



Vivekanand Education Society's

Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai,, Approved by AICTE & Recognized by Govt. of Maharashtra
Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.

Department of Information Technology

A.Y. 2024-25

Advance DevOps Lab

Experiment 06

Aim: To Build, change, and destroy AWS / GCP /Microsoft Azure/ DigitalOcean infrastructure Using Terraform.

Roll No.	44
Name	GANESH SANJAY PANDHRE
Class	D15B
Subject	Advance DevOps Lab
LO Mapped	LO1: To understand the fundamentals of Cloud Computing and be fully proficient with Cloud based DevOps solution deployment options to meet your business requirements. LO3: To apply best practices for managing infrastructure as code environments and use terraform to define and deploy cloud infrastructure.
Grade:	

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

THEORY:

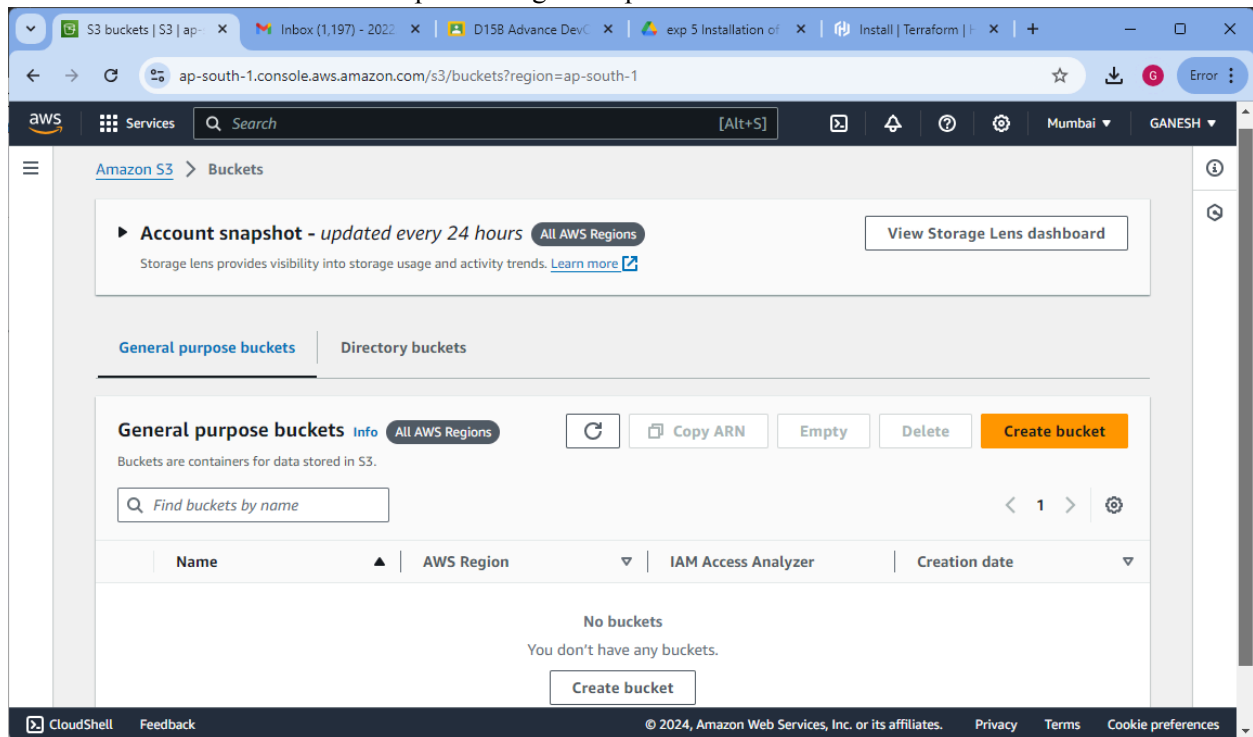
Terraform is a powerful Infrastructure as Code (IaC) tool that allows users to define, build, change, and manage cloud infrastructure across various providers like AWS, Google Cloud, Microsoft Azure, and DigitalOcean. By using Terraform, infrastructure is treated as code, enabling automation, consistency, and version control in managing resources such as S3 buckets, EC2 instances, and other cloud components.

