

Advanced DevOps Lab

Experiment 6

Name: Yash Rahate

Class: D15B

Roll No.: 48

Prerequisite:

- 1) **Must have an AWS Access Key ID and Secret Access Key**

Aim:

To create an AWS Lambda function that logs the message "An Image has been added" when an object is uploaded to a specific S3 bucket.

Theory:

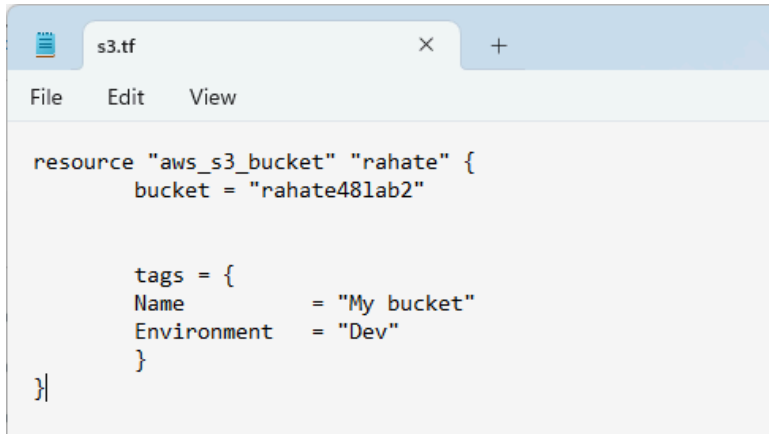
Terraform, as an Infrastructure as Code (IaC) tool, allows you to define and manage your infrastructure using declarative configuration files. This tutorial focuses on creating, modifying, and destroying an AWS S3 bucket using Terraform, which automates the management of resources in a repeatable and predictable way.

Terraform Lifecycle in AWS S3 Management:

1. **Build:**
 - **Write Configuration:** You create a configuration file that defines the desired state of an S3 bucket.
 - **Initialize:** Terraform initializes the working directory, installs the necessary AWS provider, and sets up the workspace.
 - **Plan:** Terraform compares the configuration with the current state and provides a plan showing the resources to be created.
 - **Apply:** Terraform applies the changes, creating the S3 bucket as specified.
2. **Change:**
 - If you want to modify any attribute of the S3 bucket (such as bucket policies, versioning, etc.), you simply update the configuration file.
 - Terraform compares the new desired state with the current state in the `terraform.tfstate` file and generates an execution plan for changes.
 - When you apply the plan, Terraform makes the necessary updates to the S3 bucket.
3. **Destroy:**
 - When you no longer need the S3 bucket or associated resources, you use `terraform destroy`. This command deletes the S3 bucket and any other resources that were defined in the configuration.

Steps to Build, Change, and Destroy AWS S3 Infrastructure:

