



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.	42
Name	Naikwadi Yash Shivdas
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:

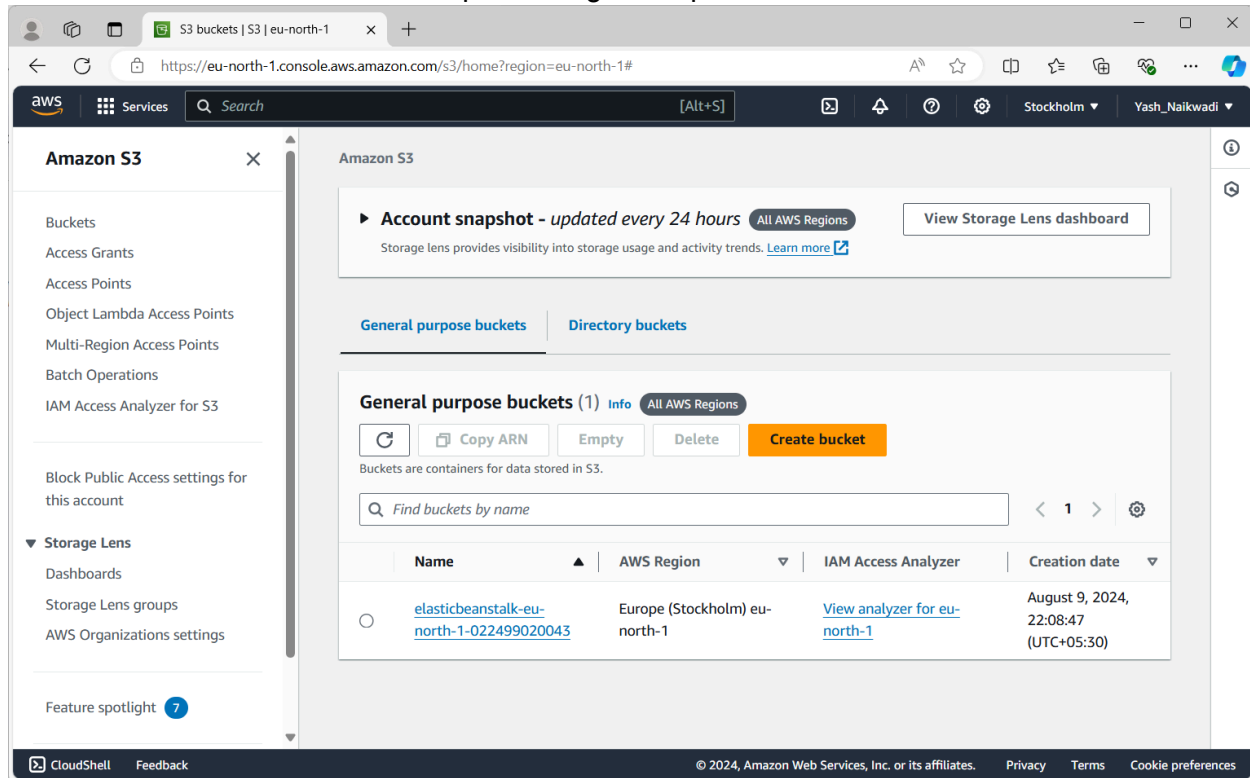
Terraform is a powerful Infrastructure as Code (IaC) tool that allows users to define, build, change, and manage cloud infrastructure across various providers like AWS, Google Cloud, Microsoft Azure, and DigitalOcean. By using Terraform, infrastructure is treated as code, enabling automation, consistency, 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.

```
s3.tf
provider.tf

File Edit View

resource "aws_s3_bucket" "yash" {
  bucket = "my-bj-terraform-test-bucket847398468342"

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

Ln 1, Col 1 | 145 characters | 100% | Windows (CRLF) | UTF-8