Creating an S3 Bucket with Terraform

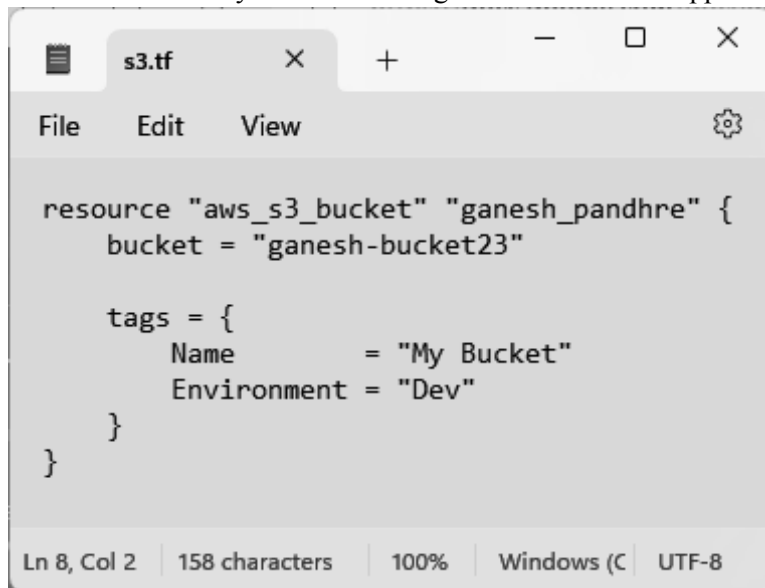
When using Terraform to create an S3 bucket on AWS, the process involves defining the desired state of the infrastructure through configuration files written in HashiCorp Configuration Language (HCL). These files specify the cloud provider, resources, and other configurations required to set up the infrastructure.

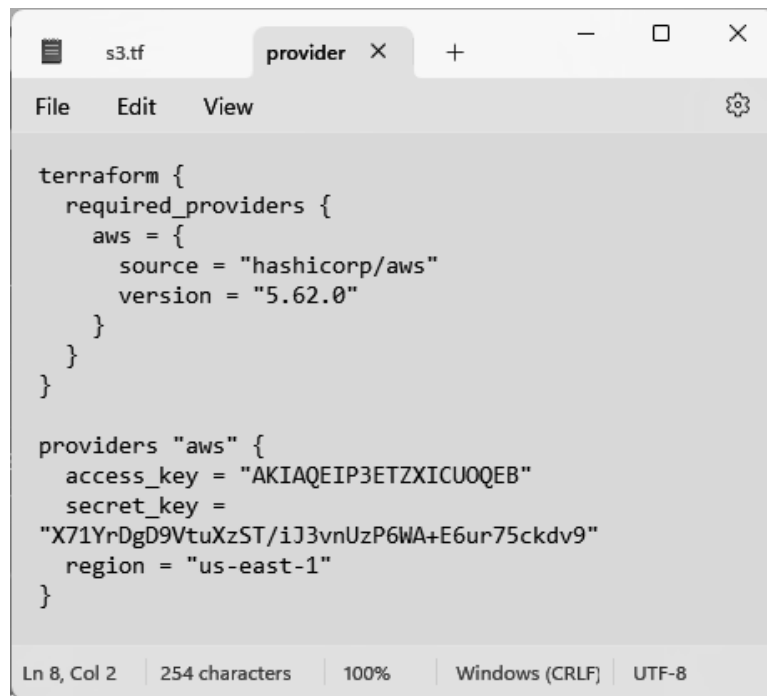
1. **Provider Configuration:** Terraform uses providers to interact with different cloud platforms. In this case, the AWS provider is configured with the necessary credentials, such as the Access Key ID and Secret Access Key, to authenticate and authorize Terraform's actions on the AWS cloud.
2. **Resource Definition:** The core of Terraform's functionality lies in its ability to define and manage resources. For creating an S3 bucket, a resource block is used to specify the properties of the bucket, such as its name, region, and access control settings. Terraform then ensures that the specified bucket is created with these properties.
3. **Infrastructure as Code (IaC):** By writing the configuration in code, Terraform enables the infrastructure to be versioned, shared, and reused across different environments. This approach not only improves collaboration among teams but also ensures that the infrastructure can be easily replicated or modified as needed.
4. **Lifecycle Management:** Terraform's lifecycle commands—`init`, `plan`, `apply`, and `destroy`—allow users to manage the entire lifecycle of their infrastructure. These commands initialize the environment, preview changes, apply the configuration, and eventually destroy the infrastructure when it is no longer needed. This level of control ensures that resources are managed efficiently, avoiding unnecessary costs and maintaining an organized cloud environment.
5. **State Management:** Terraform maintains a state file that tracks the current state of the managed infrastructure. This state file is crucial for determining what changes need to be applied when updating the infrastructure. It ensures that the live infrastructure remains in sync with the configuration files, allowing Terraform to make precise and minimal changes.

