Name: Ruthik Jadhav
Roll No.: 321087
Gr. No.: 22120243
Batch & Division: A3

# Assignment No. 5

**Aim:** Write IaC using terraform to create EC2 machine on AWS or azure or google cloud. (Compulsory to use Input and output variable files).

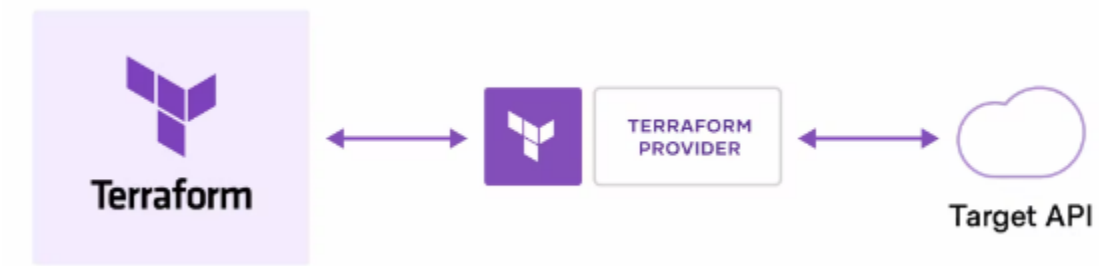
**Theory:**

## What is terraform?

- Terraform Cloud enables infrastructure automation for provisioning, compliance, and management of any cloud, datacenter, and service.
- It is an open-source tool for provisioning and managing cloud infrastructure. Terraform can provision resources on any cloud platform.
- Terraform allows you to create infrastructure in configuration files(tf files) that describe the topology of cloud resources.
- These resources include virtual machines, storage accounts, and networking interfaces or virtually any resource you want

## How does Terraform work?

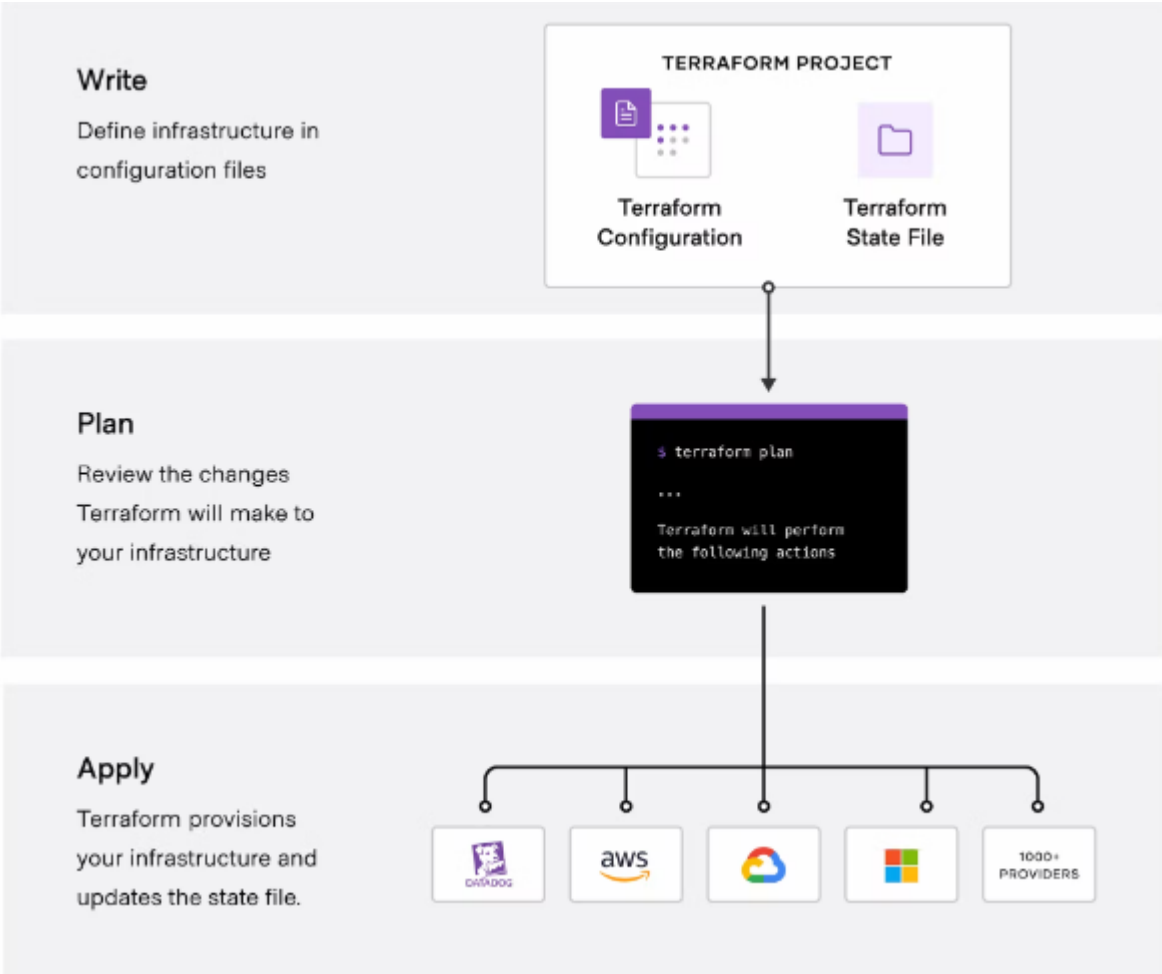
Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.




HashiCorp and the Terraform community have already written thousands of providers to manage many different types of resources and services. You can find all publicly available providers on the Terraform Registry, including Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog, and many more.

