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.	43
Name	Harsh Pramod Padyal
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.

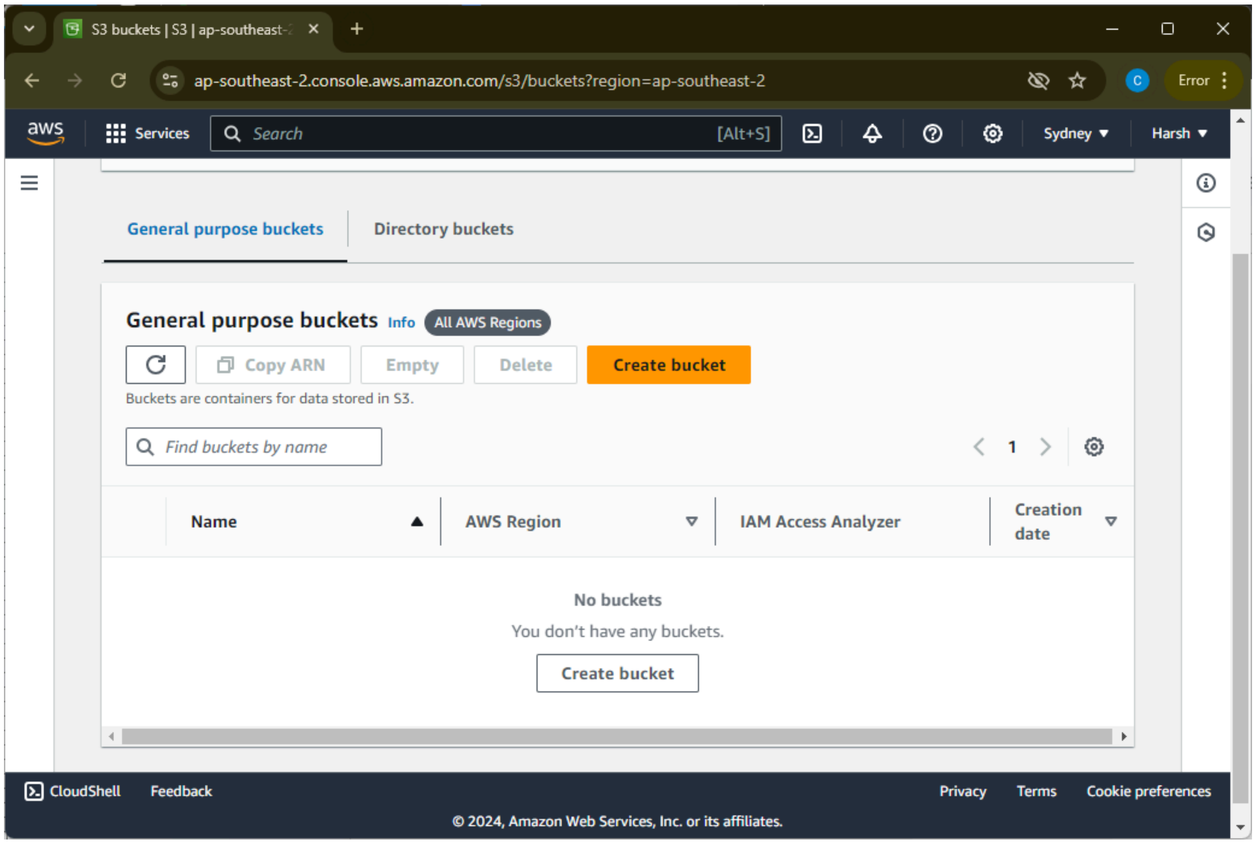
2. Resource Definition: The core of Terraform's functionality lies in its ability to define and manage resources. For creating an S3 bucket, a resource block is used to specify the properties of the bucket, such as its name, region, and access control settings. Terraform then ensures that the specified 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.

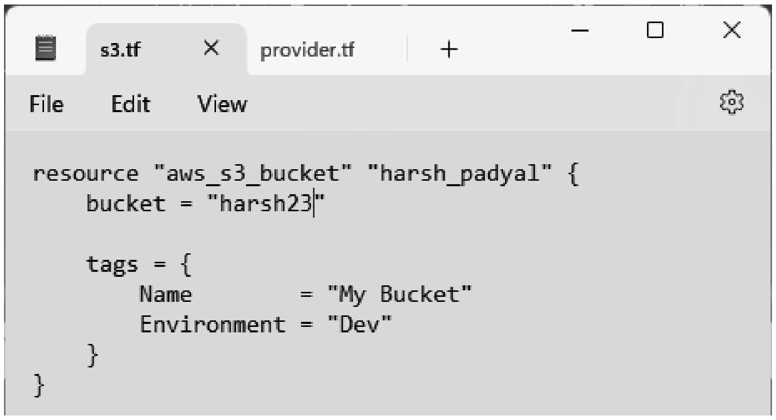
4. Lifecycle Management: Terraform's lifecycle commands—`init`, `plan`, `apply`, and `destroy`—allow users to manage the entire lifecycle of their infrastructure. These commands initialize the environment, preview changes, apply the configuration, and eventually destroy the infrastructure when it is 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.

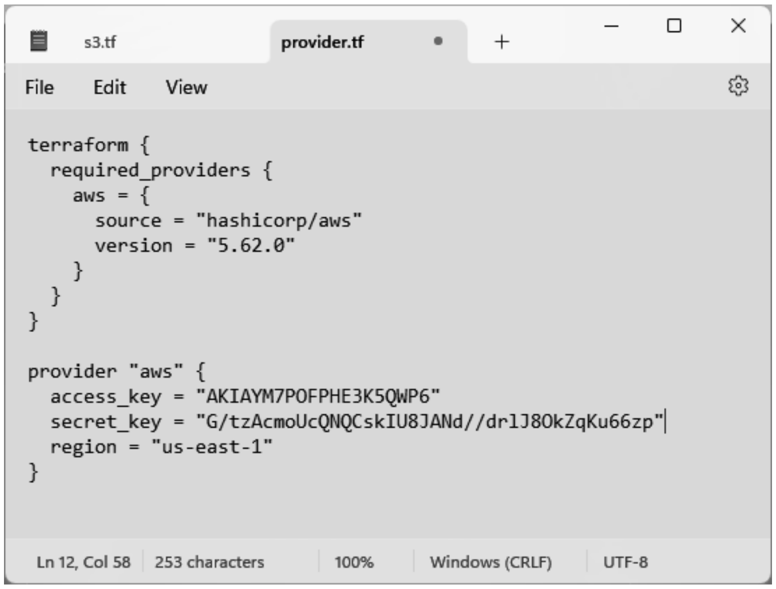
AWS S3 bucket dashboard before performing the experiment.


