Assignment 6

Aim : To understand terraform lifecycle, core concepts/ terminologies and install it.

Theory :

Terraform, developed by HashiCorp, is an open-source infrastructure as code (IaC) software tool that allows users to define and provision data center infrastructure using a declarative configuration language. In simpler terms, it lets you codify your infrastructure, enabling consistent and reproducible deployments. It is used for defining, provisioning, and managing cloud infrastructure using a declarative language.

Core Concepts:

Providers: Terraform uses providers to interact with cloud services. Examples include AWS, Azure, Google Cloud, and many others. Each provider offers resource types that can be managed.

Resources: These are the primary components in Terraform. A resource might be a physical component such as an EC2 instance in AWS or a database in Azure.
State: Terraform maintains a state file that maps real-world resources to your configuration. This state is used to determine what Terraform will do on the next apply.

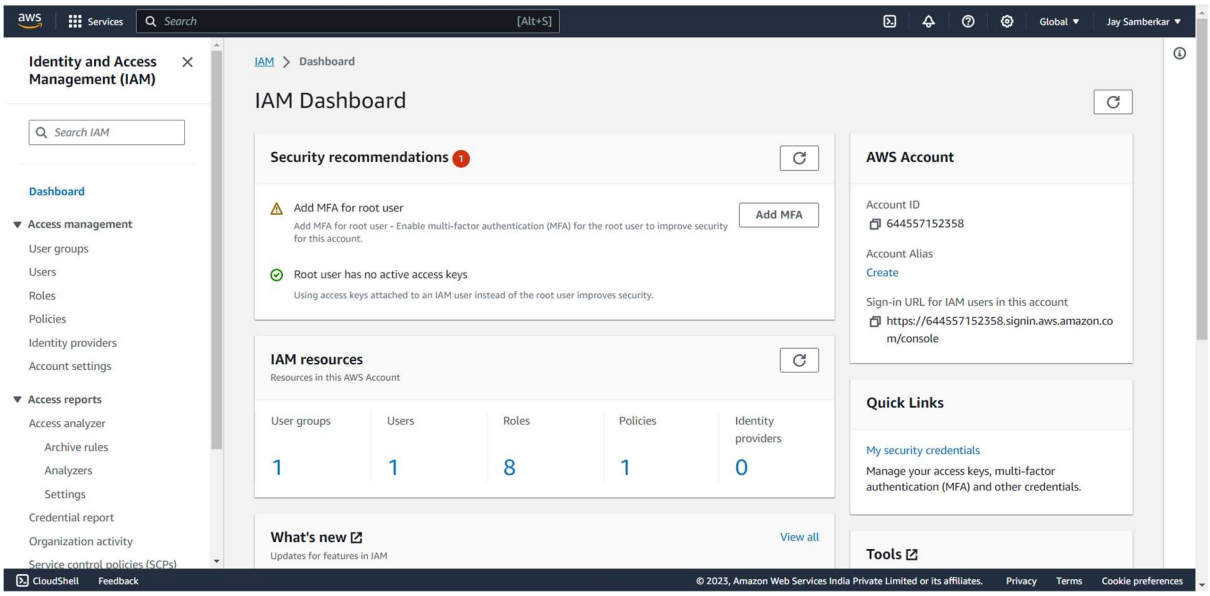
Modules: These are containers for multiple resources that are used together. They provide a way to group resources, and they can be used to create reusable infrastructure components.

Variables and Outputs: Variables allow for parameterization of the Terraform configuration, making it more dynamic and flexible. Outputs are a way to get information about the infrastructure, like IP addresses or DNS names.