AWS S3 bucket dashboard before performing the experiment.



Write a Terraform Script for creating S3 Bucket on Amazon AWS and provider.tf file. Save both the files in the same directory Terraform along with the terraform application.





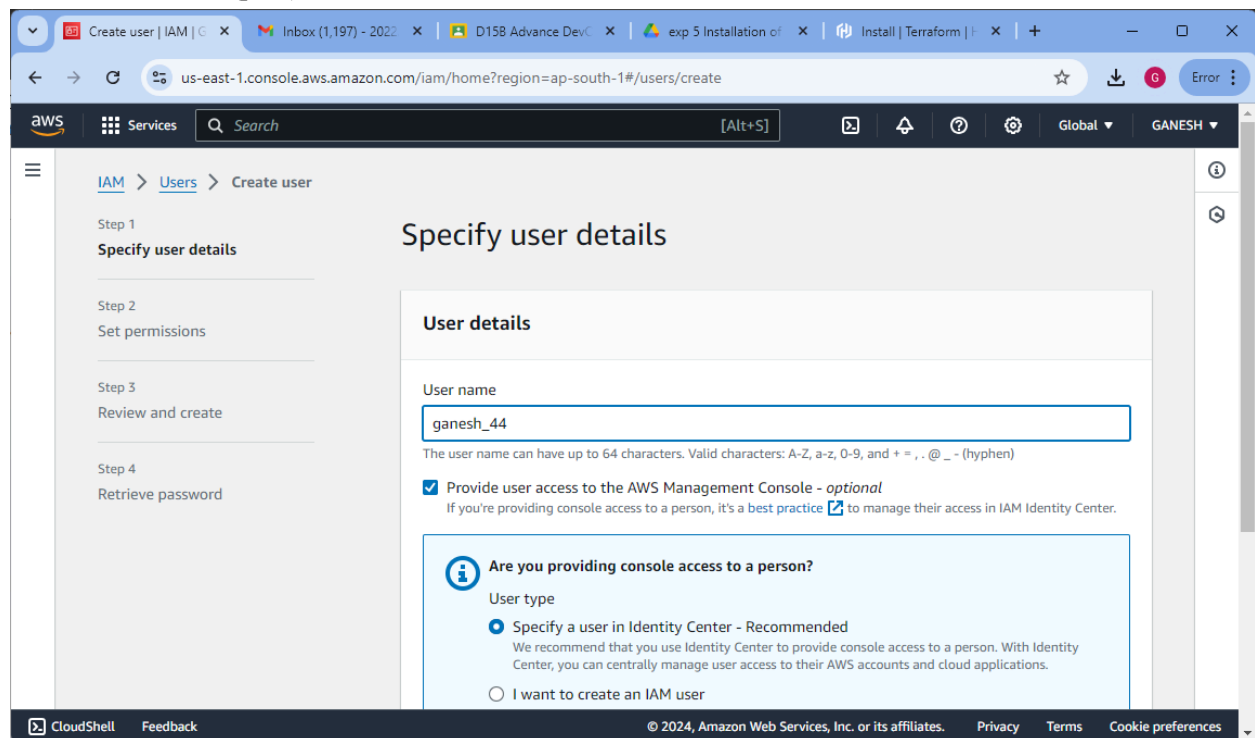
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.62.0"
    }
  }
}

