NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**

CASE STUDY::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.
My case study topic is as follows:-
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

Introduction to Infrastructure as Code (IaC) with Terraform

Infrastructure as Code (IaC) allows the automation of infrastructure deployment and management using code, offering efficiency, repeatability, and version control. Terraform is one of the most widely used IaC tools, allowing developers to define and manage infrastructure in cloud environments such as AWS, Google Cloud, and Azure. Terraform uses declarative configuration files that describe the desired state of the infrastructure, enabling consistent provisioning across environments. In this scenario, you are asked to use Terraform to provision two AWS services: an EC2 instance and an S3 bucket, and to deploy a static website on the S3 bucket while using the EC2 instance to interact with the bucket as a backend server.

Significance of Using Terraform in AWS

1. Automation: With Terraform, cloud resources can be created, modified, or destroyed automatically, reducing the need for manual intervention.
2. Version Control: The infrastructure code can be versioned and maintained in repositories, enabling teams to track changes, roll back configurations, and collaborate effectively.
3. Consistency: Infrastructure can be provisioned consistently across different environments, reducing configuration drift and ensuring that development, testing, and production environments are identical.
4. Multi-Cloud Support: Terraform provides the flexibility to manage resources not only on AWS but also across multiple cloud providers with a single tool.

Key Concepts and Services Involved are as follows:-

Terraform

- Significance: Terraform automates the provisioning and management of cloud resources. It allows organizations to treat their infrastructure as code, providing benefits like automation, collaboration, and consistency.

NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**

AWS EC2 (Elastic Compute Cloud)

- Significance: In this context, the EC2 instance will serve as the backend server that can interact with the S3 bucket. It will log the actions related to the deployment and management of the static website hosted on S3.
- Applications: EC2 is used for running web applications, backend services, databases, and other computing tasks. In this project, it plays a crucial role in managing and logging the interactions with the S3 bucket.

AWS S3 (Simple Storage Service)

- Significance: The S3 bucket will host the static website. S3 provides high availability and content delivery, making it a suitable platform for website hosting.
- Applications: AWS S3 is widely used for static website hosting, backup storage, big data analytics, and content distribution.

Static Website Hosting on S3

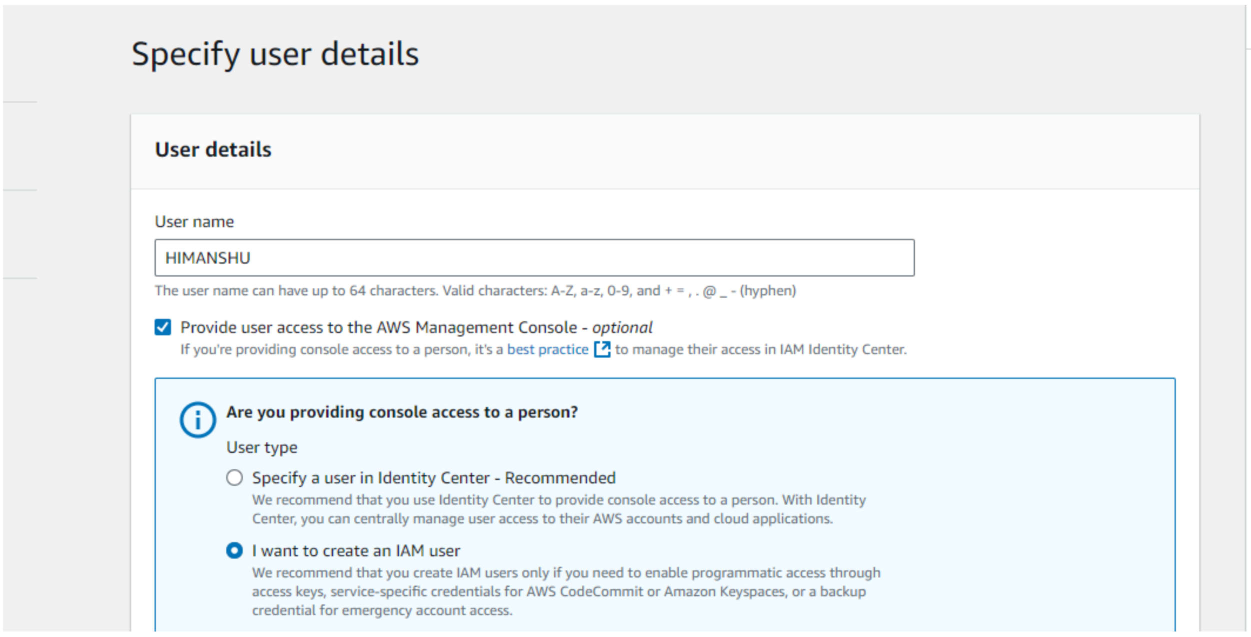
- Significance: Using S3 to host static websites is cost-effective and scalable. By leveraging this, the project eliminates the need for a dedicated web server, offloading the hosting duties to S3.
- Applications: This is useful for content-heavy websites, landing pages, and blogs that do not require server-side processing.

Applications of the Solution are as follows:-

1. Infrastructure Automation: The ability to automate infrastructure provisioning for websites, applications, or backend services significantly reduces deployment time and errors.
2. Scalable Website Hosting: Hosting static websites on S3 provides a scalable and highly available solution for website deployment. This is especially useful for static content, where high performance and low costs are critical.
3. Log Management and Monitoring: Using EC2 to interact with the S3 bucket for logging purposes provides deeper insight into website management. This setup can be extended to gather and analyze logs, track usage, or monitor performance metrics.
4. Collaboration and Versioning: By using Terraform, teams can collaborate effectively with shared infrastructure definitions, enabling version control of infrastructure similar to how they manage application code.

NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**

Step 1: Install AWS CLI



Specify user details

User details

User name

HIMANSHU

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.

Are you providing console access to a person?

