Name : Shiven Bansal **Roll no : 03** **Class : D15C**

Advance DevOps Practicals

Case Study :

3. Infrastructure as Code with Terraform

- **Concepts Used**: Terraform, AWS S3, and EC2.
- **Problem Statement**: "Use Terraform to provision an AWS EC2 instance and an S3 bucket. Deploy a sample static website on the S3 bucket using the EC2 instance as the backend server."
- **Tasks**:
  - Write a Terraform script to create an EC2 instance and an S3 bucket.
  - Deploy the static website on the S3 bucket.
  - Use the EC2 instance to interact with the S3 bucket and log the actions.

1. Introduction

Case Study Overview:

This case study delves into leveraging **Terraform**, an open-source Infrastructure-as-Code (IaC) tool, to automate the provisioning of AWS resources, specifically **EC2 instances** and **S3 buckets**. Terraform enables the declaration of cloud infrastructure in code, making it easier to automate, scale, and manage cloud environments. By using configuration files, it allows infrastructure to be provisioned, modified, and destroyed in a repeatable and version-controlled manner.

In this particular case study, we focus on the following components:

- **Amazon EC2 (Elastic Compute Cloud)**: These are virtual servers in the cloud, providing scalable computing capacity. EC2 allows users to launch instances (virtual machines) with specific operating systems and configurations.
- **Amazon S3 (Simple Storage Service)**: A storage service designed for scalable object storage. S3 is commonly used for storing and retrieving large amounts of data, backups, and content delivery.

Name : Shiven Bansal **Roll no : 03** **Class : D15C**

The purpose of this case study is to showcase how Terraform automates the creation of both EC2 instances for compute resources and S3 buckets for storage requirements. This process is managed through concise and reusable configuration files written in HashiCorp Configuration Language (HCL), eliminating the need for manual intervention via AWS Management Console or CLI. This automation not only speeds up infrastructure provisioning but also minimizes the risks of human error, making it ideal for production environments where consistency and scalability are critical.

This study also explores the use of Terraform modules and state management, allowing for efficient collaboration, infrastructure scaling, and resource management. By utilizing Terraform's declarative approach, infrastructure becomes easier to manage and can be updated or replicated in different environments with minimal effort.

Key Feature and Application:

The **key feature** of this case study is the automated provisioning of cloud infrastructure using Terraform. By writing Terraform configuration files, users can define the infrastructure they need, such as the number and type of EC2 instances, security groups, VPCs (Virtual Private Clouds), and S3 buckets. When the configuration is applied, Terraform automatically provisions these resources in AWS.

Some practical applications of this feature include:

1. **Automated EC2 Instance Deployment**:
    - EC2 instances are automatically created based on the specifications in the configuration file, such as the instance type (e.g., t2.micro), operating system, security groups, and more. This automation eliminates the need to manually launch instances through the AWS console or CLI, greatly reducing setup time.
    - Terraform also enables scalable infrastructure by defining autoscaling groups, ensuring that the infrastructure can dynamically adapt to changing workloads by adding or removing instances as needed.

- Practical application: This feature is ideal for setting up development, testing, or production environments in a consistent and repeatable manner, allowing DevOps teams to spin up environments on demand.

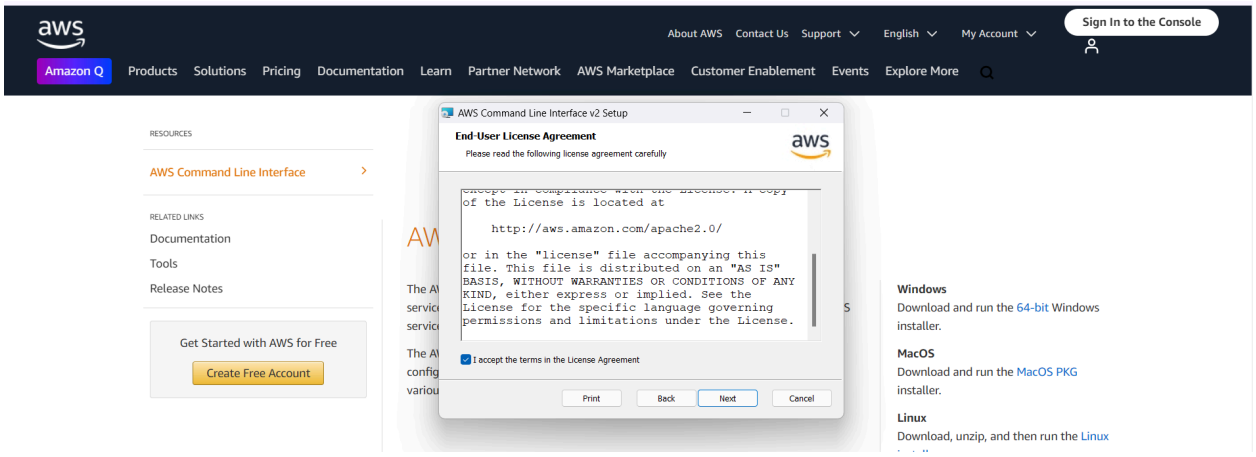
2. **Automated S3 Bucket Provisioning**:
   - Terraform automates the creation of S3 buckets, where you can specify bucket names, versioning, lifecycle rules, and access policies. This ensures that storage is always available as per the configuration without manual intervention.
   - S3 buckets can also be linked with other services such as Lambda for serverless computing, which further extends the automation capabilities.
   - Practical application: Automating S3 bucket provisioning is useful for managing large-scale backups, log storage, data archiving, or delivering static website content. It guarantees that storage is consistently set up as required across multiple environments.
3. **State Management**:
   - Terraform keeps track of the state of your infrastructure, allowing it to detect changes and apply only the necessary updates (referred to as "incremental provisioning"). This ensures that infrastructure updates are efficiently managed without needing to re-provision all resources from scratch.
   - Practical application: In production environments where downtime must be minimized, Terraform's ability to apply only incremental updates allows for safe and controlled changes.
4. **Version Control and Reusability**:
   - Terraform configurations can be version-controlled using Git or other tools, making it easy to track changes and collaborate with other team members. Terraform modules can also be reused across different projects, speeding up infrastructure provisioning and promoting consistency.
   - Practical application: For large-scale organizations with multiple teams working in parallel, Terraform allows for centralized management of infrastructure code, improving collaboration and reducing the risk of configuration drift.

