

# Lesson 11 Demo 01

## Implementing Workflow for Managing Configurations in Terraform

**Objective:** To implement a workflow for managing AWS infrastructure configurations in Terraform efficiently and effectively

**Tools required:** Terraform, AWS, and Visual Studio Code

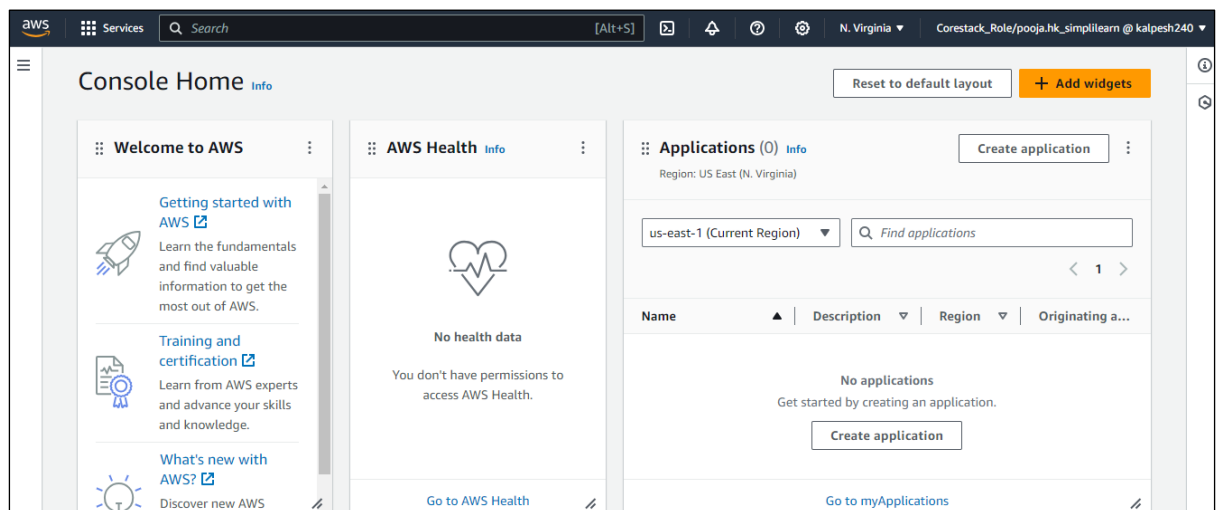
**Prerequisites:** None

Steps to be followed:

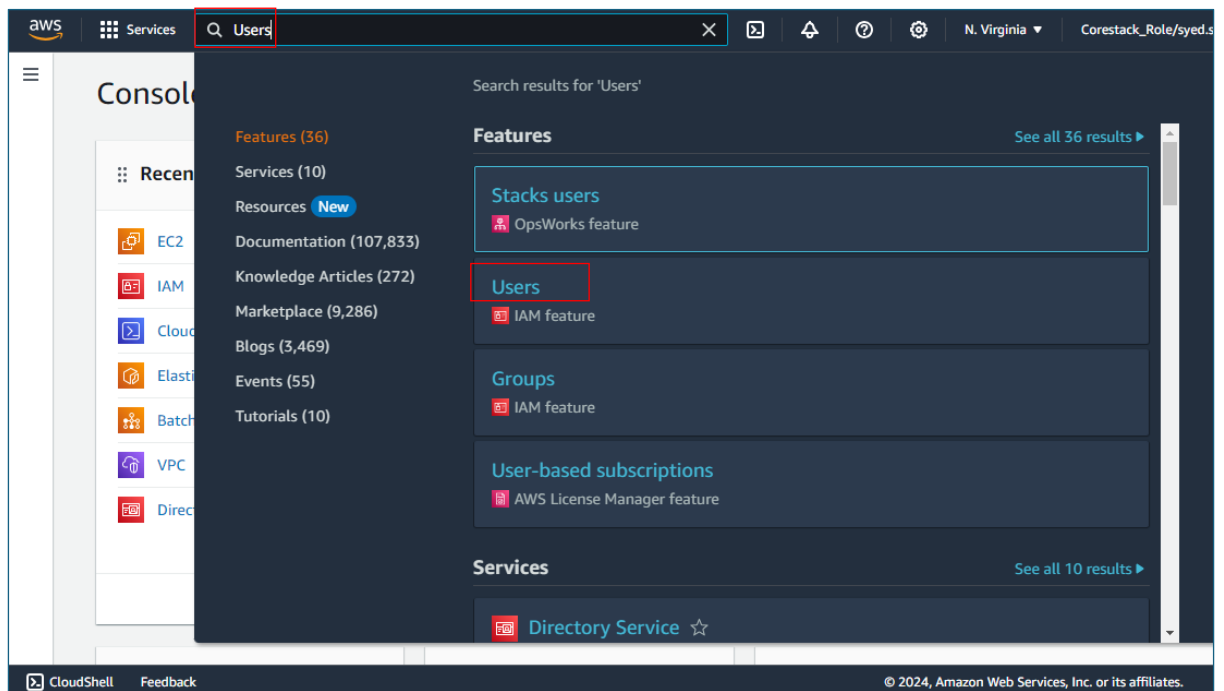
1. Create an IAM user
2. Create access and secret key
3. Implement a workflow for managing configurations in Terraform

### Step 1: Create an IAM user

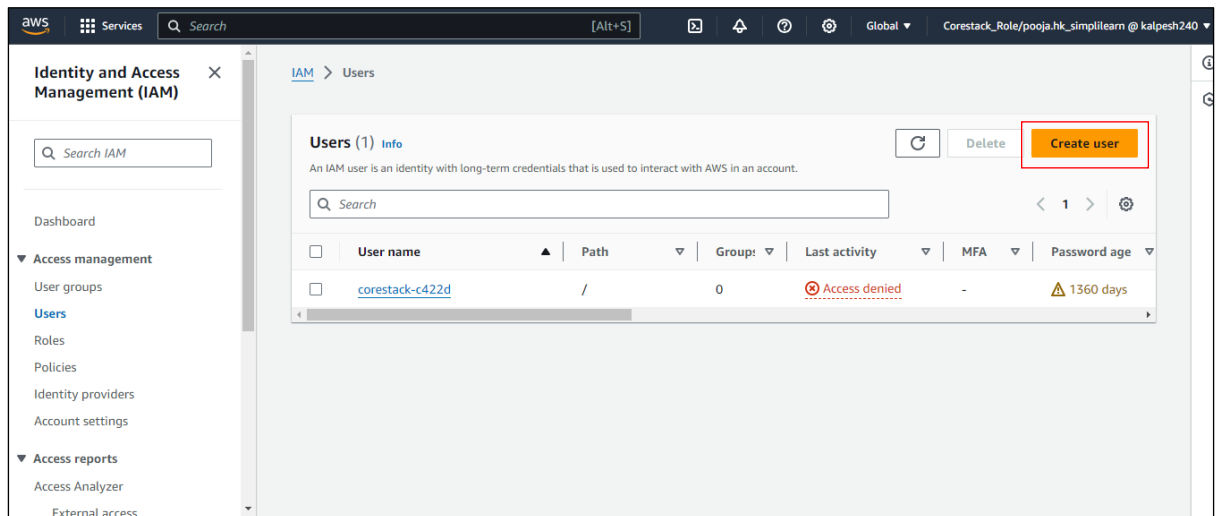
1.1 Navigate to the AWS console as shown in the screenshot below:



1.2 Search for and click on **Users** as shown in the screenshot below:



1.3 Click on **Create user** as shown in the screenshot below:



1.4 Enter a desired name in the **User name** field as shown in the screenshot below:

The screenshot shows the AWS IAM console interface for creating a new user. The breadcrumb navigation indicates the path: IAM > Users > Create user. The left-hand navigation pane lists three steps: Step 1 (Specify user details), Step 2 (Set permissions), and Step 3 (Review and create). Step 1 is currently selected. The main content area is titled 'Specify user details' and contains a 'User details' section. Within this section, the 'User name' field is highlighted with a red rectangular box and contains the text 'Myuser'. Below the field, a note states: 'The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ \_ - (hyphen)'. There is an unchecked checkbox for 'Provide user access to the AWS Management Console - optional'. A blue information box at the bottom of the section provides guidance on generating credentials for AWS CodeCommit or Amazon Keyspaces. At the bottom right of the form, there are 'Cancel' and 'Next' buttons, with 'Next' being highlighted in orange.

1.5 Click on the check box of the user access, select **Custom password**, and enter the password as shown in the screenshot below:

This screenshot shows the 'Review and create' step of the AWS IAM user creation process. The left-hand navigation pane now shows Step 3 as the active step, with Step 4 (Retrieve password) listed below it. The main content area displays the 'User name' field with 'Myuser' and the 'Provide user access to the AWS Management Console - optional' checkbox, which is now checked and highlighted with a red box. Below this, the 'Console password' section shows the 'Custom password' radio button selected and highlighted with a red box. The text 'Enter a custom password for the user.' is followed by a password input field containing eight asterisks. A list of password requirements is shown below the field: 'Must be at least 8 characters long' and '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) = [ ] { } |'. There is an unchecked 'Show password' checkbox. A checked checkbox at the bottom states 'Users must create a new password at next sign-in - Recommended'. A blue information box at the bottom provides guidance on generating credentials for AWS CodeCommit or Amazon Keyspaces. The 'Next' button remains highlighted in orange.

1.6 Untick the check box and then click on **Next** as shown in the screenshot below:

aws Services Search [Alt+S] Global Corestack\_Role/pooja.hk\_simplilearn @ kalpesh240

Retrieve password

☒ 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.

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](#)**

Cancel **Next**

1.7 Click on **Attach policies directly** as shown in the screenshot below:

aws Services Search [Alt+S] Global Corestack\_Role/pooja.hk\_simplilearn @ kalpesh240

Step 1  
[Specify user details](#)