User type

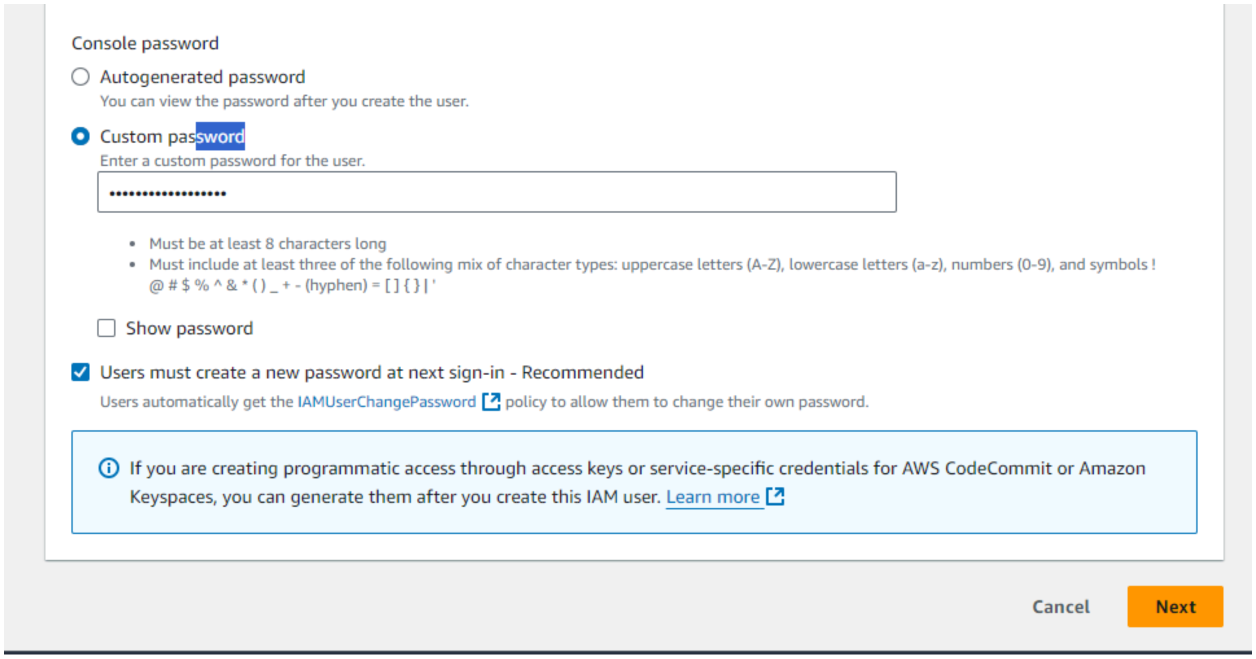
( ) Specify a user in Identity Center - Recommended

We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

(●) I want to create an IAM user

We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

Step 2: IAM user and access key creation



Console password

( ) Autogenerated password

You can view the password after you create the user.

(●) Custom password

Enter a custom password for the user.

••••••••••••••••••

- Must be at least 8 characters long
- Must include at least three of the following mix of character types: uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), and symbols ! @ # $ % ^ & * ( ) _ + - (hyphen) = [ ] { } | '

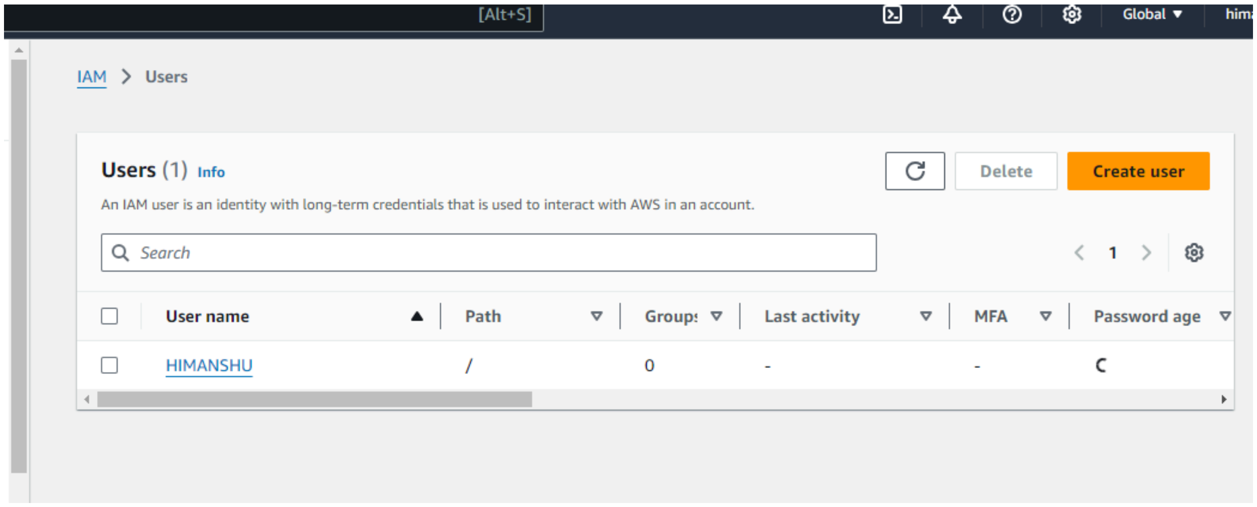
[ ] Show password

[✓] Users must create a new password at next sign-in - Recommended

Users automatically get the IAMUserChangePassword policy to allow them to change their own password.

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

NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**



[Alt+S] Global ▼ him

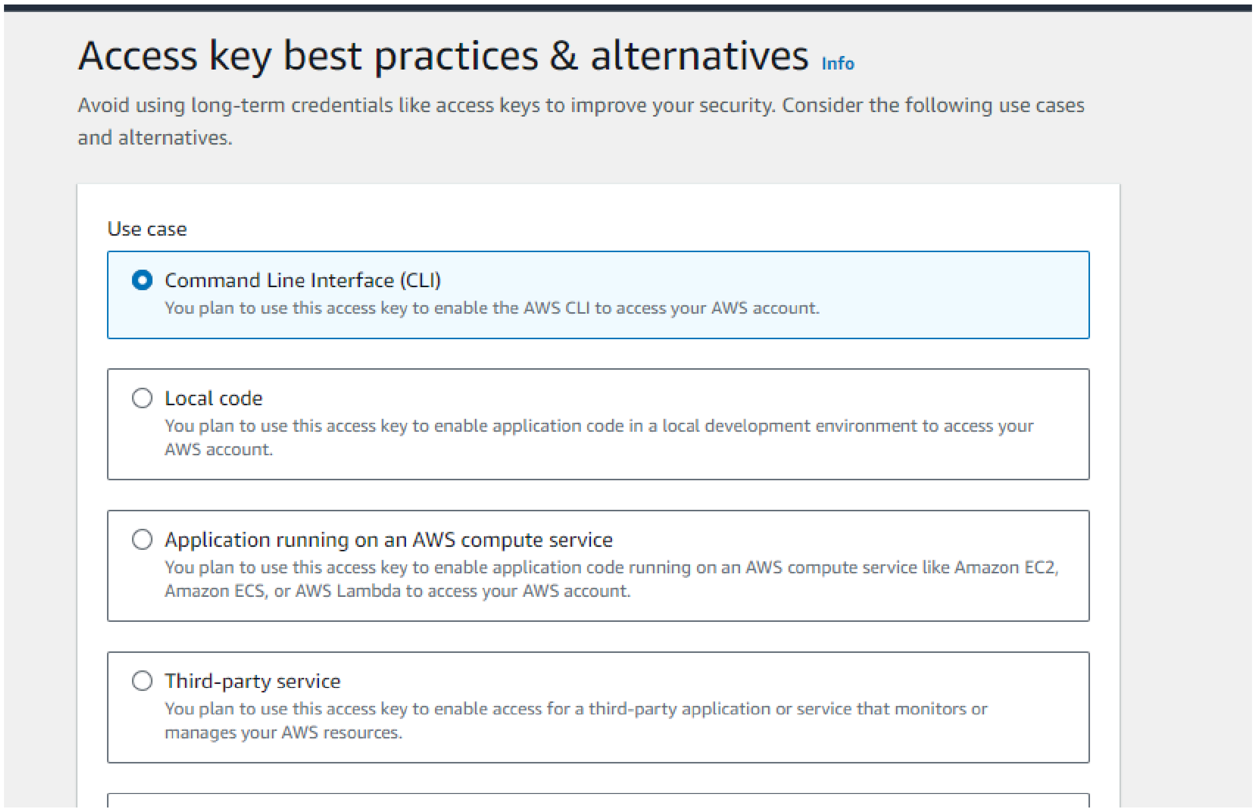
IAM > Users

Users (1) Info

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 activity	MFA	Password age
HIMANSHU	/	0	-	-	C

Create a new Access key inside the IAM User by navigating to Security section. This access key would be asked while trying to run 'aws configure'.