## The core Terraform workflow consists of three stages:

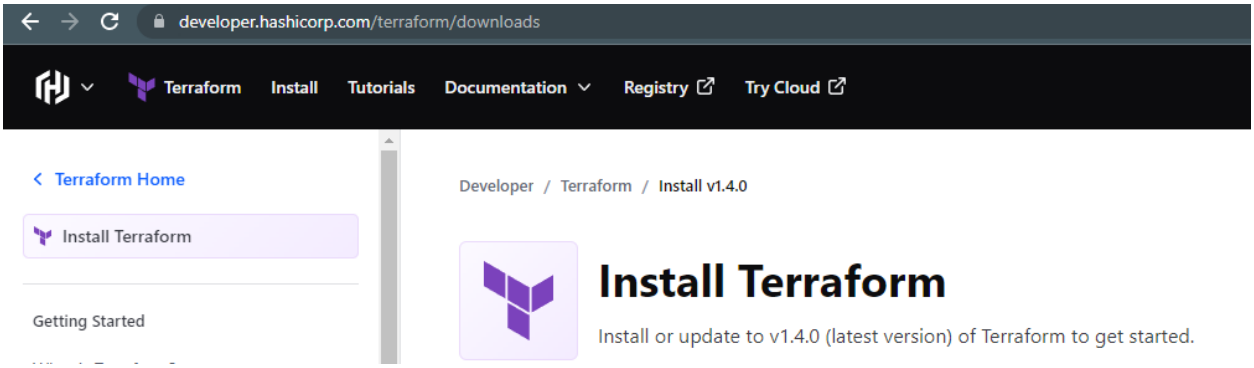
- **Write:** You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.
- **Plan:** Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration.
- **Apply:** On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies. For example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machines.



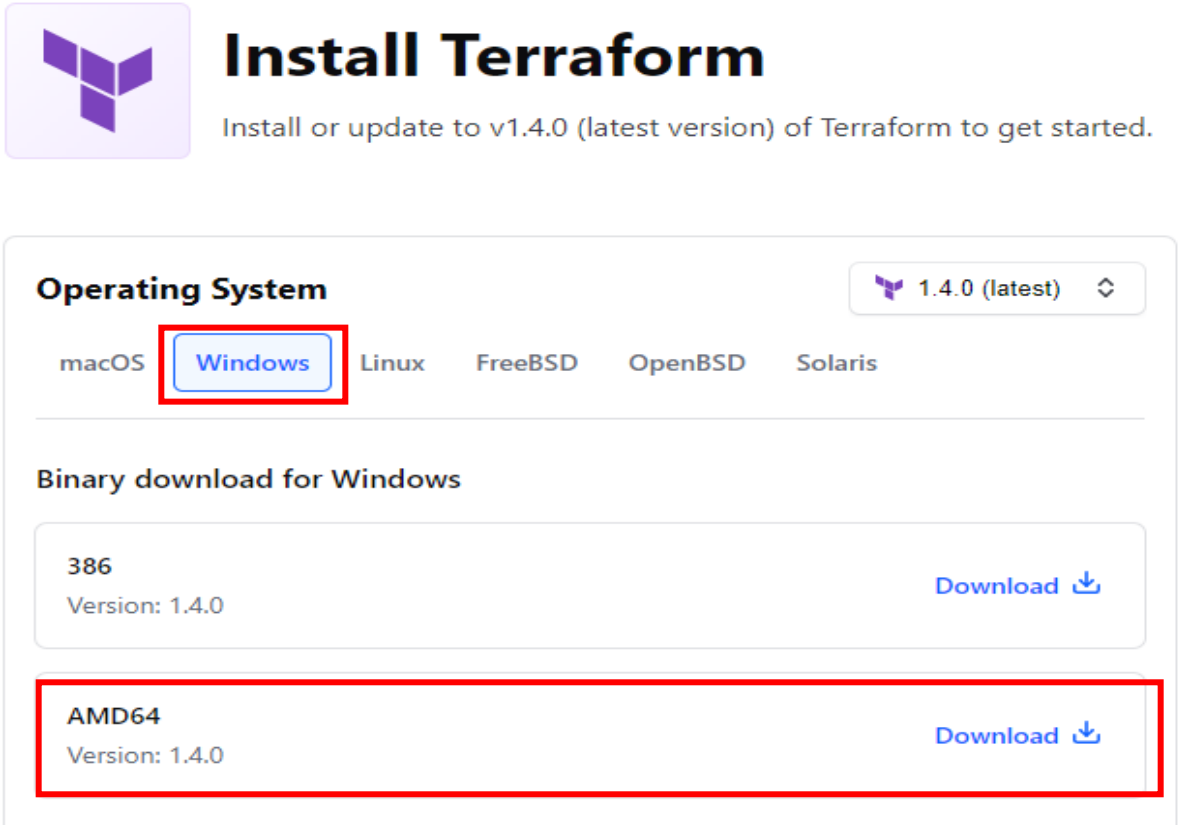
## Step-by-step screenshot to install and configure Terraform

### 1. Download it from official site

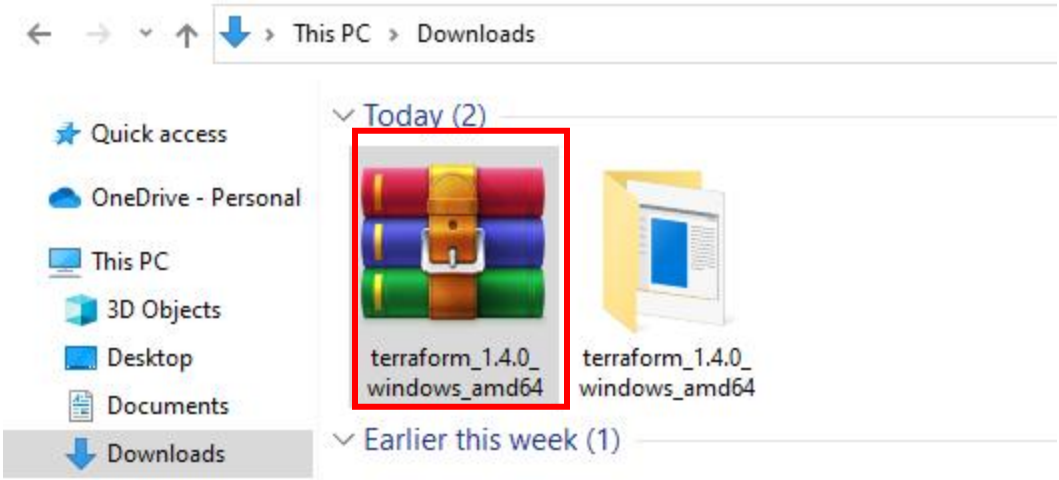
https://developer.hashicorp.com/terraform/downloads