Name : Shiven Bansal **Roll no : 03** **Class : D15C**

The **application** of this automation is particularly useful in scenarios where cloud infrastructure needs to be deployed quickly, consistently, and at scale. By automating EC2 and S3 resource creation, DevOps teams can ensure that environments are set up exactly as defined, allowing them to focus on higher-level tasks like application development and performance optimization instead of infrastructure management.

2. Step-by-Step Explanation

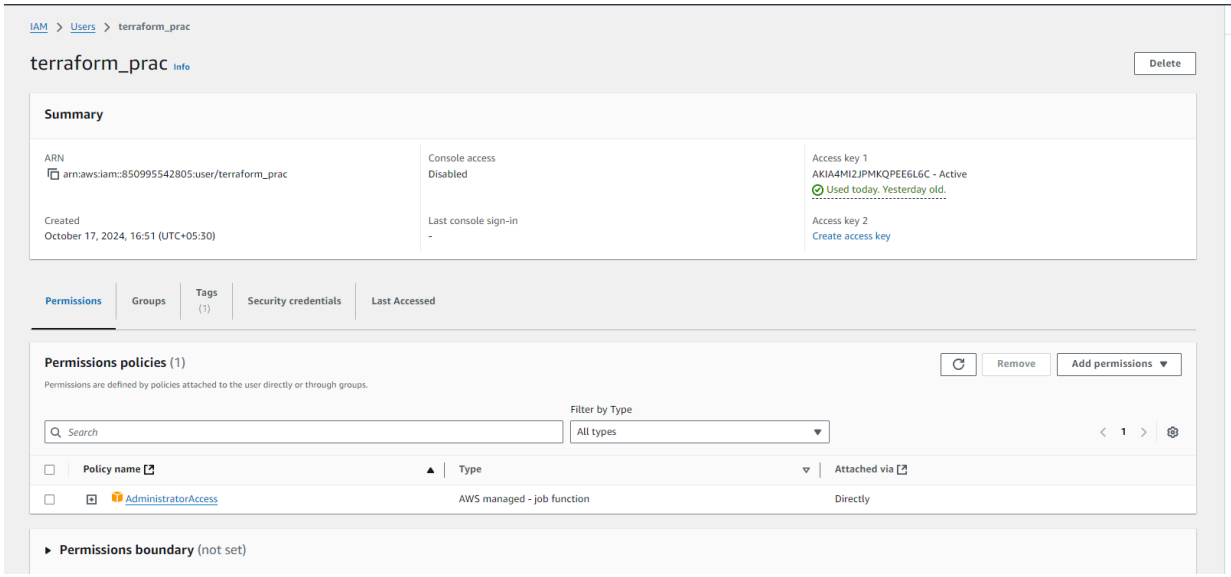
1) Install aws cli for your device from the following url https://aws.amazon.com/cli/ . Complete all the steps and confirm this by checking its version in console by running the command aws –version



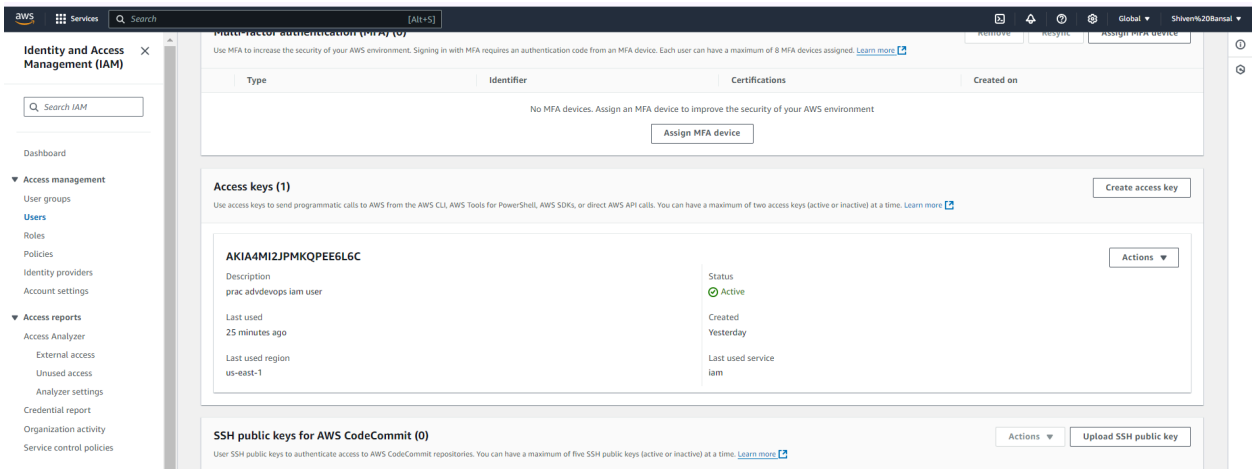

```
C:\Users\ADMIN>aws --version
aws-cli/2.18.8 Python/3.12.6 Windows/11 exe/AMD64
```

2. Create a new IAM user . Give its permission as AdministratorAccess

Name : Shiven Bansal **Roll no : 03** **Class : D15C**



3. Create a new Access key inside the IAM User by navigating to Security section inside your already created user . After creating the access key you will see something similar to this :-



4. Now create a new directory i have named it terraform-ec2-s3 in my desktop .
Contents inside terraform-ec2-s3 folder :

a. website(directory)
   - index.html
   - error.html

b. main.tf

The main.tf file inside this directory will hold our terraform code

Name : Shiven Bansal **Roll no : 03** **Class : D15C**

```
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>cd Desktop

C:\Users\ADMIN\Desktop>mkdir terraform-ec2-s3

C:\Users\ADMIN\Desktop>cd terraform-ec2-s3

C:\Users\ADMIN\Desktop\terraform-ec2-s3>touch main.tf
'touch' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\ADMIN\Desktop\terraform-ec2-s3>echo. > main.tf

C:\Users\ADMIN\Desktop\terraform-ec2-s3>mkdir website
```