Step 2  
**Set permissions**

Step 3  
Review and create

Step 4  
Retrieve password

### 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.

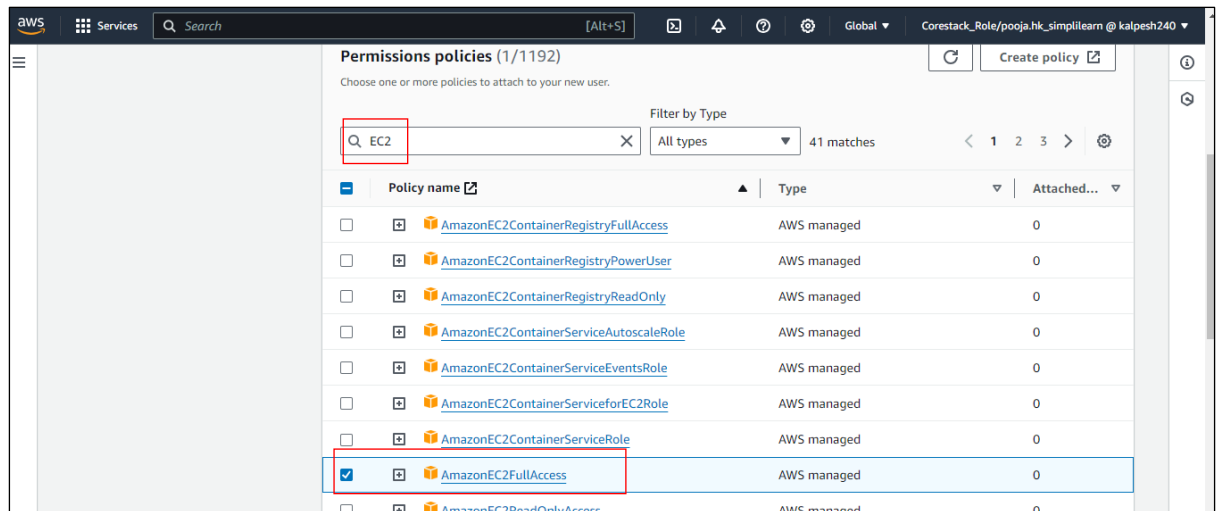
**Permissions policies (1192)**

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

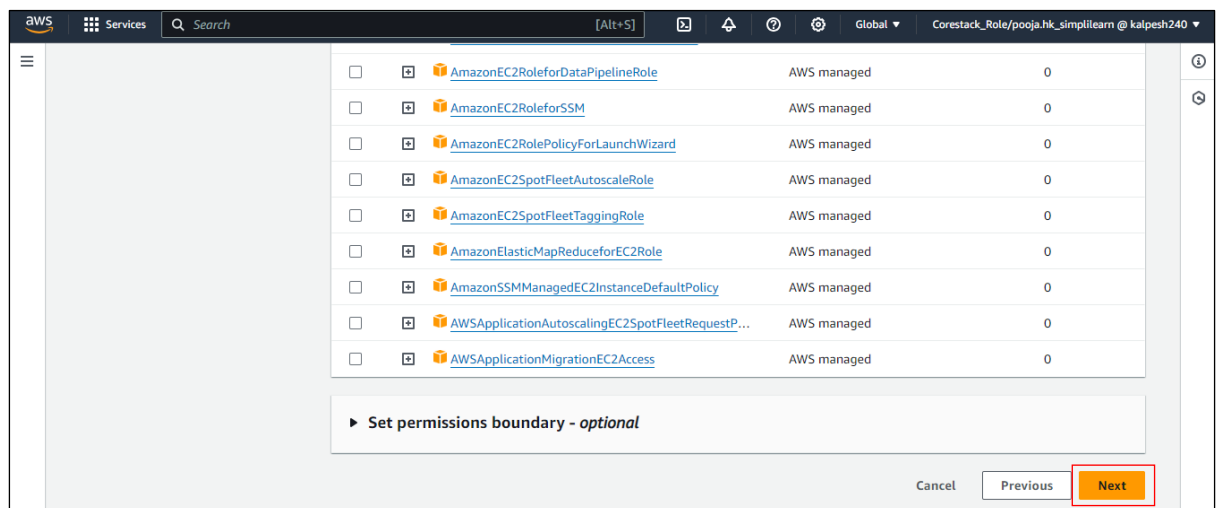
Search Filter by Type All types

< 1 2 3 4 5 6 7 ... 60 >

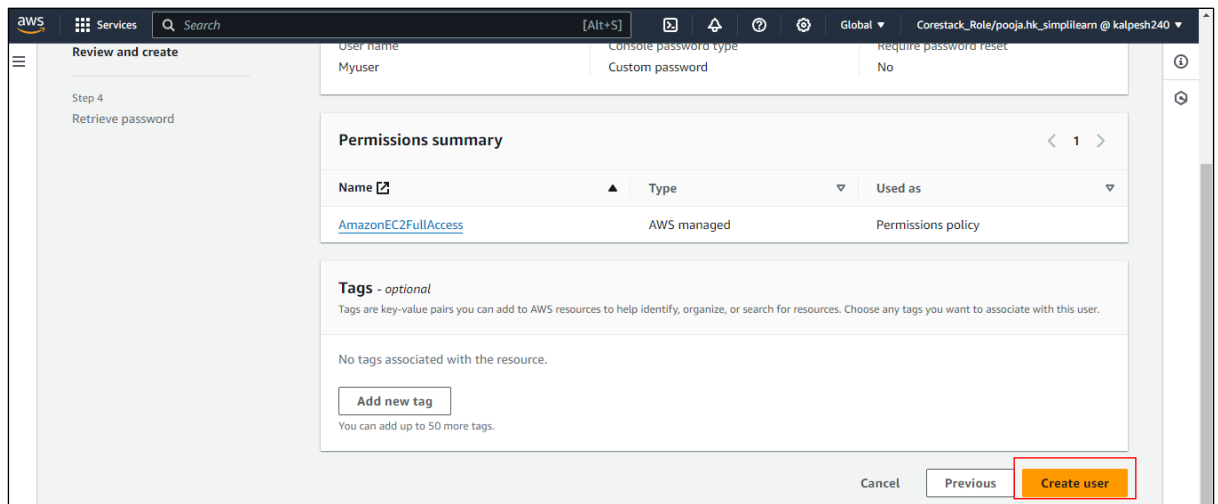
1.8 Search for EC2 in the **Permissions policies** section and select **AmazonEC2FullAccess** as shown in the screenshot below:



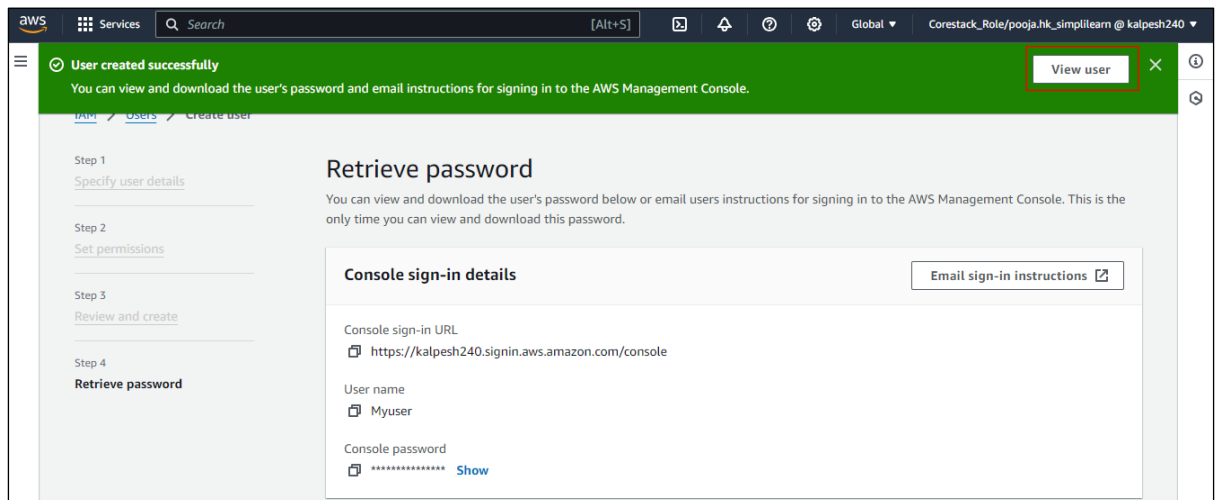
1.9 Scroll down and click on the **Next** button as shown in the screenshot below:



1.10 Click on **Create user** as shown in the screenshot below:

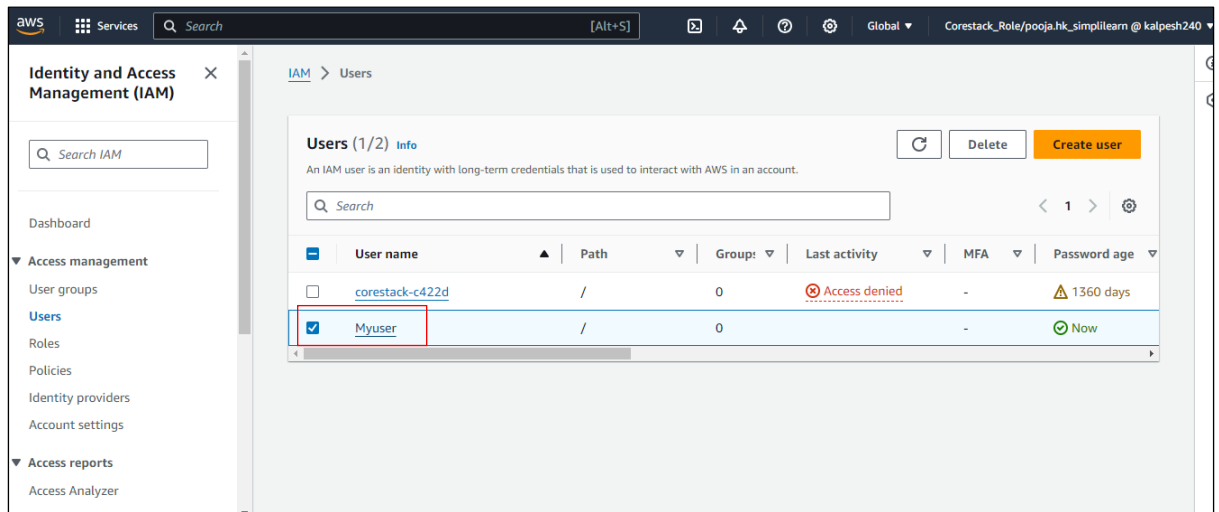


1.11 Click on **View user** as shown in the screenshot below:

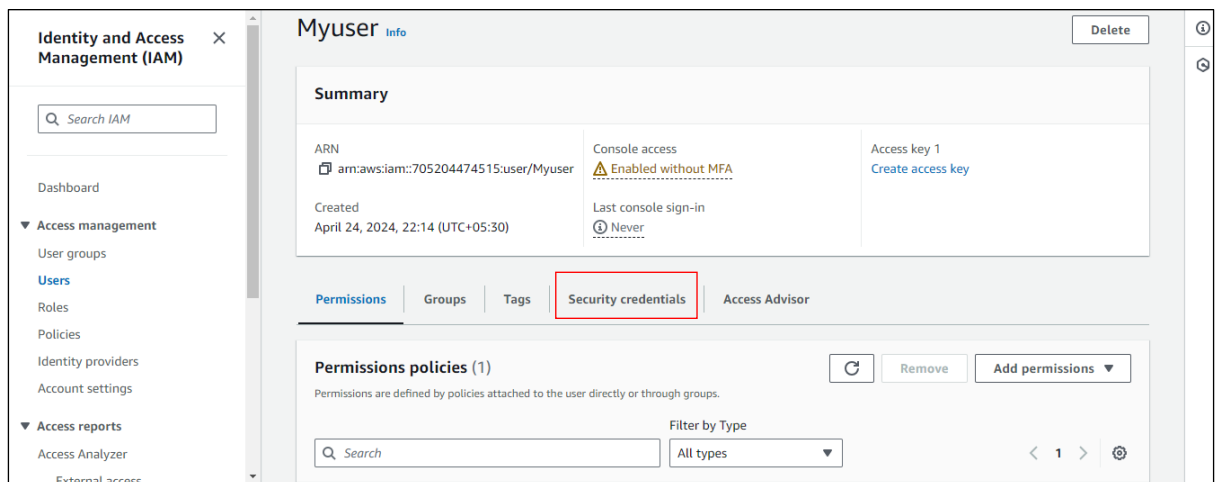


## Step 2: Create access and secret key

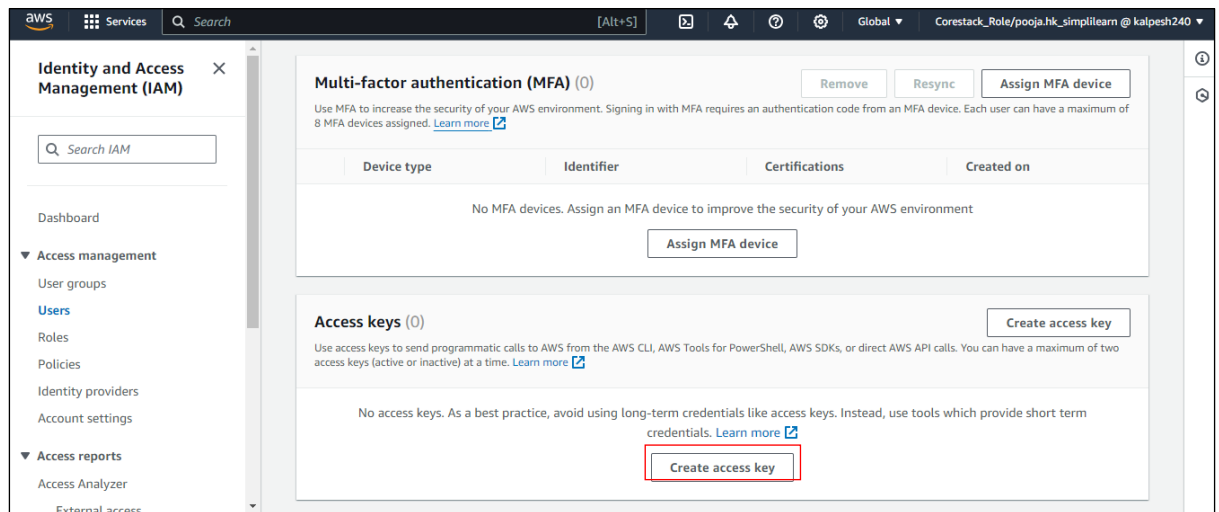
2.1 Click on the username that you created as shown in the screenshot below:



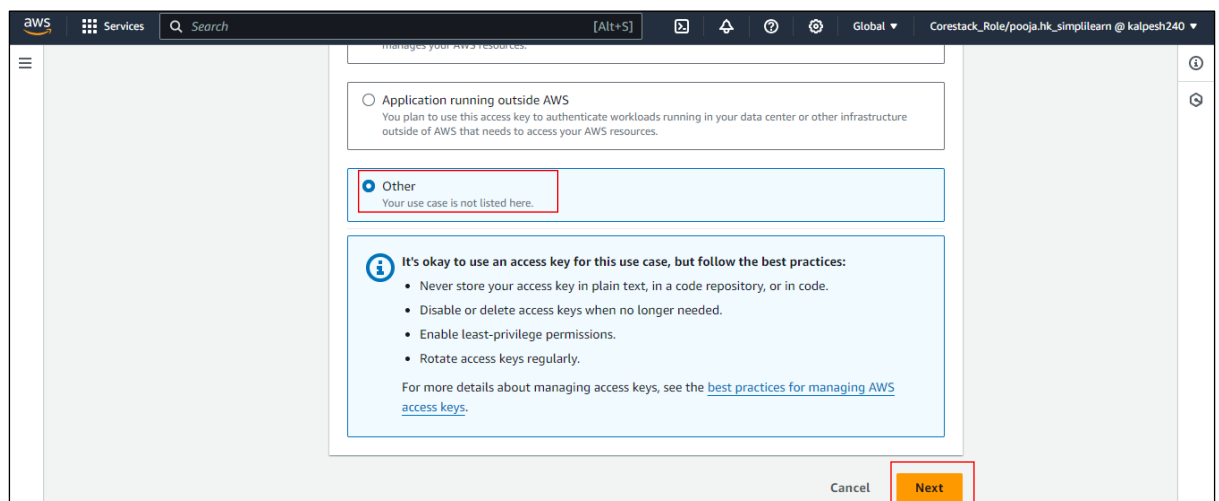
2.2 Click on **Security credentials** as shown in the screenshot below:



2.3 Scroll down to **Access keys** section and then click on **Create access key** as shown in the screenshot below:



2.4 Select **Other** from the listed options and then click on **Next** as shown in the screenshot below:





2.5 Enter a desired name in the **Description tag value** section and then click on **Create access key** as shown in the screenshot below:

The screenshot shows the AWS IAM console interface for creating an access key. The breadcrumb trail is IAM > Users > Myuser > Create access key. The left sidebar shows the progress: Step 1 (Access key best practices & alternatives), Step 2 - optional (Set description tag), and Step 3 (Retrieve access keys). The main content area is titled 'Set description tag - optional' with an info icon. Below the title, it states: 'The description for this access key will be attached to this user as a tag and shown alongside the access key.' A text input field labeled 'Description tag value' contains the text 'Mykey'. Below the field, a note says: 'Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.' Another note specifies: 'Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: \_ . : / = + - @'. At the bottom right, there are three buttons: 'Cancel', 'Previous', and 'Create access key' (highlighted with a red box).