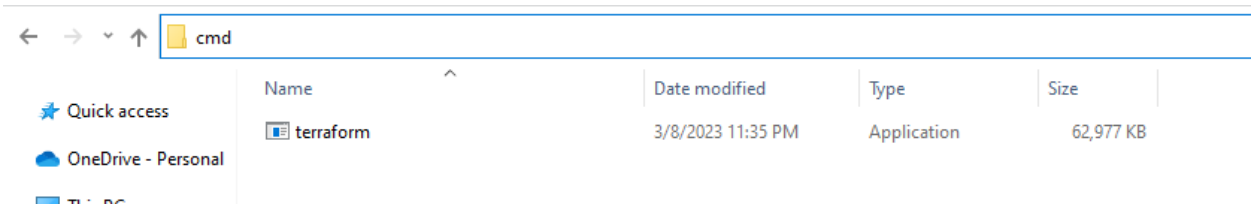

### 2. Select the OS to download



3. Extract the files from **zip**

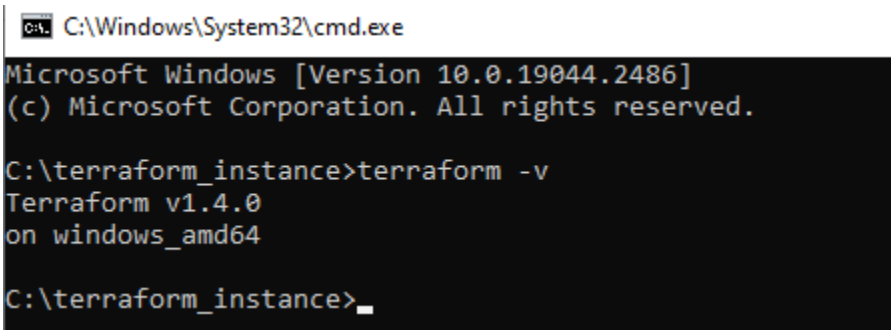



4. Open **cmd prompt** where the terraform  files are extracted



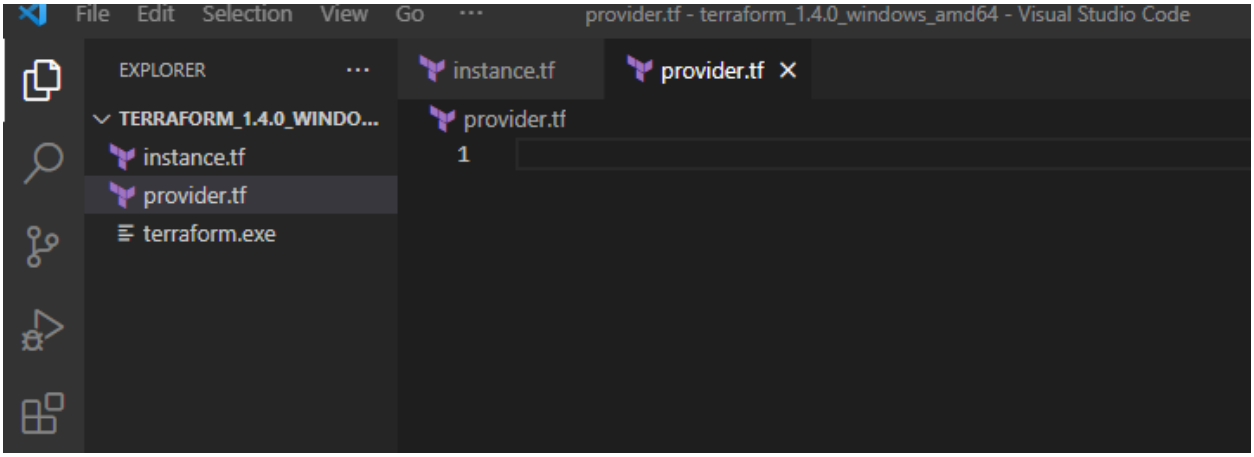

-

**5. Check for version** if it is successfully install on system

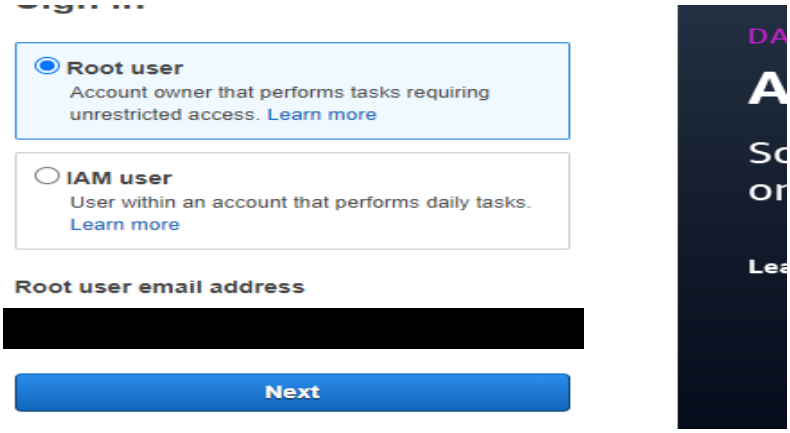


# Terraform script to create Infrastructure on any cloud platform

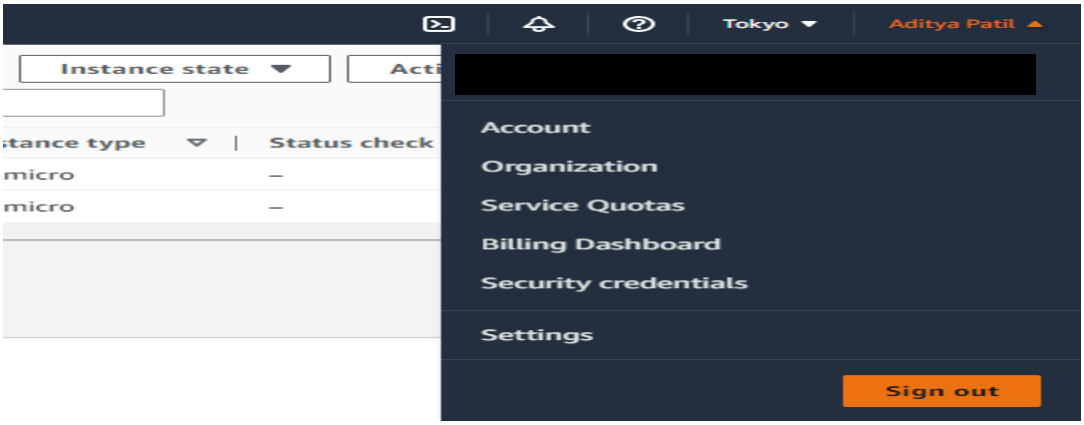
Step 1] Create two files as mention below **instance.tf and provider.tf**