providers "aws" {
  access_key = "AKIAQEIP3ETZXICUOQEB"
  secret_key = "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"
  region = "us-east-1"
}
```

Ln 8, Col 2 | 254 characters | 100% | Windows (CRLF) | UTF-8

(access key and secret key will be generated further)

Go to aws account (paid) & create IAM user.



The screenshot shows the AWS IAM console 'Create user' page. The breadcrumb navigation is 'IAM > Users > Create user'. A sidebar on the left lists four steps: 'Specify user details' (Step 1), 'Set permissions' (Step 2), 'Review and create' (Step 3), and 'Retrieve password' (Step 4). The main content area is titled 'Specify user details' and contains a 'User details' section. In this section, the 'User name' field is filled with 'ganesh_44'. Below this, there is a checkbox labeled 'Provide user access to the AWS Management Console - optional' which is checked. A note states: 'If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.' Below this is a section titled 'Are you providing console access to a person?' with two radio button options: 'Specify a user in Identity Center - Recommended' (which is selected) and 'I want to create an IAM user'. The footer of the console shows 'CloudShell', 'Feedback', and copyright information for Amazon Web Services, Inc. or its affiliates.

Set the permissions as follows at step 2.

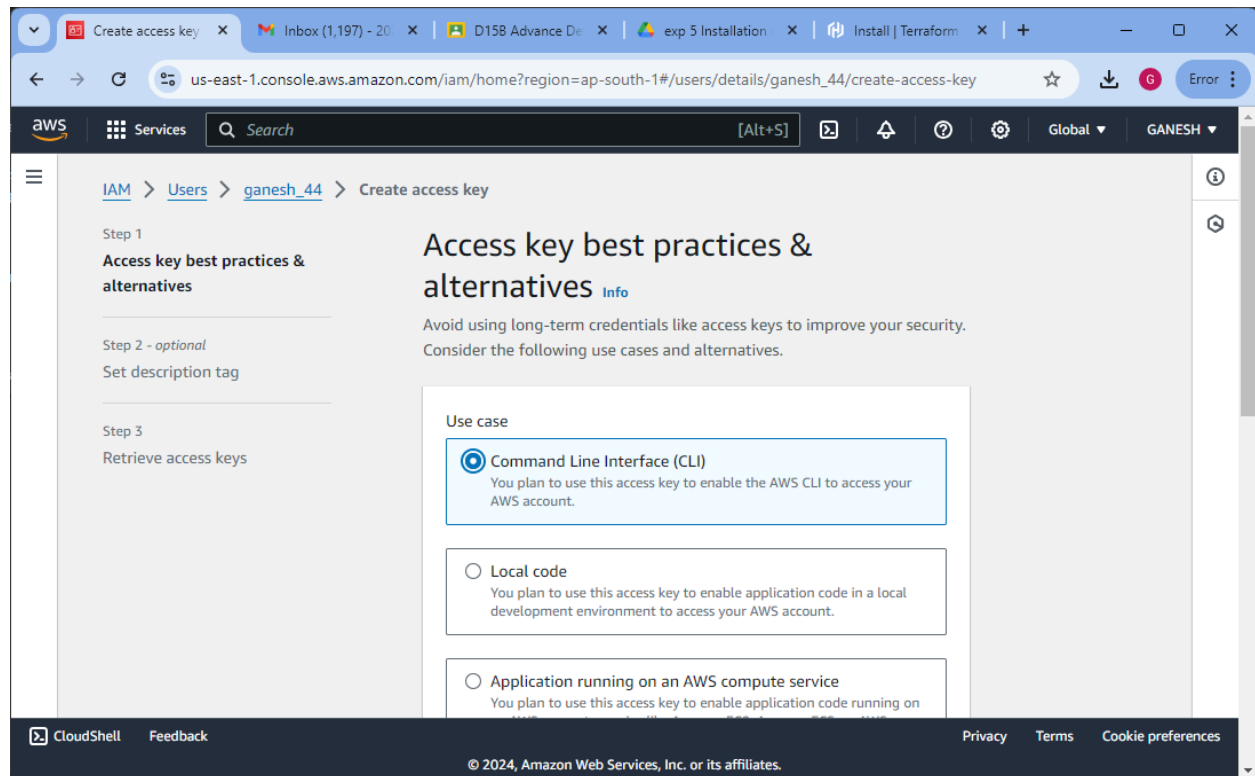
The screenshot shows the AWS IAM console 'Create user' wizard, Step 2: Set permissions. The 'Attach policies directly' option is selected. The 'Permissions policies' table shows the following policies attached to the user:

Policy name	Type	Attached entities
AccessAnalyzerSer...	AWS managed	0
AdministratorAccess	AWS managed - job func...	0

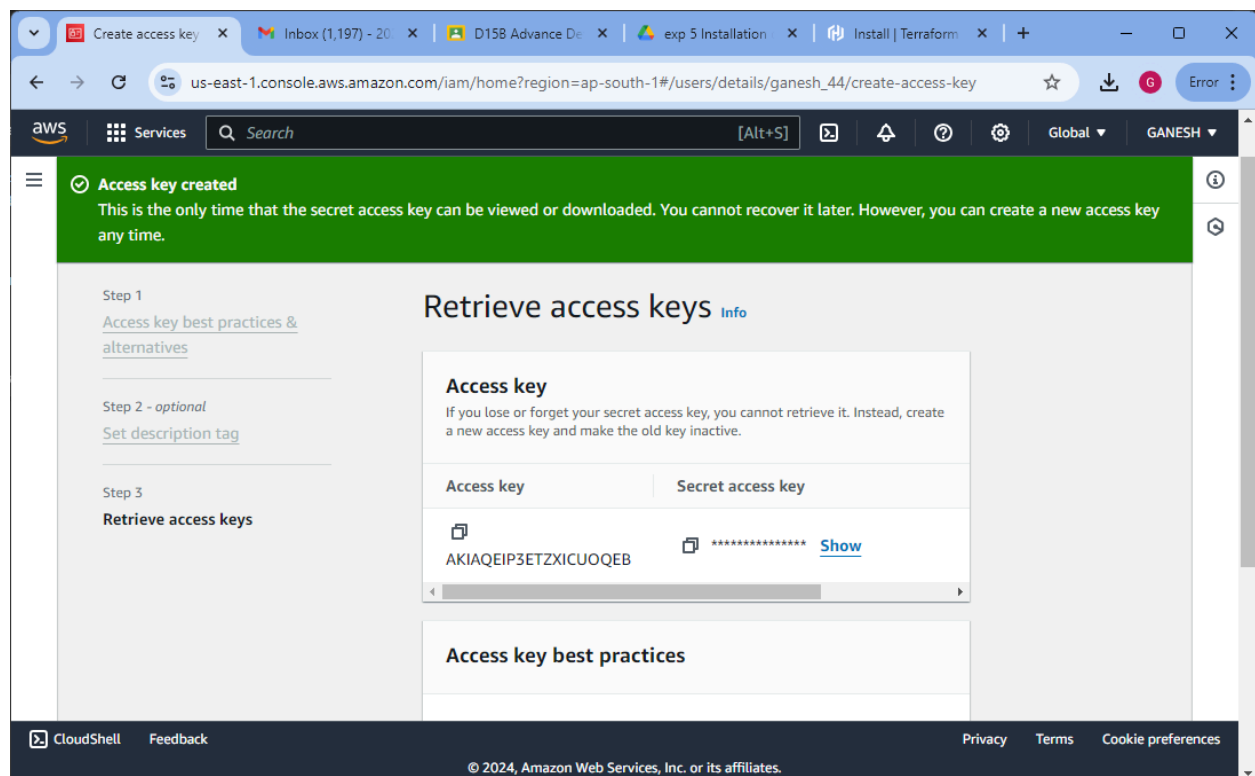
The bottom screenshot shows the 'Users' page in the IAM console. A green banner indicates 'User created successfully'. The 'Users' table shows the following user:

User name	Path	Group	Last activity	MFA	Password age
ganesh_44	/	0	-	-	-

We need to create an access key for that IAM user. Thus, select **create access key** and choose command line interface (CLI).



As the access key is created, copy the access key and secret key so that it can be pasted in the provider.tf file.



Go to the Terraform directory where both the files are saved in the command prompt and execute Terraform Init command to initialize the resources.