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

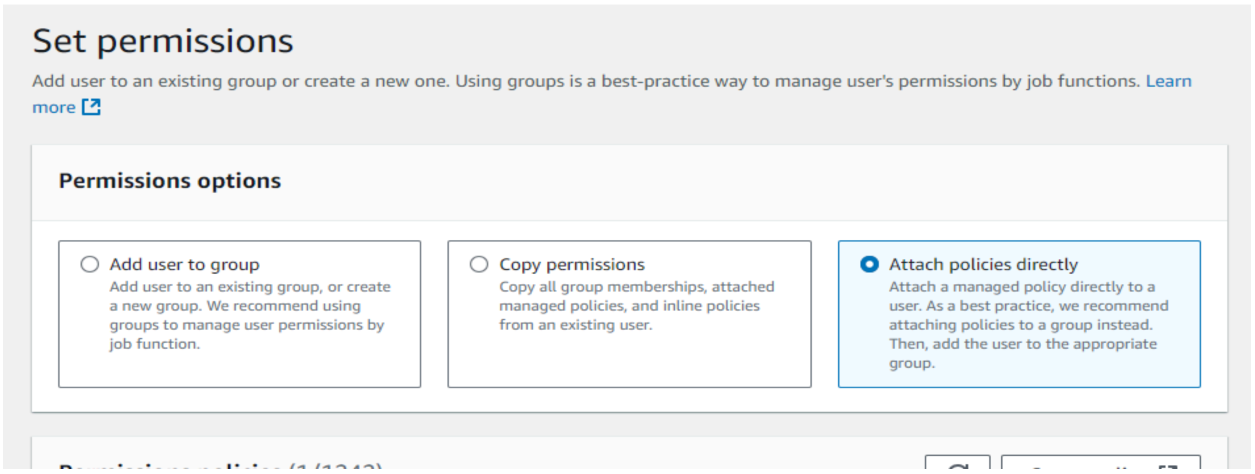
[✓ Command Line Interface (CLI)]
You plan to use this access key to enable the AWS CLI to access your AWS account.

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.

NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**

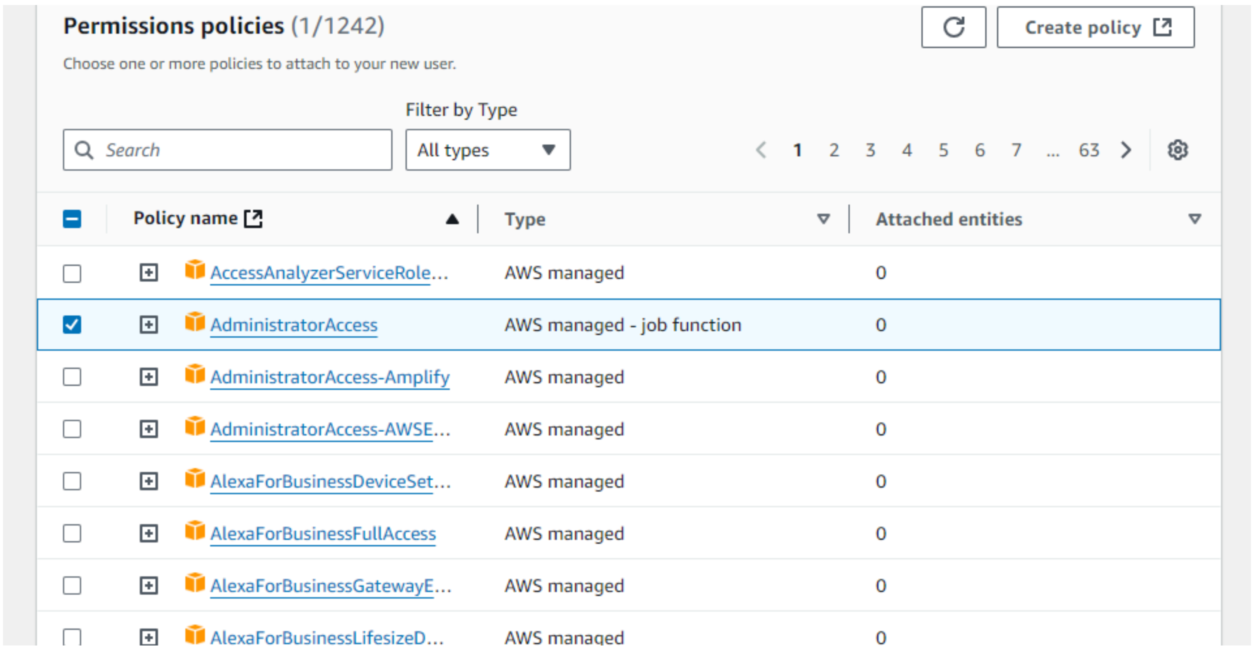


Set permissions

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 (1/1242)

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

Filter by Type: All types

	Policy name	Type	Attached entities
☐	AccessAnalyzerServiceRole...	AWS managed	0
✓	AdministratorAccess	AWS managed - job function	0
☐	AdministratorAccess-Amplify	AWS managed	0
☐	AdministratorAccess-AWSE...	AWS managed	0
☐	AlexaForBusinessDeviceSet...	AWS managed	0
☐	AlexaForBusinessFullAccess	AWS managed	0
☐	AlexaForBusinessGatewayE...	AWS managed	0
☐	AlexaForBusinessLifesizeD...	AWS managed	0

