

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?

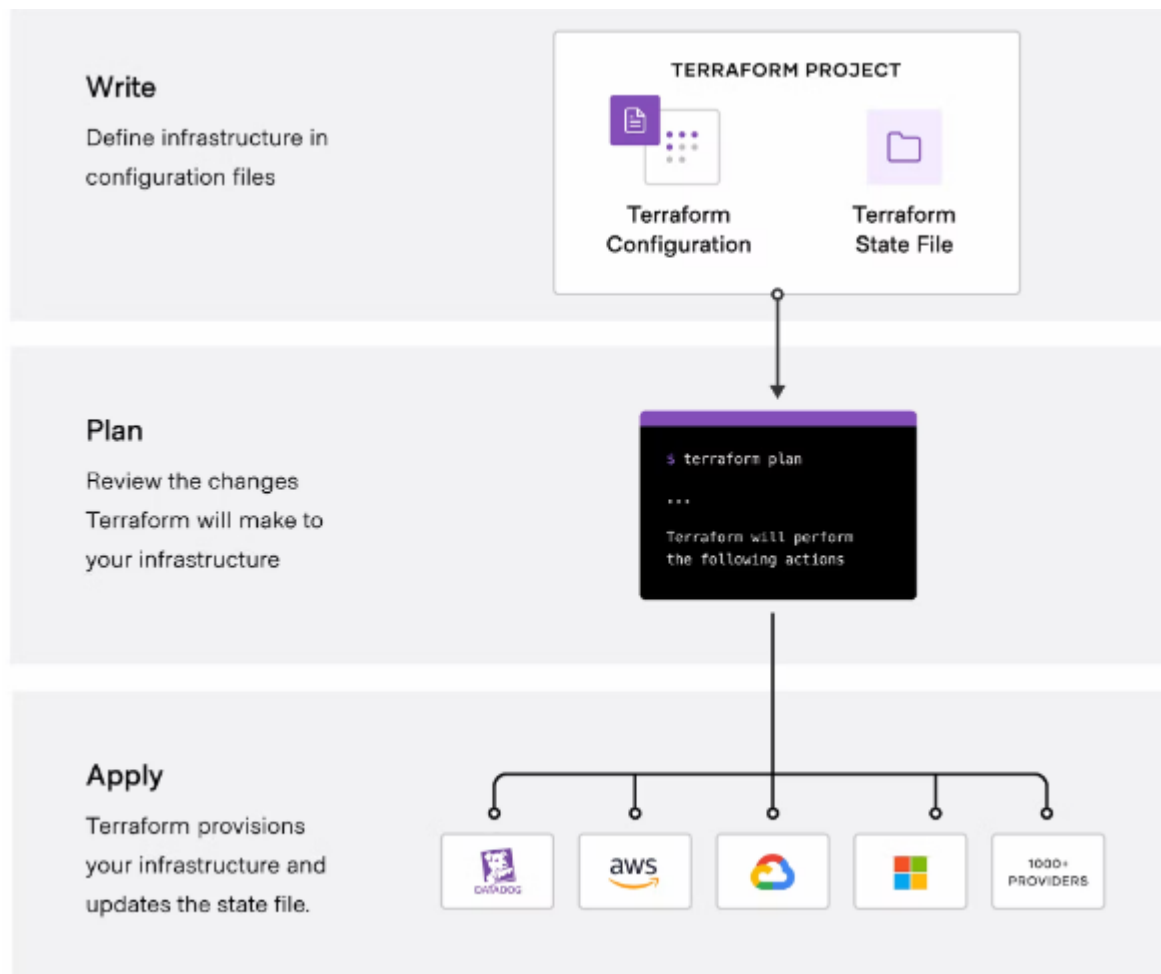
Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.



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:

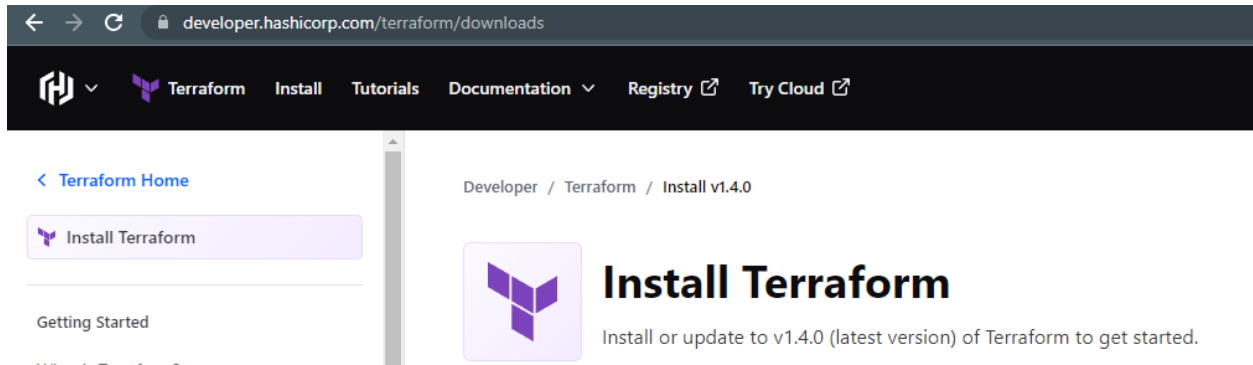
- **Write:** You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.
- **Plan:** Terraform creates an 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.