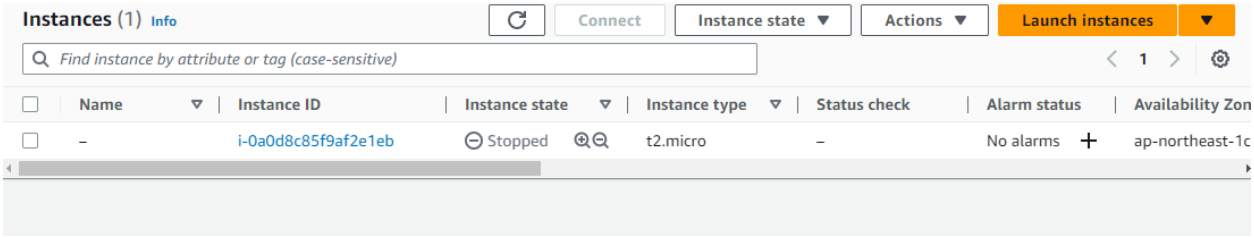

**Step 2]** Login into aws console.



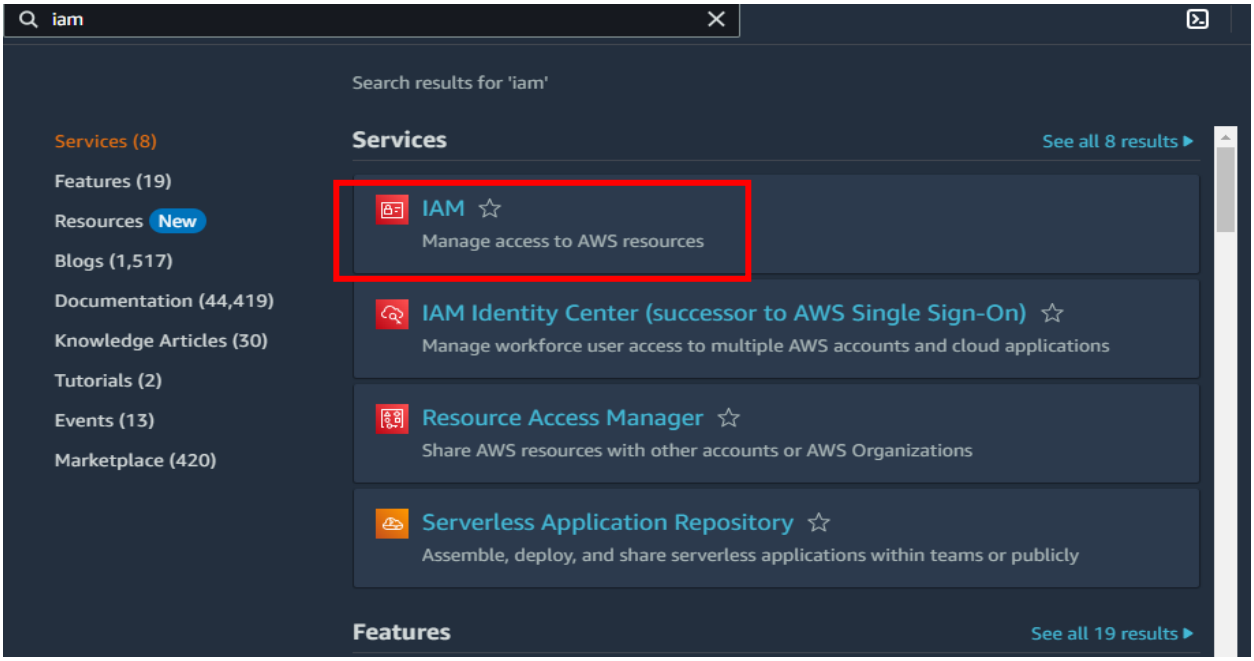

**Step 3]** Login Successful



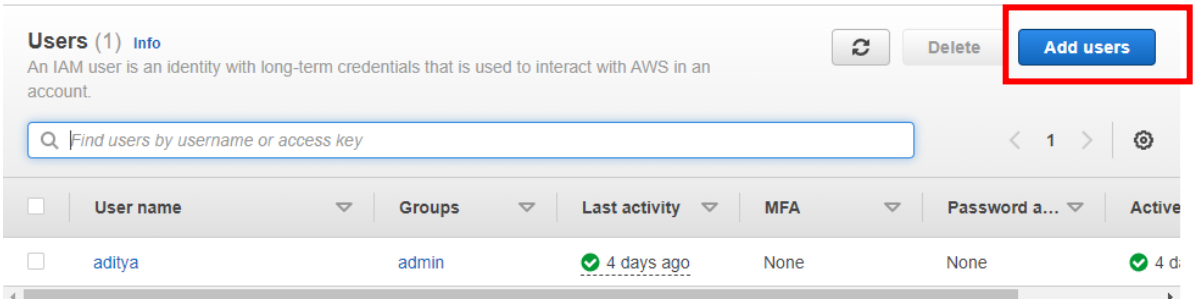
**Step 4] One instance** is their already now we have to create another instance using Terraform.