S3 buckets | S3 | ap-southeast-2
ap-southeast-2.console.aws.amazon.com/s3/buckets?region=ap-southeast-2
Services
Search [Alt+S]
Sydney
Harsh
General purpose buckets
Directory buckets
General purpose buckets Info All AWS Regions
Copy ARN
Empty
Delete
Create bucket
Buckets are containers for data stored in S3.
Find buckets by name
Name
AWS Region
IAM Access Analyzer
Creation date
No buckets
You don't have any buckets.
Create bucket
CloudShell
Feedback
Privacy
Terms
Cookie preferences
© 2024, Amazon Web Services, Inc. or its affiliates.

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.



```
resource "aws_s3_bucket" "harsh_padyal" {
    bucket = "harsh23"

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



s3.tf provider.tf

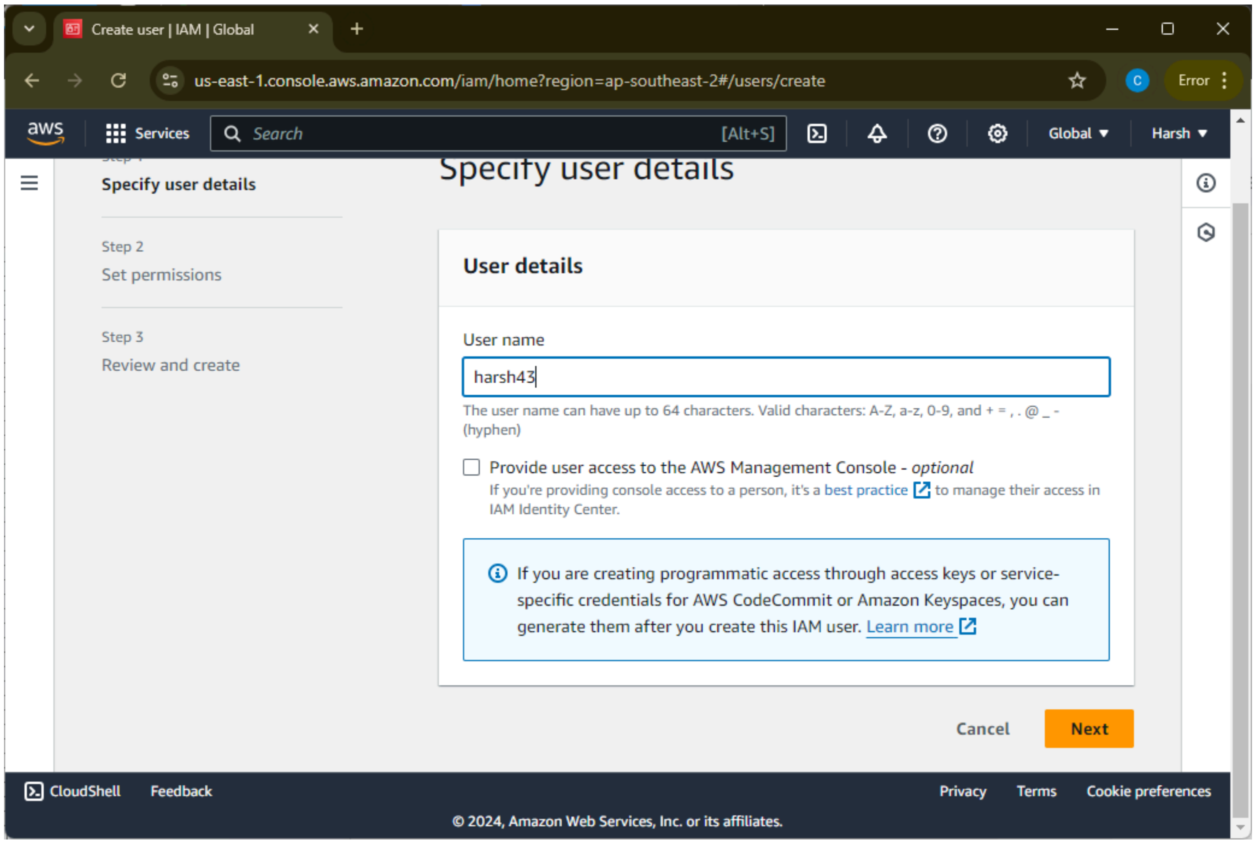
File Edit View

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

provider "aws" {
  access_key = "AKIAYM7POFPHE3K5QWP6"
  secret_key = "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"
  region = "us-east-1"
}
```

Ln 12, Col 58 | 253 characters | 100% | Windows (CRLF) | UTF-8

(access key and secret key will be generated further)

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



Create user | IAM | Global

us-east-1.console.aws.amazon.com/iam/home?region=ap-southeast-2#/users/create

Specify user details

Step 2
Set permissions

Step 3
Review and create

Specify user details

User details

User name

harsh43

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.

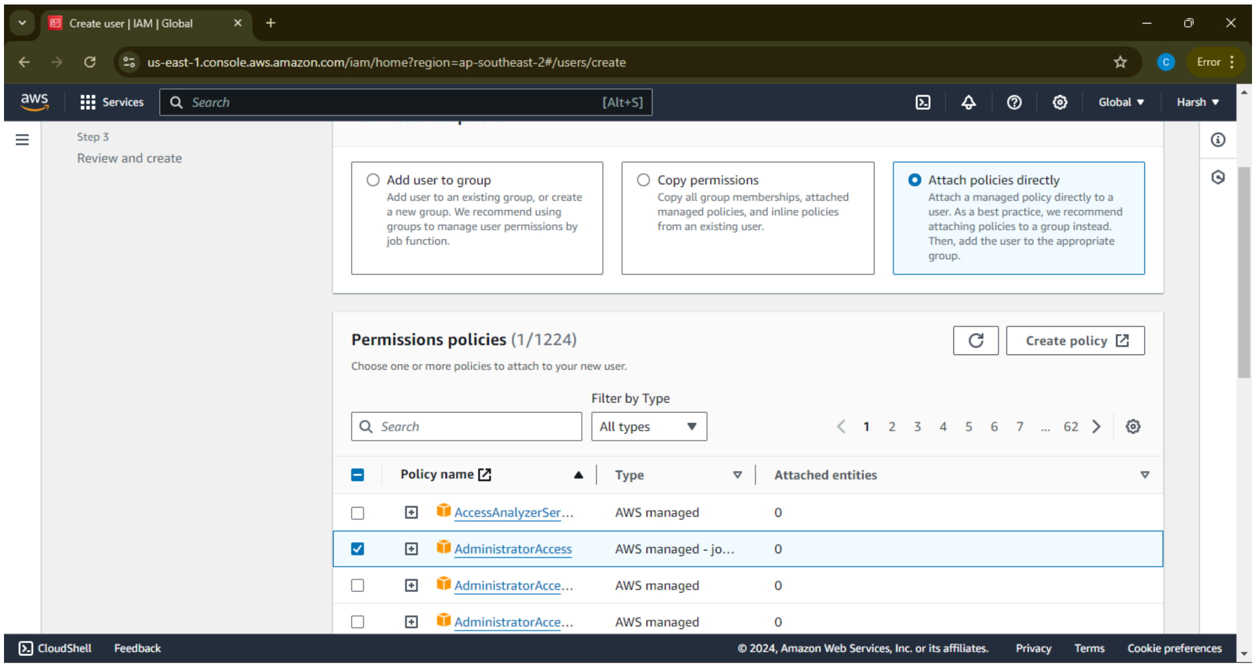
If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. Learn more

Cancel Next

CloudShell Feedback Privacy Terms Cookie preferences

© 2024, Amazon Web Services, Inc. or its affiliates.

