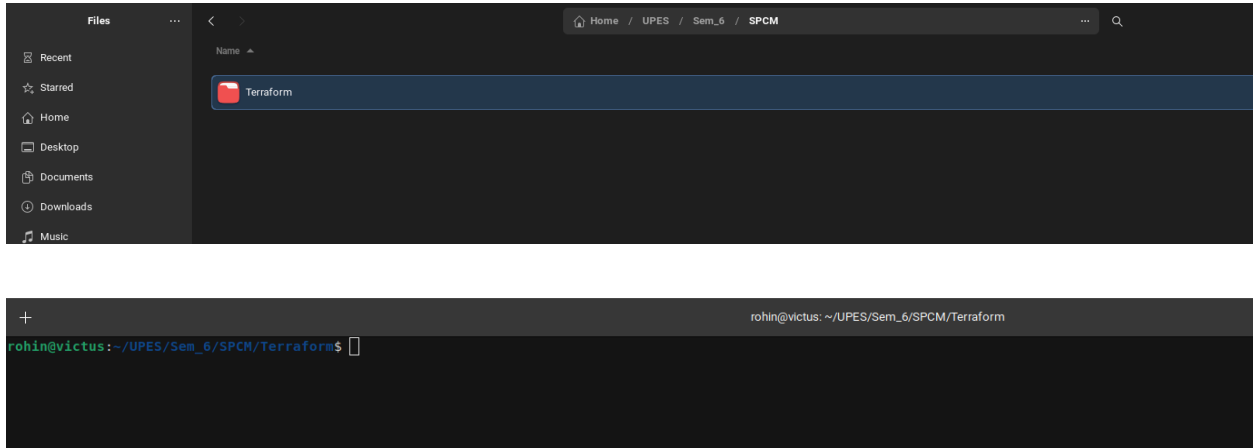


## SPCM Lab 2

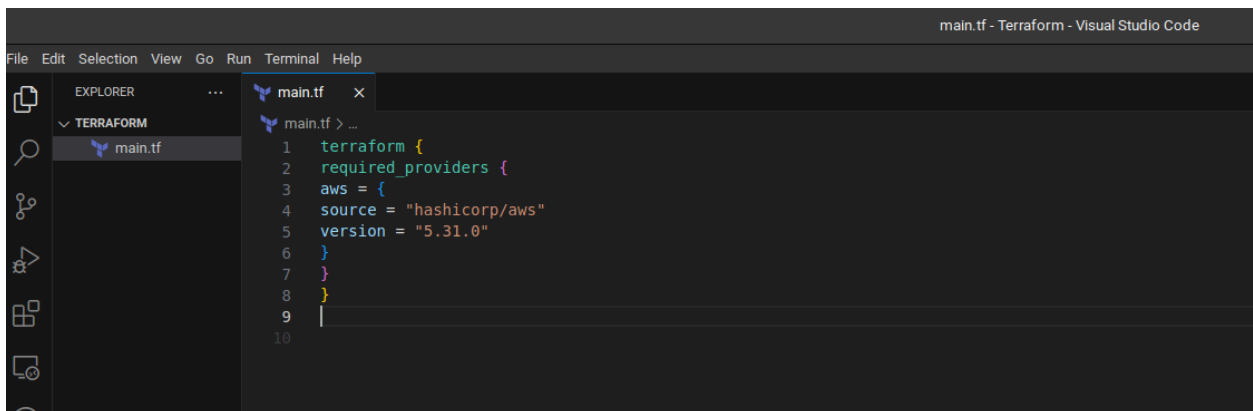
### Step 1: Create a New Directory:

Create a new directory for your Terraform configuration:



### Step 2: Create Terraform Configuration File (main.tf):

Create a file named main.tf with the following content:



## Create a new IAM user:

The screenshot shows the 'Specify user details' step in the AWS IAM console. The 'User name' field is filled with 'terraform-user'. Below it, there is a checkbox for 'Provide user access to the AWS Management Console - optional' and a blue information box stating that if creating programmatic access, credentials can be generated after creation.

**Specify user details**

**User details**

User name  
terraform-user

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.

[i](#) 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.

## Give admin access to user

The screenshot shows the 'Set permissions' step in the AWS IAM console. The 'Attach policies directly' option is selected. Below, a table lists available permissions policies, with 'AdministratorAccess' selected.

**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 (1/1171)**

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

Search: [ ] Filter by Type: All types

	Policy name	Type	Attached entities
<input type="checkbox"/>	<a href="#">AccessAnalyzerServiceRolePolicy</a>	AWS managed	0
<input checked="" type="checkbox"/>	<a href="#">AdministratorAccess</a>	AWS managed - job function	0
<input type="checkbox"/>	<a href="#">AdministratorAccess-Amplify</a>	AWS managed	0
<input type="checkbox"/>	<a href="#">AdministratorAccess-AWSElasticBeanstalk</a>	AWS managed	0

## Create Access-Key for the newly created user:

The screenshot shows the 'Access keys' section in the AWS IAM console. It indicates that there are 0 access keys and provides a 'Create access key' button. A note advises against using long-term credentials like access keys.

**Access keys (0)**

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. [Learn more](#)

No access keys. As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. [Learn more](#)

Create access key

## Provide access keys in the main.tf file:

```
10 provider "aws" {
11   region = "ap-south-1"
12   access_key = "AKIAWMVRZHDWFAYQF7SU"
13   secret_key = "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"
14 }
15
16 |
```

This script defines an AWS provider and provisions an EC2 instance.

### Step 3: Initialize Terraform:

Run the following command to initialize your Terraform working directory:

```
+ rohin@victus: ~/UPES/Sem_6/SPCM/Terraform
rohin@victus:~/UPES/Sem_6/SPCM/Terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.31.0"...
- Installing hashicorp/aws v5.31.0...
- Installed hashicorp/aws v5.31.0 (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.
rohin@victus:~/UPES/Sem_6/SPCM/Terraform$
```