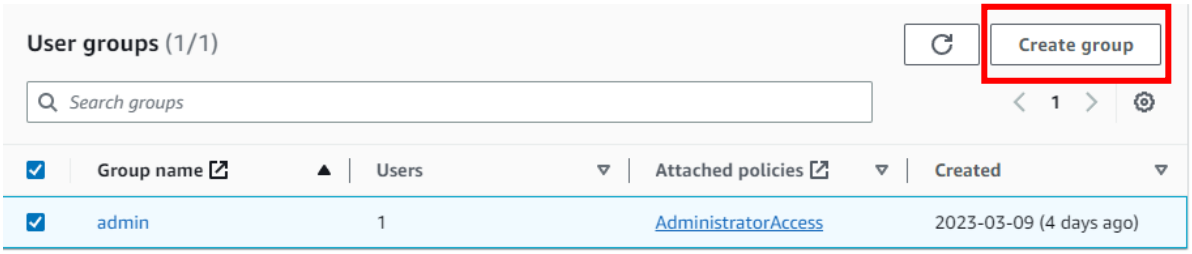

Step 5] Go to **IAM** to create a **new user**



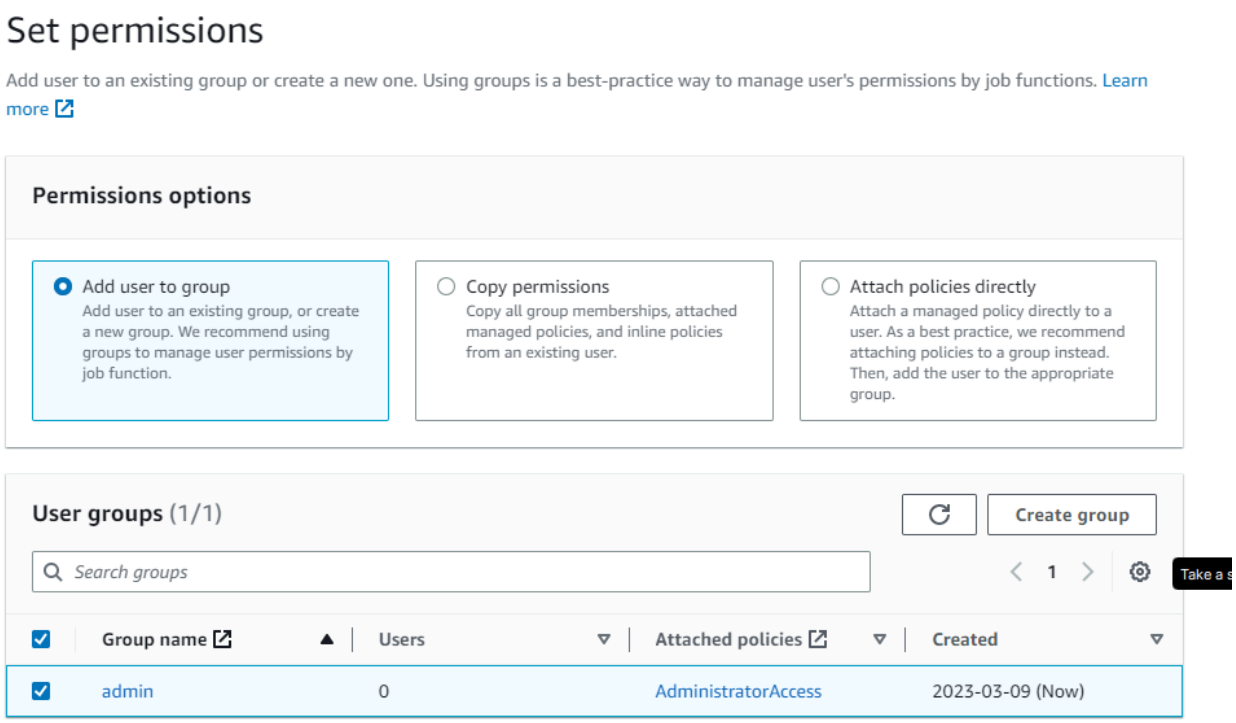

Step 6]  Click on the add User.



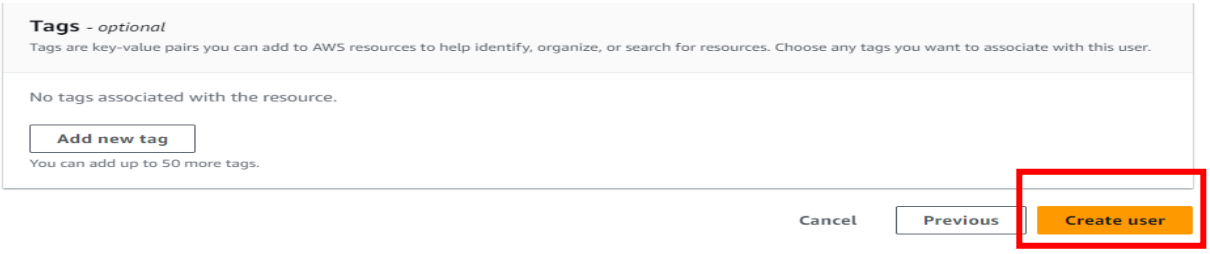
Step 7] As there is no user group lets create group of admin.