5. Initialize the terraform in the directory by running this command : terraform init

```
C:\Users\ADMIN\Desktop\terraform-ec2-s3>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (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.

C:\Users\ADMIN\Desktop\terraform-ec2-s3>
```

7. Following are the contents for the files index.html , error.html and main.tf

a) For index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Static Website</title>
```

```
</head>
<body>
    <h1>Welcome to My Static Website!</h1>
    <p>This website is hosted on AWS S3.</p>
    <p>This is Case study 3 which was performed by Shiven Bansal</p>
</body>
</html>
```

b) For error.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Page Not Found</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            padding: 50px;
        }
        h1 {
            font-size: 50px;
        }
        p {
            font-size: 20px;
        }
        a {
            text-decoration: none;
            color: #007BFF;
        }
    </style>
</head>
<body>
```

```
    <h1>Oops! Page Not Found</h1>
    <p>We're sorry, but the page you were looking for doesn't exist.</p>
    <p><a href="index.html">Go back to the homepage</a></p>
</body>
</html>
```

c) for main.tf

```
# AWS Provider
provider "aws" {
  region = "us-east-1"  # Use the region you configured with aws configure
}

# IAM Role for EC2 to Access S3
resource "aws_iam_role" "ec2_role" {
  name = "ec2_role"

  assume_role_policy = jsonencode({
    Version: "2012-10-17",
    Statement: [{
      Action: "sts:AssumeRole",
      Effect: "Allow",
      Principal: {
        Service: "ec2.amazonaws.com"
      }
    }]
  })
}

# Attach S3 Full Access Policy to the EC2 Role
resource "aws_iam_policy_attachment" "ec2_s3_access" {
  name       = "ec2-s3-access"
  roles      = [aws_iam_role.ec2_role.name]
  policy_arn = "arn:aws:iam::aws:policy/AmazonS3FullAccess"
}
```

Name : Shiven Bansal **Roll no : 03** **Class : D15C**

```
# Instance Profile for EC2
resource "aws_iam_instance_profile" "ec2_profile" {
  name = "ec2_profile"
  role = aws_iam_role.ec2_role.name
}

# EC2 Instance
resource "aws_instance" "example" {
  ami                  = "ami-06b21ccaeff8cd686"  # Change this AMI based on your
region if necessary
  instance_type        = "t2.micro"
  iam_instance_profile  = aws_iam_instance_profile.ec2_profile.name

  tags = {
    Name = "Terraform-EC2"
  }
}

resource "aws_s3_bucket" "website_bucket" {
  bucket = "bucket05122004advdevopsshivenbansal03"

  # Remove the acl argument
  # Other parameters, such as versioning, lifecycle rules, etc.
}

#resource "aws_s3_bucket_acl" "website_bucket_acl" {
  #bucket = aws_s3_bucket.website_bucket.id
  #acl    = "private"  # Set the desired ACL here
#}

resource "aws_s3_bucket_public_access_block" "website_bucket_public_access" {
  bucket                  = aws_s3_bucket.website_bucket.id
  block_public_acls       = false
  ignore_public_acls      = false
```

```
  block_public_policy    = false
  restrict_public_buckets = false
}



resource "aws_s3_bucket_website_configuration" "website_config" {
  bucket = aws_s3_bucket.website_bucket.id

  index_document {
    suffix = "index.html"
  }

  error_document {
    key = "error.html"
  }
}

# Upload files to the S3 bucket
resource "aws_s3_object" "website_index" {
  bucket = aws_s3_bucket.website_bucket.bucket
  key    = "index.html"
  source = "./website/index.html"  # Adjust the path if necessary
  #acl    = "public-read"  # Public read access for the object
}

resource "aws_s3_object" "website_error" {
  bucket = aws_s3_bucket.website_bucket.bucket
  key    = "error.html"
  source = "./website/error.html"  # Adjust the path if necessary
  #acl    = "public-read"  # Public read access for the object
}
```

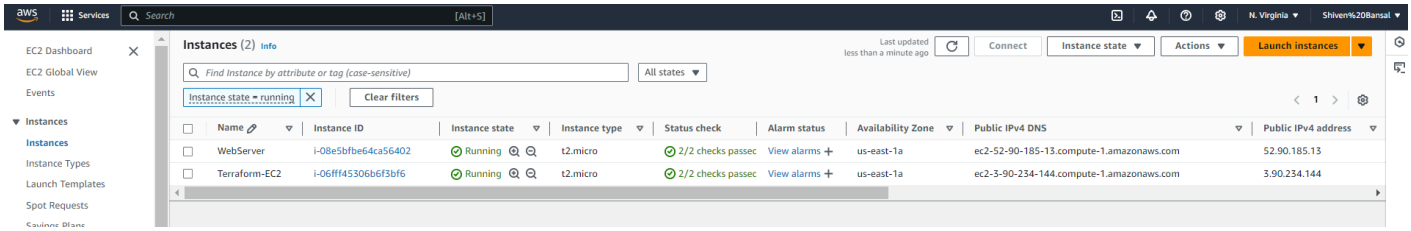
8. Execute terraform.apply command and it should run successfully