```
Command Prompt

C:\Users\Admin>cd ..

C:\Users>cd ..

C:\>cd Terraform

C:\Terraform>dir
Volume in drive C has no label.
Volume Serial Number is 96B8-ADDB

Directory of C:\Terraform

12-08-2024  04.19 PM    <DIR>          .
12-08-2024  04.16 PM             4,922 LICENSE.txt
12-08-2024  04.29 PM             267 provider.tf
12-08-2024  04.11 PM             165 s3.tf
12-08-2024  04.16 PM       91,051,696 terraform.exe
               4 File(s)      91,057,050 bytes
               1 Dir(s)  46,528,745,472 bytes free

C:\Terraform>
```

```
Command Prompt

C:\Terraform>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.62.0"...
- Installing hashicorp/aws v5.62.0...
- Installed hashicorp/aws v5.62.0 (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.

C:\Terraform>
```

Execute Terraform plan to see the available resources

```
Command Prompt
C:\Terraform>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.ganesh_pandhre will be created
+ resource "aws_s3_bucket" "ganesh_pandhre" {
  + acceleration_status = (known after apply)
  + acl                  = (known after apply)
  + arn                  = (known after apply)
  + bucket               = "ganesh-bucket23"
  + 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"         = "My Bucket"
  }
  + tags_all              = {
    + "Environment" = "Dev"
    + "Name"         = "My Bucket"
  }
  + website_domain         = (known after apply)
  + website_endpoint       = (known after apply)
}

+ tags_all              = {
  + "Environment" = "Dev"
  + "Name"         = "My 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)

+ 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.