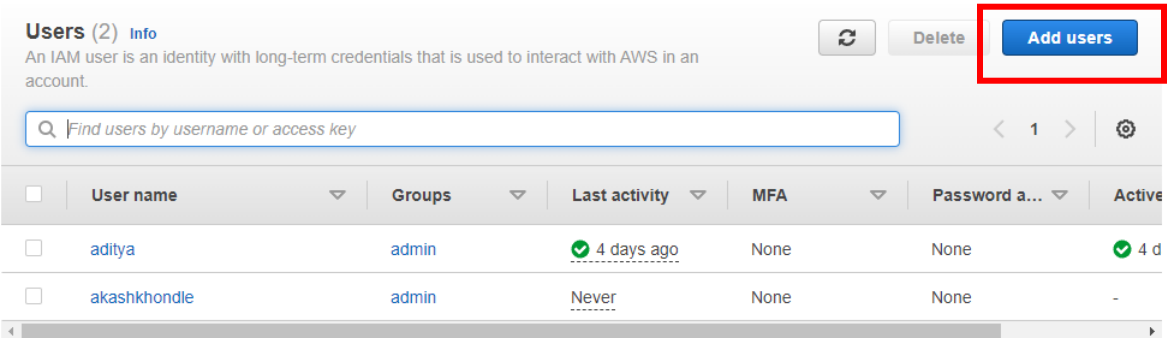

Step 8] User group created add new user to **admin group**



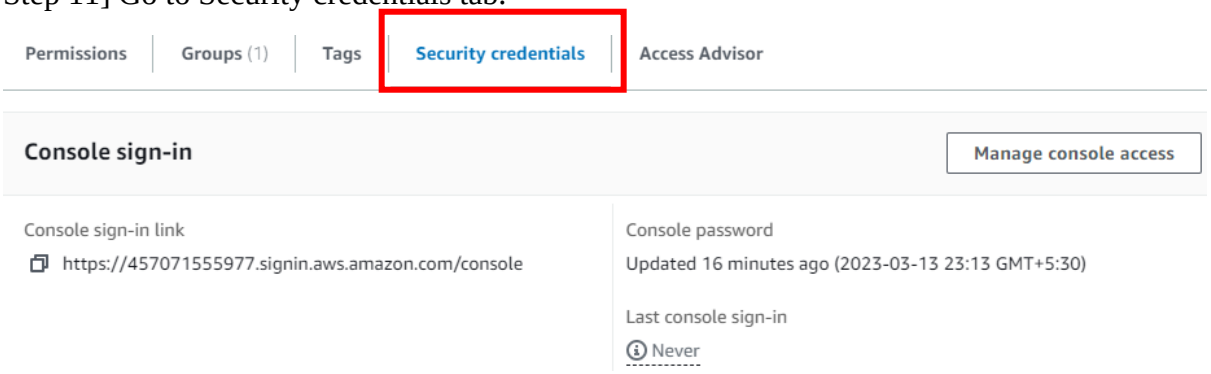

Step 9] Click on **Create User**



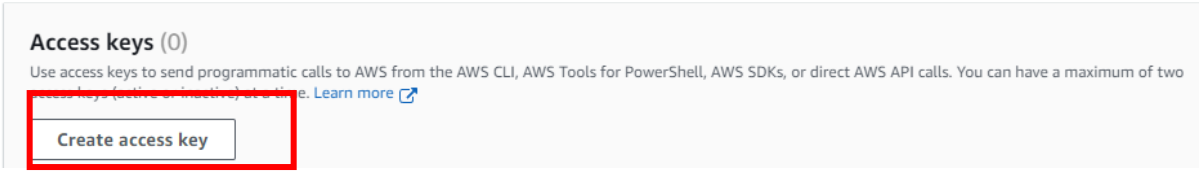
Step 10] Now here the New User has been **created** and click on Username which you created.



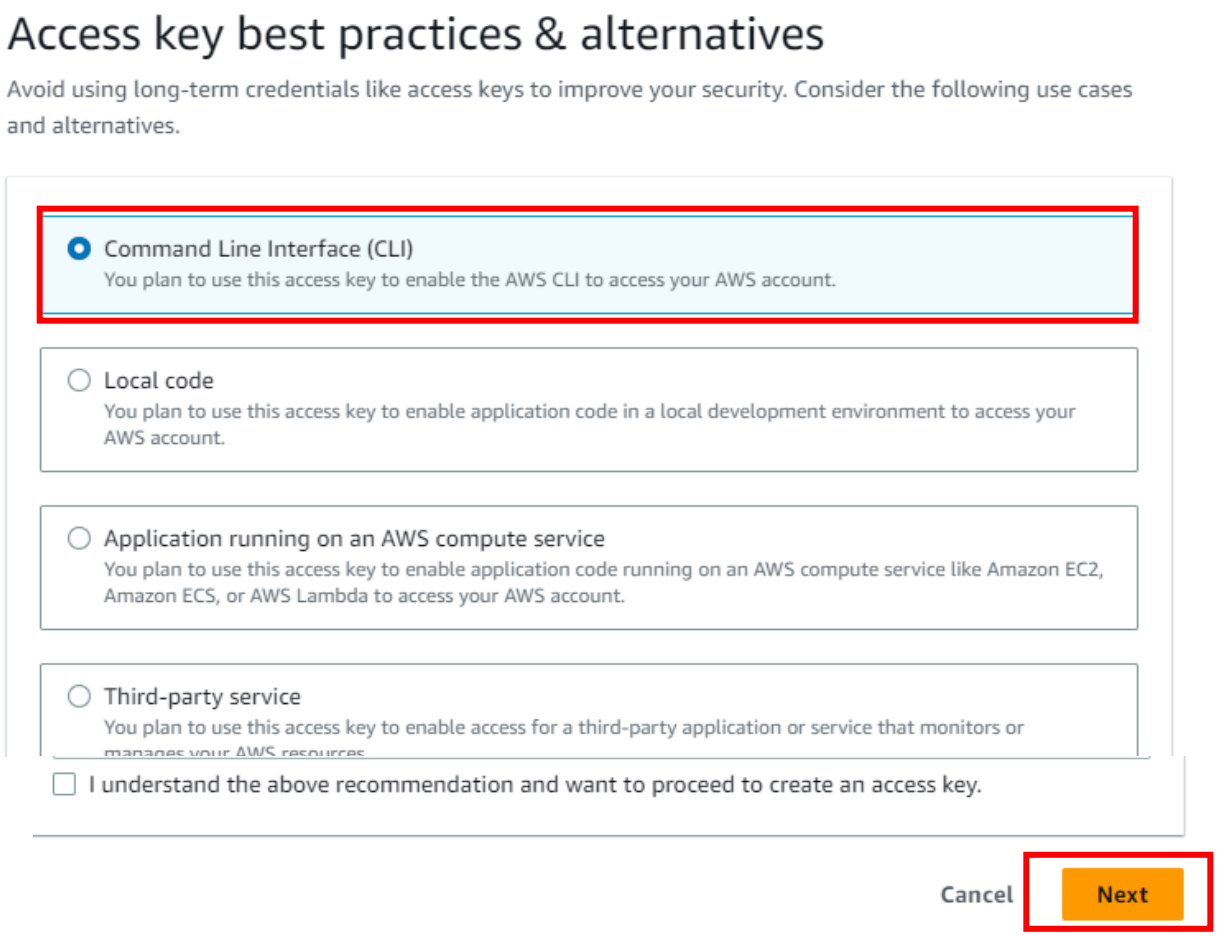

Step 11] Go to Security credentials tab.