Set the permissions as follows at step 2.


Create user | IAM | Global

us-east-1.console.aws.amazon.com/iam/home?region=ap-southeast-2#/users/create

Step 3
Review and create

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.

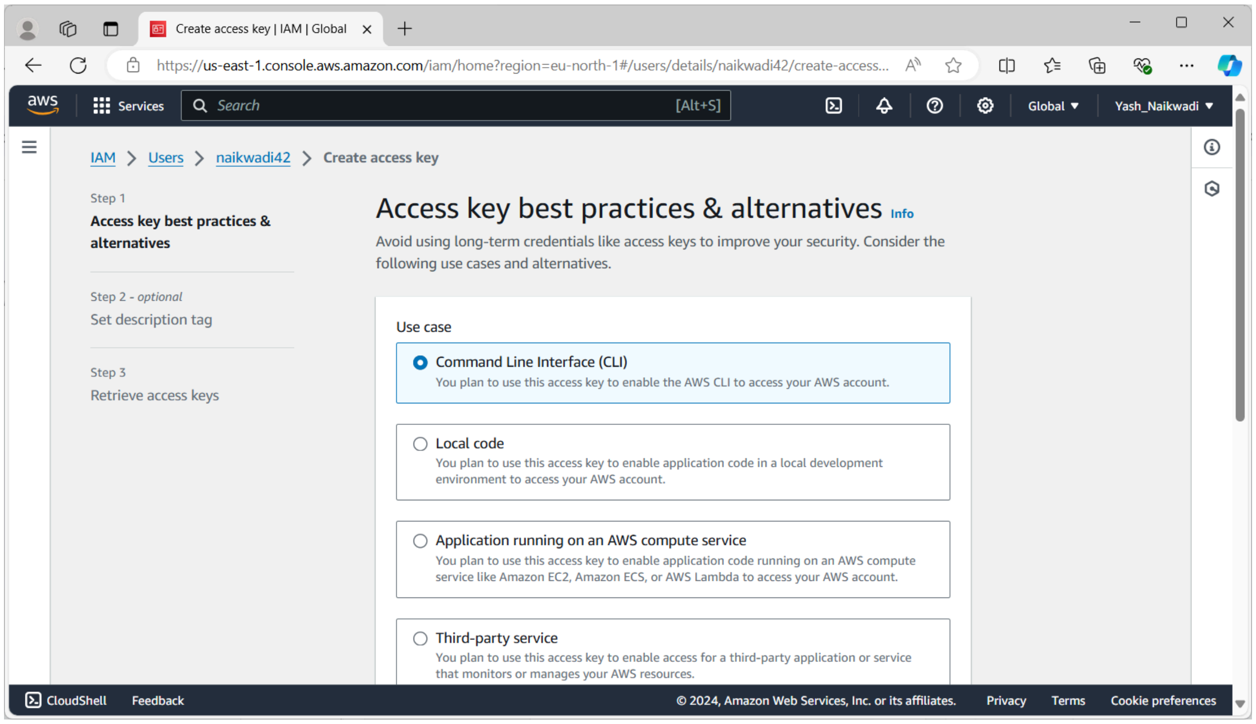
Permissions policies (1/1224)
Choose one or more policies to attach to your new user.

Filter by Type: All types

	Policy name	Type	Attached entities
☐	AccessAnalyzerSer…	AWS managed	0
✓	AdministratorAccess	AWS managed - jo…	0
☐	AdministratorAcce…	AWS managed	0
☐	AdministratorAcce…	AWS managed	0

© 2024, Amazon Web Services, Inc. or its affiliates.

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


Create access key | IAM | Global

https://us-east-1.console.aws.amazon.com/iam/home?region=eu-north-1#/users/details/naikwadi42/create-access...

IAM > Users > naikwadi42 > Create access key

Step 1
Access key best practices & alternatives

Step 2 - *optional*
Set description tag

Step 3
Retrieve access keys

Access key best practices & alternatives Info

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Use case

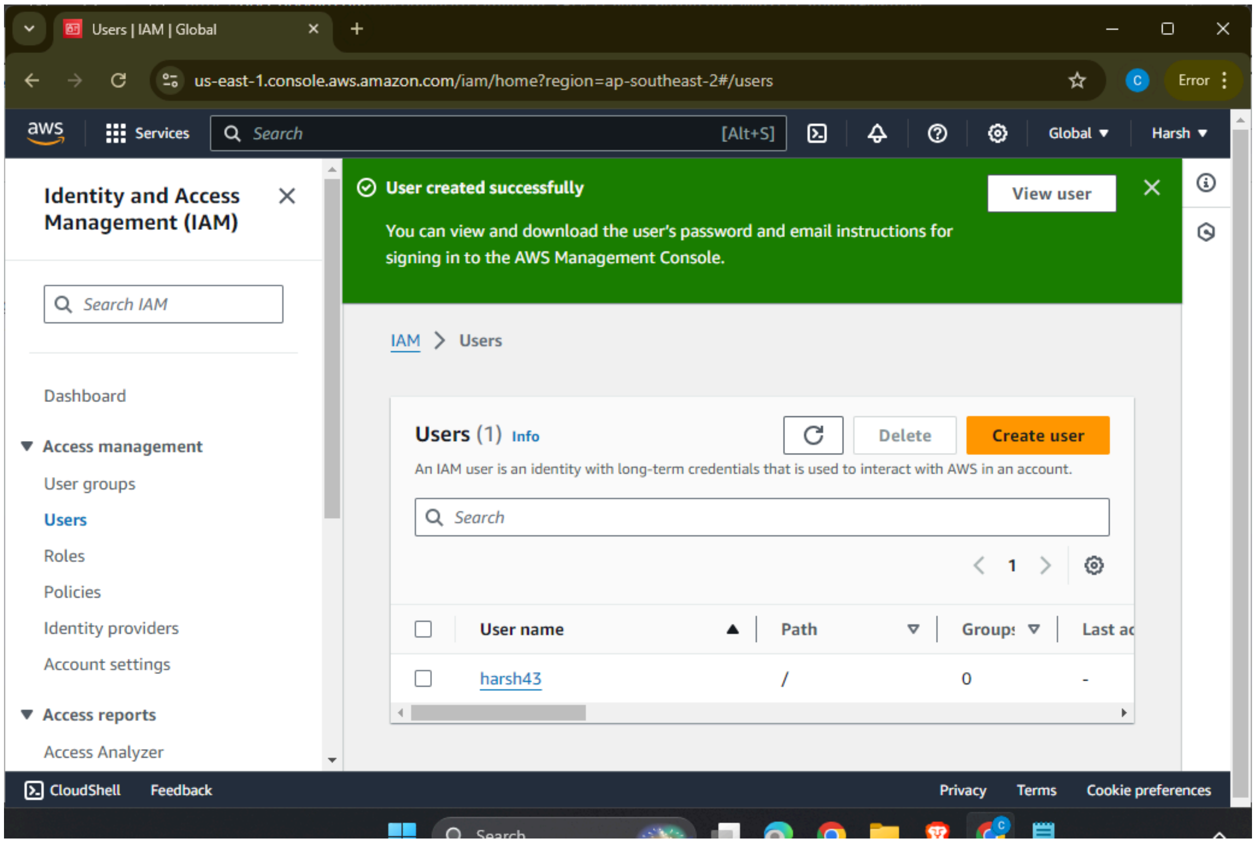
[✓ Command Line Interface (CLI)]
You plan to use this access key to enable the AWS CLI to access your AWS account.

Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.

Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

Third-party service
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.

© 2024, Amazon Web Services, Inc. or its affiliates.


User created successfully

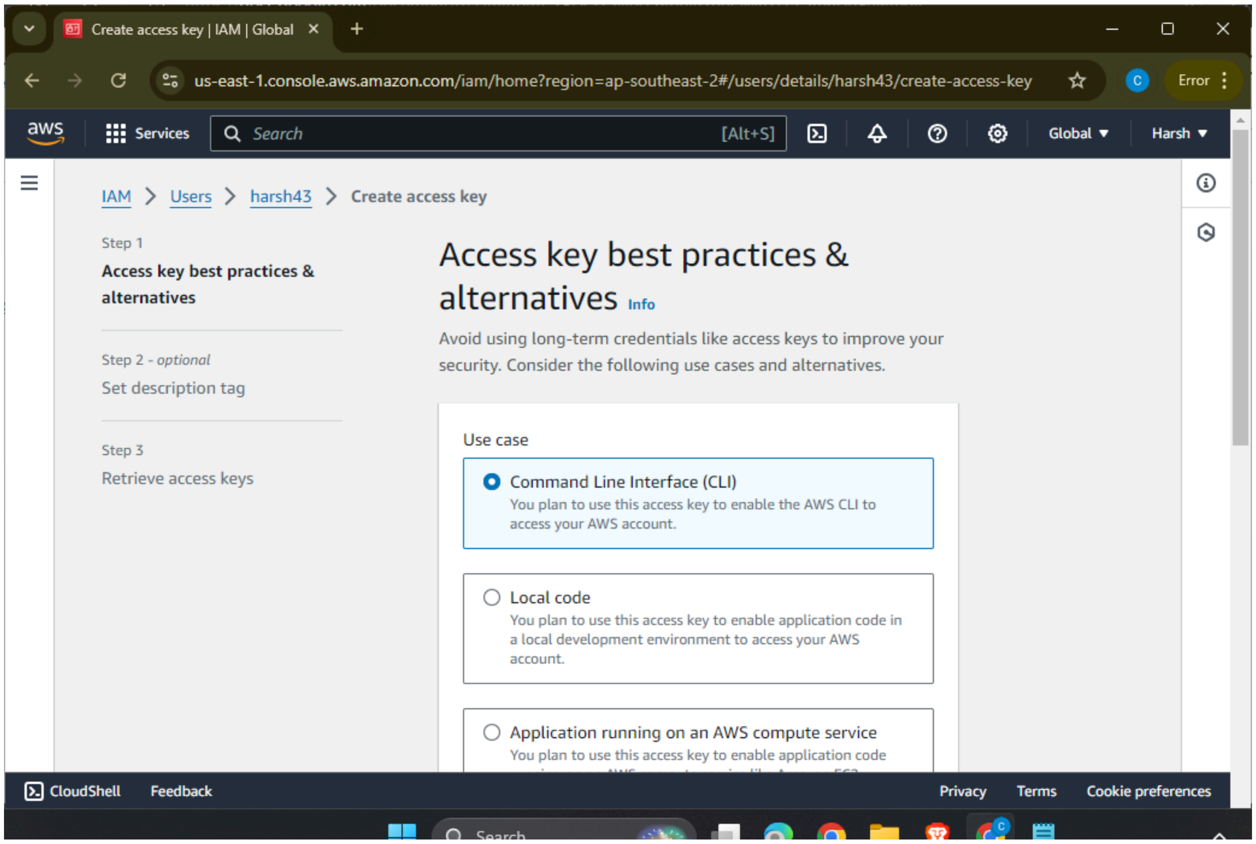
You can view and download the user's password and email instructions for signing in to the AWS Management Console.

Users (1)