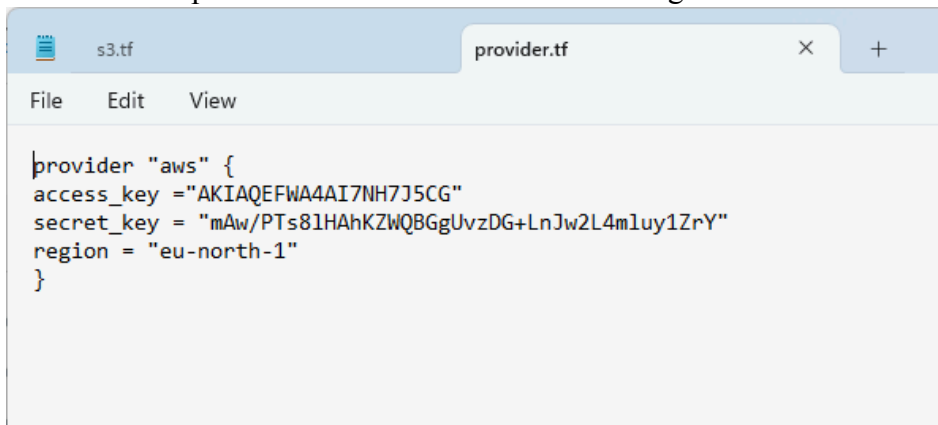
Step 1: Write a Terraform Script in Atom for creating S3 Bucket on Amazon AWS



```
resource "aws_s3_bucket" "rahate" {
  bucket = "rahate48lab2"

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

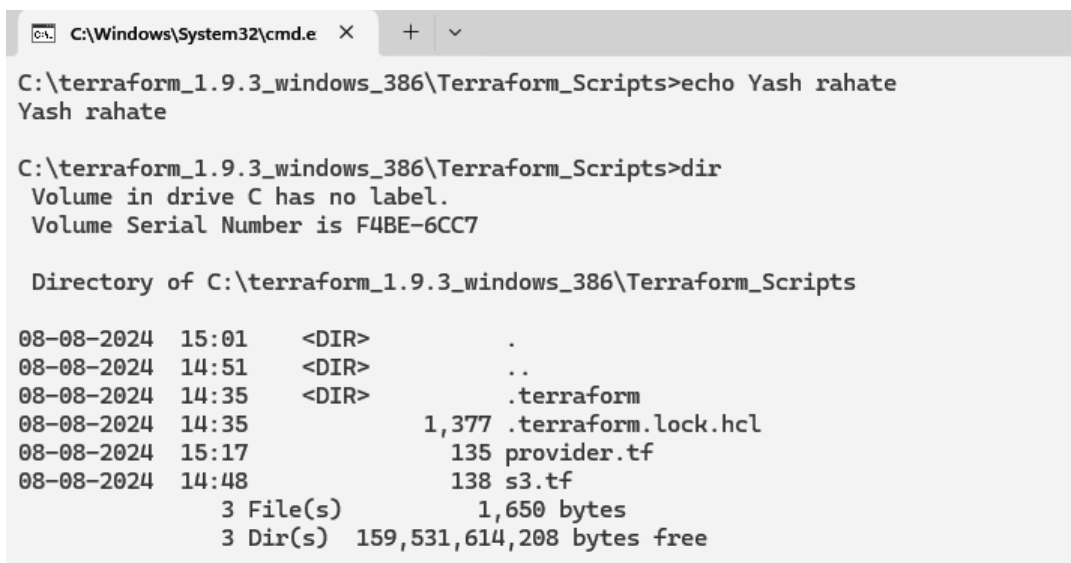
Create a new provider.tf file and write the following contents into it.



```
provider "aws" {
  access_key = "AKIAQEFWA4AI7NH7J5CG"
  secret_key = "mAw/PTs81HAhKZWQBGuVzDG+LnJw2L4m1uy1ZrY"
  region = "eu-north-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:\Windows\System32\cmd.e  X  +  v

C:\terraform_1.9.3_windows_386\Terraform_Scripts>echo Yash rahate
Yash rahate

C:\terraform_1.9.3_windows_386\Terraform_Scripts>dir
Volume in drive C has no label.
Volume Serial Number is F4BE-6CC7