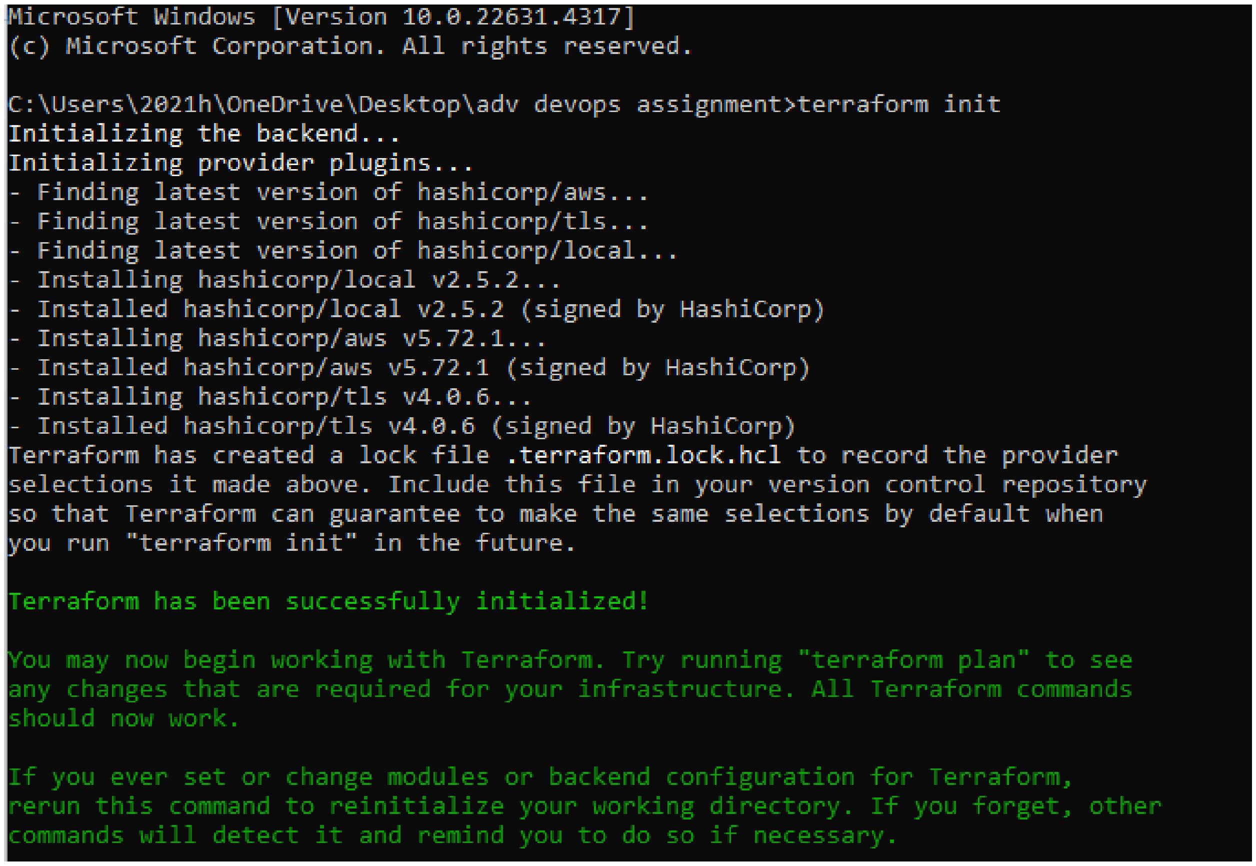
Step 3: Terraform usage

Now, we are supposed to create a new Directory. I named it ADV DEVOPS CASE STUDY

Create 3 files in it …

1. main.tf
2. Index.html
3. error.html

Now, initialize terraform in the directory by running terraform init command



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

C:\Users\2021h\OneDrive\Desktop\adv devops assignment>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Finding latest version of hashicorp/tls...
- Finding latest version of hashicorp/local...
- Installing hashicorp/local v2.5.2...
- Installed hashicorp/local v2.5.2 (signed by HashiCorp)
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (signed by HashiCorp)
- Installing hashicorp/tls v4.0.6...
- Installed hashicorp/tls v4.0.6 (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.
```

Write this in index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Welcome to My Static Website</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This is a sample static website hosted on S3.</p>
</body>
</html>
```

Write error.html like this

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Error Page</title>
</head>
<body>
  <h1>Oops!</h1>
  <p>Sorry, the page you are looking for does not exist.</p>
</body>
</html>
```

Now, modify the code in main.tf like this.
This code contains

1. **IAM user connection using access key id**
2. **EC2 instance and bucket configuration**
3. **Code snippets for connecting index.html and error.html**
4. **Code snippets for creating a separate public and private key**

```
# 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
```

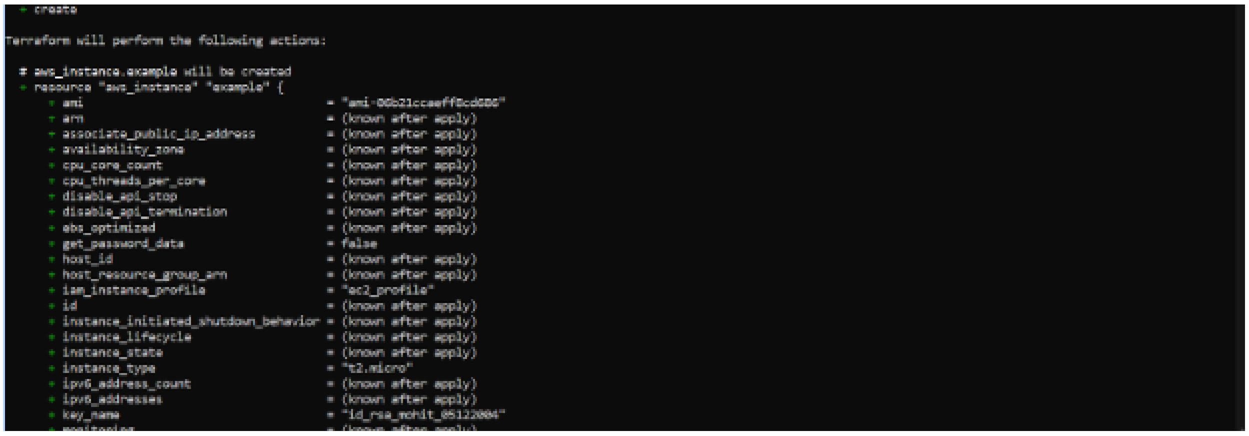
NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**

```
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_himanshu_05122004"
  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_qwe"  # 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 = "advance devops"
  }
}

# S3 Bucket
resource "aws_s3_bucket" "website_bucket" {
  bucket = "himanshu18102003advdevops"
}

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 = "index.html"
}

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

NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**

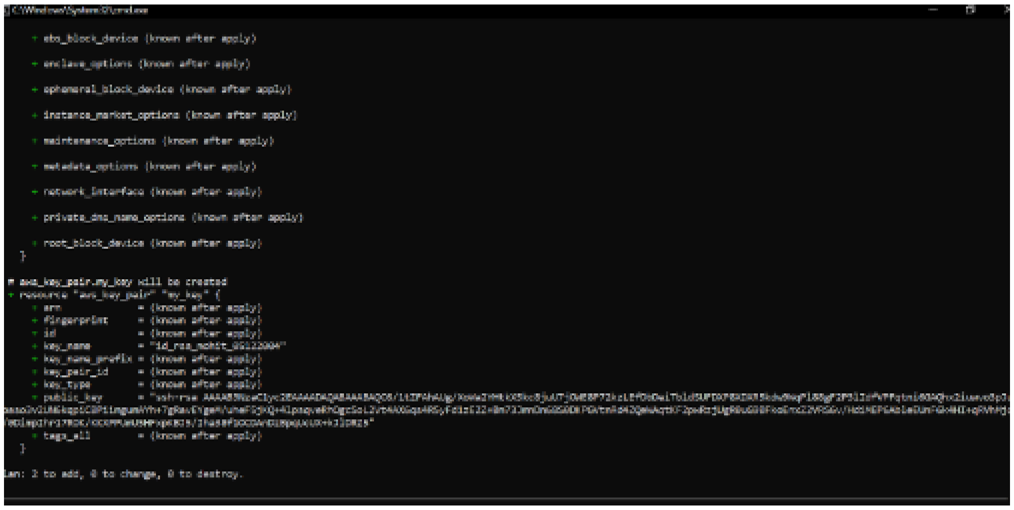
So, after commenting out the **acl attribute** in bucket and both html files, terraform plan and apply commands started working correctly