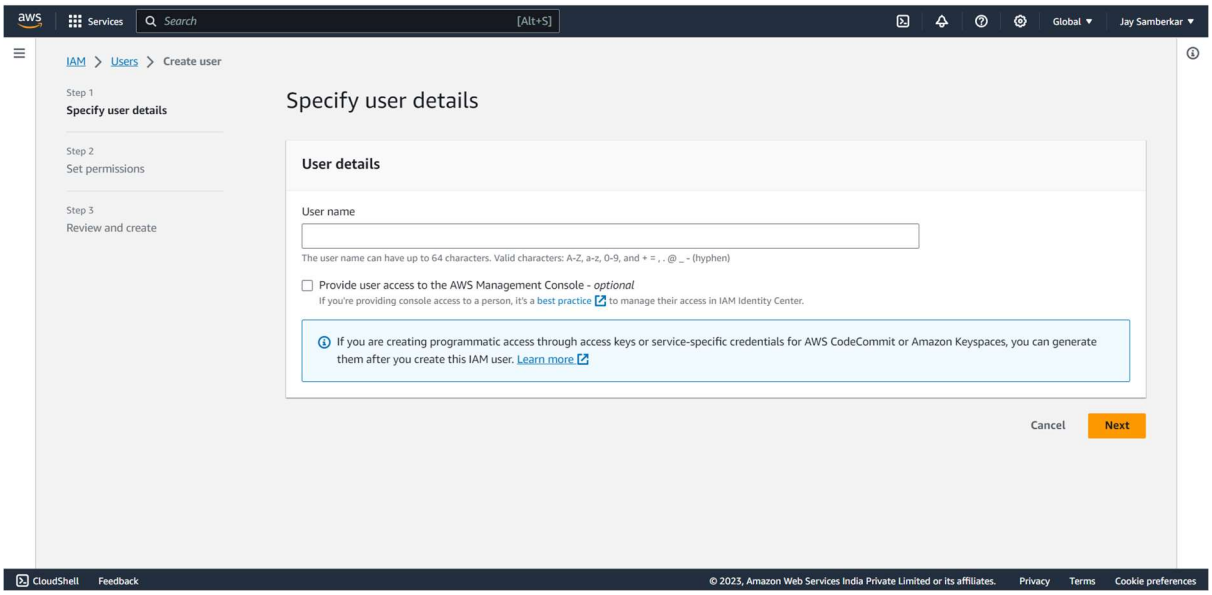
Provisioners: While not always recommended due to their imperative nature, provisioners can be used to model specific actions on the local machine or on a remote machine, like executing scripts.

Steps :

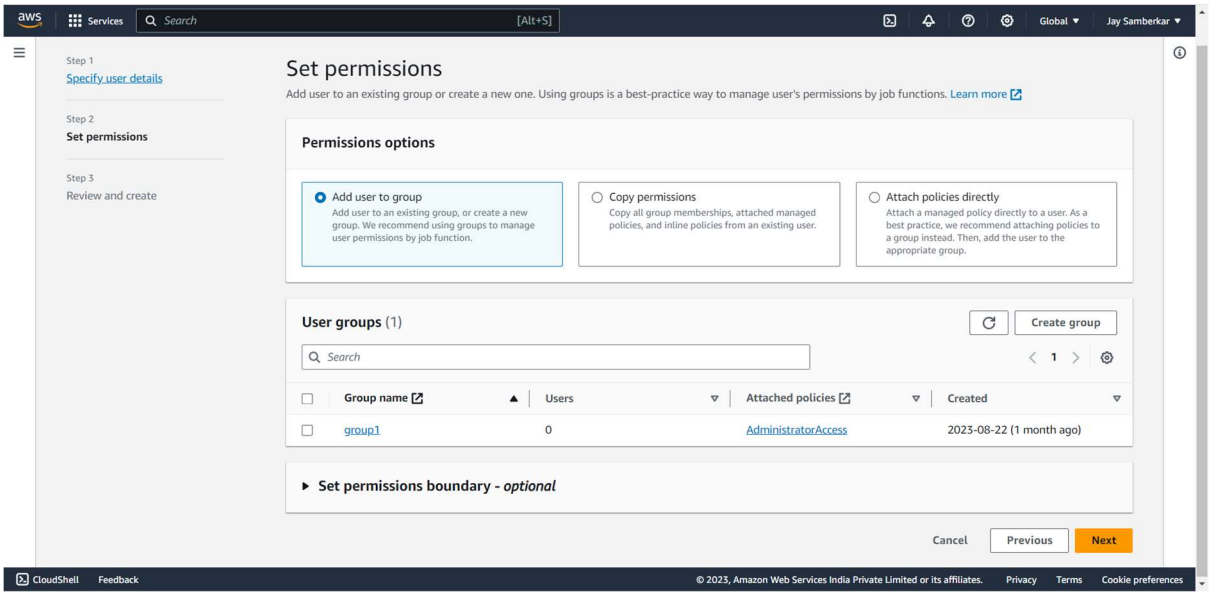
1. Login and Search IAM on AWS Console and select the first option.