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>
```


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

```
Command Prompt
C:\Terraform>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.ganesh_pandhre will be created
+ resource "aws_s3_bucket" "ganesh_pandhre" {
  + acceleration_status      = (known after apply)
  + acl                      = (known after apply)
  + arn                      = (known after apply)
  + bucket                   = "ganesh-bucket23"
  + 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"        = "My Bucket"
  }
  + tags_all              = {
    + "Environment" = "Dev"
    + "Name"        = "My Bucket"
  }
  + website_domain        = (known after apply)
  + website_endpoint      = (known after apply)
  + cors_rule (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)
+ 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.

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

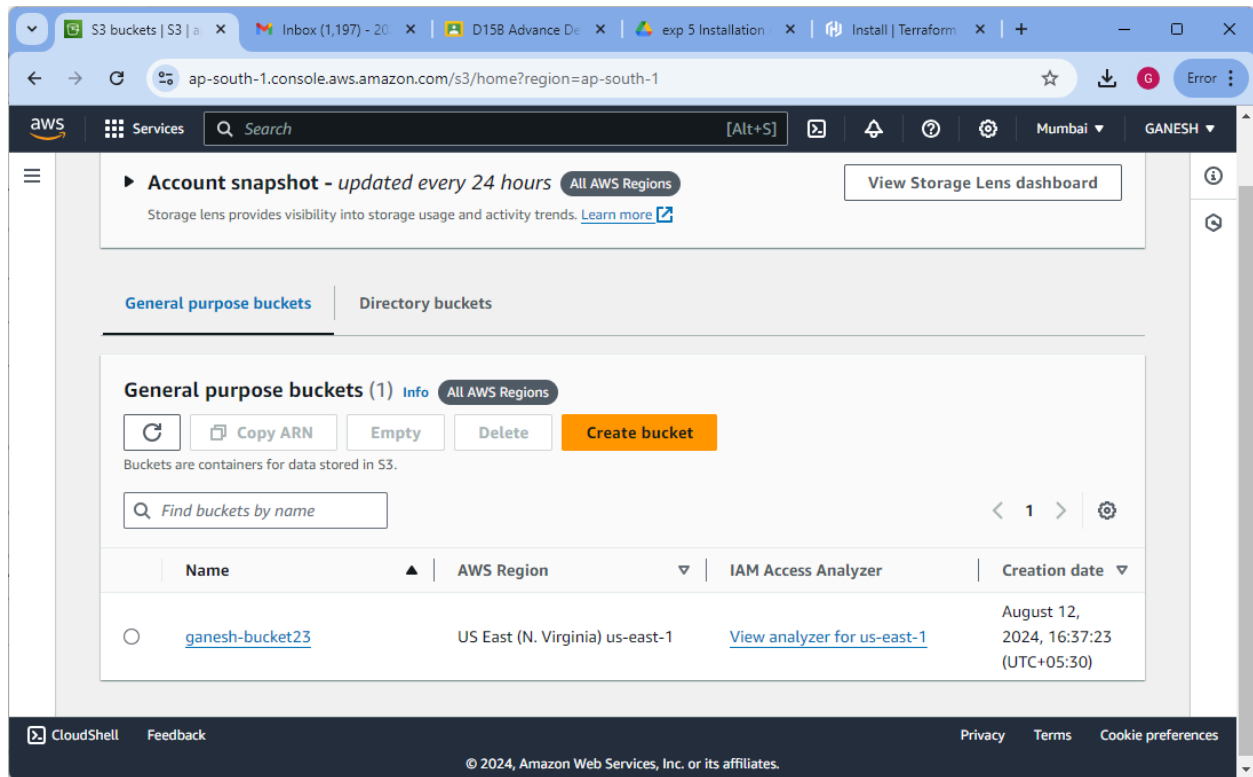
Enter a value: yes

aws_s3_bucket.ganesh_pandhre: Creating...
aws_s3_bucket.ganesh_pandhre: Creation complete after 5s [id=ganesh-bucket23]

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

C:\Terraform>
```

AWS S3 Bucket dashboard, after Executing Apply step.



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