Terraform plan output



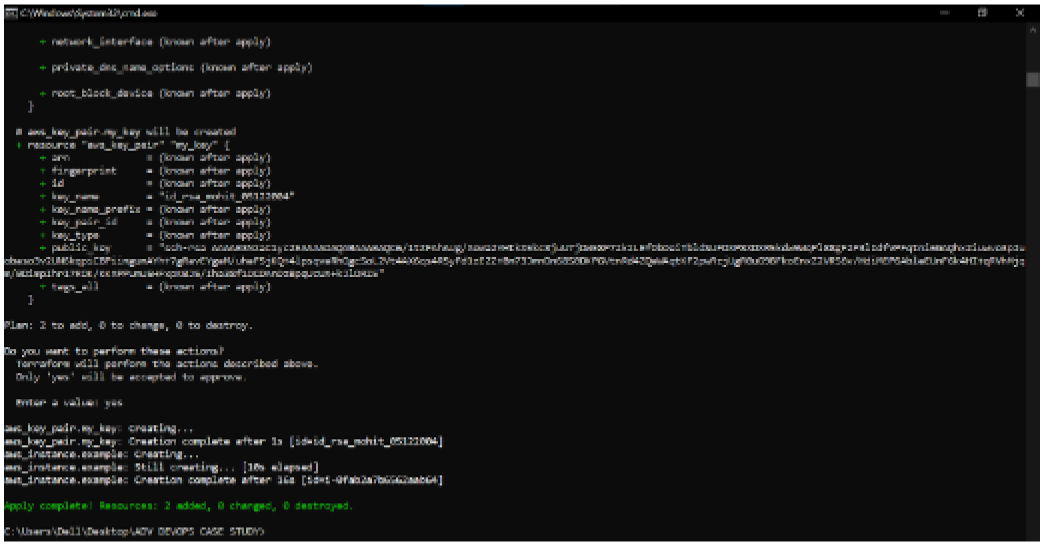
```
  + create

Terraform will perform the following actions:

  # aws_instance.example will be created
  + resource "aws_instance" "example" {
      + ami                                  = "ami-00b21ccaeff0cd686"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = "ec2_profile"
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = "id_rsa_mohit_05122004"
      + monitoring                           = (known after apply)
```

Terraform apply output

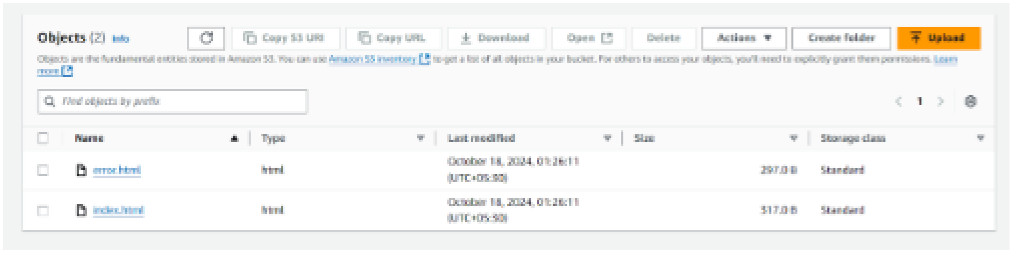


```
C:\Windows\System32\cmd.exe

      + ebs_block_device (known after apply)

      + enclave_options (known after apply)

      + ephemeral_block_device (known after apply)

      + instance_market_options (known after apply)

      + maintenance_options (known after apply)

      + metadata_options (known after apply)

      + network_interface (known after apply)

      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

  # aws_key_pair.my_key will be created
  + resource "aws_key_pair" "my_key" {
      + arn             = (known after apply)
      + fingerprint     = (known after apply)
      + id              = (known after apply)
      + key_name        = "id_rsa_mohit_05122004"
      + key_name_prefix = (known after apply)
      + key_pair_id     = (known after apply)
      + key_type        = (known after apply)
      + public_key      = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQ[illegible]"
      + tags_all        = (known after apply)
    }

lan: 2 to add, 0 to change, 0 to destroy.
```

NAME :HIMANSHU NAIK ROLL NO:63 DIV:D15C



```
C:\Windows\System32\cmd.exe
      + network_interface (known after apply)

      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

  # aws_key_pair.my_key will be created
  + resource "aws_key_pair" "my_key" {
      + arn             = (known after apply)
      + fingerprint     = (known after apply)
      + id              = (known after apply)
      + key_name        = "id_rsa_mohit_05122004"
      + key_name_prefix = (known after apply)
      + key_pair_id     = (known after apply)
      + key_type        = (known after apply)
      + public_key      = "ssh-rsa [illegible]"
      + tags_all        = (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_key_pair.my_key: Creating...
aws_key_pair.my_key: Creation complete after 1s [id=id_rsa_mohit_05122004]
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 16s [id=i-0fab2a7b6562aab64]

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

C:\Users\Dell\Desktop\ADV DEVOPS CASE STUDY>
```



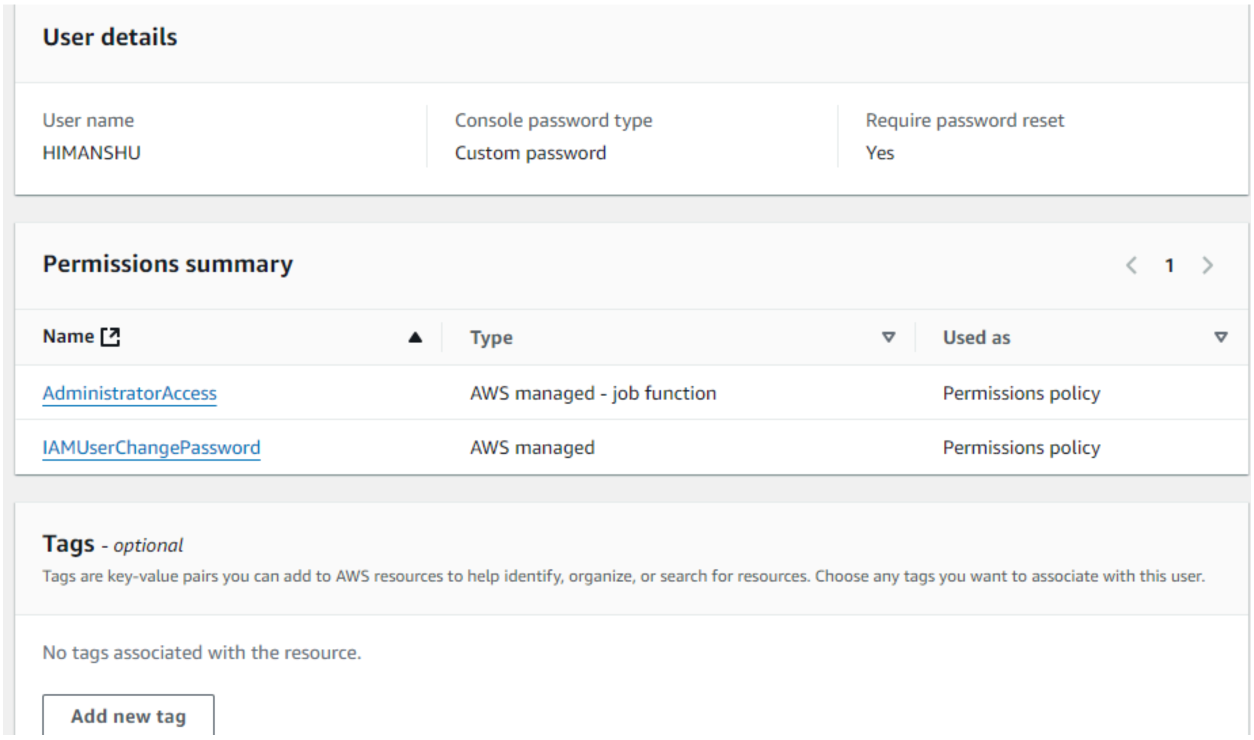
Objects (2) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
error.html	html	October 18, 2024, 01:26:11 (UTC+05:30)	297.0 B	Standard
index.html	html	October 18, 2024, 01:26:11 (UTC+05:30)	517.0 B	Standard