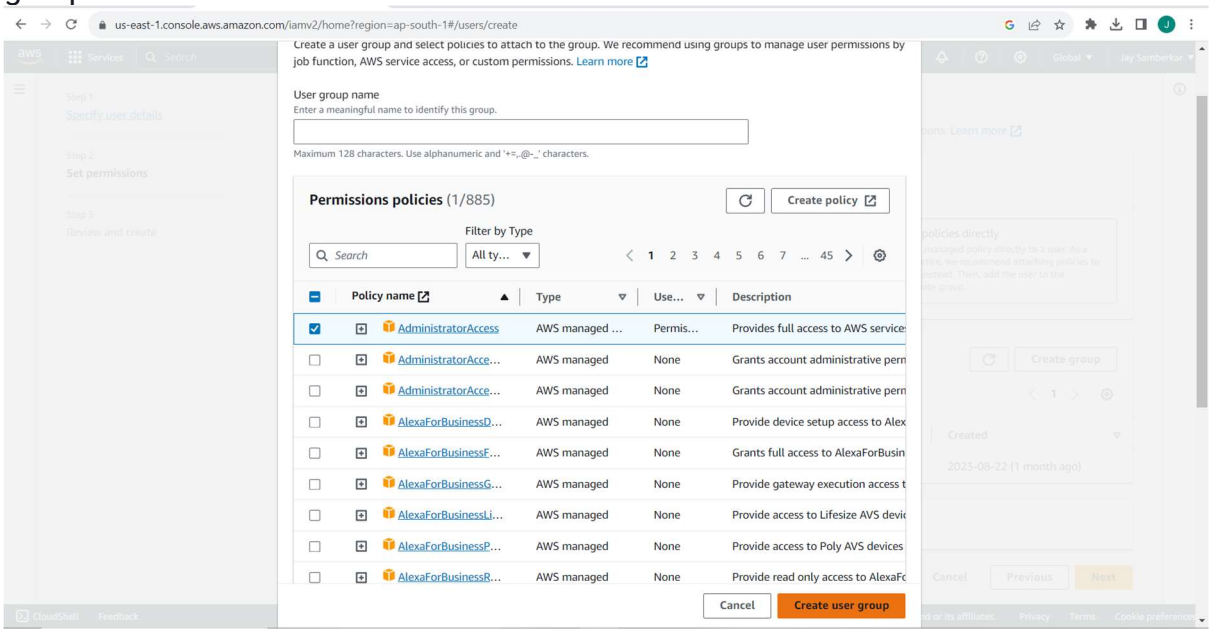
2. Head to users and Click on Create User