```
s3.tf
provider.tf

File Edit View

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

provider "aws" {
  access_key = "AKIAQKPILZUFRJPQ74ZS"
  secret_key = "JWArlyXIsKJGwJTGfI9b16RezV017njDNhCuZLmrH"
  region = "us-east-1"
}
```

Ln 13, Col 26 | 257 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' wizard at the 'Specify user details' step. The user name 'naikwadi42' is entered in the 'User name' field. Below the field, a note states: 'The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)'. There is an unchecked checkbox for 'Provide user access to the AWS Management Console - optional' with a note: 'If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.' A blue information box contains text about generating credentials for programmatic access. At the bottom right, there are 'Cancel' and 'Next' buttons.

Step 1
Specify user details

Step 2
Set permissions

Step 3
Review and create

Specify user details

User details

User name

naikwadi42

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - optional

If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

Info If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. [Learn more](#)

Cancel Next

Set the permissions as follows at step 2.

The screenshot shows the AWS IAM console 'Create user' wizard at the 'Set permissions' step. The 'Attach policies directly' option is selected under 'Permissions options'. Below, the 'Permissions policies (1/1228)' section shows a table of policies. The 'AdministratorAccess' policy is selected with a checkbox. The table has columns for 'Policy name', 'Type', and 'Attached entities'.

Step 1
[Specify user details](#)

Step 2
Set permissions

Step 3
Review and create