The ec2 instance has also been created ..
Now, we will use it to interact with our S3 bucket

NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**



User details

User name	Console password type	Require password reset
HIMANSHU	Custom password	Yes

Permissions summary

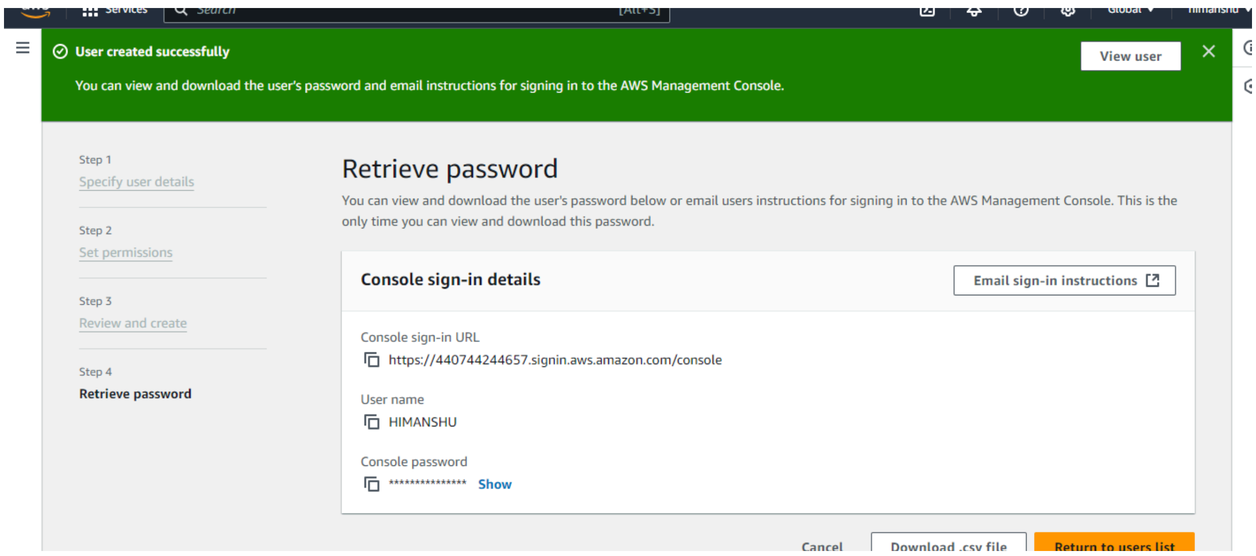
Name	Type	Used as
AdministratorAccess	AWS managed - job function	Permissions policy
IAMUserChangePassword	AWS managed	Permissions policy

Tags - *optional*

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.

Add new tag



User created successfully

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

Step 1 Specify user details
Step 2 Set permissions
Step 3 Review and create
Step 4 Retrieve password

Retrieve password

You can view and download the user's password below or email users instructions for signing in to the AWS Management Console. This is the only time you can view and download this password.

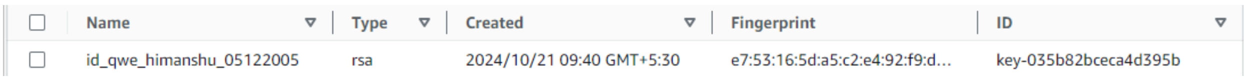
Console sign-in details

Console sign-in URL
https://440744244657.signin.aws.amazon.com/console

User name
HIMANSHU

Console password
*************** Show

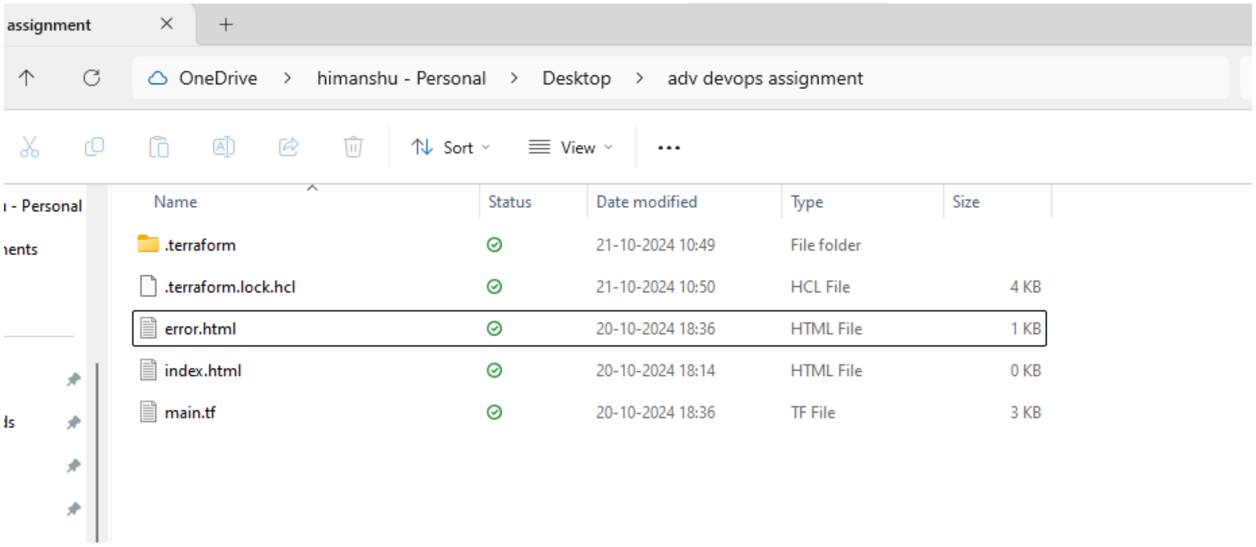
In our main.tf code, we have mentioned two code blocks where a public and a private key is to be created. So that accessing our EC2 instance should not be problematic
Public key "id_qwe_himanshu_05122004" is shown to have been created in AWS key pair section