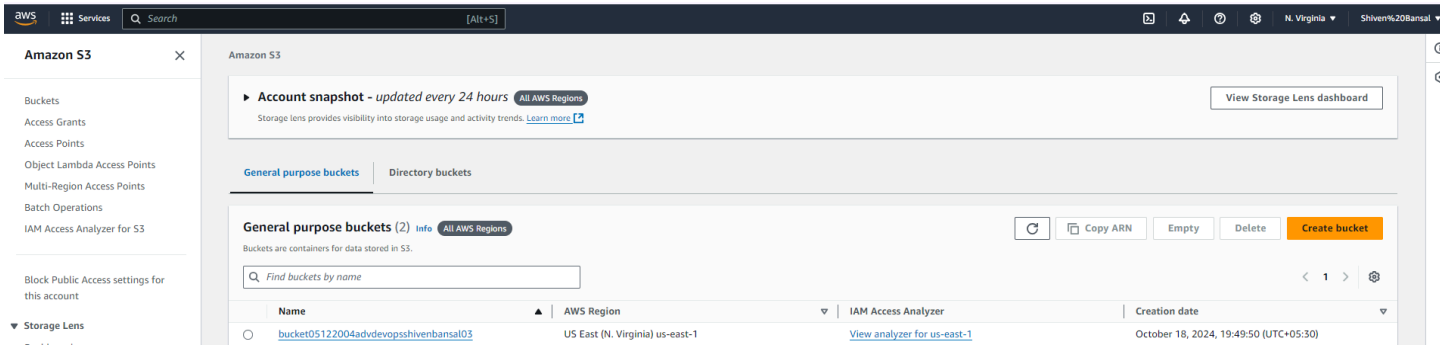
Name : Shiven Bansal **Roll no : 03** **Class : D15C**

```
C:\Users\ADMIN\Desktop\terraform-ec2-s3>terraform apply
aws_iam_role.ec2_role: Refreshing state... [id=ec2_role]
aws_s3_bucket.website_bucket: Refreshing state... [id=bucket05122004advdevopsshivenbansal03]
aws_iam_policy_attachment.ec2_s3_access: Refreshing state... [id=ec2-s3-access]
aws_iam_instance_profile.ec2_profile: Refreshing state... [id=ec2_profile]
aws_instance.example: Refreshing state... [id=i-06fff45306b6f3bf6]
aws_s3_bucket_public_access_block.website_bucket_public_access: Refreshing state... [id=bucket05122004advdevopsshivenbansal03]
aws_s3_bucket_website_configuration.website_config: Refreshing state... [id=bucket05122004advdevopsshivenbansal03]

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_object.website_error will be created
  + resource "aws_s3_object" "website_error" {
      + acl                    = (known after apply)
      + arn                    = (known after apply)
      + bucket                 = "bucket05122004advdevopsshivenbansal03"
      + bucket_key_enabled     = (known after apply)
      + checksum_crc32         = (known after apply)
      + checksum_crc32c        = (known after apply)
      + checksum_sha1          = (known after apply)
      + checksum_sha256        = (known after apply)
      + content_type           = (known after apply)
      + etag                   = (known after apply)
      + force_destroy          = false
      + id                     = (known after apply)
      + key                    = "error.html"
      + kms_key_id             = (known after apply)
      + server_side_encryption = (known after apply)
      + source                 = "./website/error.html"
      + storage_class          = (known after apply)
      + tags_all               = (known after apply)
      + version_id             = (known after apply)
    }

  # aws_s3_object.website_index will be created
  + resource "aws_s3_object" "website_index" {
      + acl                    = (known after apply)
      + arn                    = (known after apply)
      + bucket                 = "bucket05122004advdevopsshivenbansal03"
      + bucket_key_enabled     = (known after apply)
      + checksum_crc32         = (known after apply)
      + checksum_crc32c        = (known after apply)
      + checksum_sha1          = (known after apply)
      + checksum_sha256        = (known after apply)
      + content_type           = (known after apply)
      + etag                   = (known after apply)
      + force_destroy          = false
      + id                     = (known after apply)
      + key                    = "index.html"
      + kms_key_id             = (known after apply)
      + server_side_encryption = (known after apply)
      + source                 = "./website/index.html"
      + storage_class          = (known after apply)
      + tags_all               = (known after apply)
      + version_id             = (known after apply)
    }

Plan: 2 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_object.website_index: Creating...
aws_s3_object.website_error: Creating...
aws_s3_object.website_index: Creation complete after 2s [id=index.html]
aws_s3_object.website_error: Creation complete after 2s [id=error.html]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

C:\Users\ADMIN\Desktop\terraform-ec2-s3>
```

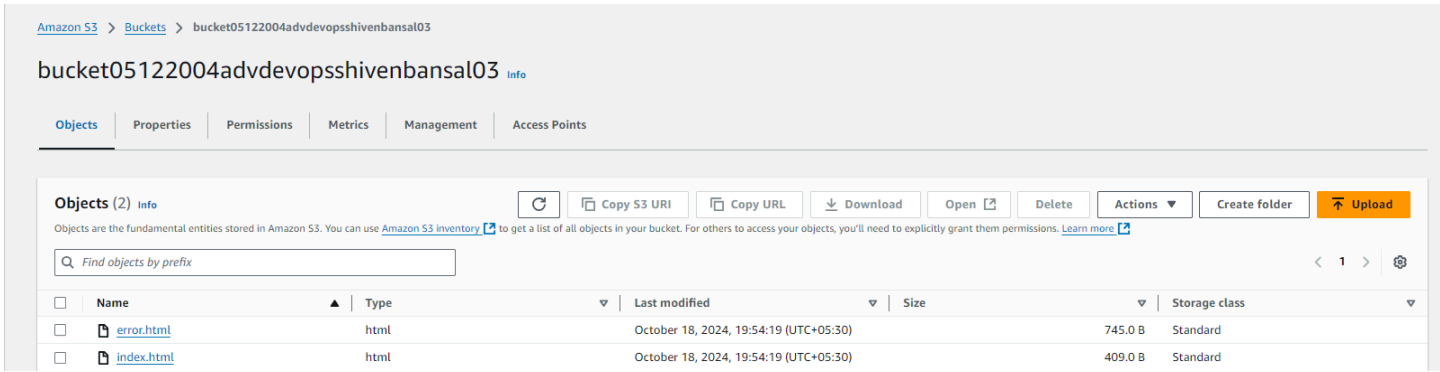
Name : Shiven Bansal **Roll no : 03** **Class : D15C**