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

☐ Add user to group

Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

☐ Copy permissions

Copy all group memberships, attached managed policies, and inline policies from an existing user.

☒ Attach policies directly

Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1228)

Choose one or more policies to attach to your new user.

Filter by Type

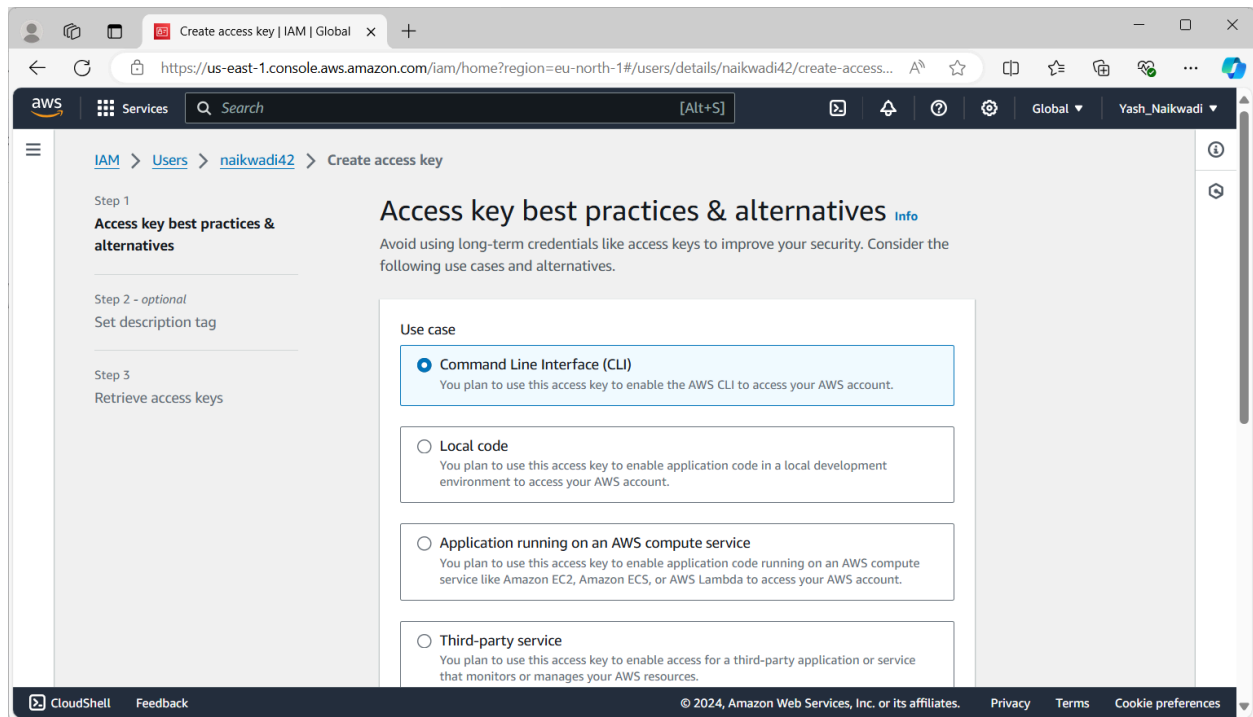
Search

All ty...

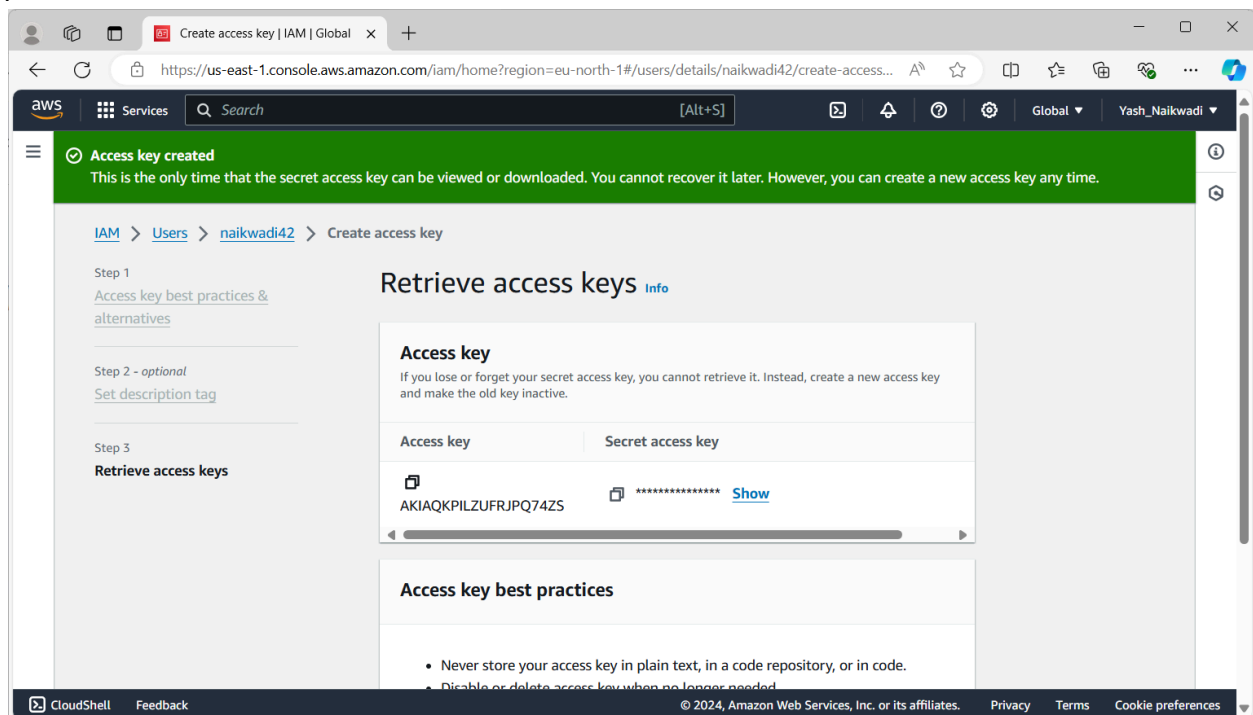
< 1 2 3 4 5 6 7 ... 62 >

	Policy name	Type	Attached entities
<input type="checkbox"/>	AccessAnalyzerSer...	AWS managed	0
<input checked="" type="checkbox"/>	AdministratorAccess	AWS managed - job function	0

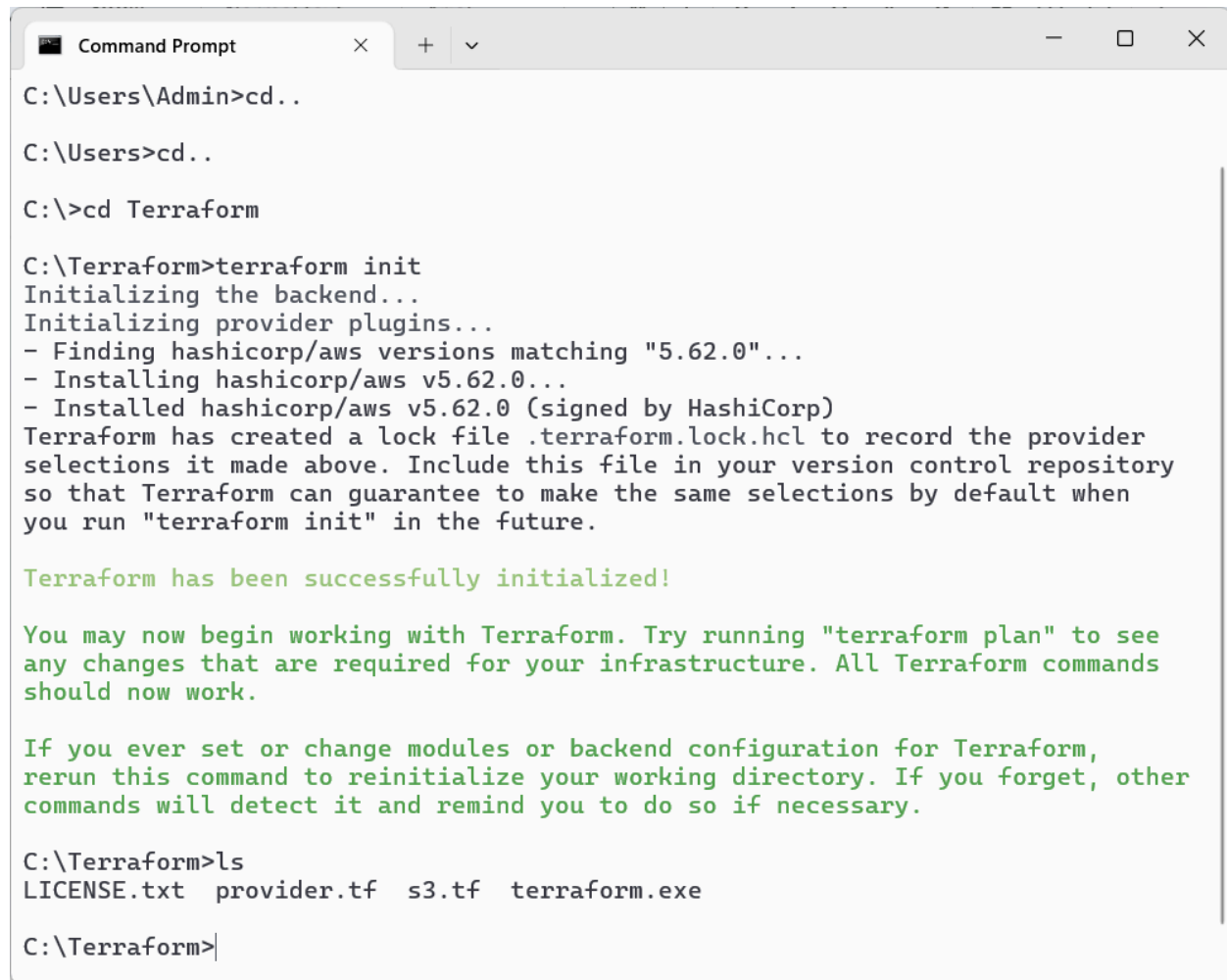
We need to create an access key for that IAM user. Thus, select **create access key** and choose command line interface (CLI).