Name	Type	Created	Fingerprint	ID
id_qwe_himanshu_05122005	rsa	2024/10/21 09:40 GMT+5:30	e7:53:16:5d:a5:c2:e4:92:f9:d...	key-035b82bceca4d395b

vate key "id_rsa" is created in the same directory as our main.tf file

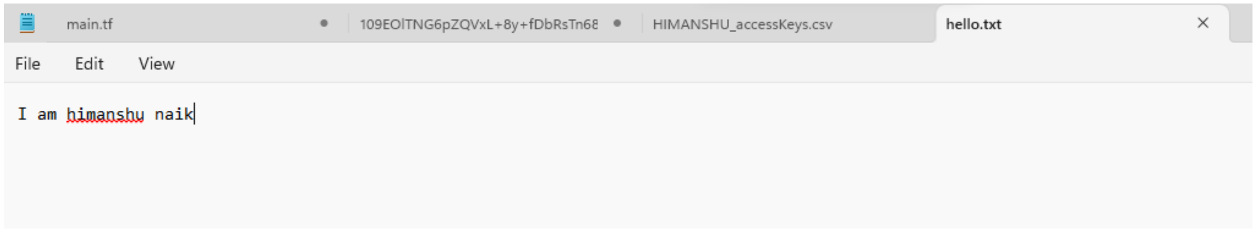
NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**



assignment > OneDrive > himanshu - Personal > Desktop > adv devops assignment

Name	Status	Date modified	Type	Size
.terraform	✓	21-10-2024 10:49	File folder	
.terraform.lock.hcl	✓	21-10-2024 10:50	HCL File	4 KB
error.html	✓	20-10-2024 18:36	HTML File	1 KB
index.html	✓	20-10-2024 18:14	HTML File	0 KB
main.tf	✓	20-10-2024 18:36	TF File	3 KB

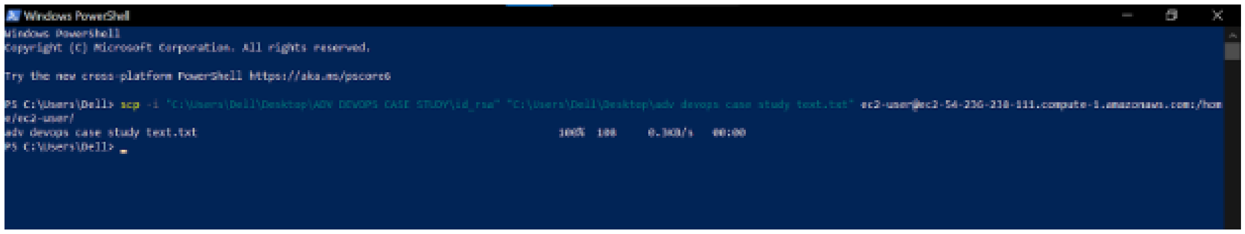
Use this command to remotely login onto your ec2 instance
ssh -i "<path to your main.tf file/ has to be the same path>\id_rsa" ec2-user@ec2-54-236-238-111.compute-1.amazonaws.com
MAKE A TEXT FILE



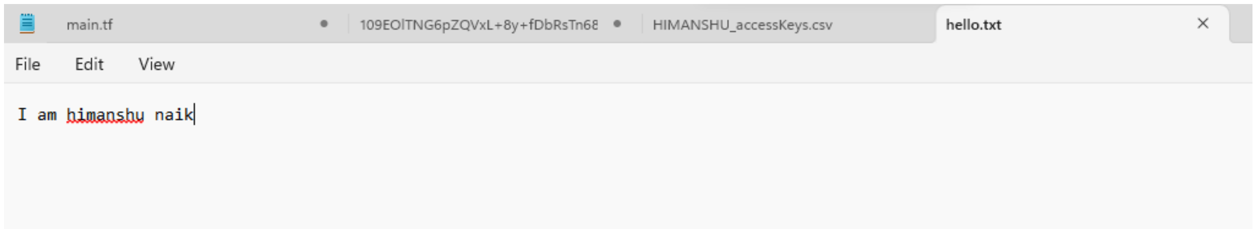
main.tf | 109EOlTNG6pZQVxL+8y+fDbRsTn68 | HIMANSHU_accessKeys.csv | hello.txt

```
I am himanshu naik
```

Open new terminal and run this command to upload our newly made text file on our instance
scp -i "path_to_your_key.pem" "path_to_local_file" ec2-user@your-ec2-public-ip:/home/ec2-user/.
This command is used to upload a random text file onto our instance



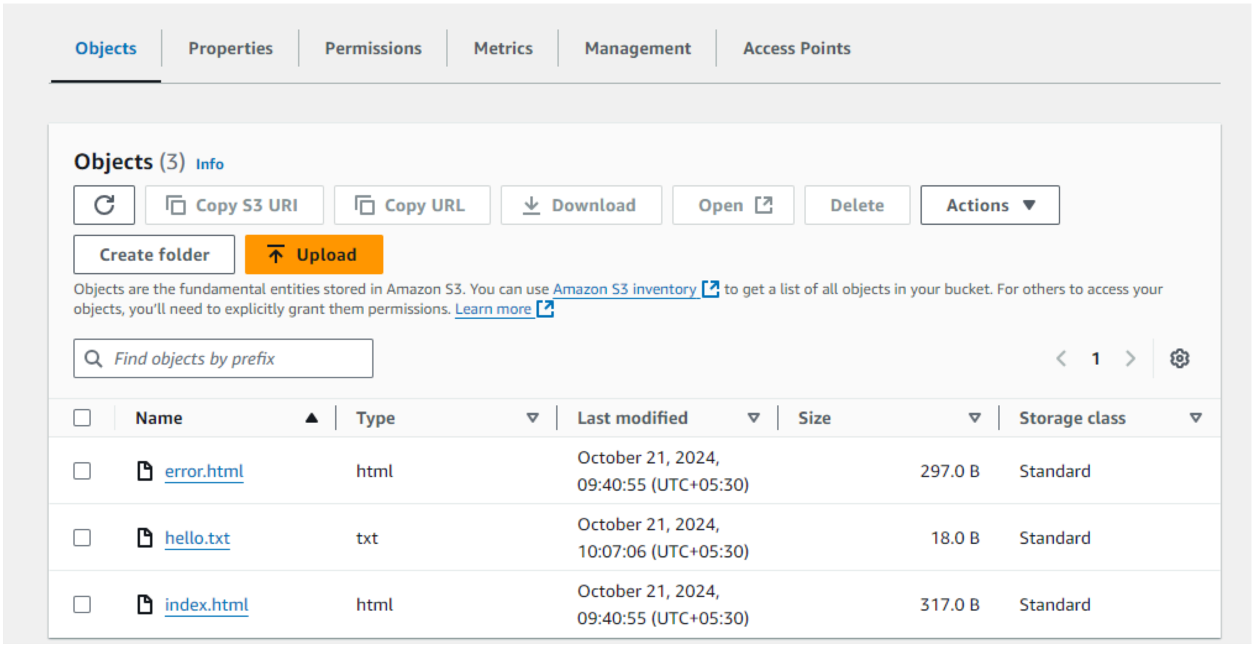
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Dell> scp -i "C:\Users\Dell\Desktop\ADV DEVOPS CASE STUDY\id_rsa" "C:\Users\Dell\Desktop\adv devops case study text.txt" ec2-user@ec2-54-236-238-111.compute-1.amazonaws.com:/home/ec2-user/
adv devops case study text.txt                                  100%  100     0.3KB/s   00:00
PS C:\Users\Dell> _
```



main.tf | 109EOlTNG6pZQVxL+8y+fDbRsTn68 | HIMANSHU_accessKeys.csv | hello.txt