```

C:\Terraform>terraform destroy
aws_s3_bucket.ganesh_pandhre: Refreshing state... [id=ganesh-bucket23]

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.ganesh_pandhre will be destroyed
- resource "aws_s3_bucket" "ganesh_pandhre" {
  - arn                = "arn:aws:s3:::ganesh-bucket23" -> null
  - bucket             = "ganesh-bucket23" -> null
  - bucket_domain_name = "ganesh-bucket23.s3.amazonaws.com" -> null
  - bucket_regional_domain_name = "ganesh-bucket23.s3.us-east-1.amazonaws.com" -> null
  - force_destroy      = false -> null
  - hosted_zone_id     = "Z3AQBSTGFYJSTF" -> null
  - id                 = "ganesh-bucket23" -> null
  - object_lock_enabled = false -> null
  - region             = "us-east-1" -> null
  - request_payer      = "BucketOwner" -> null
  - tags               = {
    - "Environment" = "Dev"
    - "Name"        = "My Bucket"
  } -> null
  - tags_all           = {
    - "Environment" = "Dev"
    - "Name"        = "My Bucket"
  } -> null
  # (3 unchanged attributes hidden)

- grant {
  - id           = "313ffa3bcb0d85bf057b1739ee521473758c15847255605908914e6de7a14f6" -> null
  - permissions = [
    - "FULL_CONTROL",
  ]
}

```

```
Command Prompt
- "FULL_CONTROL",
] -> null
- type = "CanonicalUser" -> null
# (1 unchanged attribute hidden)
}

- server_side_encryption_configuration {
- rule {
- bucket_key_enabled = false -> null

- apply_server_side_encryption_by_default {
- sse_algorithm = "AES256" -> null
# (1 unchanged attribute hidden)
}
}
}

- versioning {
- enabled = false -> null
- mfa_delete = false -> null
}
}

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

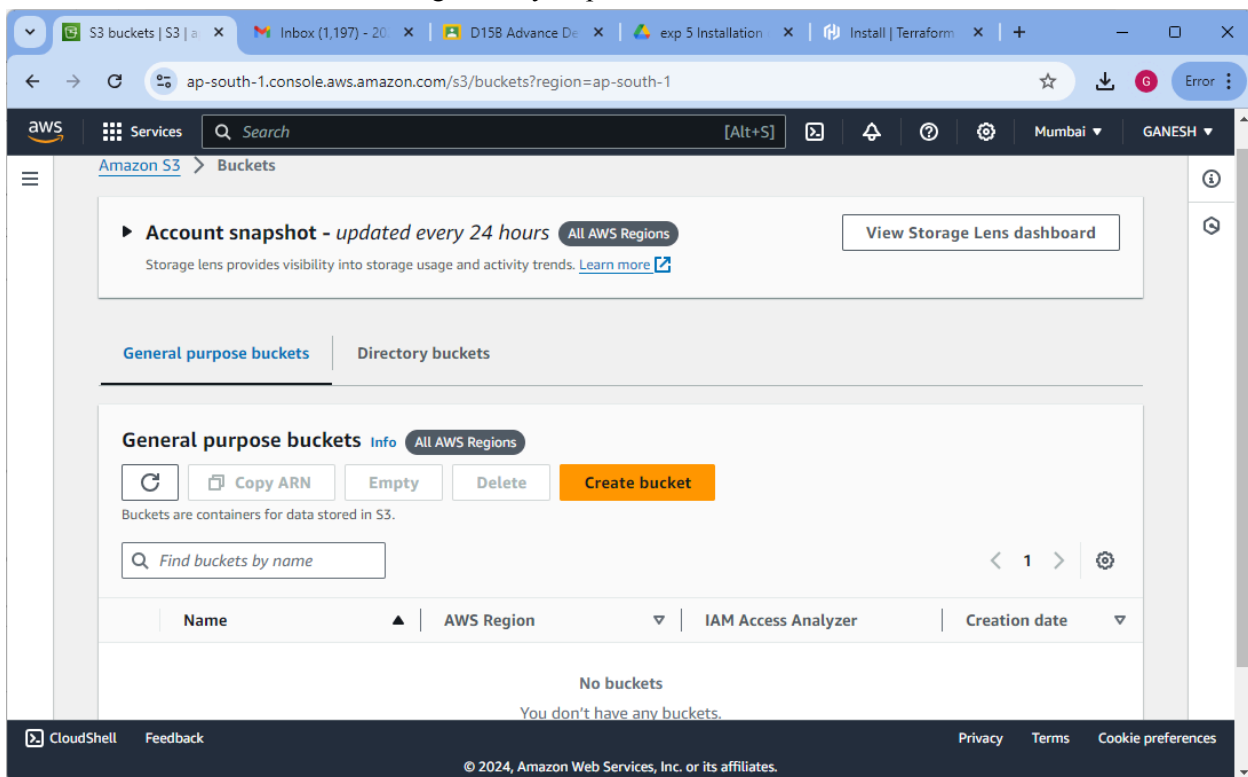
Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