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>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.

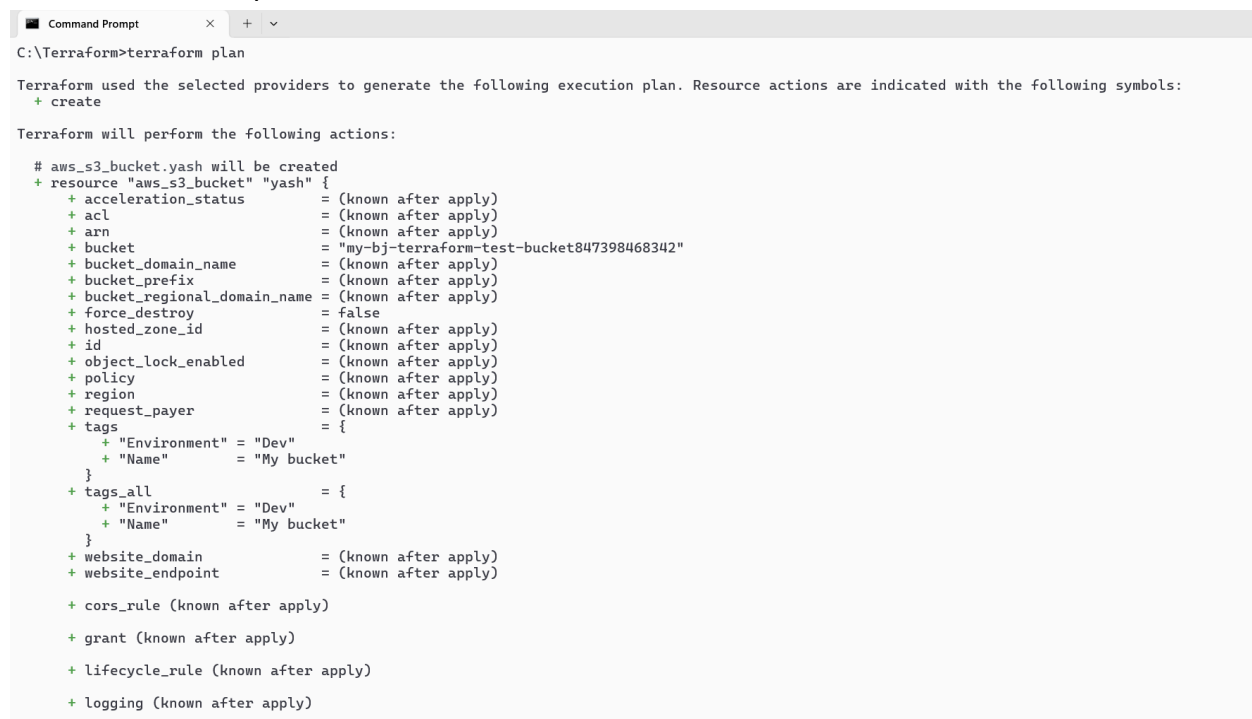
Terraform has been successfully initialized!

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>ls
LICENSE.txt  provider.tf  s3.tf  terraform.exe
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.yash will be created
+ resource "aws_s3_bucket" "yash" {
+   acceleration_status      = (known after apply)
+   acl                      = (known after apply)
+   arn                     = (known after apply)
+   bucket                  = "my-bj-terraform-test-bucket847398468342"
+   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)

+ grant (known after apply)

+ lifecycle_rule (known after apply)

+ logging (known after apply)
```