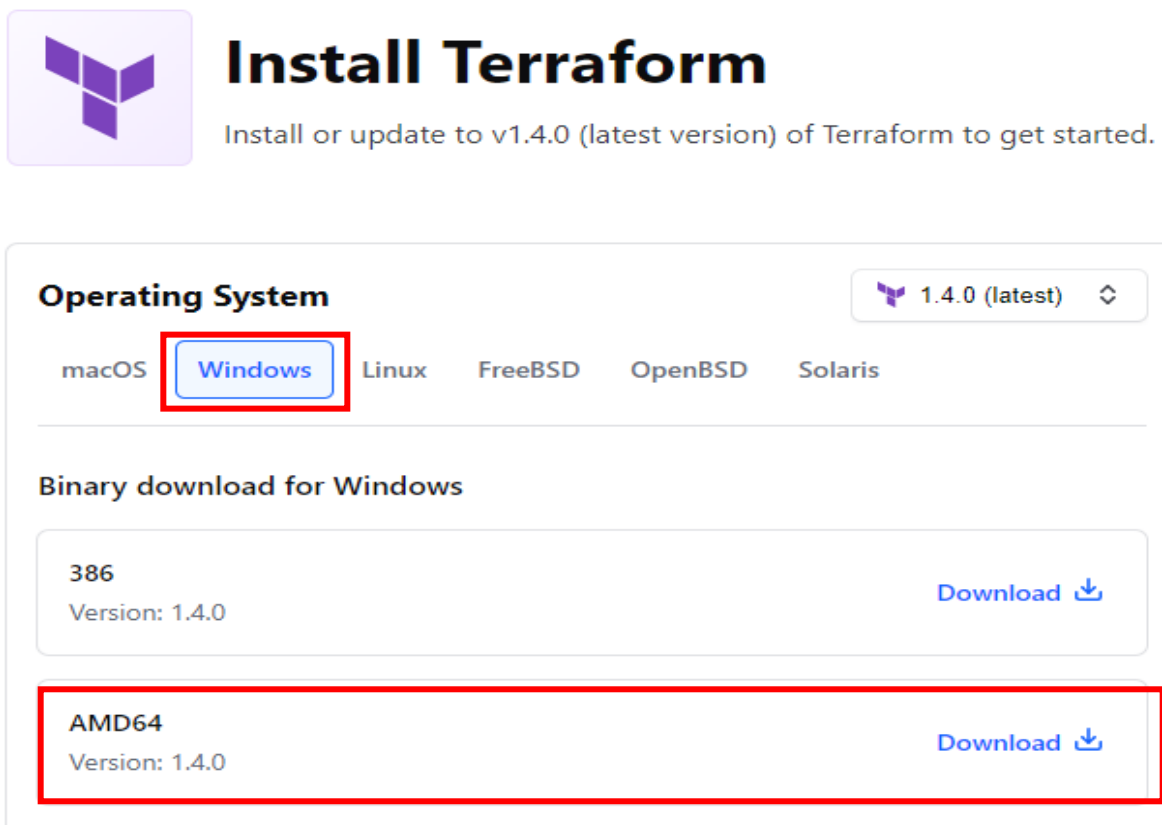
Step-by-step screenshot to install and configure Terraform