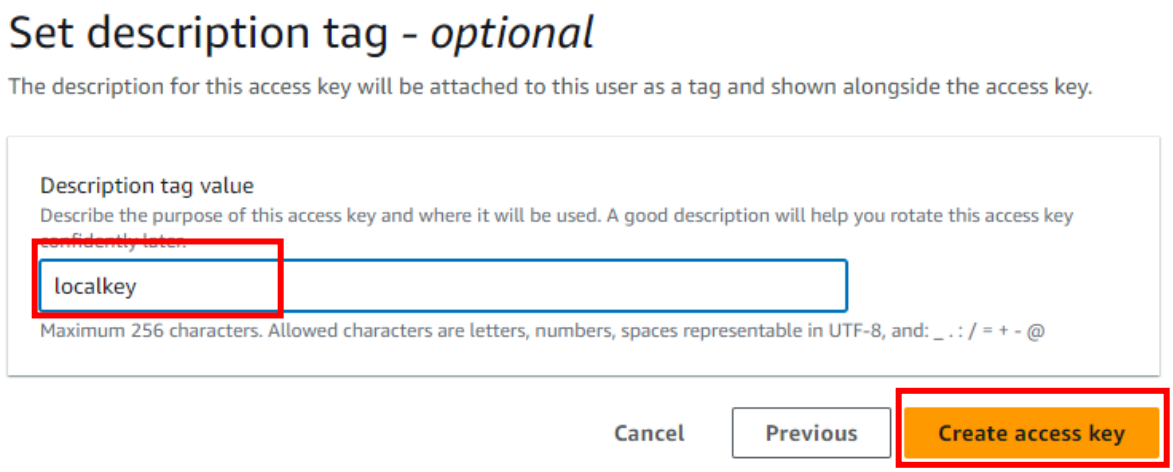

Step 12] Click on Create access Key button.



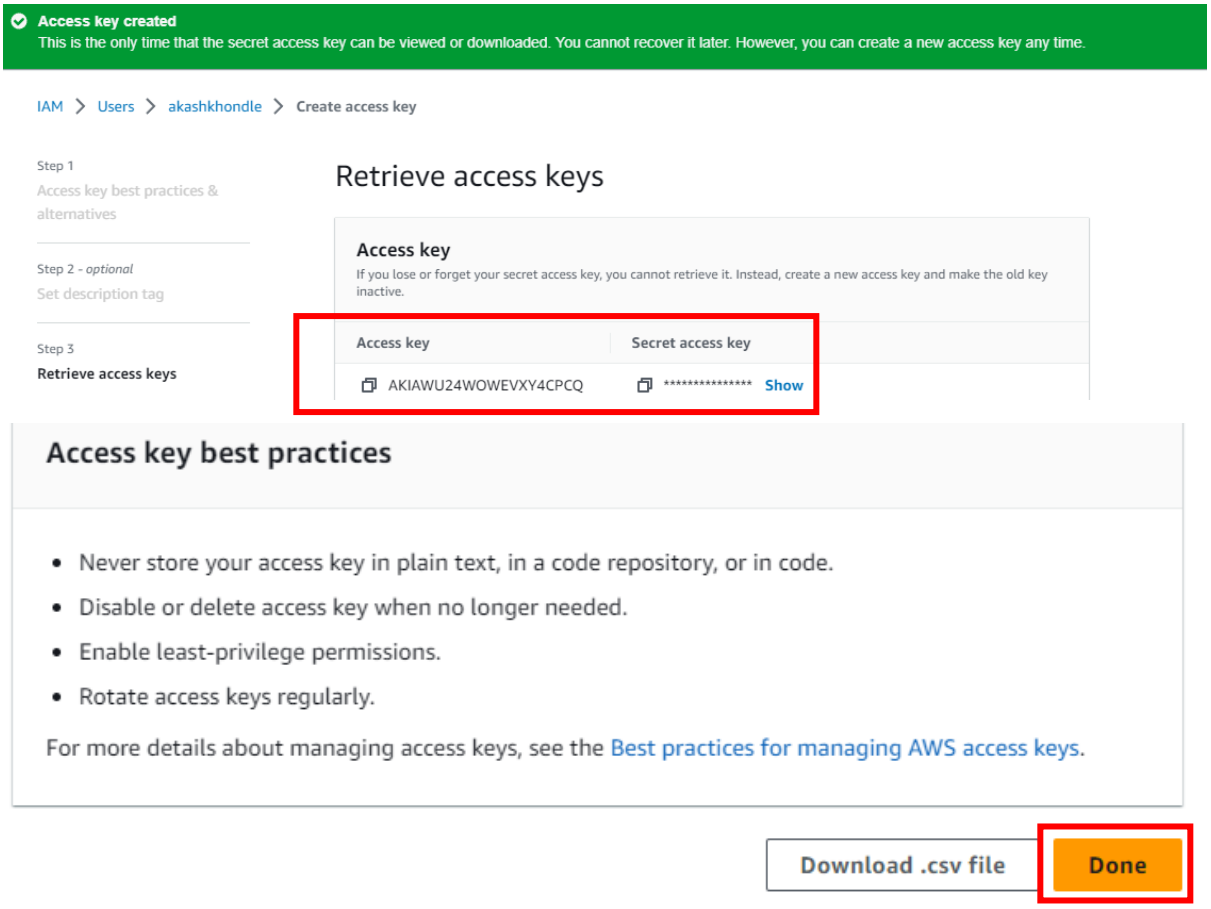

Step 13] Choose Command Line interface and click on next.