2.6 Copy the **Access key** and **Secret access key** and save it in Notepad for the next steps as shown in the screenshot below:

The screenshot shows the AWS IAM console interface for retrieving access keys. A green banner at the top states: '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.' The breadcrumb trail is IAM > Users > Myuser > Retrieve access keys. The left sidebar shows the progress: Step 1 (Access key best practices & alternatives), Step 2 - optional (Set description tag), and Step 3 (Retrieve access keys). The main content area is titled 'Retrieve access keys' with an info icon. Below the title, it states: '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.' A table displays the access key details:

Access key	Secret access key
AKIAZIMLWKJXC7SHIZW	***** <a href="#">Show</a>

Below the table, there is a section titled 'Access key best practices' with a list of guidelines:

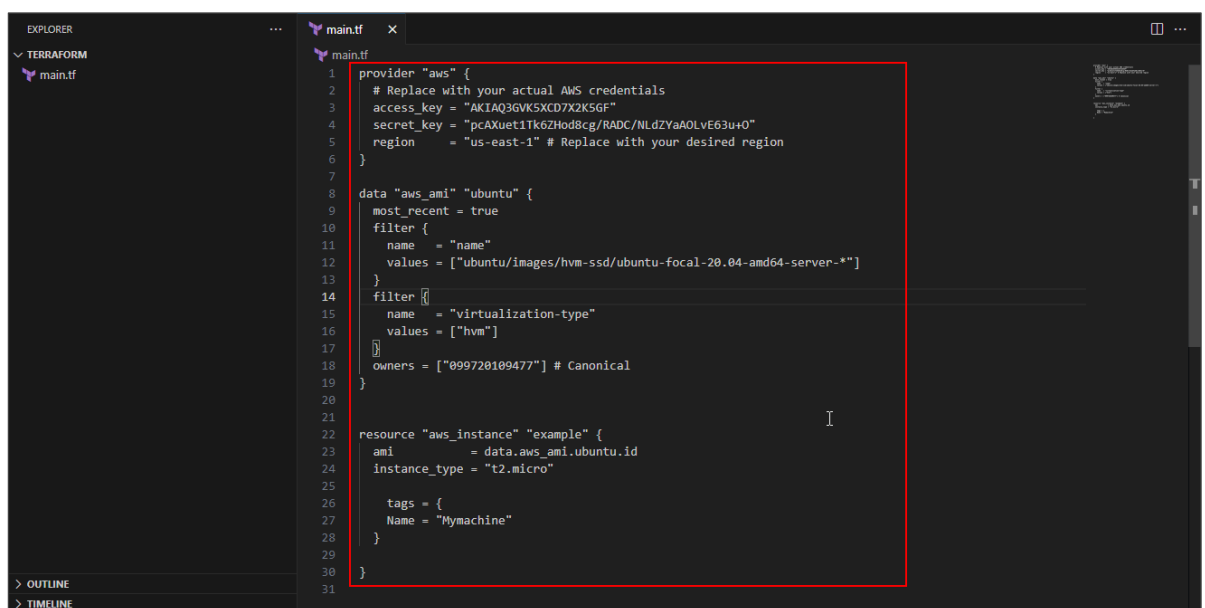
- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

**Note:** Save the Access key and Secret access key in notepad to use in next steps

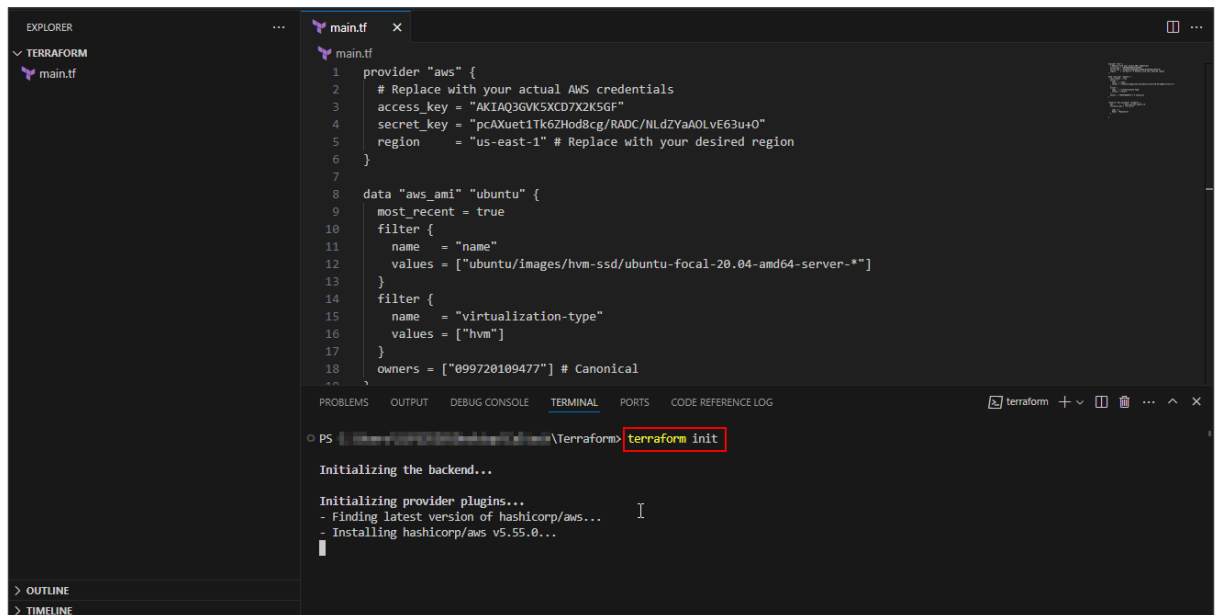
### Step 3: Implement a workflow for managing configurations in Terraform

- 3.1 Open Visual Studio Code, create a file named **main.tf**, and add the following configuration code as shown in the screenshot below:

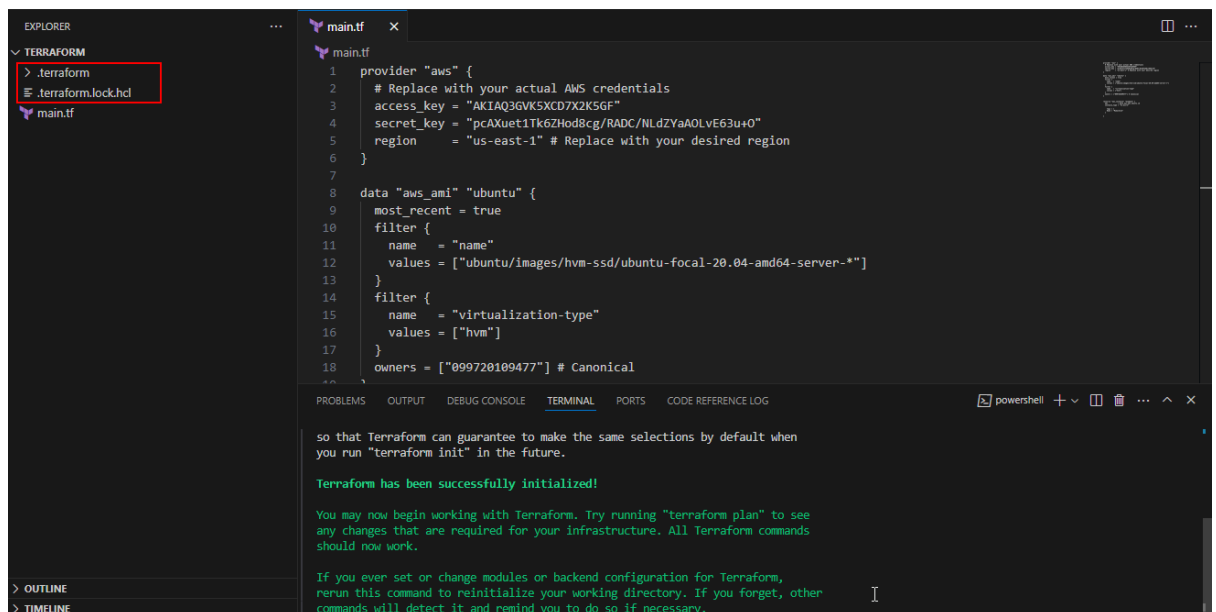
```
provider "aws" {  
  # Replace with your actual AWS credentials  
  access_key = "YOUR_ACCESS_KEY"  
  secret_key = "YOUR_SECRET_KEY"  
  region     = "us-east-1" # Replace with your desired region  
}  
  
data "aws_ami" "ubuntu" {  
  most_recent = true  
  filter {  
    name     = "name"  
    values   = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]  
  }  
  filter {  
    name     = "virtualization-type"  
    values   = ["hvm"]  
  }  
  owners = ["099720109477"] # Canonical  
}  
  
resource "aws_instance" "example" {  
  ami           = data.aws_ami.ubuntu.id  
  instance_type = "t2.micro"  
  tags = {  
    Name = "Mymachine"  
  }  
}
```