9. Since the terraform apply was executed successfully use should now be able to see a new ec2 instance “Terraform-EC2” and new s3 bucket named “bucket05122004advdevopsshivenbansal03” like the following screenshots



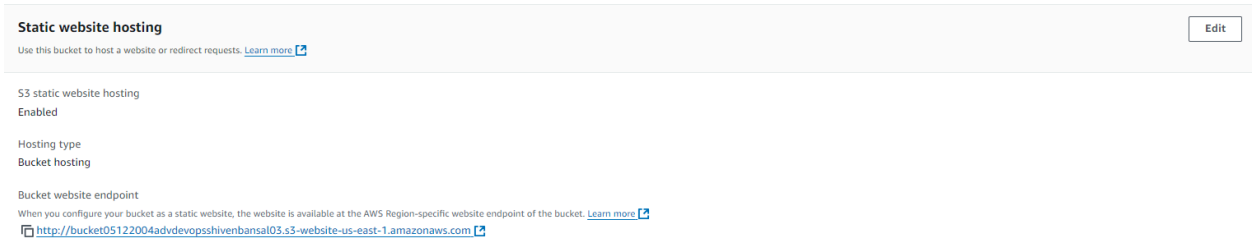


We can see in the bucket that error.html and index.html are successfully deployed through terraform



10. In the S3 bucket settings, confirm that the **Static website hosting** option is enabled.

Name : Shiven Bansal **Roll no : 03** **Class : D15C**



11. Modify your terraform code in main.tf as follows :

```
# AWS Provider
provider "aws" {
  region = "us-east-1"  # Use the region you configured with aws configure
}

# IAM Role for EC2 to Access S3
resource "aws_iam_role" "ec2_role" {
  name = "ec2_role"

  assume_role_policy = jsonencode({
    Version: "2012-10-17",
    Statement: [{
      Action: "sts:AssumeRole",
      Effect: "Allow",
      Principal: {
        Service: "ec2.amazonaws.com"
      }
    }]
  })
}

# Attach S3 Full Access Policy to the EC2 Role
resource "aws_iam_policy_attachment" "ec2_s3_access" {
  name       = "ec2-s3-access"
  roles      = [aws_iam_role.ec2_role.name]
  policy_arn = "arn:aws:iam::aws:policy/AmazonS3FullAccess"
```

```
}

# Instance Profile for EC2
resource "aws_iam_instance_profile" "ec2_profile" {
  name = "ec2_profile"
  role = aws_iam_role.ec2_role.name
}

# Generate an SSH Key Pair
resource "tls_private_key" "my_key" {
  algorithm = "RSA"
  rsa_bits  = 2048
}

resource "aws_key_pair" "my_key" {
  key_name   = "id_rsa_shiven_public"
  public_key = tls_private_key.my_key.public_key_openssh
}

# Save the private key to a file (for local access)
resource "local_file" "private_key" {
  filename = "${path.module}/id_rsa_private"  # Adjust the path if necessary
  content  = tls_private_key.my_key.private_key_pem
}

# EC2 Instance
resource "aws_instance" "example" {
  ami                  = "ami-06b21ccaeff8cd686"  # Change this AMI based on your
region if necessary
  instance_type        = "t2.micro"
  iam_instance_profile  = aws_iam_instance_profile.ec2_profile.name
  key_name             = aws_key_pair.my_key.key_name  # Associate the key pair
with the instance

  tags = {
```

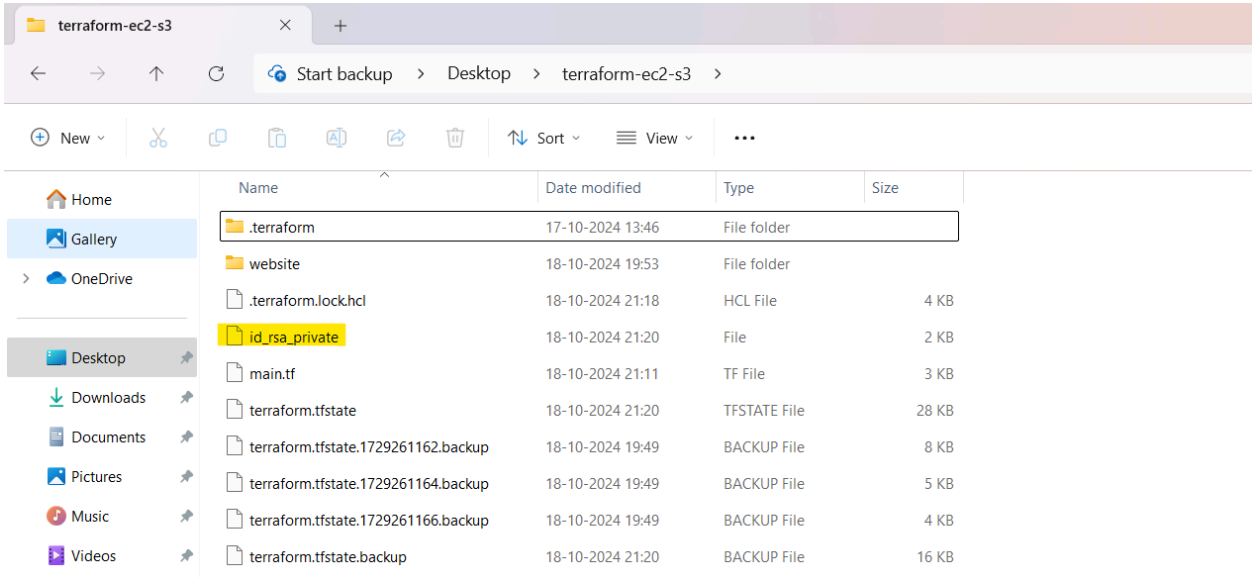
```
    Name = "Terraform-EC2"
  }
}

# S3 Bucket
resource "aws_s3_bucket" "website_bucket" {
  bucket = "bucket05122004advdevopsshivenbansal03"
}

resource "aws_s3_bucket_public_access_block" "website_bucket_public_access" {
  bucket                  = aws_s3_bucket.website_bucket.id
  block_public_acls       = false
  ignore_public_acls      = false
  block_public_policy     = false
  restrict_public_buckets = false
}

resource "aws_s3_bucket_website_configuration" "website_config" {
  bucket = aws_s3_bucket.website_bucket.id

  index_document {
    suffix = "index.html"
  }

  error_document {
    key = "error.html"
  }
}

# Upload files to the S3 bucket
resource "aws_s3_object" "website_index" {
  bucket = aws_s3_bucket.website_bucket.bucket
  key    = "index.html"
  source = "./website/index.html"
}
```