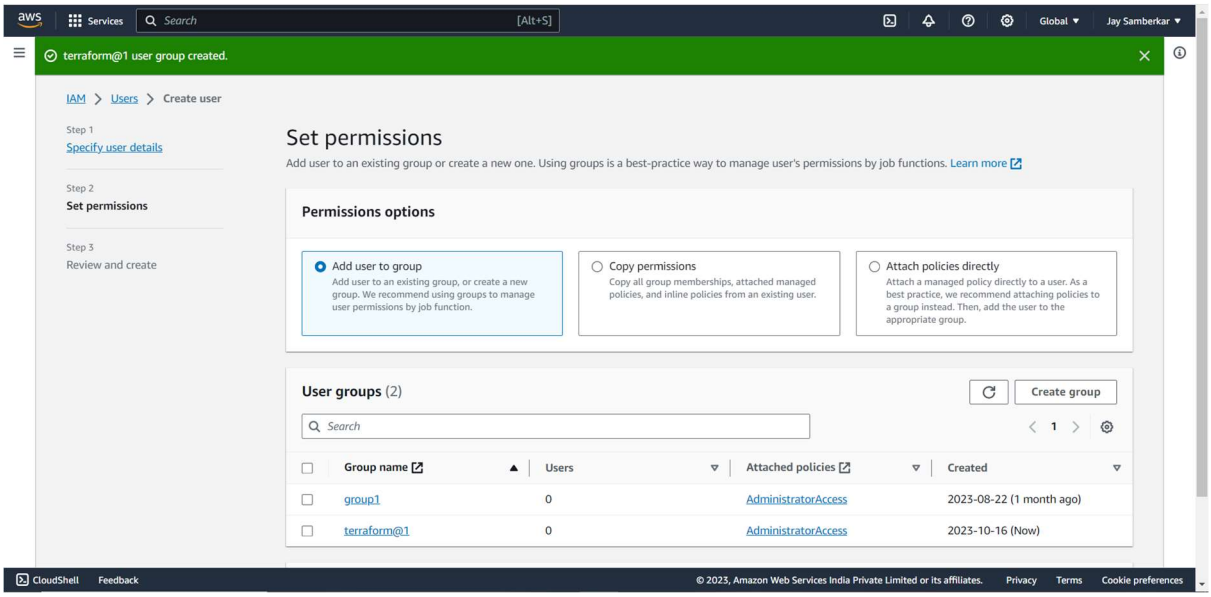
3. Enter a new username and click on next
4. Click on create group



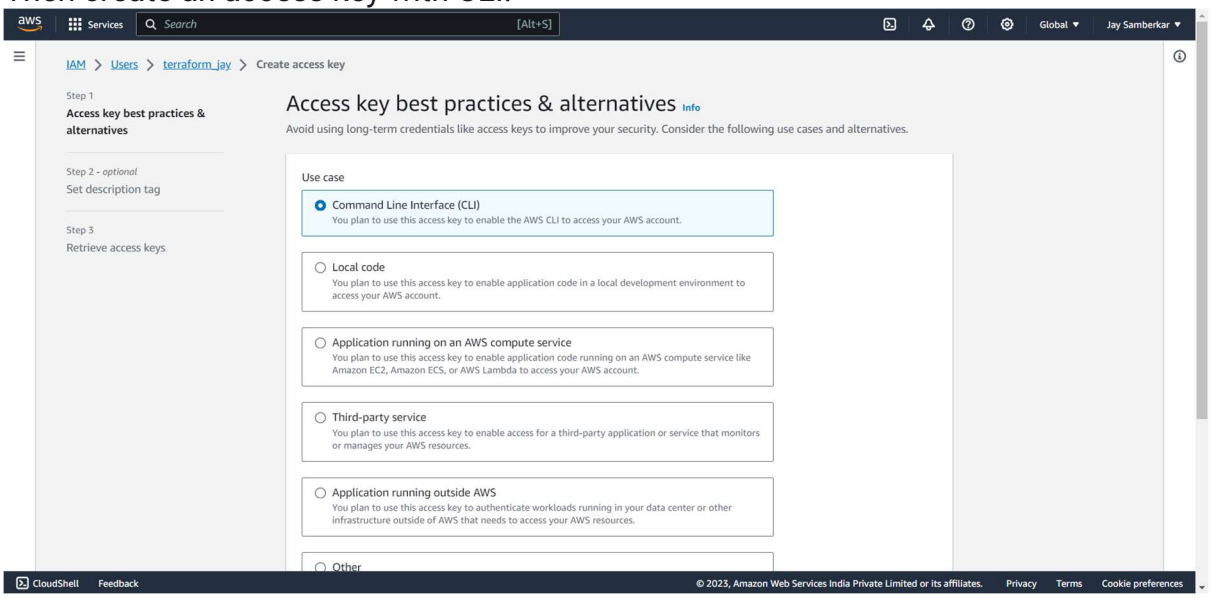
5. Select the first Policy will full access and enter a group name and create the group.