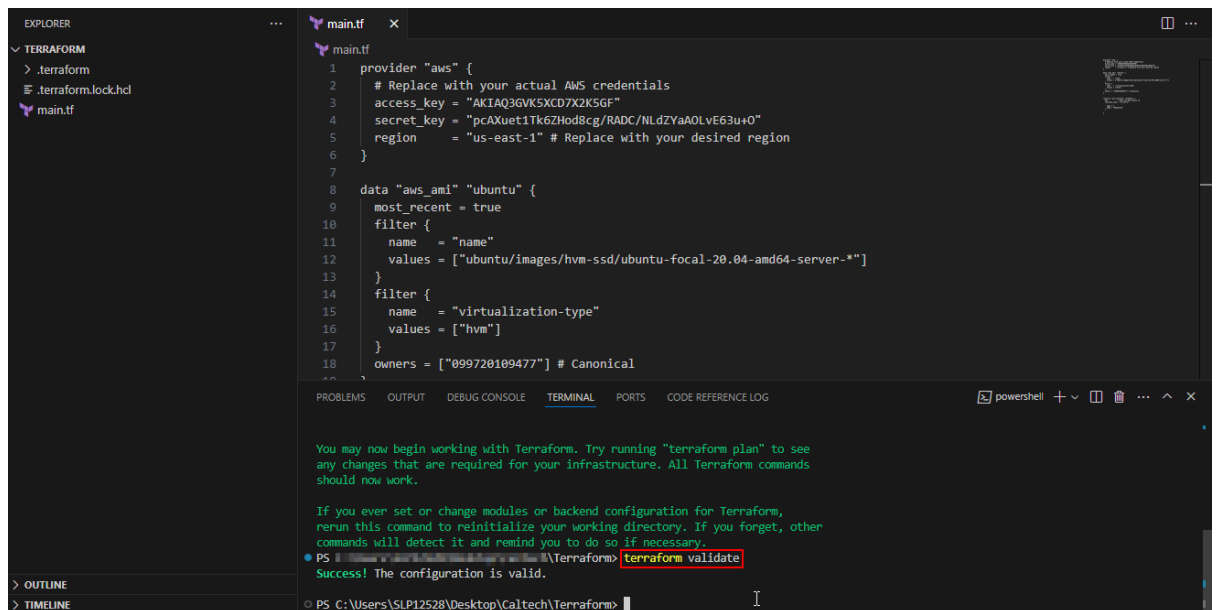
### 3.2 Initialize the configuration using the following command as shown in the screenshot below: **terraform init**



Once the Terraform has been initialized successfully, providers are downloaded as shown in the screenshot below:



3.3 Validate the configuration using the following command as shown in the screenshot below:  
**terraform validate**



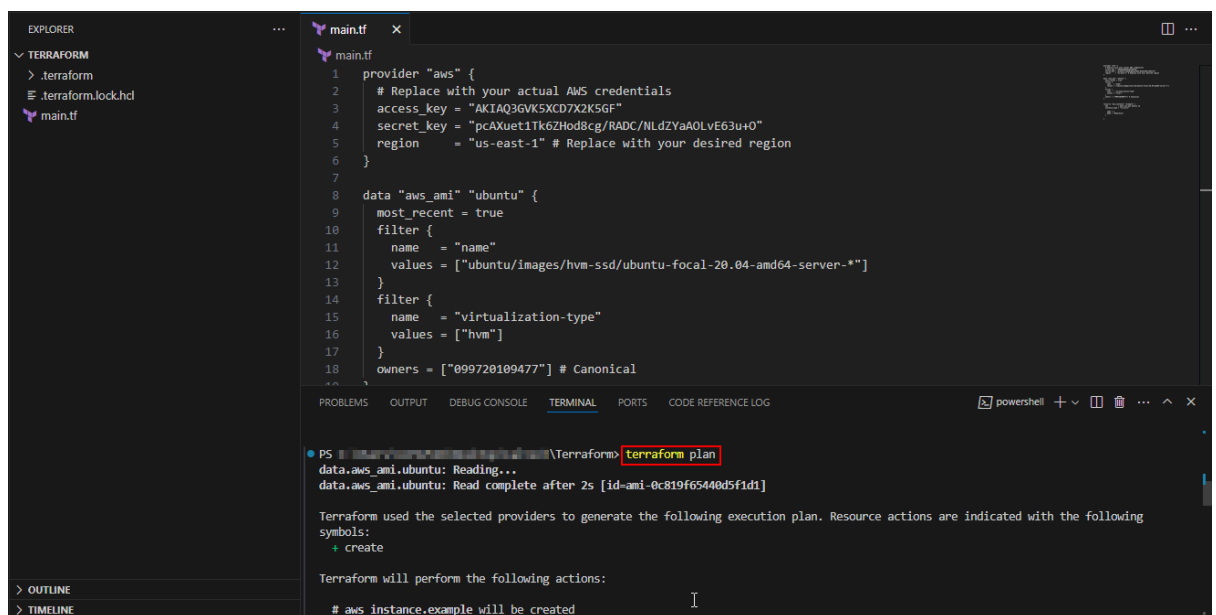
The screenshot shows a Visual Studio Code editor with a Terraform configuration file named `main.tf` open. The Explorer sidebar on the left shows the project structure: `TERRAFORM` > `.terraform` > `.terraform.lock.hcl` and `main.tf`. The `main.tf` file contains the following HCL code:

```
1 provider "aws" {
2   # Replace with your actual AWS credentials
3   access_key = "AKIAQ3GVK5XCD7X2K5GF"
4   secret_key = "pcAXuet1Tk6ZHod8cg/RADC/NLdZYaAOLvE63u+0"
5   region     = "us-east-1" # Replace with your desired region
6 }
7
8 data "aws_ami" "ubuntu" {
9   most_recent = true
10  filter {
11    name     = "name"
12    values   = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
13  }
14  filter {
15    name     = "virtualization-type"
16    values   = ["hvm"]
17  }
18  owners    = ["099720109477"] # Canonical
19 }
```

The terminal window at the bottom shows the output of the `terraform validate` command:

```
PS C:\Users\SLP12528\Desktop\Caltech\Terraform> terraform validate
Success! The configuration is valid.
```

