider.tf
22-08-2024  13:57               160 s3.tf
               2 File(s)          1,125 bytes
               2 Dir(s)  209,743,204,352 bytes free
```

Step 3: Set keys from both ends.

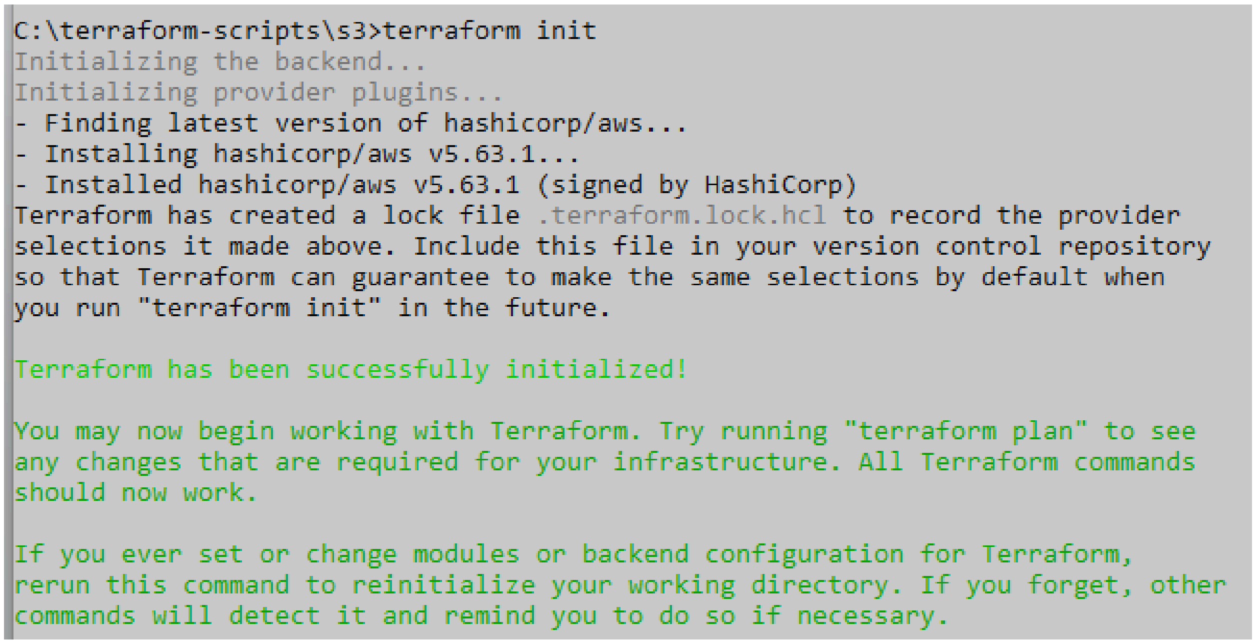


```
[cloudshell-user@ip-10-130-75-28 ~]$ export aws_access_key_id=ASIA2XEZRFUCADHRJIE5
[cloudshell-user@ip-10-130-75-28 ~]$ export aws_secret_access_key=ehRMYv0jJoVst1S/OkQhGenG3r0mUgwBVeCSpW3O
```



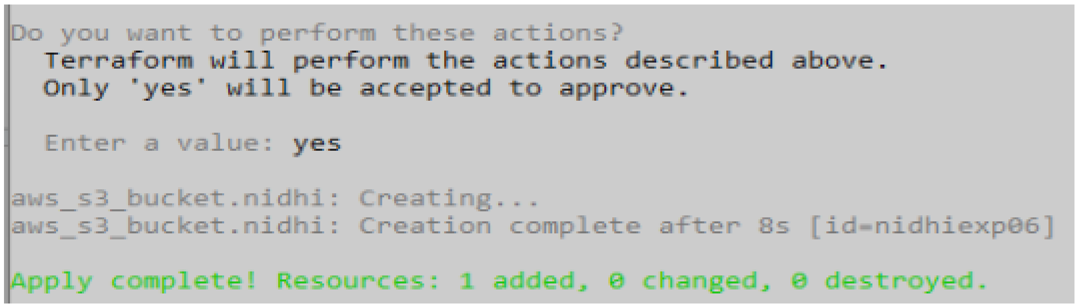
```
C:\terraform-scripts\s3>set access_key = "ASIA2XEZRFUCADHRJIE5"