1. Download it from official site

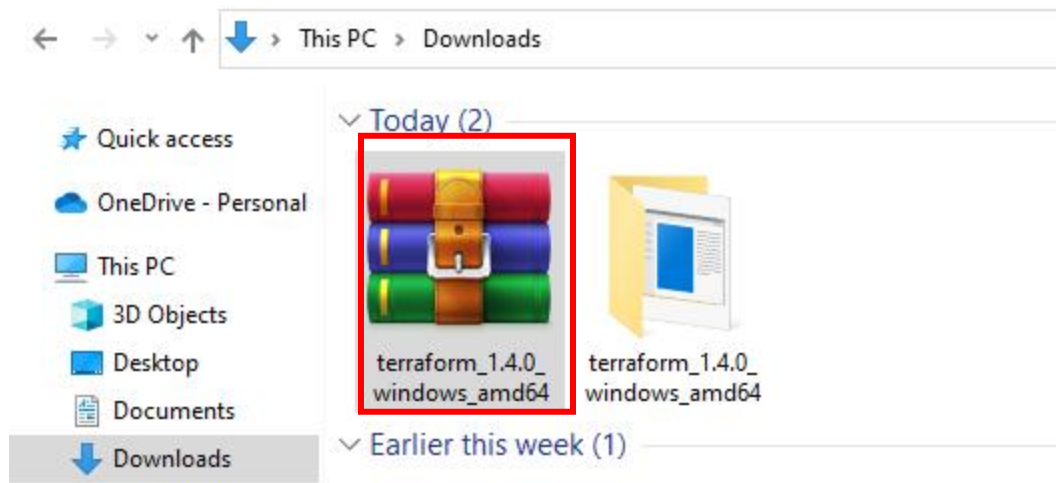
<https://developer.hashicorp.com/terraform/downloads>



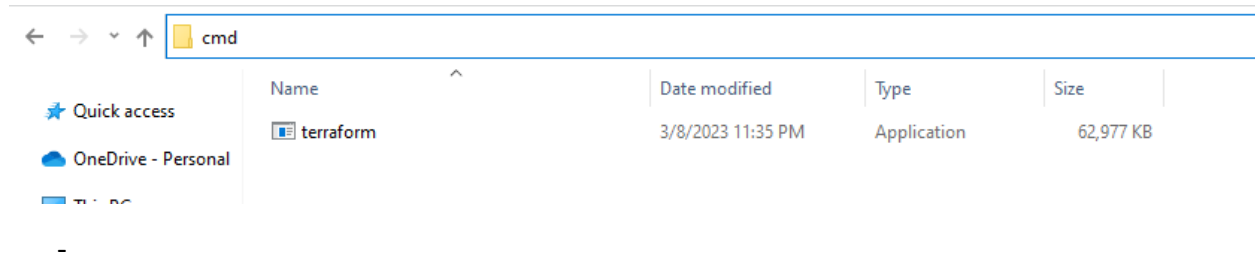
2. Select the OS to download



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

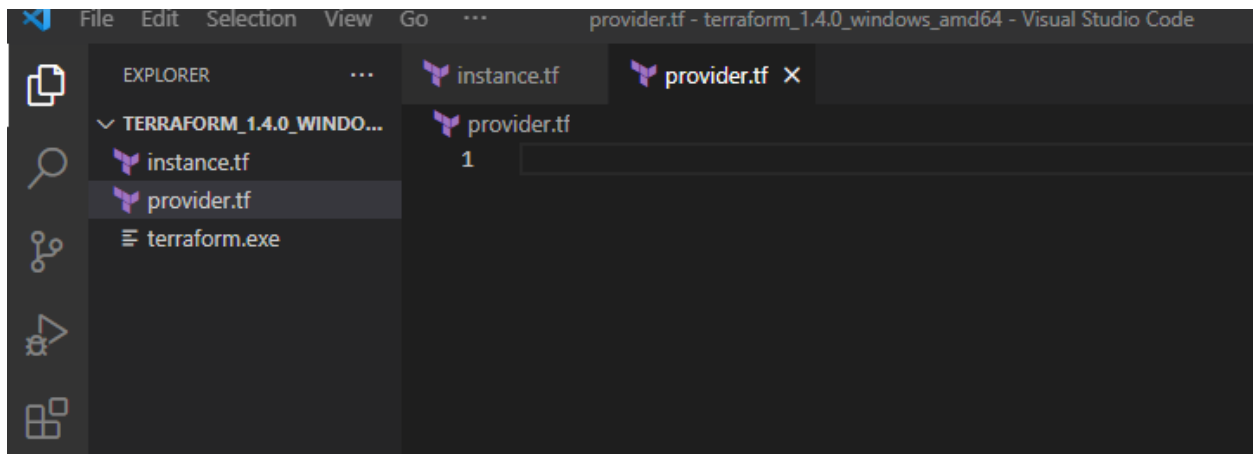
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2486]
(c) Microsoft Corporation. All rights reserved.