3.4 Plan the changes using the following command as shown in the screenshot below:  
**terraform plan**



The screenshot shows the same Visual Studio Code editor with the `main.tf` file. The terminal window now shows the output of the `terraform plan` command:

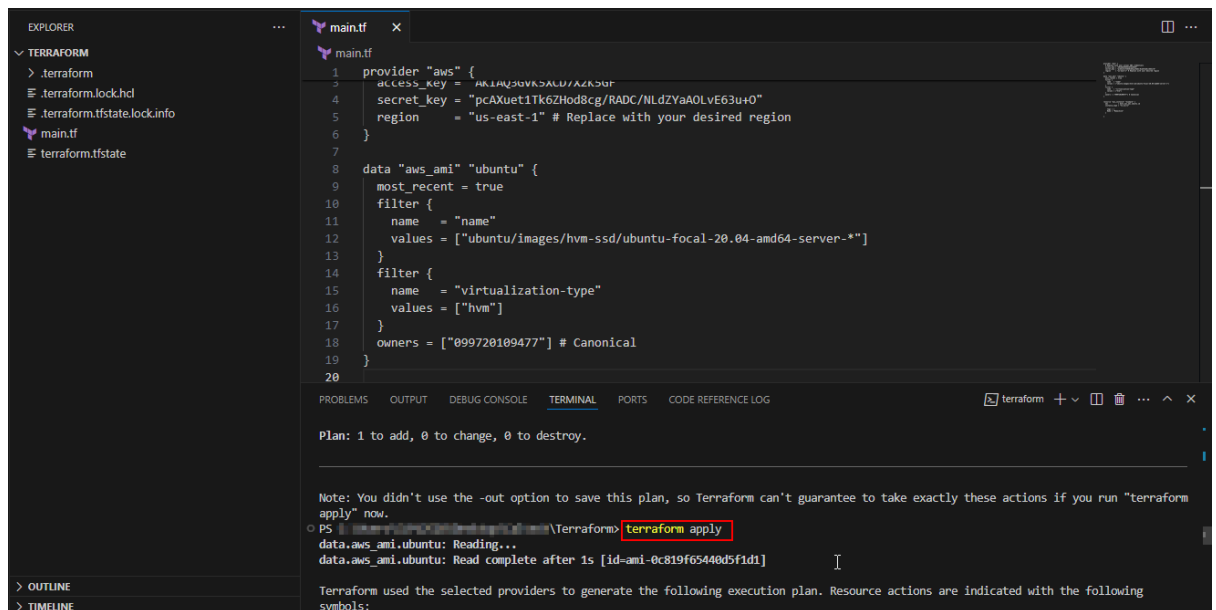
```
PS C:\Users\SLP12528\Desktop\Caltech\Terraform> terraform plan
data.aws_ami.ubuntu: Reading...
data.aws_ami.ubuntu: Read complete after 2s [id=ami-8c819f65440d5f1d1]

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_instance.example will be created
```

3.5 Apply the configuration using the following command as shown in the screenshot below:  
**terraform apply**



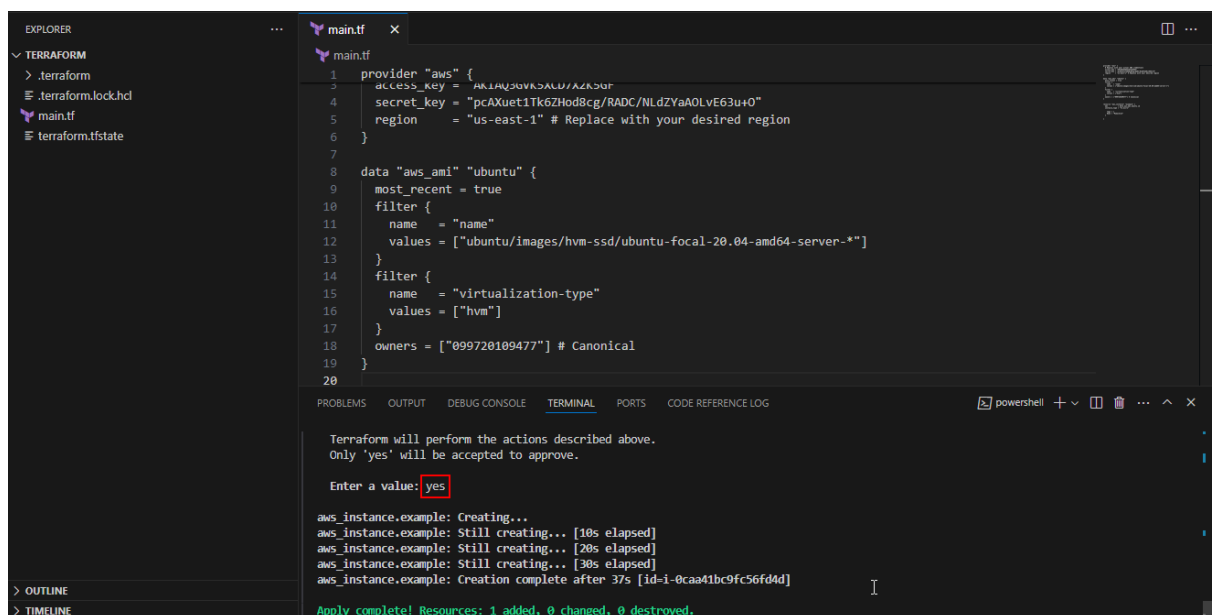
The screenshot shows the Visual Studio Code interface with the Terraform configuration file `main.tf` open. The configuration defines an AWS provider and an `aws_ami` data source. The terminal window shows the command `terraform apply` being executed, resulting in the following output:

```
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.
PS C:\Users\user> terraform apply
data.aws_ami.ubuntu: Reading...
data.aws_ami.ubuntu: Read complete after 1s [id=ami-0c819f65440d5f1d1]

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

3.6 Type **yes** and click on the **enter** key to proceed further as shown in the screenshot below:



The screenshot shows the Visual Studio Code interface with the Terraform configuration file `main.tf` open. The terminal window shows the command `terraform apply` being executed, resulting in the following output:

```
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

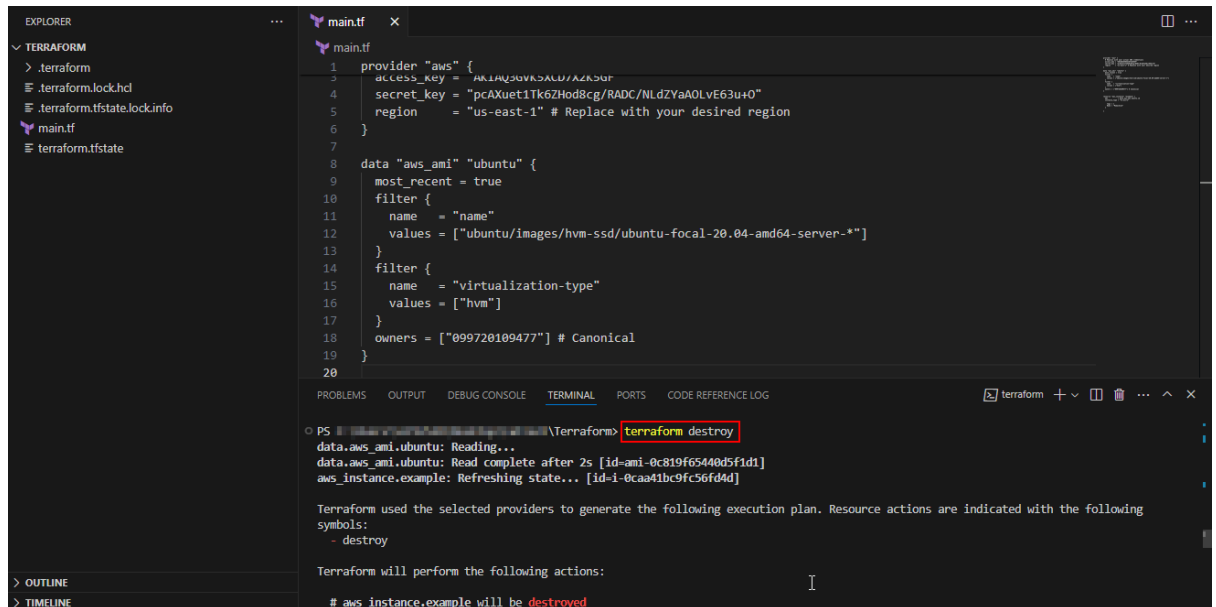
Enter a value: yes

aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Still creating... [20s elapsed]
aws_instance.example: Still creating... [30s elapsed]
aws_instance.example: Creation complete after 37s [id=i-0caa41bc9fc56fd4d]

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

3.7 Execute the following command to remove all the resources defined in the Terraform configuration as shown in the screenshot below:

**terraform destroy**



The screenshot shows the VS Code interface with the Terraform configuration file `main.tf` open. The configuration defines an AWS provider and an AWS AMI data source. The terminal window shows the command `terraform destroy` being executed. The output indicates that the resources are being destroyed and the execution plan is generated.

```
1 provider "aws" {
2   access_key = "AKIAJ3GVK3ALU/AZK3GUF"
3   secret_key = "pcAXuet1Tk6ZHoD8cg/RADC/NLdZYaAOLvE63u+0"
4   region     = "us-east-1" # Replace with your desired region
5 }
6
7
8 data "aws_ami" "ubuntu" {
9   most_recent = true
10  filter {
11    name     = "name"
12    values   = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
13  }
14  filter {
15    name     = "virtualization-type"
16    values   = ["hvm"]
17  }
18  owners    = ["099720109477"] # Canonical
19 }
20
```

PS [C:\Users\user> terraform destroy

data.aws\_ami.ubuntu: Reading...

data.aws\_ami.ubuntu: Read complete after 2s [id=ami-0c819f65440d5f1d1]

aws\_instance.example: Refreshing state... [id=i-0caa41bc9fc56fd4d]

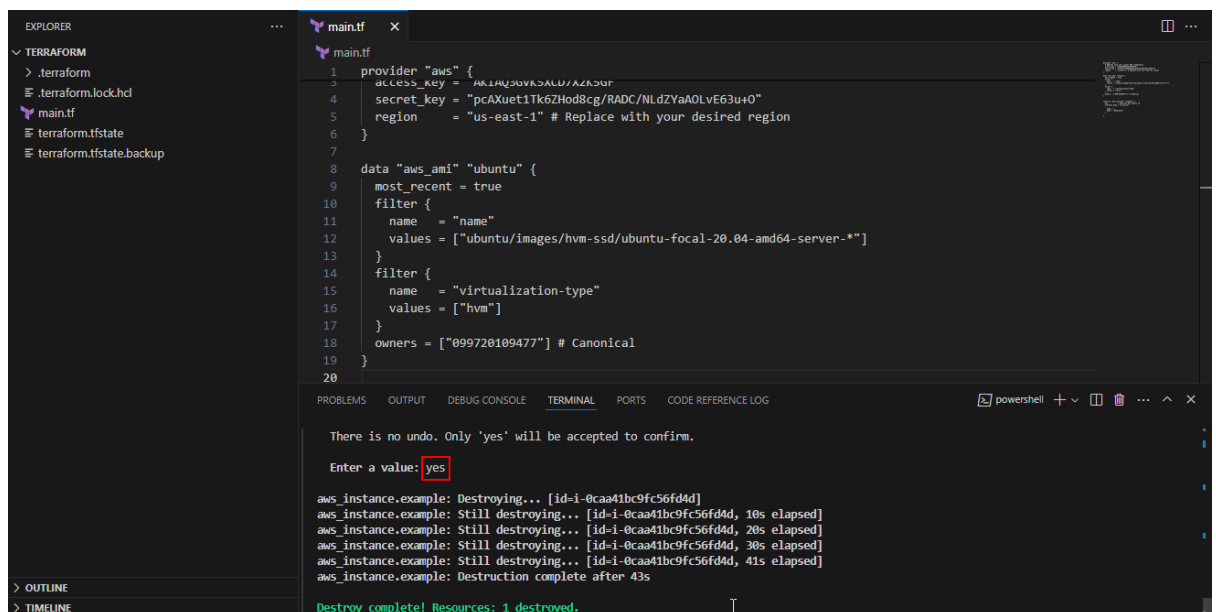
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

- destroy

Terraform will perform the following actions:

- # aws\_instance.example will be **destroyed**

3.8 Type **yes** and click on the **Enter** key to proceed further as shown in the screenshot below:



The screenshot shows the VS Code interface with the Terraform configuration file `main.tf` open. The terminal window shows the command `terraform destroy` being executed. The output indicates that the resources are being destroyed and the execution plan is generated. The terminal shows the confirmation prompt "There is no undo. Only 'yes' will be accepted to confirm." and the user enters "yes".

```
1 provider "aws" {
2   access_key = "AKIAJ3GVK3ALU/AZK3GUF"
3   secret_key = "pcAXuet1Tk6ZHoD8cg/RADC/NLdZYaAOLvE63u+0"
4   region     = "us-east-1" # Replace with your desired region
5 }
6
7
8 data "aws_ami" "ubuntu" {
9   most_recent = true
10  filter {
11    name     = "name"
12    values   = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
13  }
14  filter {
15    name     = "virtualization-type"
16    values   = ["hvm"]
17  }
18  owners    = ["099720109477"] # Canonical
19 }
20
```

There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: **yes**

aws\_instance.example: Destroying... [id=i-0caa41bc9fc56fd4d]

aws\_instance.example: Still destroying... [id=i-0caa41bc9fc56fd4d, 10s elapsed]

aws\_instance.example: Still destroying... [id=i-0caa41bc9fc56fd4d, 20s elapsed]

aws\_instance.example: Still destroying... [id=i-0caa41bc9fc56fd4d, 30s elapsed]

aws\_instance.example: Still destroying... [id=i-0caa41bc9fc56fd4d, 41s elapsed]

aws\_instance.example: Destruction complete after 43s

**Destroy complete! Resources: 1 destroyed.**

By following these steps, you have successfully implemented a workflow for efficiently and effectively managing AWS infrastructure configurations in Terraform.