Name : Shiven Bansal **Roll no : 03** **Class : D15C**

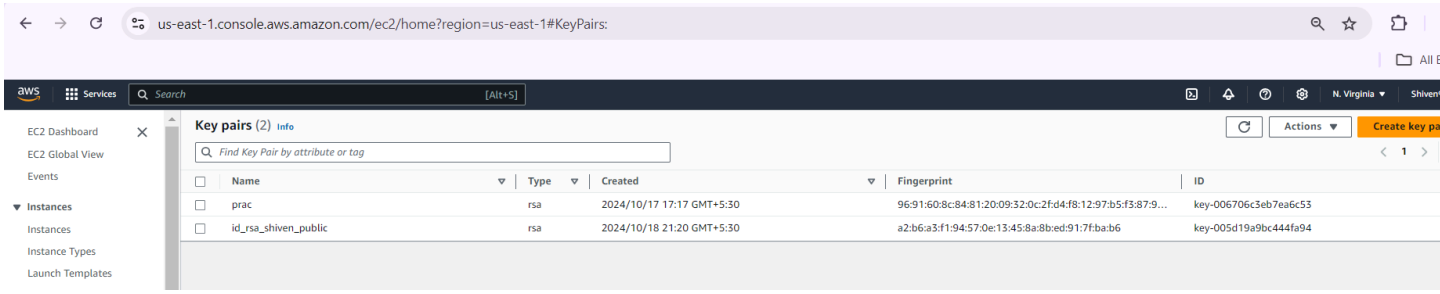
```
resource "aws_s3_object" "website_error" {
  bucket = aws_s3_bucket.website_bucket.bucket
  key    = "error.html"
  source = "./website/error.html"
}
```

We have added public and private keys in the new code.
Now run terraform init , terraform plan , terraform apply again .

12. Now you can see the new key pair file named “id_rsa_private” has been downloaded in the same directory as all other files in the terraform-ec2-s3 directory



We can also see the new public key name “id_rsa_shiven_public” in aws key pairs section



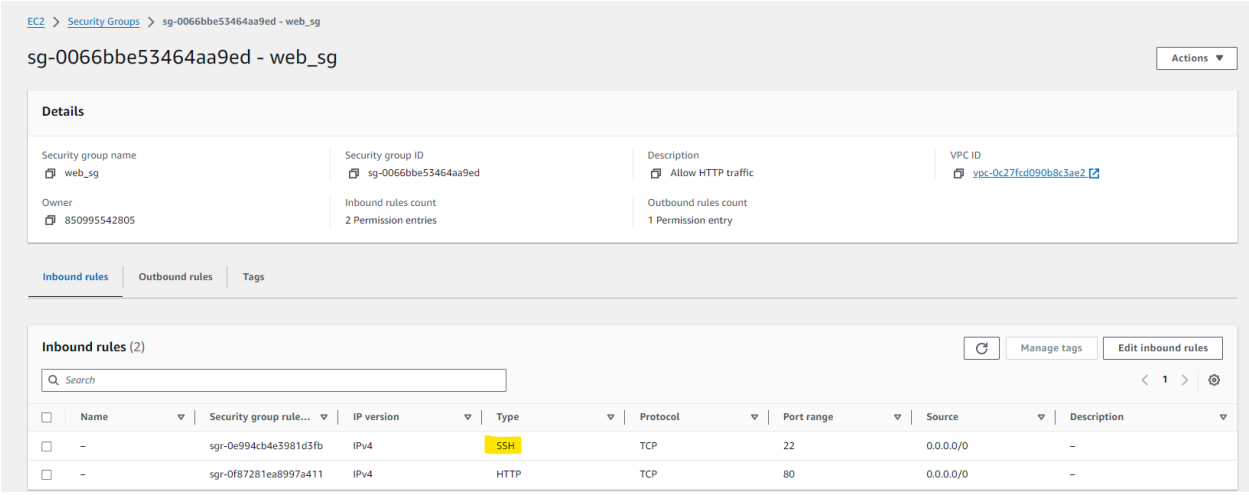
Name : Shiven Bansal **Roll no : 03** **Class : D15C**

Use this command to remotely login into your ec2 instance
ssh -i "<path to your main.tf file/ has to be the same path>\id_rsa_private"
ec2-user@ec2-54-236-238-111.compute-1.amazonaws.com