Directory of C:\terraform_1.9.3_windows_386\Terraform_Scripts

08-08-2024  15:01    <DIR>          .
08-08-2024  14:51    <DIR>          ..
08-08-2024  14:35    <DIR>          .terraform
08-08-2024  14:35                1,377 .terraform.lock.hcl
08-08-2024  15:17                135 provider.tf
08-08-2024  14:48                138 s3.tf
               3 File(s)              1,650 bytes
               3 Dir(s)  159,531,614,208 bytes free
```

Step 3: Execute Terraform Init command to initialize the resources

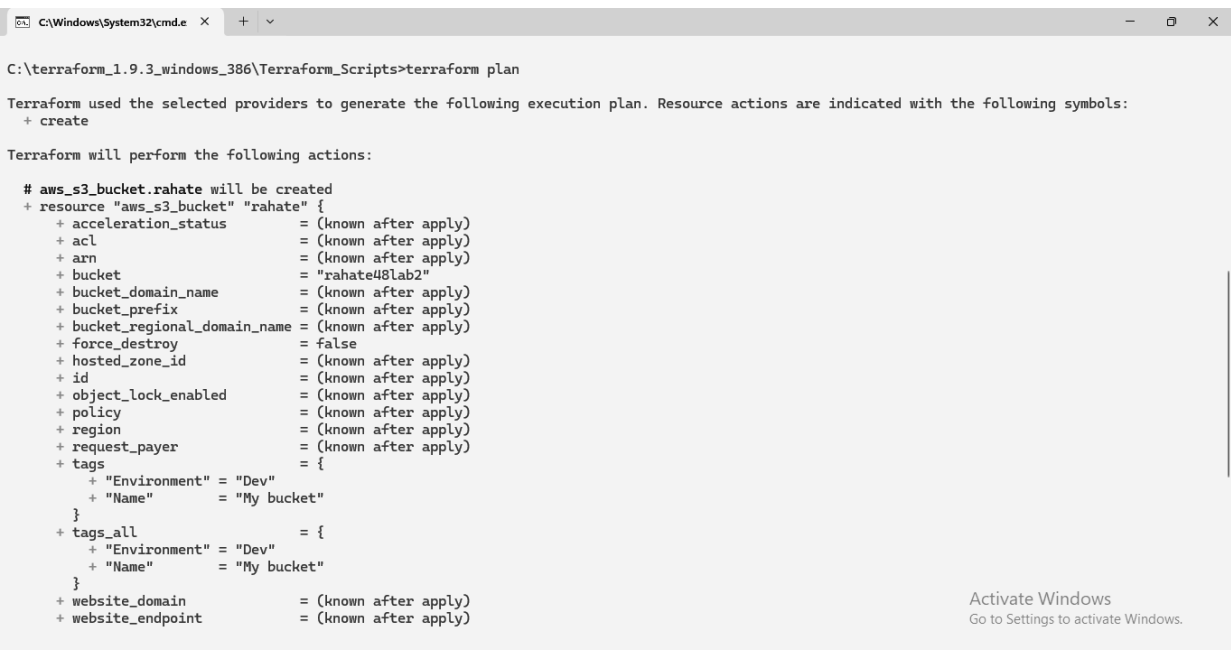
```
C:\terraform_1.9.3_windows_386\Terraform_Scripts>terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.61.0

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 4: Execute Terraform plan to see the available resources



```
C:\Windows\System32\cmd.exe
C:\terraform_1.9.3_windows_386\Terraform_Scripts>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.rahate will be created
+ resource "aws_s3_bucket" "rahate" {
  + acceleration_status = (known after apply)
  + acl                 = (known after apply)
  + arn                 = (known after apply)
  + bucket              = "rahate48lab2"
  + 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)
```

Activate Windows
Go to Settings to activate Windows.