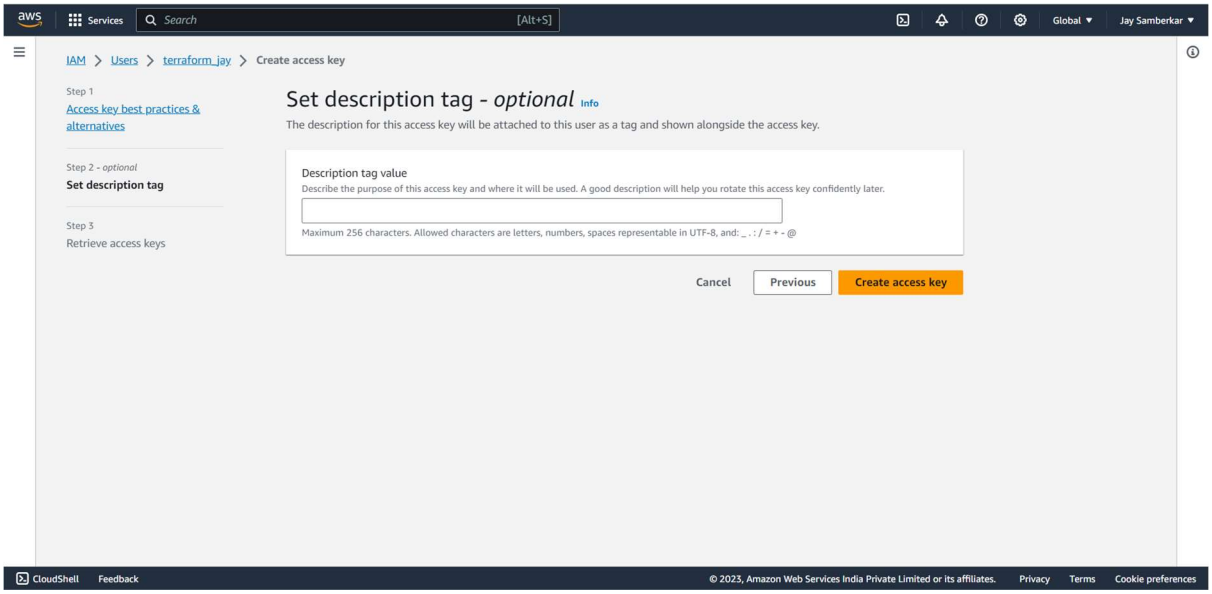
6. Then click on next, select the policy and create user.



7. Then create an access key with CLI.



8. Leave the description field empty and create a key. Download the access key in the .csv file.



9. Install terraform.

```
Last login: Sat Aug 19 17:44:56 on ttys000
❯ brew tap hashicorp/tap
Running `brew update --auto-update`...
Installing from the API is now the default behaviour!
You can save space and time by running:
  brew untap homebrew/core
  brew untap homebrew/cask
==> Auto-updated Homebrew!
Updated 3 taps (mongodb/brew, homebrew/core and homebrew/cask).
==> New Formulae
arm-none-eabi-binutils                  medusa
arm-none-eabi-gcc                       mjml
arm-none-eabi-gdb                       mongodb/brew/mongodb-community@6.0
asnmap                                  mongodb/brew/mongodb-enterprise@6.0
cargo-auditable                         mongodb/brew/mongodb-mongocryptd@6.0
cdi                                     mysql-client@8.0
cloudlist                               mysql@8.0
coder                                   ollama
ctpv                                    proxify
czkawka                                 python-certifi
dnsrobocert                             riff
dolphie                                 riscv64-elf-binutils
ebook2cw                                riscv64-elf-gcc
go@1.20                                 riscv64-elf-gdb
img2pdf                                 rpmspectool
```