C:\terraform_instance>terraform -v
Terraform v1.4.0
on windows_amd64

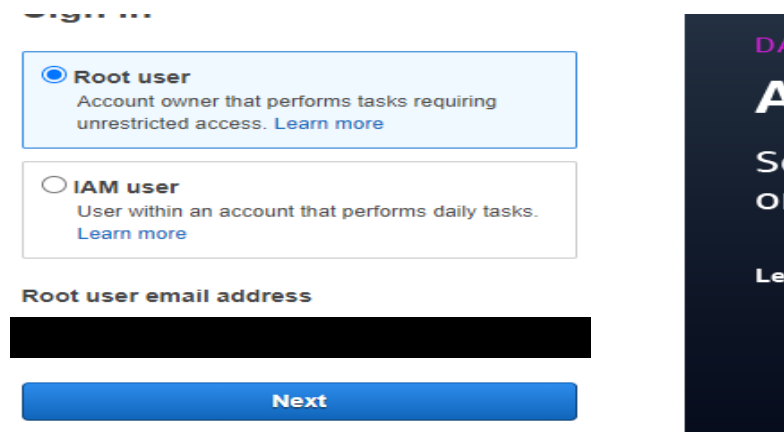
C:\terraform_instance>
```

Terraform script to create Infrastructure on any cloud platform

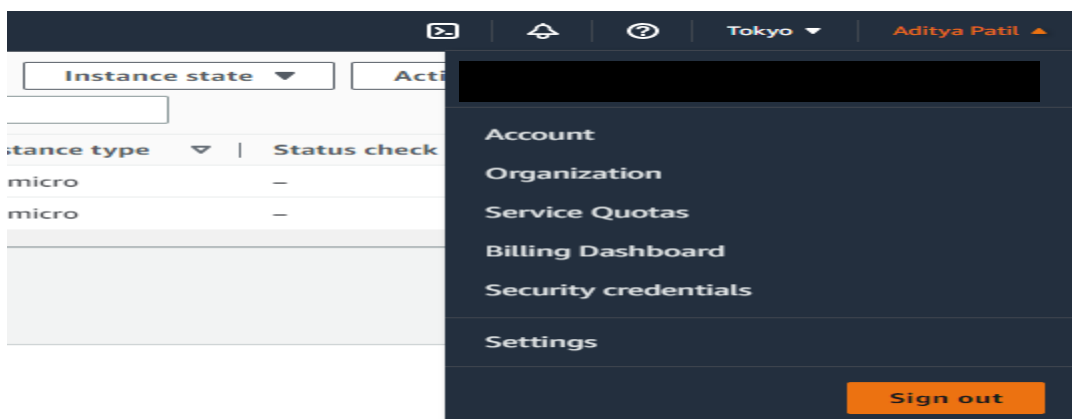
Step 1] Create two files as mention below **instance.tf** and **provider.tf**



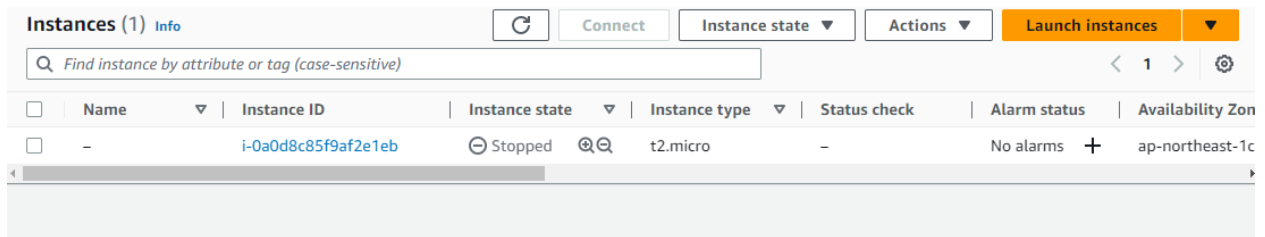
Step 2] Login into aws console.



Step 3] Login Successful



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

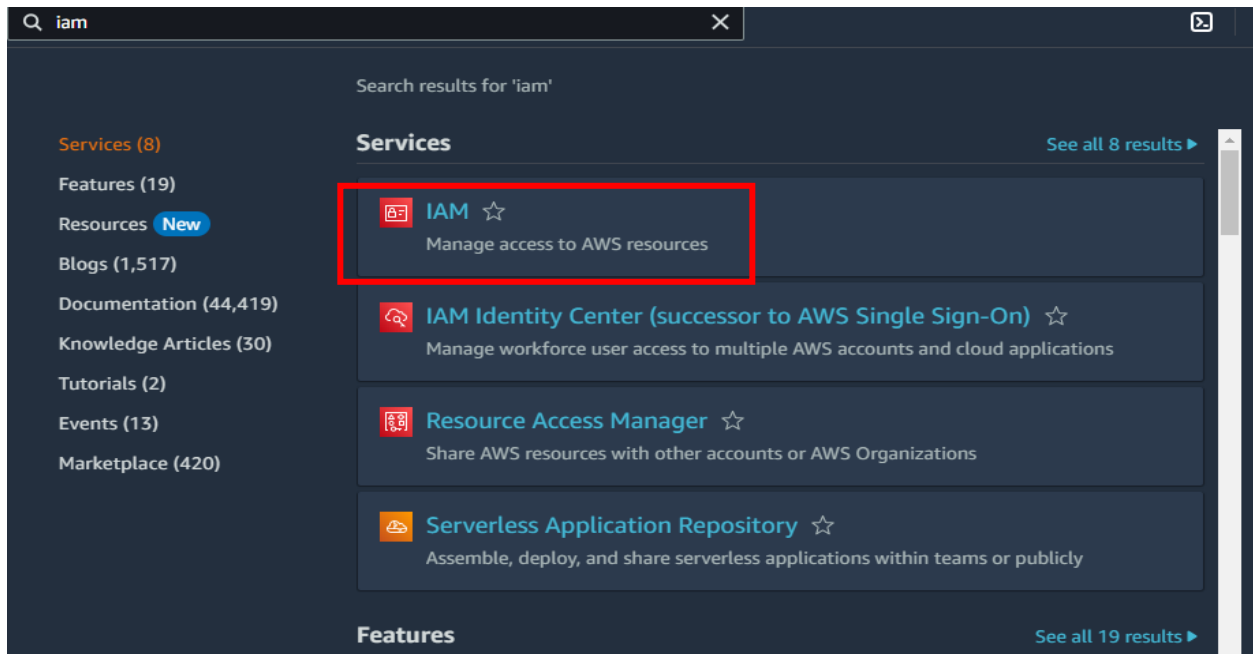


Instances (1) Info

Find instance by attribute or tag (case-sensitive)

| | Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Availability Zone |
|--------------------------|------|---------------------|----------------|---------------|--------------|--------------|-------------------|
| <input type="checkbox"/> | - | i-0a0d8c85f9af2e1eb | Stopped | t2.micro | - | No alarms | ap-northeast-1c |

Step 5] Go to IAM to create a new user



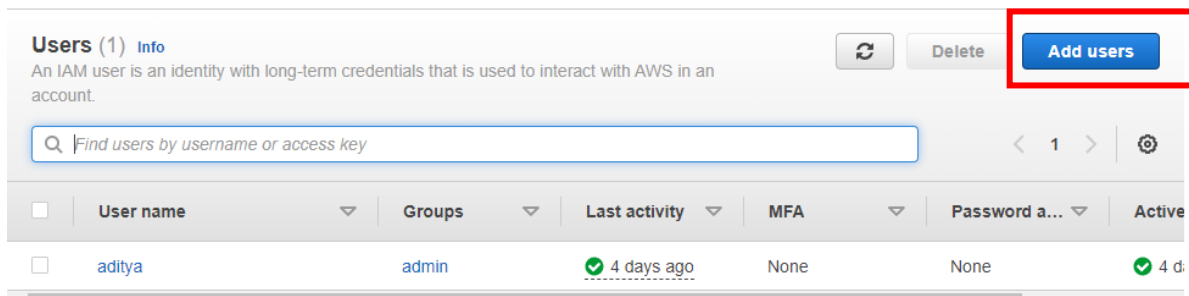
Search results for 'iam'

Services (8) See all 8 results ▶

- IAM** ☆
Manage access to AWS resources
- IAM Identity Center (successor to AWS Single Sign-On)** ☆
Manage workforce user access to multiple AWS accounts and cloud applications
- Resource Access Manager** ☆
Share AWS resources with other accounts or AWS Organizations
- Serverless Application Repository** ☆