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

User name	Path	Groups	Last ac
harsh43	/	0	-

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


IAM > Users > harsh43 > Create access key

Step 1 Access key best practices & alternatives

Step 2 - optional Set description tag

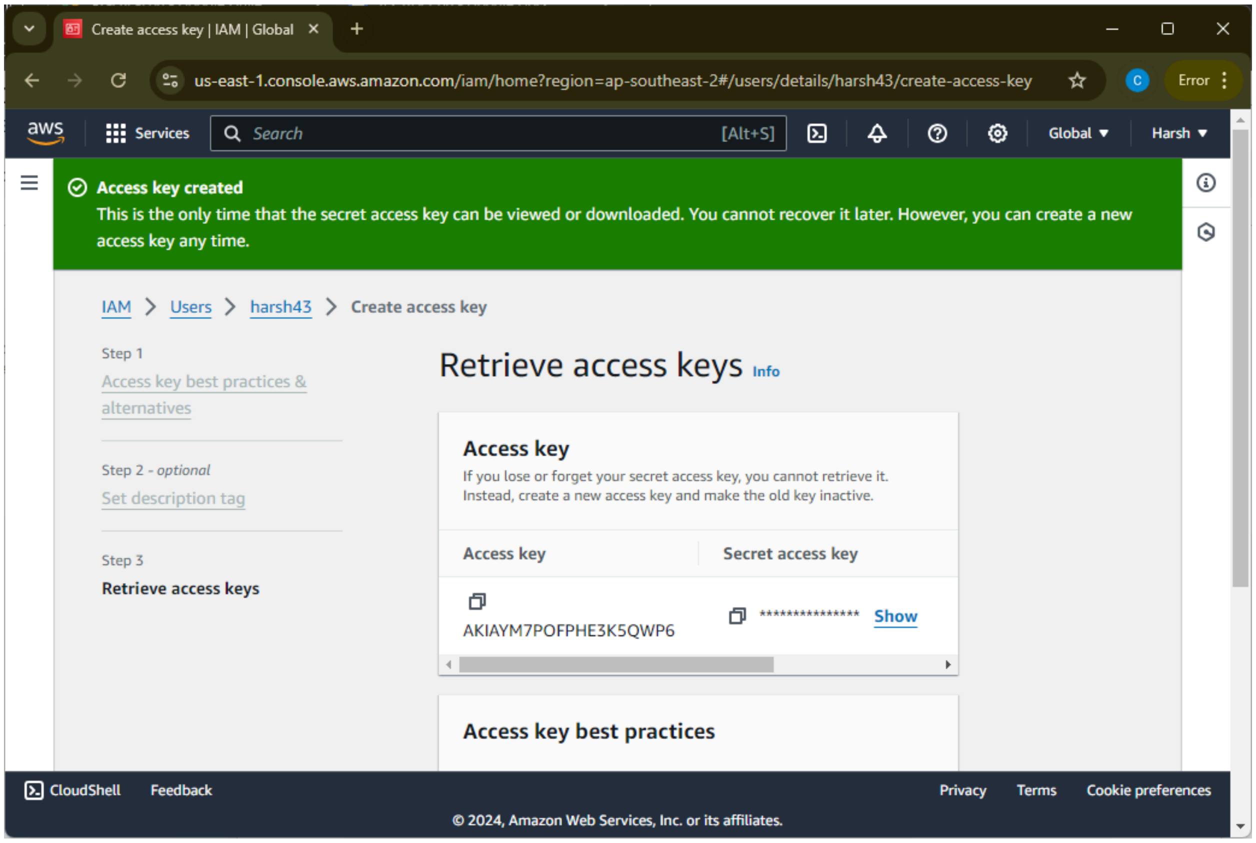
Step 3 Retrieve access keys

Access key best practices & alternatives

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Use case

- [✓ Command Line Interface (CLI)] You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code — You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service — You plan to use this access key to enable application code


Create access key | IAM | Global
us-east-1.console.aws.amazon.com/iam/home?region=ap-southeast-2#/users/details/harsh43/create-access-key
Services
Search
[Alt+S]
Global
Harsh

Access key created
This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

IAM > Users > harsh43 > Create access key

Step 1
Access key best practices & alternatives

Step 2 - *optional*
Set description tag

Step 3
Retrieve access keys

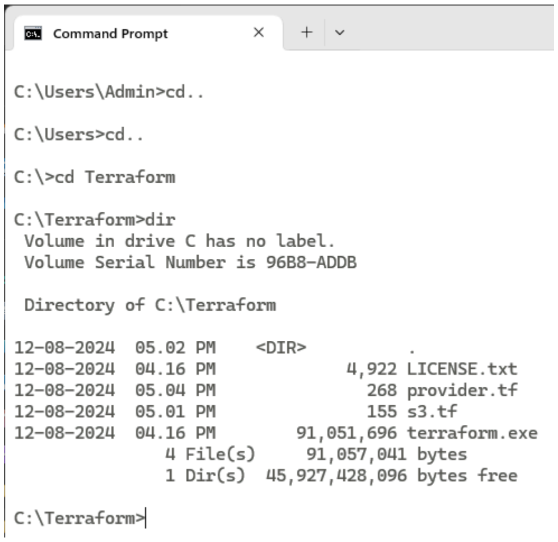
Retrieve access keys Info

Access key
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

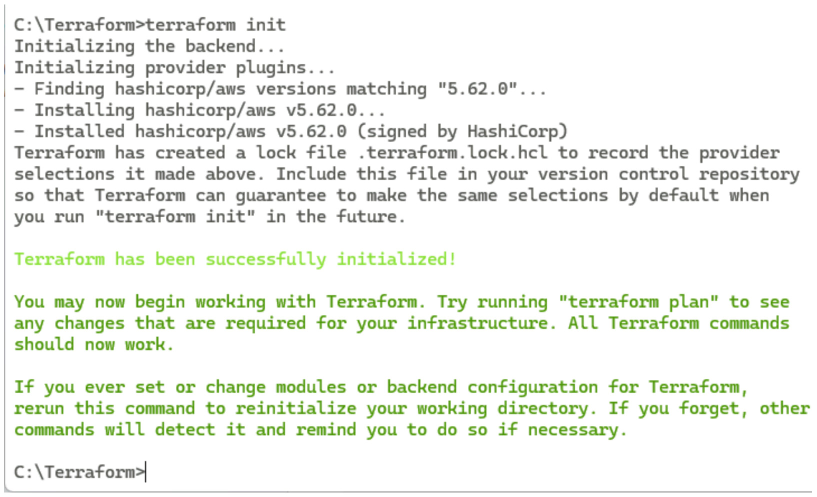
Access key	Secret access key
AKIAYM7POFPHE3K5QWP6	*************** Show

Access key best practices

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

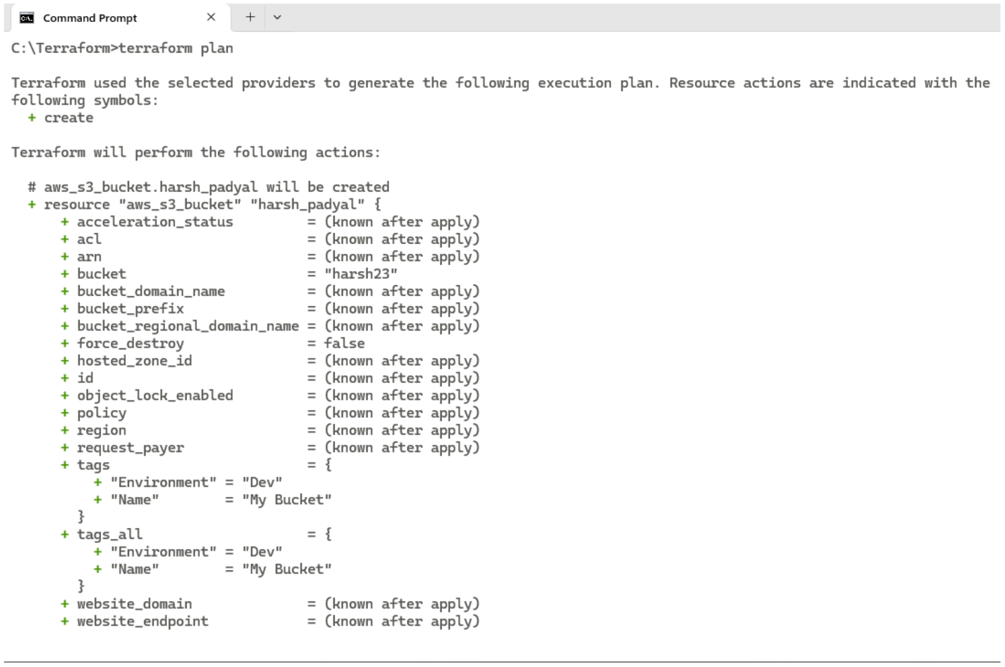


```
C:\Users\Admin>cd..

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  05.02 PM    <DIR>          .
12-08-2024  04.16 PM             4,922 LICENSE.txt
12-08-2024  05.04 PM               268 provider.tf
12-08-2024  05.01 PM               155 s3.tf
12-08-2024  04.16 PM        91,051,696 terraform.exe
               4 File(s)     91,057,041 bytes
               1 Dir(s)  45,927,428,096 bytes free

C:\Terraform>
```