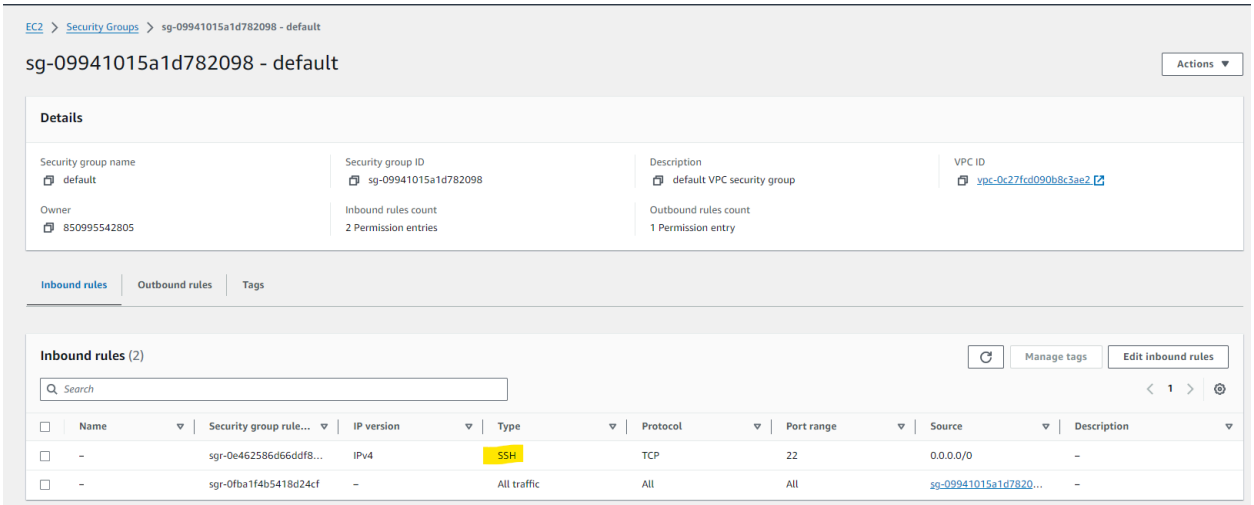
```
PS C:\Users\ADMIN> ssh -i "C:\Users\ADMIN\Desktop\terraform-ec2-s3/id_rsa_private" ec2-user@ec2-34-235-156-185.compute-1.amazonaws.com
ssh: connect to host ec2-34-235-156-185.compute-1.amazonaws.com port 22: Connection timed out
```

I encountered connection timeout issue this generally indicates that the connection request is being blocked, to resolve this

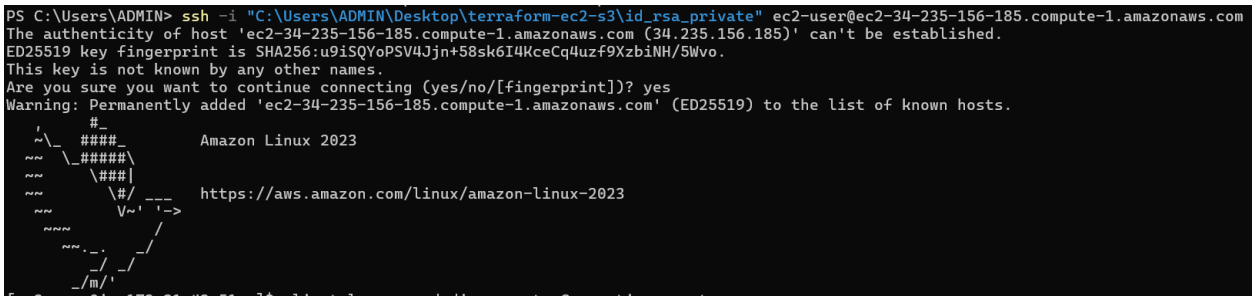
a) Go to the AWS Management Console.
b) Navigate to EC2 > Instances.
c) Select your instance and find the Security Group in the description.
d) Click on the security group link to go to the security group settings.
e) Under the Inbound rules tab, ensure there is a rule that allows SSH



Name : Shiven Bansal **Roll no : 03** **Class : D15C**



I have added SSH inbound rule in both the security groups . Now using the command of ssh -i "C:\Users\ADMIN\Desktop\terraform-ec2-s3\id_rsa_private" ec2-user@ec2-34-235-156-185.compute-1.amazonaws.com (in my case) remotely login into your ec2 instance



13. Now we will deploy another file called about.html in our bucket
Add the about.html in the website directory of the root directory terraform-ec2-s3

Name : Shiven Bansal **Roll no : 03** **Class : D15C**

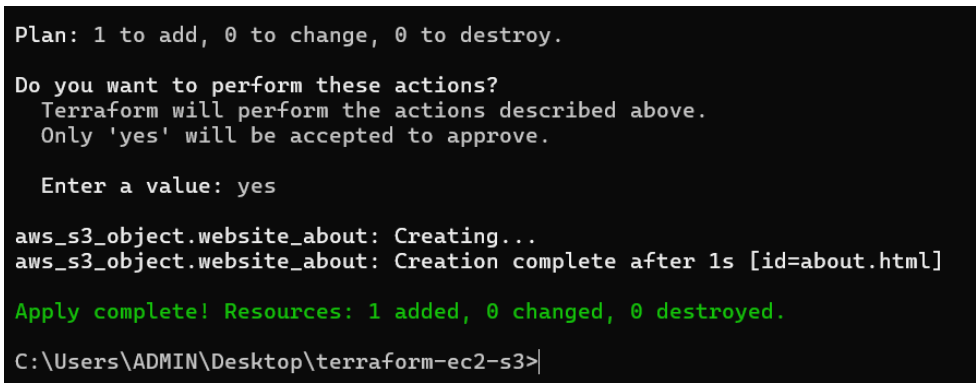
14. Add the following code at the end of main.tf

```
resource "aws_s3_object" "website_about" {
  bucket = aws_s3_bucket.website_bucket.bucket
  key    = "about.html"        # The name the file will have in S3
  source = "./website/about.html"  # The local path to your file
}
```

This is a new aws_s3_object resource for the new file.