Assemble, deploy, and share serverless applications within teams or publicly

Features (19) See all 19 results ▶

Step 6] Click on the add User.



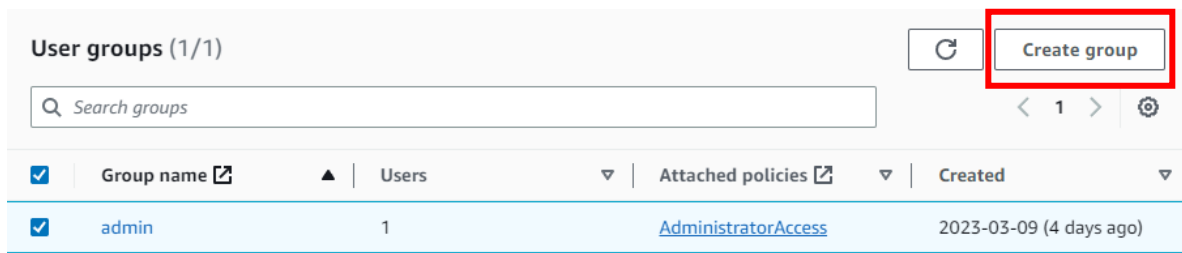
Users (1) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

Find users by username or access key

| | User name | Groups | Last activity | MFA | Password a... | Active |
|--------------------------|-----------|--------|---------------|------|---------------|--------|
| <input type="checkbox"/> | aditya | admin | 4 days ago | None | None | 4 d |

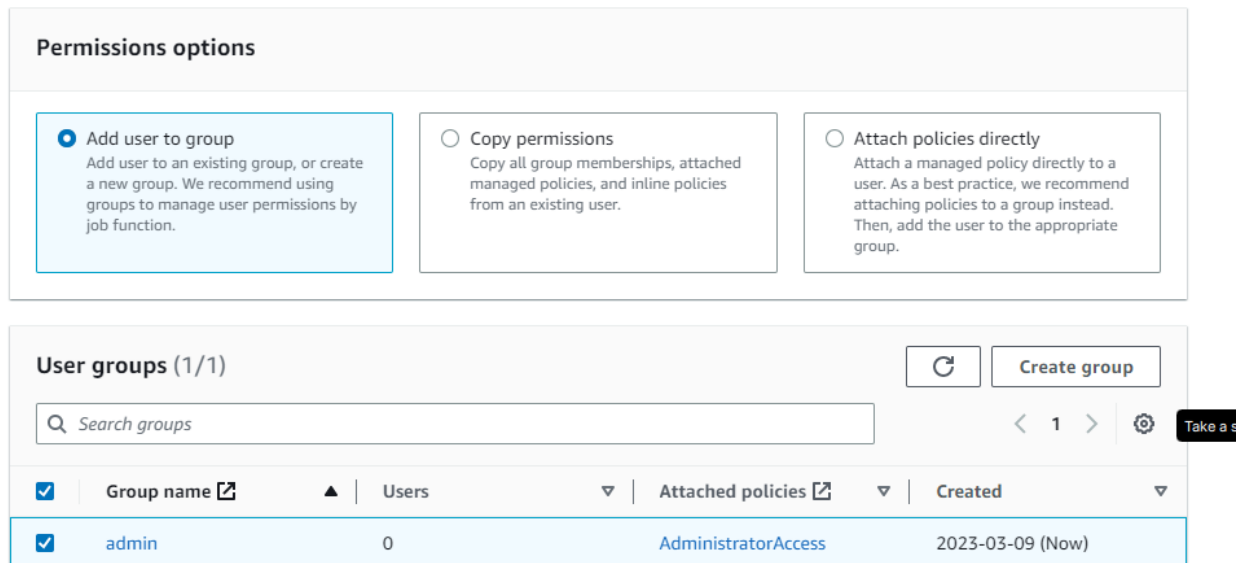
Step 7] As there is no user group lets create group of admin.



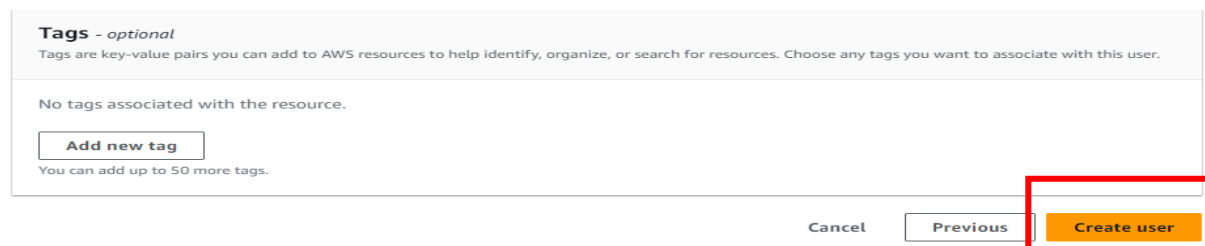
Step 8] User group created add new user to **admin** group

Set permissions

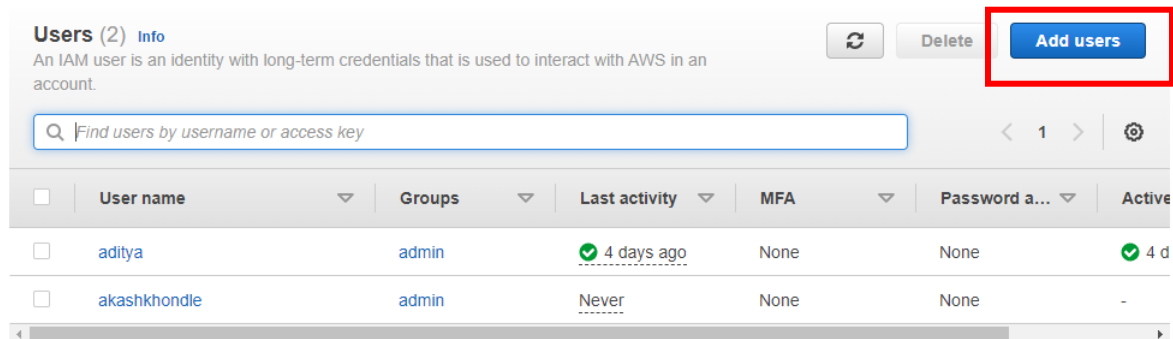
Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)



Step 9] Click on **Create User**



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



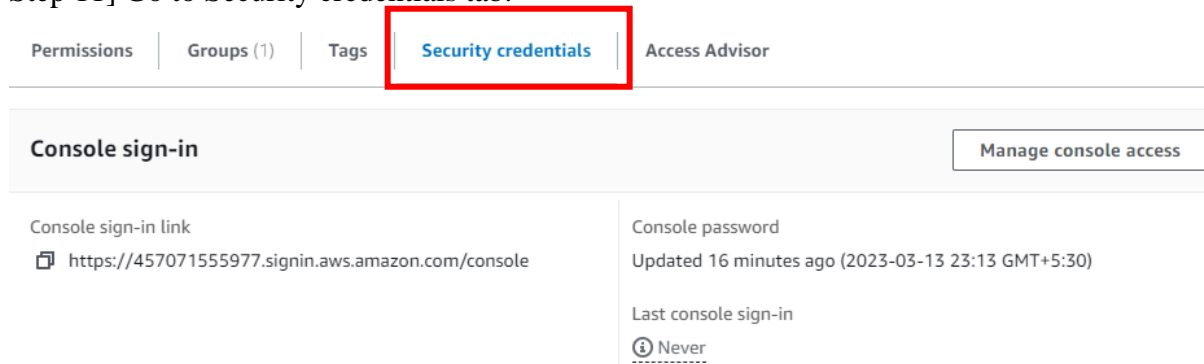
Users (2) [Info](#)

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

[Refresh](#) [Delete](#) [Add users](#)

| <input type="checkbox"/> | User name | Groups | Last activity | MFA | Password a... | Active |
|--------------------------|------------------------------|-----------------------|---------------|------|---------------|--------|