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

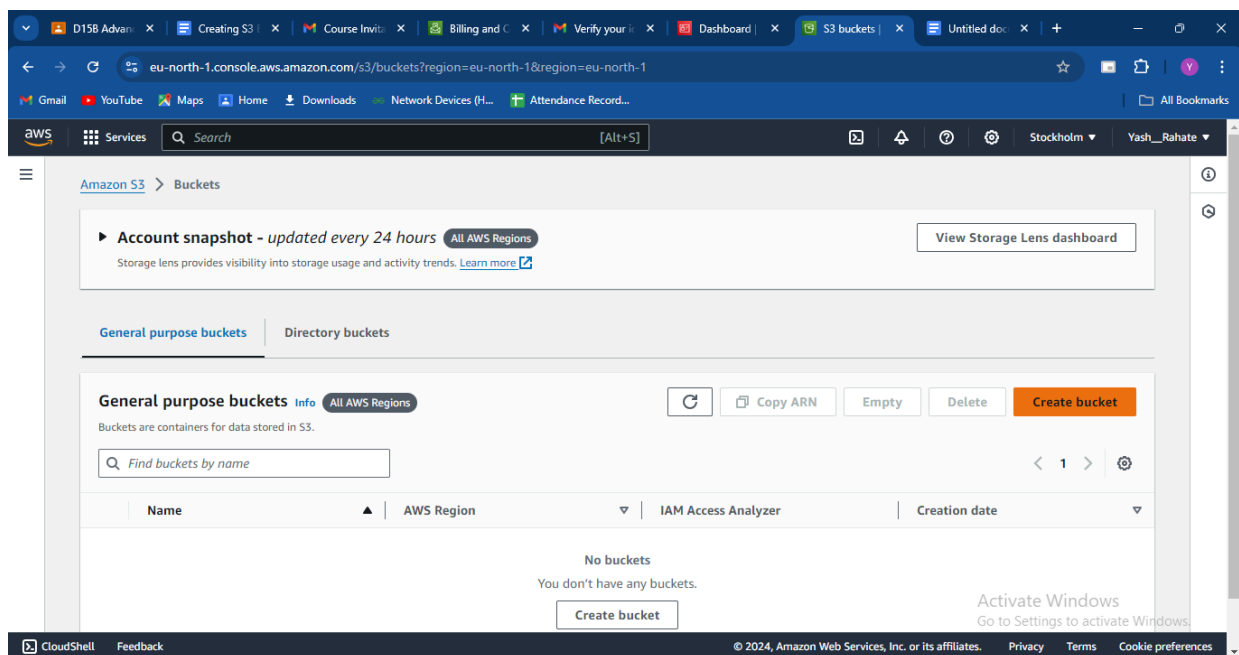
```
C:\Windows\System32\cmd.exe
C:\terraform_1.9.3_windows_386\Terraform_Scripts>terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

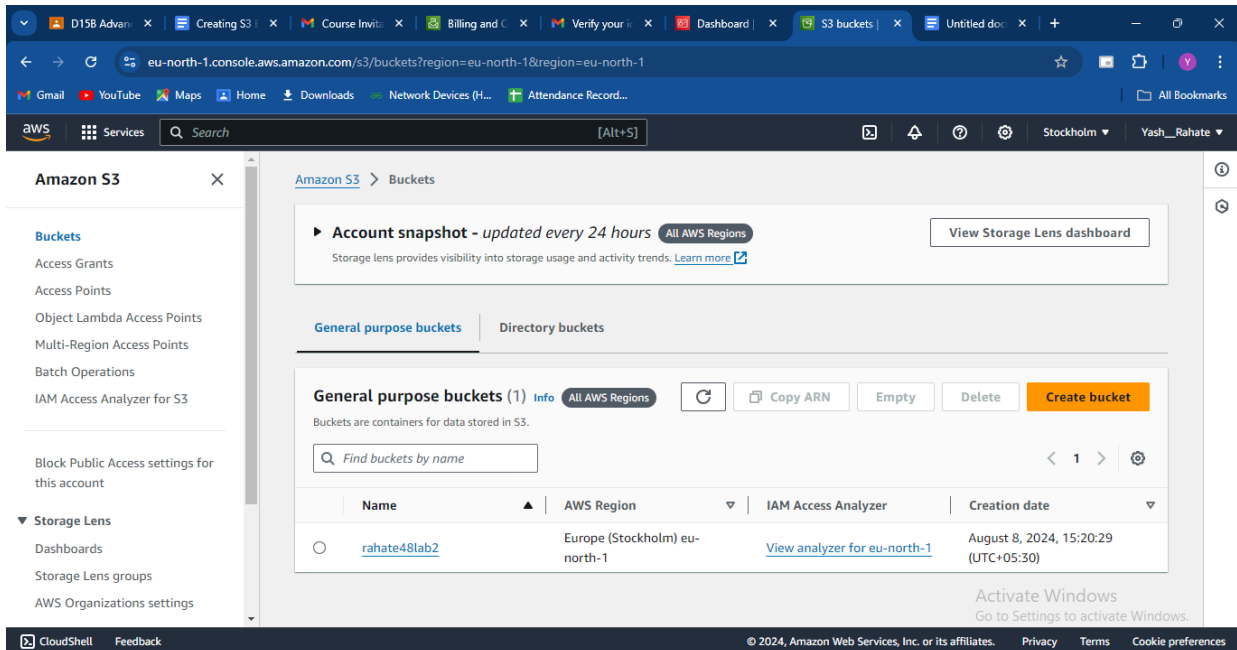
Terraform will perform the following actions:

# aws_s3_bucket.rahate will be created
+ resource "aws_s3_bucket" "rahate" {
+   acceleration_status      = (known after apply)
+   acl                     = (known after apply)
+   arn                     = (known after apply)
+   bucket                  = "rahate48lab2"
+   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)
}
```