C:\terraform-scripts\s3>set secret_key = "ehRMYv0jJoVst1S/OkQhGenG3r0mUgwBVeCSpW3O"
```

Step 4 : Execute Terraform Init command to initialize the resources.

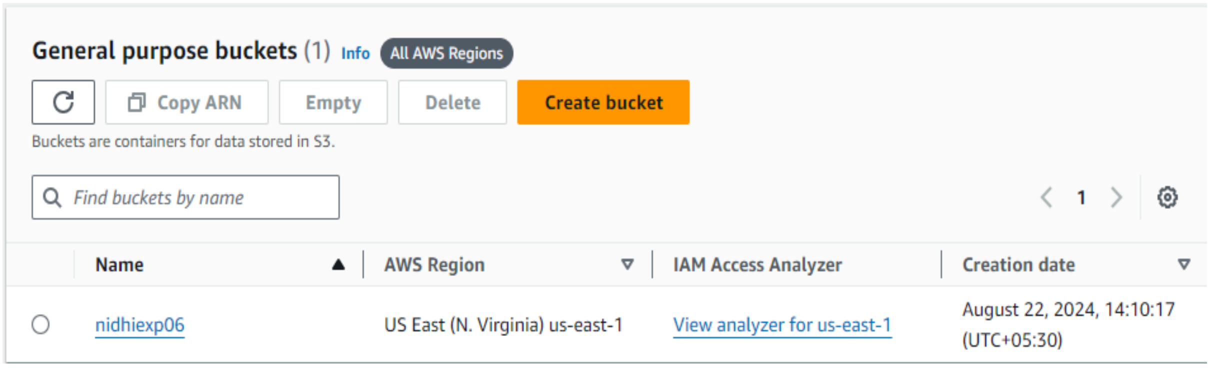


```
C:\terraform-scripts\s3>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.63.1...
- Installed hashicorp/aws v5.63.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record 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.
```

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



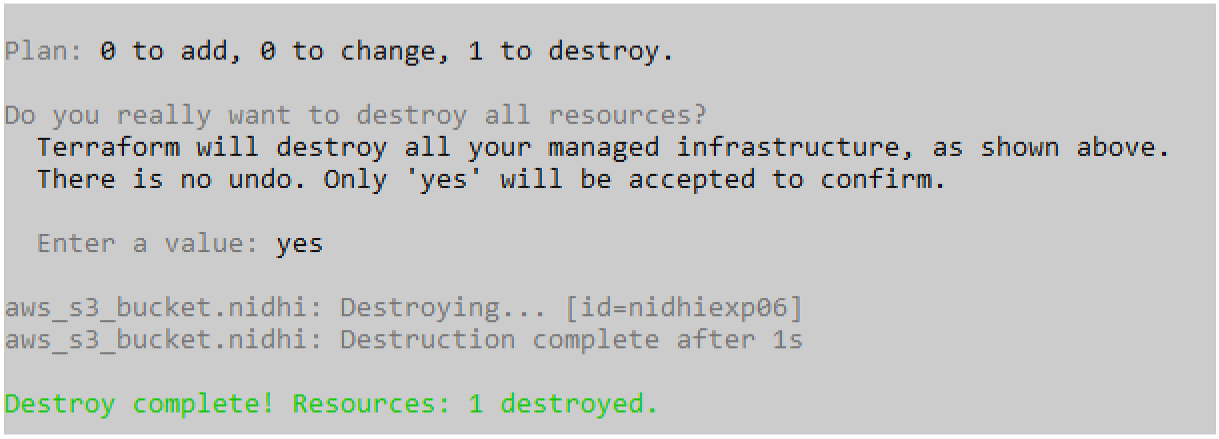
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_s3_bucket.nidhi: Creating...
aws_s3_bucket.nidhi: Creation complete after 8s [id=nidhiexp06]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```



Buckets are containers for data stored in S3.

	Name	AWS Region	IAM Access Analyzer	Creation date
○	nidhiexp06	US East (N. Virginia) us-east-1	View analyzer for us-east-1	August 22, 2024, 14:10:17 (UTC+05:30)

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



```
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.nidhi: Destroying... [id=nidhiexp06]
aws_s3_bucket.nidhi: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.
```

Conclusion:
Terraform streamlines infrastructure management across cloud providers, offering flexibility, automation, and collaboration. By treating infrastructure as code, it enhances efficiency and scalability in cloud operations.