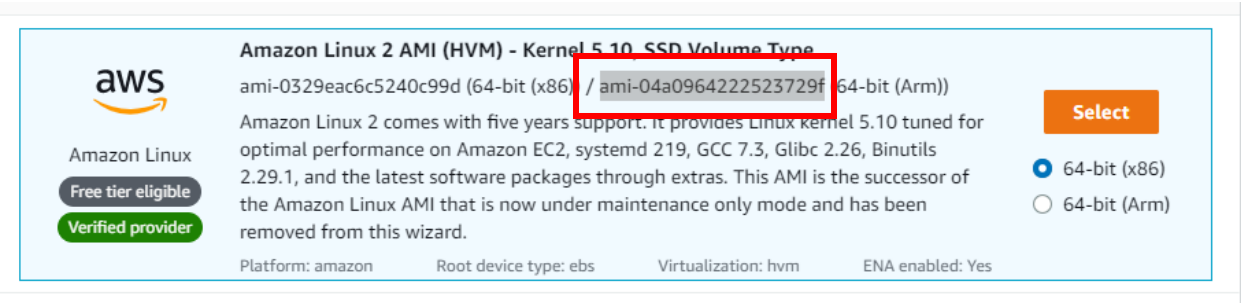

Step 14] Create the **Access Key** give the tag value in input box



Step 15] Copy the **Access key and Secret key Access** and paste in the code. Then click on Done.




Step 16] Copy the **ami img.**




Step 17] Now type **code in the files we c**reated.

instance.tf × provider.tf

instance.tf

```
resource "aws_instance" "web" {
  ami           = "ami-0329eac6c5240c99d"
  instance_type = "t2.micro"
  tags = {
    Name = "Launch-EC2-Manually"
  }
}
```

instance.tf provider.tf ×

provider.tf

```
provider "aws" {
  region     = "ap-northeast-1"
  access_key = "AKIAWU24WOWE4HX2WBGC"
  secret_key = "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"
}
```

## Commands to create instance using terraform

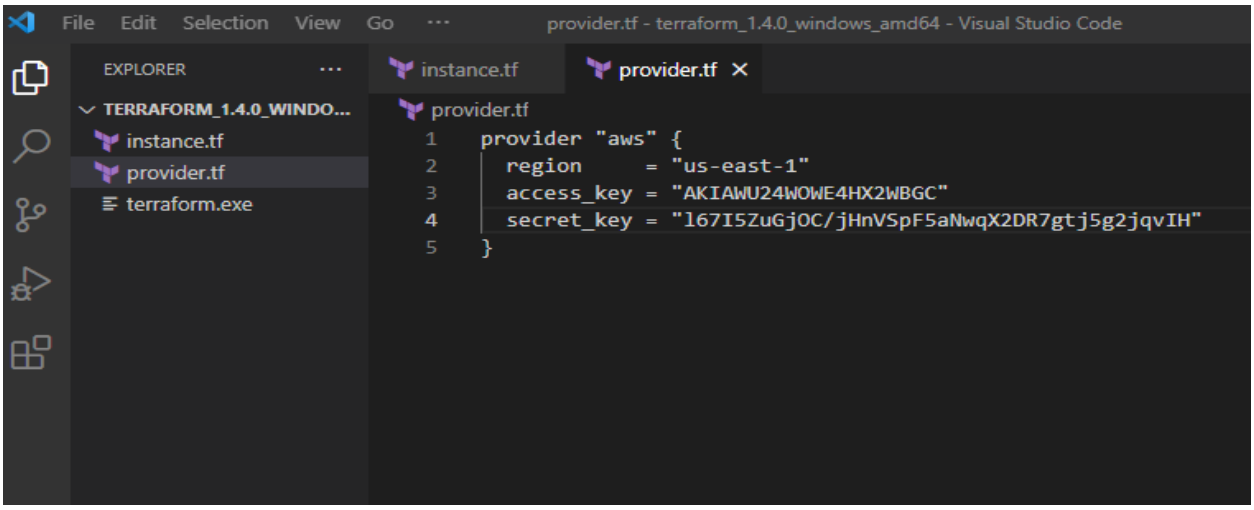
Run terraform command in the folder where files are created

- **Terraform fmt** will beautify the code

```
C:\terraform_instance>terraform fmt
instance.tf
provider.tf

C:\terraform_instance>
```

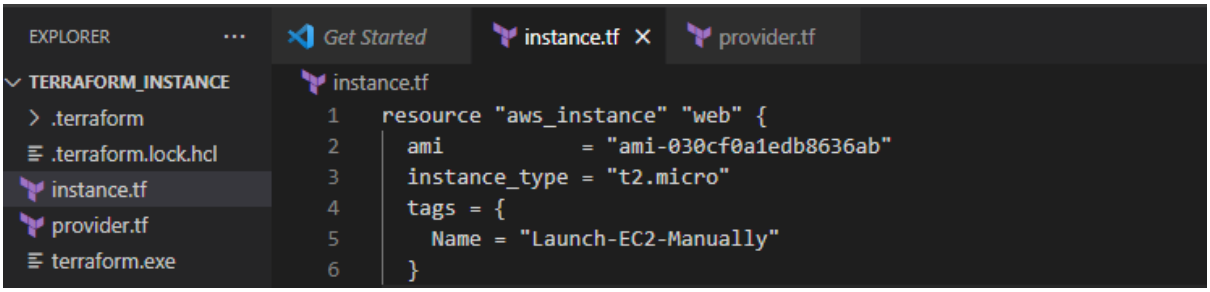
- **Terraform providers** will show the cloud providers

```
C:\terraform_instance>terraform providers

Providers required by configuration:
.
└── provider[registry.terraform.io/hashicorp/aws]


C:\terraform_instance>
```




- **terraform init** command initializes a working directory containing Terraform configuration files.

```
C:\terraform_instance>terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v4.58.0...
- Installed hashicorp/aws v4.58.0 (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.

C:\terraform_instance>
```



- **terraform plan** command creates an execution plan, which lets you preview the changes that **Terraform plans** to make to your infrastructure

```
C:\terraform_instance>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.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-030cf0a1edb8636ab"
      + 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                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
```

- **terraform apply** command performs a plan just like terraform plan does, but then actually carries out the planned changes to each resource using the relevant infrastructure provider's API.

```
C:\terraform_instance>terraform apply --auto-approve

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.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-030cf0a1edb8636ab"
      + 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                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
```

```
      + tags_all                                   = {
          + "Name" = "Launch-EC2-Manually"
        }
      + tenancy                                    = (known after apply)
      + user_data                                  = (known after apply)
      + user_data_base64                           = (known after apply)
      + user_data_replace_on_change                = false
      + vpc_security_group_ids                     = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Creation complete after 36s [id=i-0b04ee7917097f91a]

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

C:\terraform_instance>
```

## New Instance Created