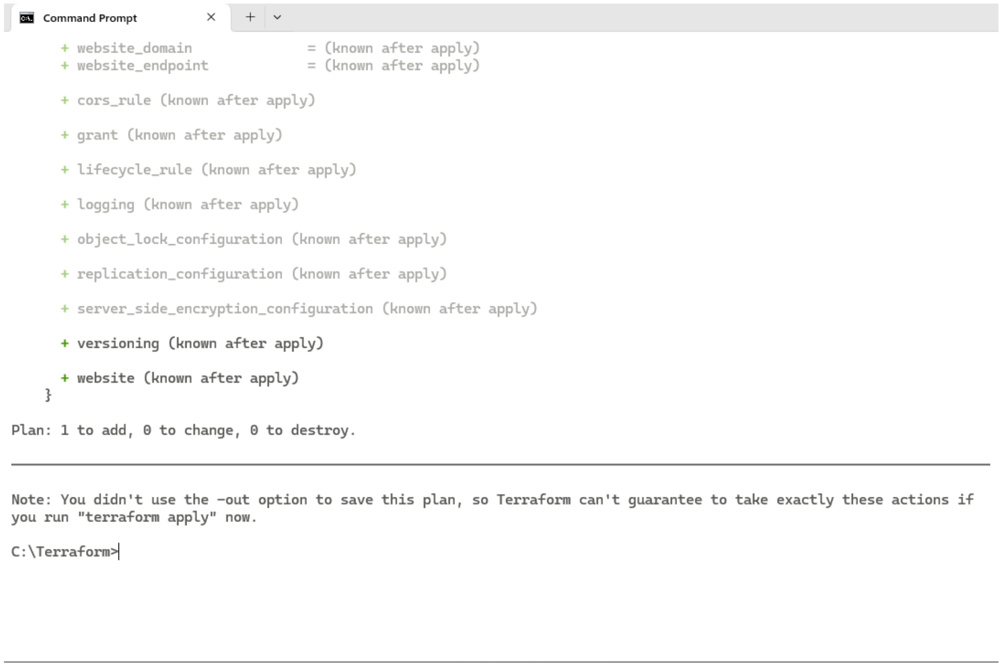


```
C:\Terraform>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.62.0"...
- Installing hashicorp/aws v5.62.0...
- Installed hashicorp/aws v5.62.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record 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>
```

Execute Terraform plan to see the available resources

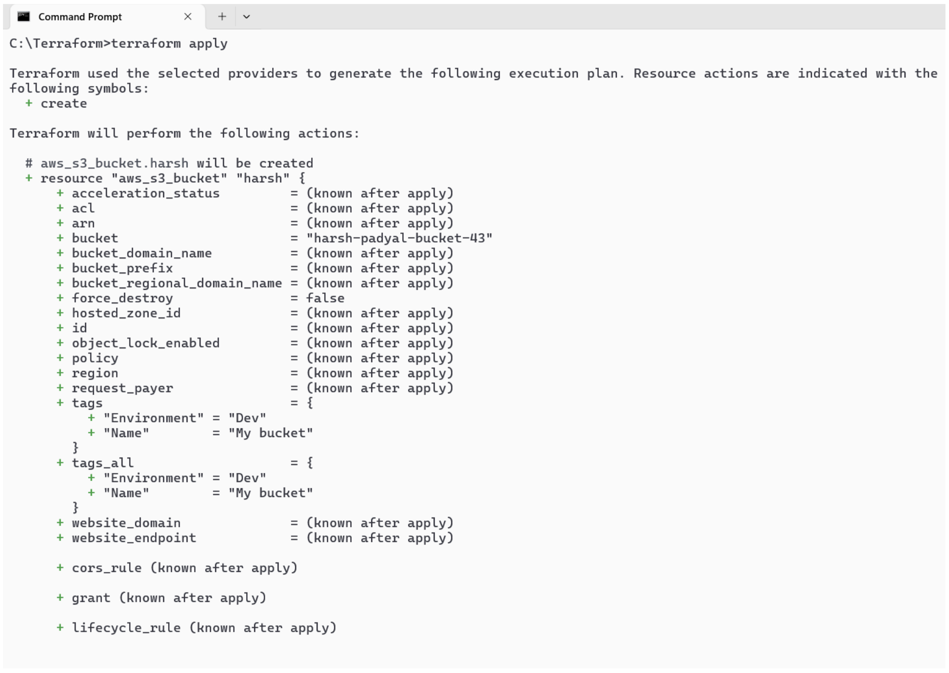


```
C:\Terraform>terraform plan

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.harsh_padyal will be created
  + resource "aws_s3_bucket" "harsh_padyal" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      + arn                         = (known after apply)
      + bucket                      = "harsh23"
      + 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)
```