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.yash will be created
+ resource "aws_s3_bucket" "yash" {
  + acceleration_status      = (known after apply)
  + acl                      = (known after apply)
  + arn                     = (known after apply)
  + bucket                  = "my-bj-terraform-test-bucket847398468342"
  + 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)

  + 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.

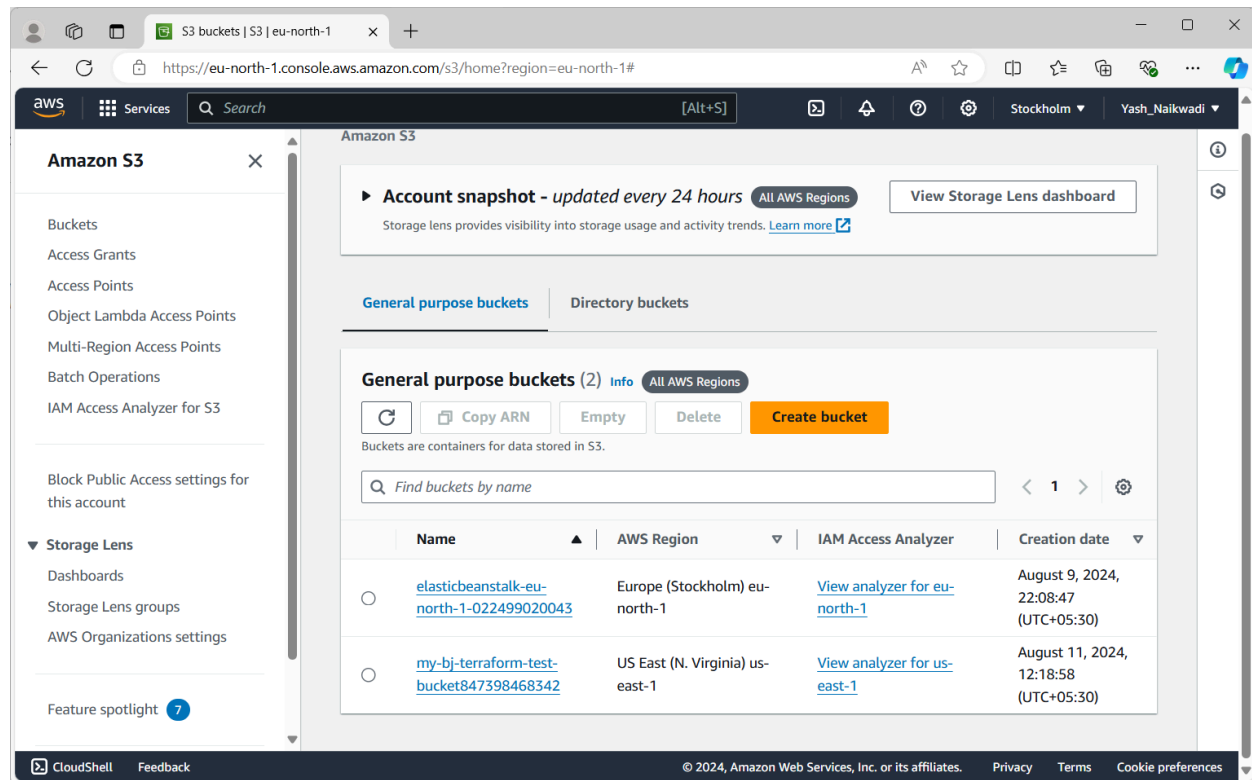
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

aws_s3_bucket.yash: Creating...
aws_s3_bucket.yash: Creation complete after 5s [id=my-bj-terraform-test-bucket847398468342]

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

AWS S3 Bucket dashboard, after Executing Apply step.