| <input type="checkbox"/> | aditya | admin | ✓ 4 days ago | None | None | ✓ 4 d |
| <input type="checkbox"/> | akashkhondle | admin | Never | None | None | - |

Step 11] Go to Security credentials tab.

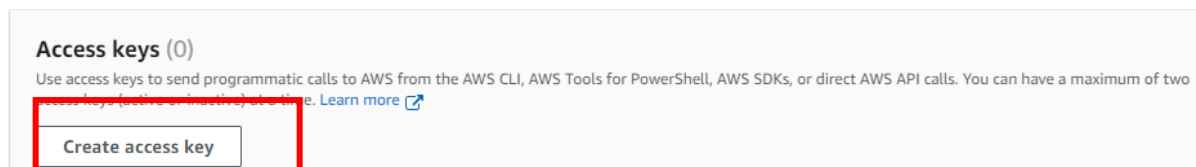


[Permissions](#) | [Groups \(1\)](#) | [Tags](#) | **[Security credentials](#)** | [Access Advisor](#)

Console sign-in [Manage console access](#)

| | |
|---|--|
| Console sign-in link https://457071555977.signin.aws.amazon.com/console | Console password Updated 16 minutes ago (2023-03-13 23:13 GMT+5:30) |
| | Last console sign-in ⓘ Never |

Step 12] Click on Create access Key button.



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](#)

[Create access key](#)

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

Access key best practices & alternatives

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

☒ **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.

☐ I understand the above recommendation and want to proceed to create an access key.

CancelNext

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

Set description tag - *optional*

The description for this access key will be attached to this user as a tag and shown alongside the access key.

Description tag value
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.

localkey

Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ . : / = + - @

CancelPreviousCreate access key

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

✓ **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.

IAM > Users > akashkhondle > Create access key

Step 1
Access key best practices & alternatives

Step 2 - optional
Set description tag

Step 3
Retrieve access keys

Retrieve access keys

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.

| Access key | Secret access key |
|----------------------|----------------------------|
| AKIAWU24WOWEVXY4CPCQ | ***** Show |

Access key best practices

- 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.

For more details about managing access keys, see the [Best practices for managing AWS access keys](#).

[Download .csv file](#) [Done](#)

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

Amazon Linux
Free tier eligible
Verified provider

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type

ami-0329eac6c5240c99d (64-bit (x86)) / **ami-04a0964222523729f** (64-bit (Arm))

Amazon Linux 2 comes with five years support. It provides Linux kernel 5.10 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is now under maintenance only mode and has been removed from this wizard.

Platform: amazon Root device type: ebs Virtualization: hvm ENA enabled: Yes

[Select](#)

☒ 64-bit (x86)
☐ 64-bit (Arm)

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