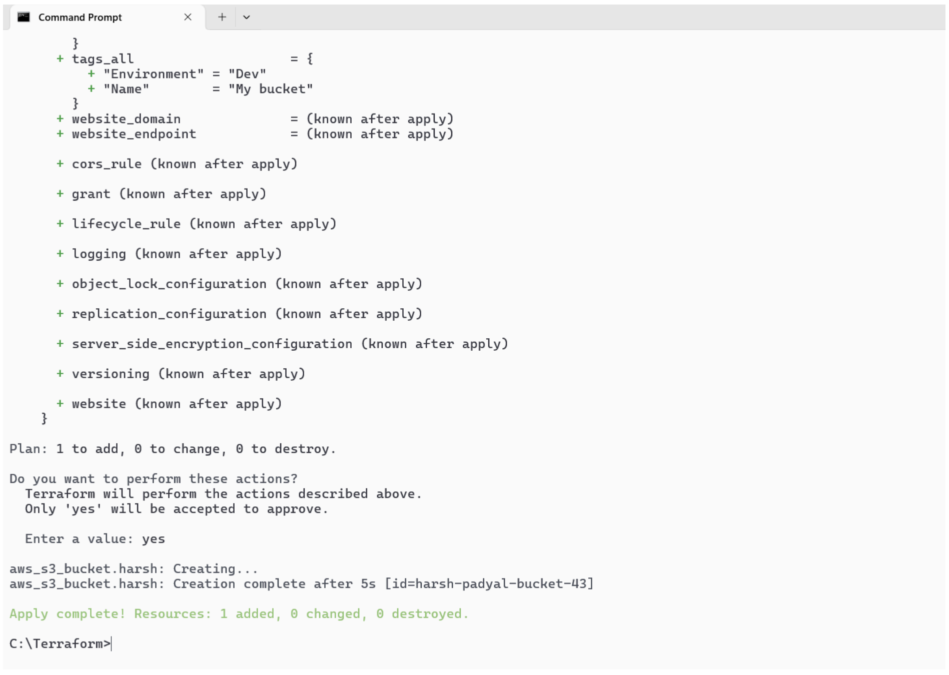


```
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + 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.

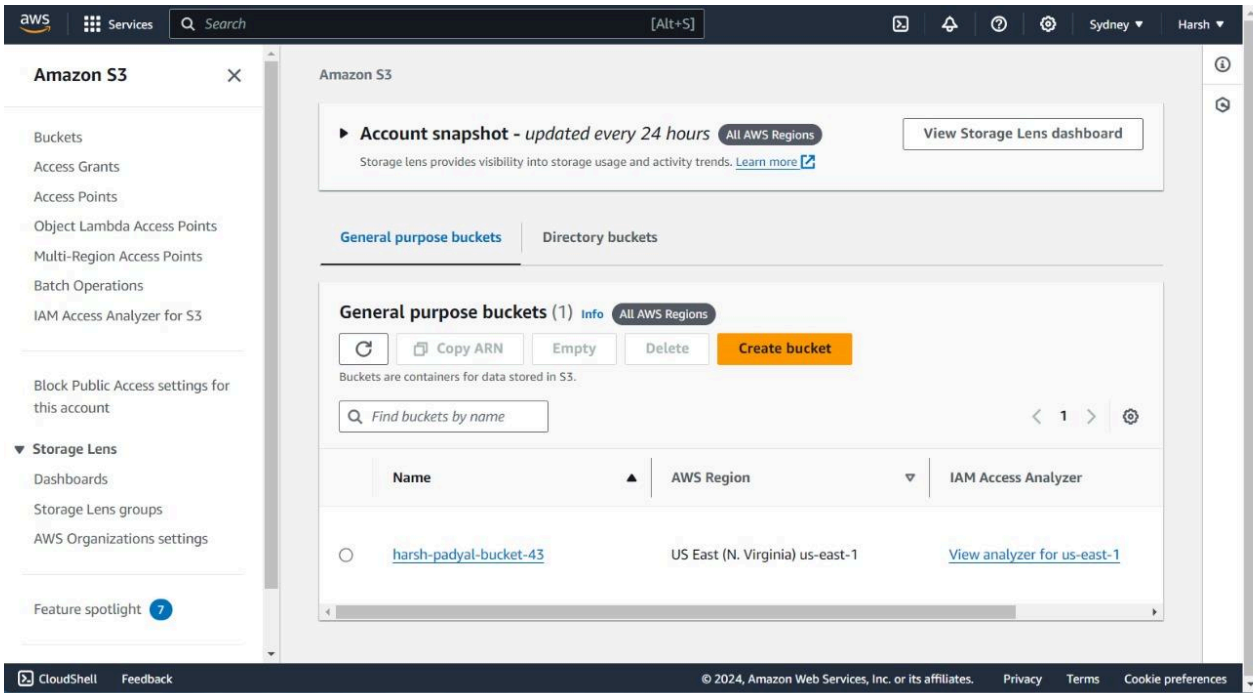


```
C:\Terraform>terraform apply

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.harsh will be created
  + resource "aws_s3_bucket" "harsh" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      + arn                         = (known after apply)
      + bucket                      = "harsh-padyal-bucket-43"
      + 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)
```



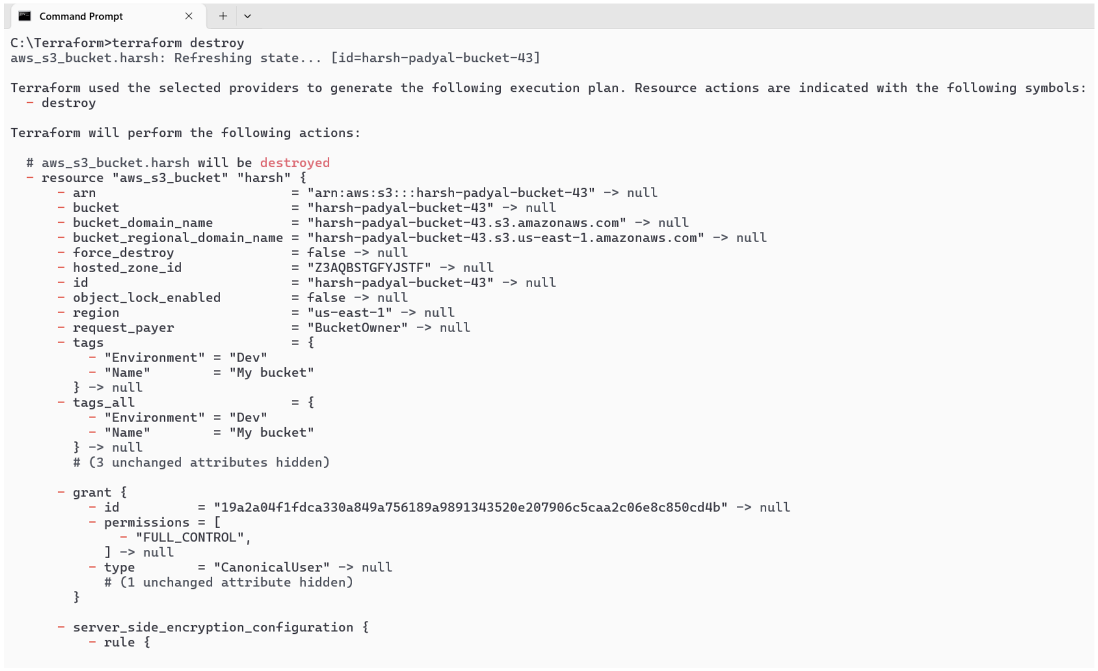
```
        }
      + tags_all                    = {
          + "Environment" = "Dev"
          + "Name"        = "My bucket"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + 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.harsh: Creating...
aws_s3_bucket.harsh: Creation complete after 5s [id=harsh-padyal-bucket-43]