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

```

Command Prompt
C:\Terraform>terraform destroy
aws_s3_bucket.yash: Refreshing state... [id=my-bj-terraform-test-bucket847398468342]

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.yash will be destroyed
- resource "aws_s3_bucket" "yash" {
  - arn                = "arn:aws:s3::my-bj-terraform-test-bucket847398468342" -> null
  - bucket             = "my-bj-terraform-test-bucket847398468342" -> null
  - bucket_domain_name = "my-bj-terraform-test-bucket847398468342.s3.amazonaws.com" -> null
  - bucket_regional_domain_name = "my-bj-terraform-test-bucket847398468342.s3.us-east-1.amazonaws.com" -> null
  - force_destroy      = false -> null
  - hosted_zone_id     = "Z3AQBSTGFYJSTF" -> null
  - id                = "my-bj-terraform-test-bucket847398468342" -> 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           = "776c5741eb306f8e3cf502ec9d885235425235aee65742e9e17ac68778e0d589" -> null
    - permissions = [
      - "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.yash: Destroying... [id=my-bj-terraform-test-bucket847398468342]
aws_s3_bucket.yash: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.

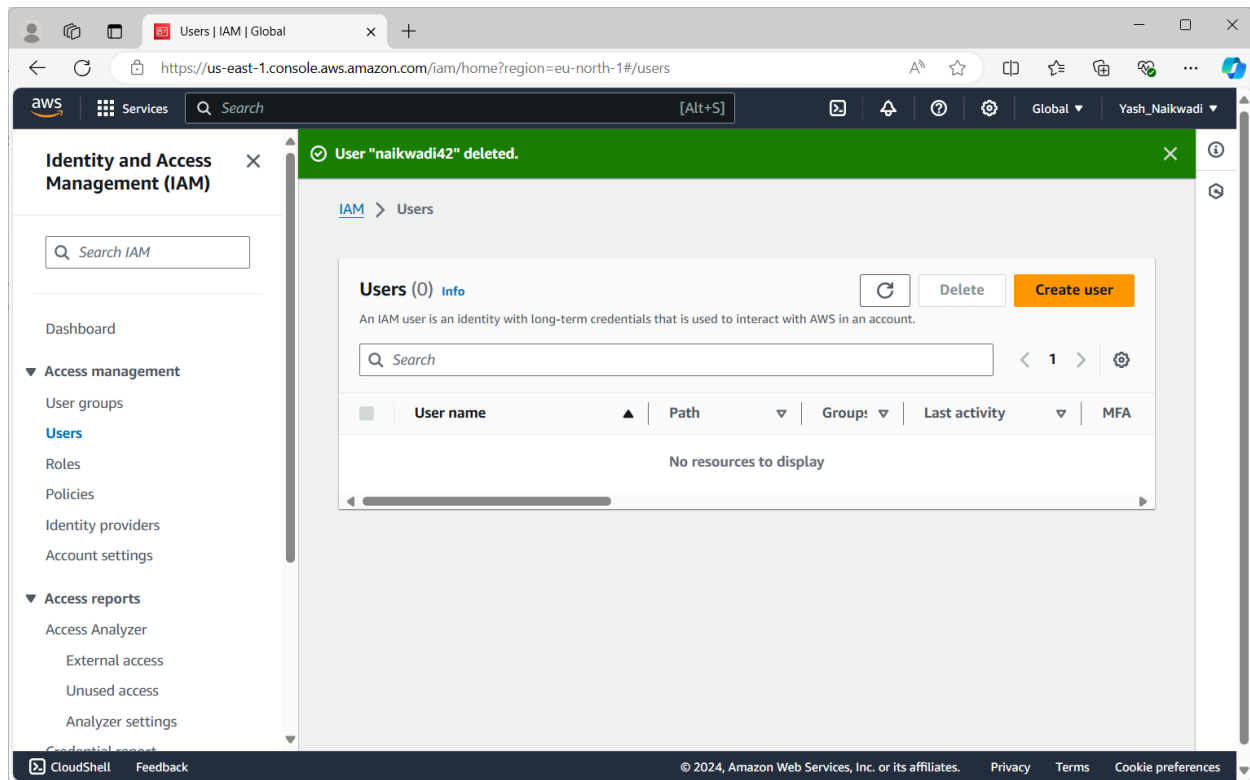
C:\Terraform>

AWS EC2 dashboard, after Executing Destroy step.

The screenshot displays the AWS S3 console for the eu-north-1 region. The left sidebar contains navigation links for Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Storage Lens, Dashboards, Storage Lens groups, AWS Organizations settings, and Feature spotlight. The main content area shows an 'Account snapshot - updated every 24 hours' with a 'View Storage Lens dashboard' button. Below this, there are tabs for 'General purpose buckets' and 'Directory buckets'. The 'General purpose buckets' section shows a 'Create bucket' button and a search bar. A table lists the existing buckets:

Name	AWS Region	IAM Access Analyzer	Creation date
elasticbeanstalk-eu-north-1-022499020043	Europe (Stockholm) eu-north-1	View analyzer for eu-north-1	August 9, 2024, 22:08:47 (UTC+05:30)

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.