```
instance.tf × provider.tf
instance.tf
1 resource "aws_instance" "web" {
2     ami           = "ami-0329eac6c5240c99d"
3     instance_type = "t2.micro"
4     tags = {
5         Name = "Launch-EC2-Manually"
6     }
7 }
```

```
instance.tf provider.tf ×
provider.tf
1 provider "aws" {
2     region     = "ap-northeast-1"
3     access_key = "AKIAWU24WOWE4HX2WBGC"
4     secret_key = "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"
5 }
```

Commands to create instance using terraform

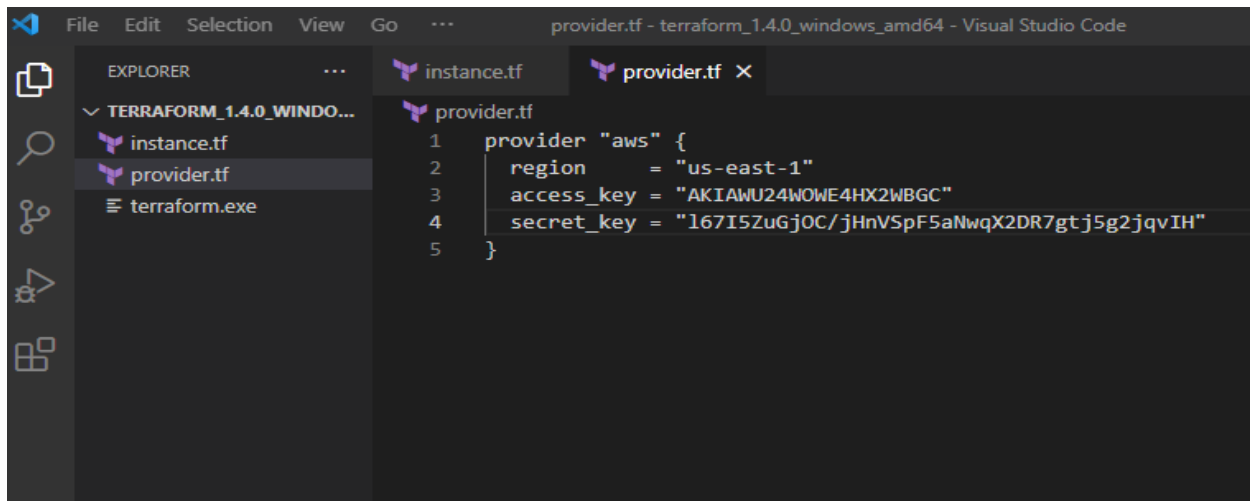
Run terraform command in the folder where files are created

- **Terraform fmt** will beautify the code

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

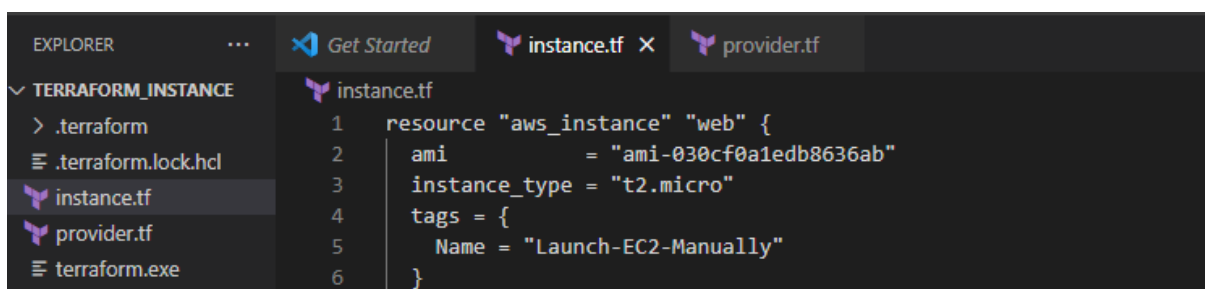
- **Terraform providers** will show the cloud providers

```
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

| Instances (3) Info | | | | | | | |
|---|-----------------|---------------------|----------------|---------------|-------------------|--------------|-------------------|
| <input type="text" value="Find instance by attribute or tag (case-sensitive)"/> | | | | | | | |
| <input type="checkbox"/> | Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Availability Zone |
| <input type="checkbox"/> | - | i-0a0d8c85f9af2e1eb | Stopped | t2.micro | - | No alarms | ap-northeast-1c |
| <input type="checkbox"/> | Launch-EC2-M... | i-0b04ee7917097f91a | Running | t2.micro | 2/2 checks passed | No alarms | ap-northeast-1c |
| <input type="checkbox"/> | Launch-EC2-M... | i-0c0f920870e9d5ce6 | Stopped | t2.micro | - | No alarms | ap-northeast-1c |