AWS S3 bucket dashboard, Before Executing Apply command:



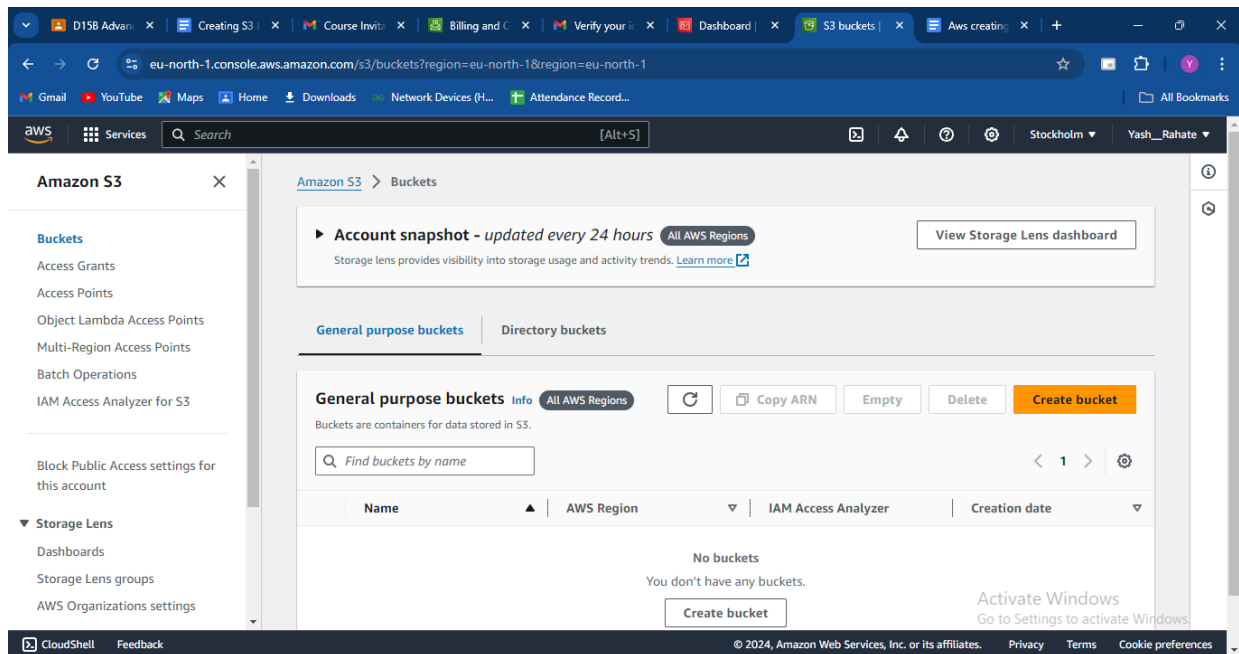
AWS S3 Bucket dashboard, After Executing Apply step:



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



AWS EC2 dashboard, After Executing Destroy step:



Conclusion:

Terraform makes it easy to build, modify, and destroy AWS infrastructure in an automated and predictable way. By using a simple declarative configuration language, Terraform enables developers and administrators to manage infrastructure resources like AWS S3 buckets with minimal manual intervention. The process of creating, changing, and destroying infrastructure is efficient and reusable, ensuring consistency across environments. Using Terraform with AWS not only simplifies infrastructure management but also provides version control, collaboration, and scaling capabilities, making it ideal for modern DevOps practices.