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

C:\Terraform>
```

AWS S3 Bucket dashboard, after Executing Apply step.

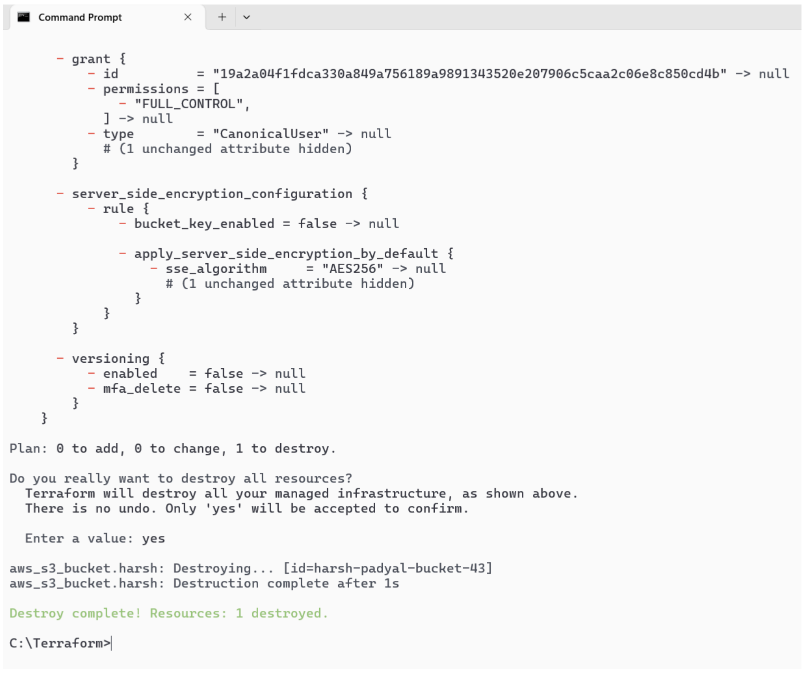

Services
Search [Alt+S]
Sydney
Harsh
Amazon S3
Buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight 7
Amazon S3
Account snapshot - updated every 24 hours
All AWS Regions
Storage lens provides visibility into storage usage and activity trends. Learn more
View Storage Lens dashboard
General purpose buckets
Directory buckets
General purpose buckets (1) Info
All AWS Regions
Copy ARN
Empty
Delete
Create bucket
Buckets are containers for data stored in S3.
Find buckets by name
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Name	AWS Region	IAM Access Analyzer
harsh-padyal-bucket-43	US East (N. Virginia) us-east-1	View analyzer for us-east-1

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

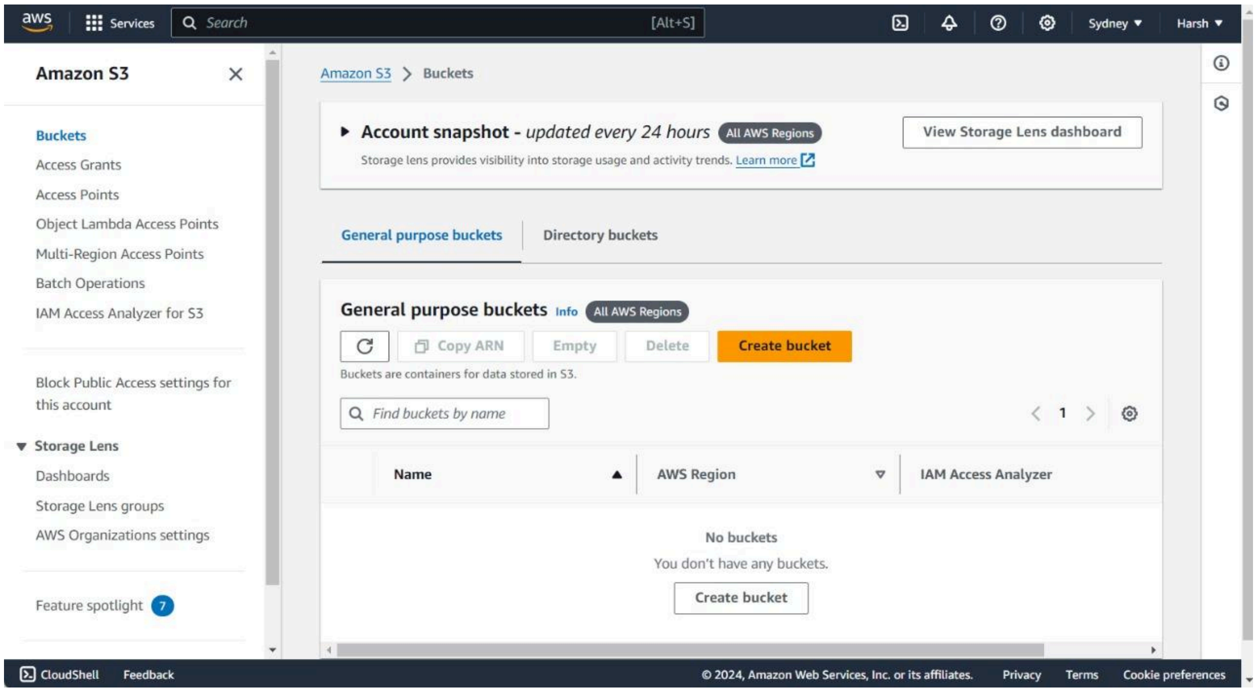


```
C:\Terraform>terraform destroy
aws_s3_bucket.harsh: Refreshing state... [id=harsh-padyal-bucket-43]

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.harsh will be destroyed
  - resource "aws_s3_bucket" "harsh" {
      - arn                         = "arn:aws:s3:::harsh-padyal-bucket-43" -> null
      - bucket                      = "harsh-padyal-bucket-43" -> null
      - bucket_domain_name          = "harsh-padyal-bucket-43.s3.amazonaws.com" -> null
      - bucket_regional_domain_name = "harsh-padyal-bucket-43.s3.us-east-1.amazonaws.com" -> null
      - force_destroy               = false -> null
      - hosted_zone_id              = "Z3AQBSTGFYJSTF" -> null
      - id                          = "harsh-padyal-bucket-43" -> 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          = "19a2a04f1fdca330a849a756189a9891343520e207906c5caa2c06e8c850cd4b" -> null
          - permissions = [
              - "FULL_CONTROL",
            ] -> null
          - type        = "CanonicalUser" -> null
            # (1 unchanged attribute hidden)
        }

      - server_side_encryption_configuration {
          - rule {
```



```
      - grant {
          - id          = "19a2a04f1fdca330a849a756189a9891343520e207906c5caa2c06e8c850cd4b" -> 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.harsh: Destroying... [id=harsh-padyal-bucket-43]
aws_s3_bucket.harsh: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.

C:\Terraform>
```

AWS EC2 dashboard, after Executing Destroy step.



Storage lens provides visibility into storage usage and activity trends. Learn more

General purpose buckets Info All AWS Regions

Buckets are containers for data stored in S3.

No buckets
You don't have any buckets.
Create bucket

© 2024, Amazon Web Services, Inc. or its affiliates.

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.