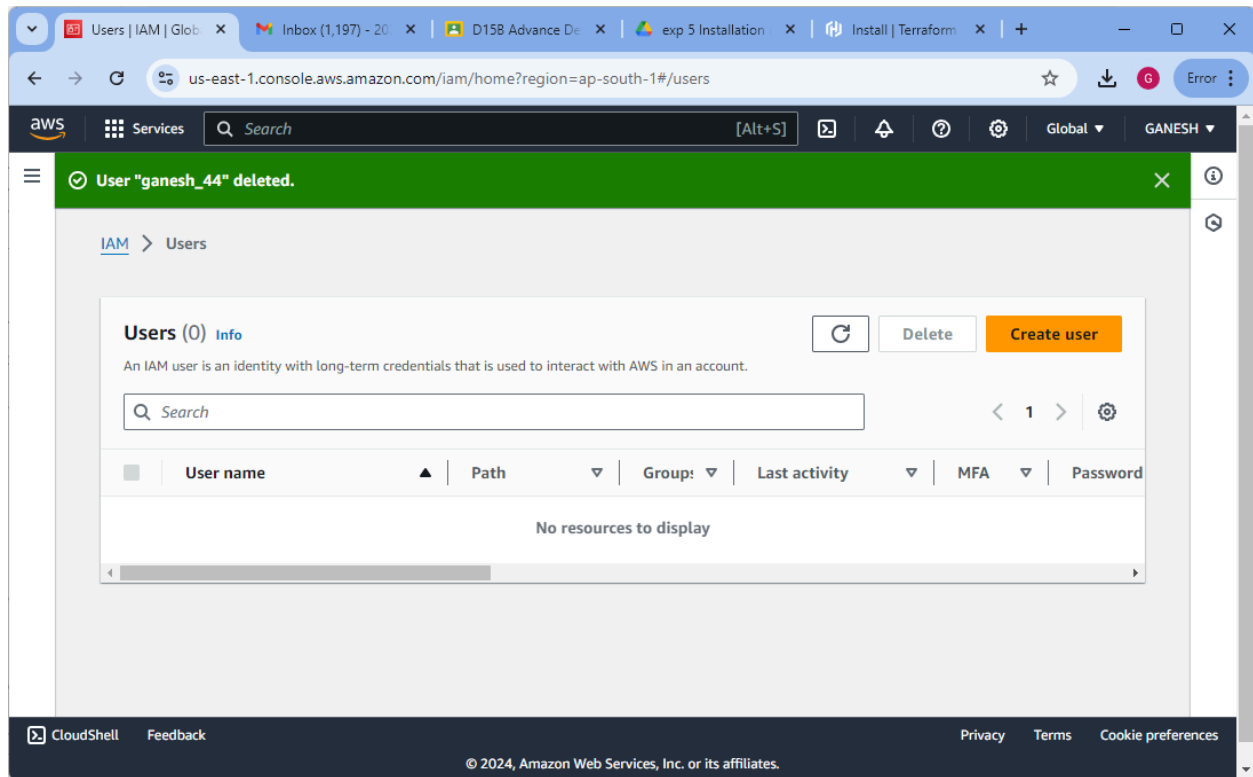
aws_s3_bucket.ganesh_pandhre: Destroying... [id=ganesh-bucket23]
aws_s3_bucket.ganesh_pandhre: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.
```

AWS EC2 dashboard, after Executing Destroy step.



Delete the IAM user which was created earlier.



CONCLUSION :

Terraform streamlines the process of managing cloud infrastructure by treating it as code, enabling automation and consistency across different cloud platforms. By using Terraform, you can efficiently create, modify, and destroy resources such as S3 buckets, ensuring a more organized and controlled approach to cloud management. Understanding these concepts is key to leveraging Terraform for advanced DevOps practices.