```
I am himanshu naik
```

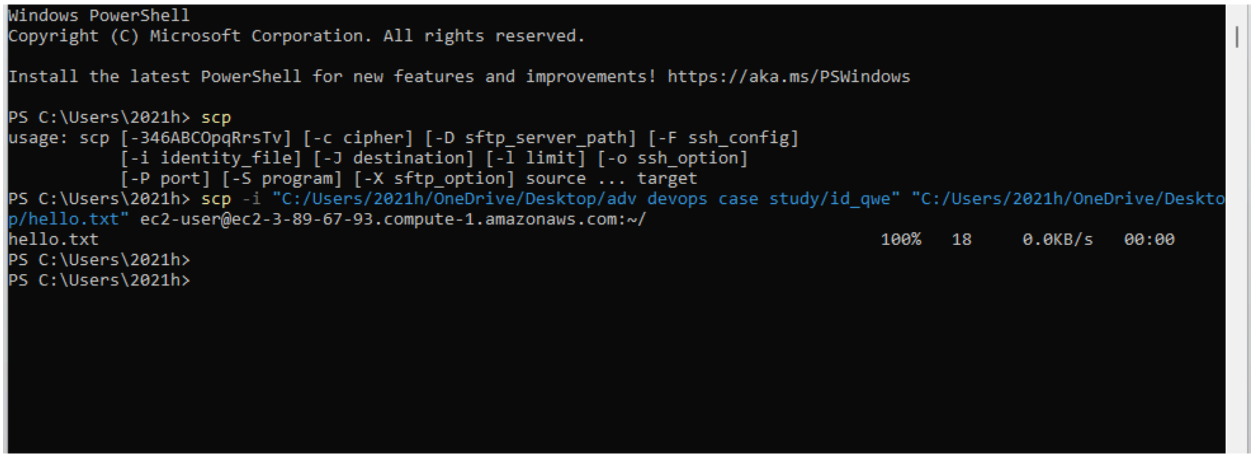
NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
error.html	html	October 21, 2024, 09:40:55 (UTC+05:30)	297.0 B	Standard
hello.txt	txt	October 21, 2024, 10:07:06 (UTC+05:30)	18.0 B	Standard
index.html	html	October 21, 2024, 09:40:55 (UTC+05:30)	317.0 B	Standard

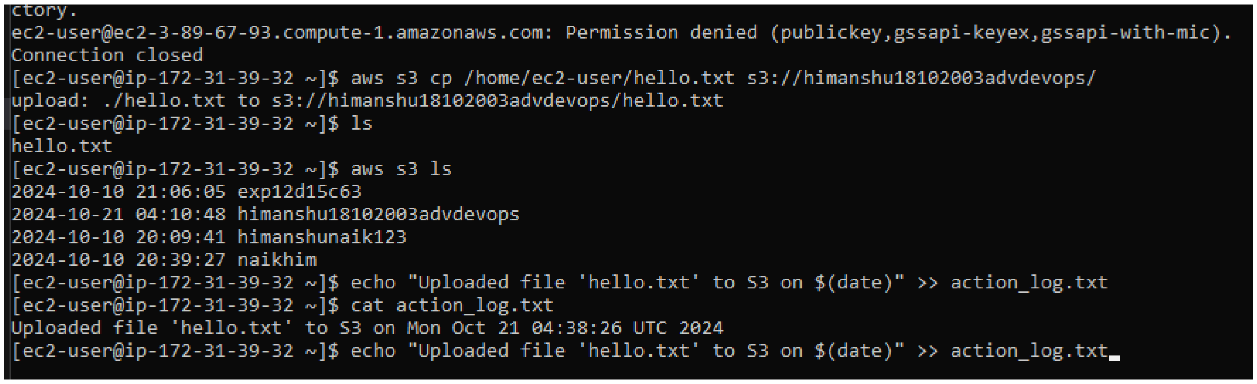
Then upload that file onto our bucket from the original terminal and list them just for confirmation
Command to upload the file onto our bucket: **aws s3 cp /home/ec2-user/you.txt s3://your-s3-bucket-name/**



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\2021h> scp
usage: scp [-346ABCOpqRrsTv] [-c cipher] [-D sftp_server_path] [-F ssh_config]
           [-i identity_file] [-J destination] [-l limit] [-o ssh_option]
           [-P port] [-S program] [-X sftp_option] source ... target
PS C:\Users\2021h> scp -i "C:/Users/2021h/OneDrive/Desktop/adv devops case study/id_qwe" "C:/Users/2021h/OneDrive/Deskto
p/hello.txt" ec2-user@ec2-3-89-67-93.compute-1.amazonaws.com:~/
hello.txt                                                                     100%   18     0.0KB/s   00:00
PS C:\Users\2021h>
PS C:\Users\2021h>
```

Use this command in the original terminal to log the actions of the S3 bucket.
echo "Uploaded file 'filename.txt' to S3 on $(date)" >> action_log.txt

NAME :HIMANSHU NAIK **ROLL NO:63** **DIV:D15C**


```
ctory.
ec2-user@ec2-3-89-67-93.compute-1.amazonaws.com: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
Connection closed
[ec2-user@ip-172-31-39-32 ~]$ aws s3 cp /home/ec2-user/hello.txt s3://himanshu18102003advdevops/
upload: ./hello.txt to s3://himanshu18102003advdevops/hello.txt
[ec2-user@ip-172-31-39-32 ~]$ ls
hello.txt
[ec2-user@ip-172-31-39-32 ~]$ aws s3 ls
2024-10-10 21:06:05 exp12d15c63
2024-10-21 04:10:48 himanshu18102003advdevops
2024-10-10 20:09:41 himanshunaik123
2024-10-10 20:39:27 naikhim
[ec2-user@ip-172-31-39-32 ~]$ echo "Uploaded file 'hello.txt' to S3 on $(date)" >> action_log.txt
[ec2-user@ip-172-31-39-32 ~]$ cat action_log.txt
Uploaded file 'hello.txt' to S3 on Mon Oct 21 04:38:26 UTC 2024
[ec2-user@ip-172-31-39-32 ~]$ echo "Uploaded file 'hello.txt' to S3 on $(date)" >> action_log.txt
```

Conclusion: Thus, by creating an IAM user, providing an access key to it , installing AWS CLI, using terraform commands in our project to create ec2 instance and s3 bucket, we were successfully able to ensure interaction between our ec2 instance and s3 bucket. We tested this by uploading a text file over to our s3 bucket by using our ec2 instance. This experiment ensured that terraform can be used to change the configuration of our resources and create and access the proper ones using the AWS CLI