15. Now that we have added the about.html in website directory . Open the terminal and again execute the commands :

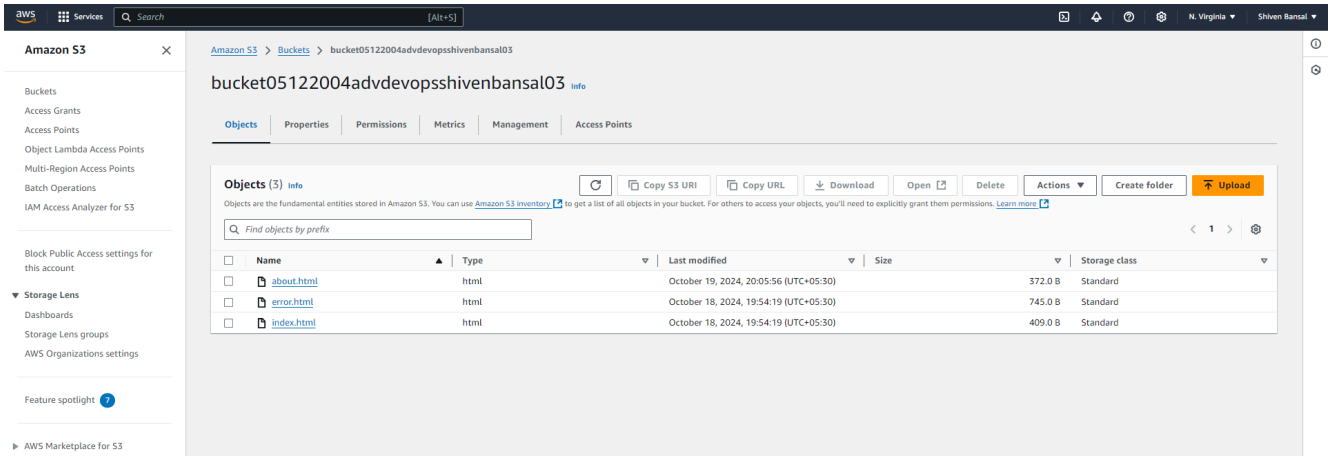
Terraform init , terraform plan and terraform apply to apply the changes



This screenshot of terraform apply says that the new resource (about.html) was successfully added in our bucket in aws called “bucket051122004advancedevopsshivenbansal03”

16. Check in the contents of your S3 bucket the new file should have been added

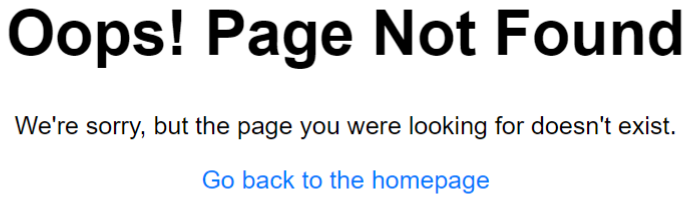
Name : Shiven Bansal **Roll no : 03** **Class : D15C**



These are the files uploaded :

a) about.html

b) error.html



c) index.html

Name : Shiven Bansal **Roll no : 03** **Class : D15C**

Alternatively we can deploy static websites using ec2 also by following these steps :

Since i was getting errors by doing this in cmd i executed the steps in gitbash

17. Upload Files from Local Machine to EC2 using the command

scp -i path/to/your/key.pem
C:\Users\ADMIN\Desktop\terraform-ec2-s3\website\test.html
ec2-user@<your-ec2-public-dns>:/home/ec2-user/

```
ADMIN@DESKTOP-LPV2RP5 MINGW64 ~
$ scp -i "/c/Users/ADMIN/Desktop/terraform-ec2-s3/id_rsa_private" "/c/Users/ADMI
N/Desktop/terraform-ec2-s3/website/test.html" ec2-user@ec2-34-235-156-185.comput
e-1.amazonaws.com:/home/ec2-user/
test.html                                     100%  281     1.3KB/s   00:00
```

17. In the console execute the following command to upload files to your S3 bucket:

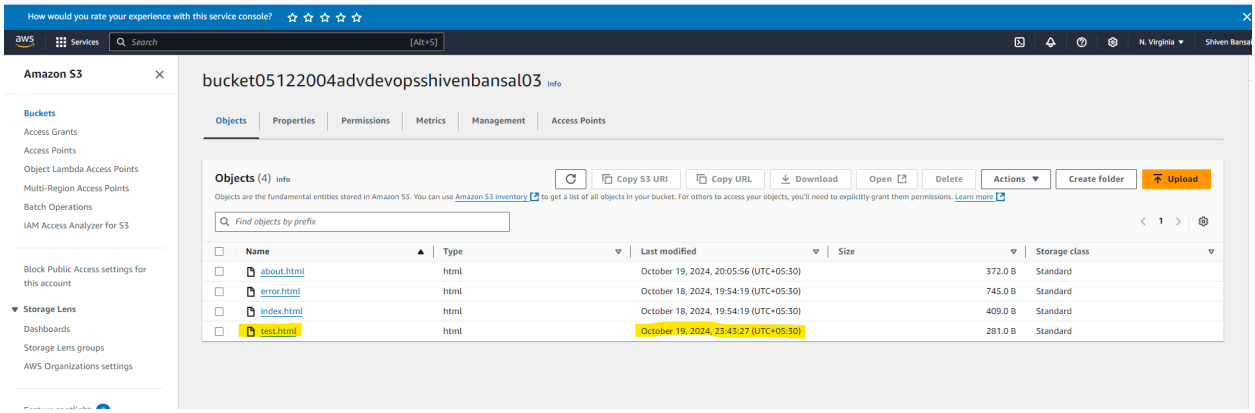
aws s3 cp /path/to/your/local/file s3://your-bucket-name/

```
[ec2-user@ip-172-31-42-51 ~]$ aws s3 cp /home/ec2-user/test.html s3://bucket0512
2004advdevopsshivenbansal03/
upload: ./test.html to s3://bucket05122004advdevopsshivenbansal03/test.html
```

Name : Shiven Bansal **Roll no : 03** **Class : D15C**

```
[ec2-user@ip-172-31-42-51 ~]$ aws s3 ls s3://bucket05122004advdevopsshivenbansal
03/
2024-10-19 14:35:56        372 about.html
2024-10-18 14:24:19        745 error.html
2024-10-18 14:24:19        409 index.html
2024-10-19 18:13:27        281 test.html
[ec2-user@ip-172-31-42-51 ~]$
```

By verifying using the ls command our file test.html was successfully uploaded to our s3 bucket



Conclusion

In this experiment, I used Terraform to automate the creation of AWS resources, including EC2 instances and an S3 bucket. After setting up the infrastructure, I deployed a sample HTML website by uploading the static files to the S3 bucket. By configuring the bucket for static website hosting, the HTML files were made publicly accessible. This process not only demonstrated how Terraform can automate cloud resource management but also how easily a simple website can be hosted on AWS using S3. The combination of automation and cloud hosting ensures a streamlined and efficient deployment process.