10. Create the terraform config file main.tf with required configurations.

```
❯ nano main.tf
❯ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.16"...
- Installing hashicorp/aws v4.67.0...
- Installed hashicorp/aws v4.67.0 (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.
~/terraform_scripts                                                  ✔ 25s
```

11. Run the command terraform plan and enter yes

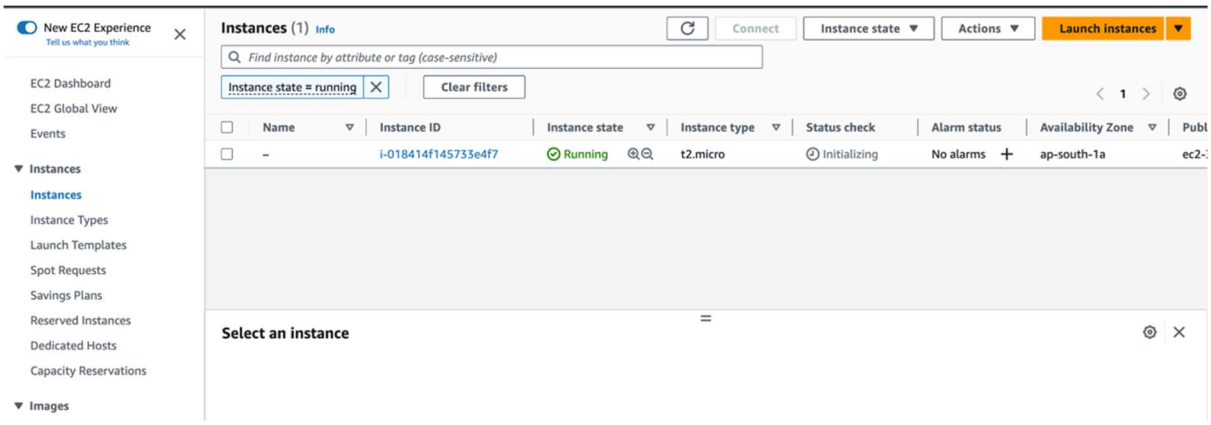
```
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
❯ terraform plan

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_instance.Ubuntu will be created
  + resource "aws_instance" "Ubuntu" {
      + ami                                  = "ami-0f5ee92e2d63afc18"
      + 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)
```

12. Check EC2 instances on AWS, you'll find an instance running started by terraform.



13. Now run terraform destroy to terminate the instance.

```
          - enable_resource_name_dns_a_record    = false -> null
          - enable_resource_name_dns_aaaa_record = false -> null
          - hostname_type                        = "ip-name" -> null
        }

      - root_block_device {
          - delete_on_termination = true -> null
          - device_name           = "/dev/sda1" -> null
          - encrypted             = false -> null
          - iops                  = 100 -> null
          - tags                  = {} -> null
          - throughput            = 0 -> null
          - volume_id             = "vol-07a64df7d00e4e7a1" -> null
          - volume_size           = 8 -> null
          - volume_type           = "gp2" -> null
        }
    }

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

Lab Outcome:
LO1, LO6 mapped.

Conclusion :
We understood the workings of Terraform to create and manage EC2 instances in AWS which provides a highly automated and reproducible infrastructure-as-code solution. The ability to effortlessly spin up and tear down instances not only enhances operational agility but also ensures optimal resource utilization.