

# EC2-Creation

## Install Terraform on 'Amazon linux'

1. `sudo yum install -y yum-utils shadow-utils`
2. `sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo`
3. `sudo yum -y install terraform`

Steps for create EC2

1. Create script file by extension '.tf'

```
provider "aws" {  
  access_key = "AKIATM4VCLAT5UMA6KN5"  
  secret_key = "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"  
  region = "ap-south-1"  
}  
  
resource "aws_instance" "ec2_instance" {  
  ami = "ami-03f4878755434977f"  
  instance_type = "t2.micro"  
  key_name = "terraform"  
}
```

@ access key & secret key create by creating IAM user

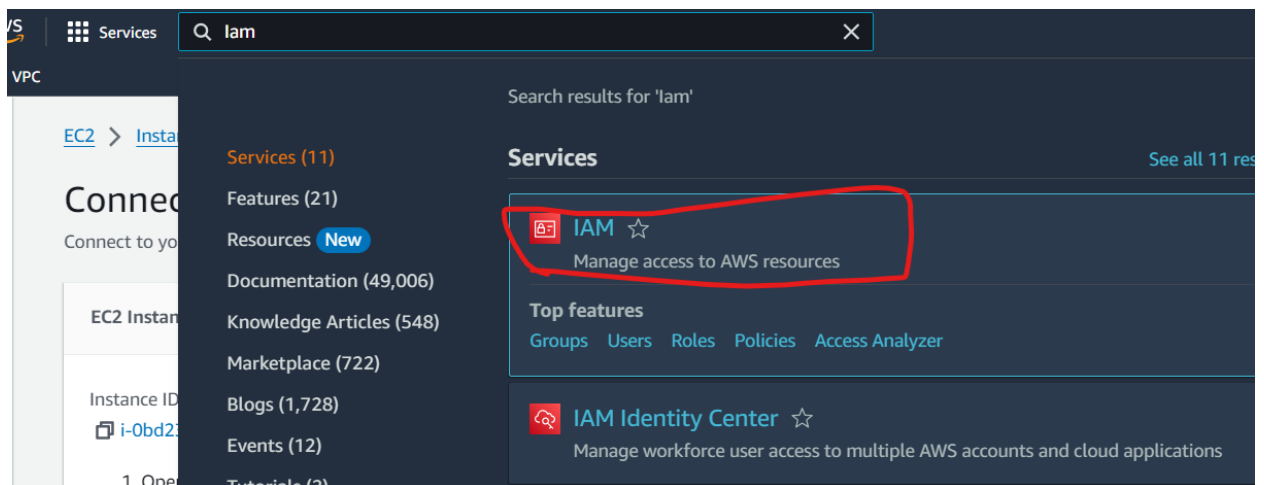
@ ami - O.S image id

@ instance type - ram

@ key name - key pair

→ Create access key:-

1. Go to IAM in AWS



2. Create user from left side

The image consists of two screenshots of the AWS IAM console interface. The top screenshot shows the 'IAM Dashboard' with a sidebar on the left. In the sidebar, under 'Access management', the 'Users' link is highlighted with a red box and a red arrow points from it to the 'Security recommendations' section in the main content area. The bottom screenshot shows the 'Users' page, which is currently empty. A red box highlights the 'Create user' button in the top right corner of the main content area, with a red arrow pointing to it from below.

3. Write name and click save

## Specify user details

### User details

User name

terraform

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.

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

User created

- Set permission by follow below steps and click 'next'

IAM > Users > Create user

Step 1  
[Specify user details](#)

Step 2  
**Set permissions**

Step 3  
[Review and create](#)

### 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](#)

☐ 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.

☒ AdministratorAccess

☐ AccessAnalyzerServiceRolePolicy

☐ AdministratorAccess-Amplify

☐ AdministratorAccess-AWSElasticBea...

☐ AlexaForBusinessDeviceSetup

☐ AlexaForBusinessFullAccess

☐ AlexaForBusinessGatewayExecution

Type

AWS managed

AWS managed - job function

AWS managed

AWS managed

AWS managed

AWS managed

AWS managed

Attached entities

0

0

0

0

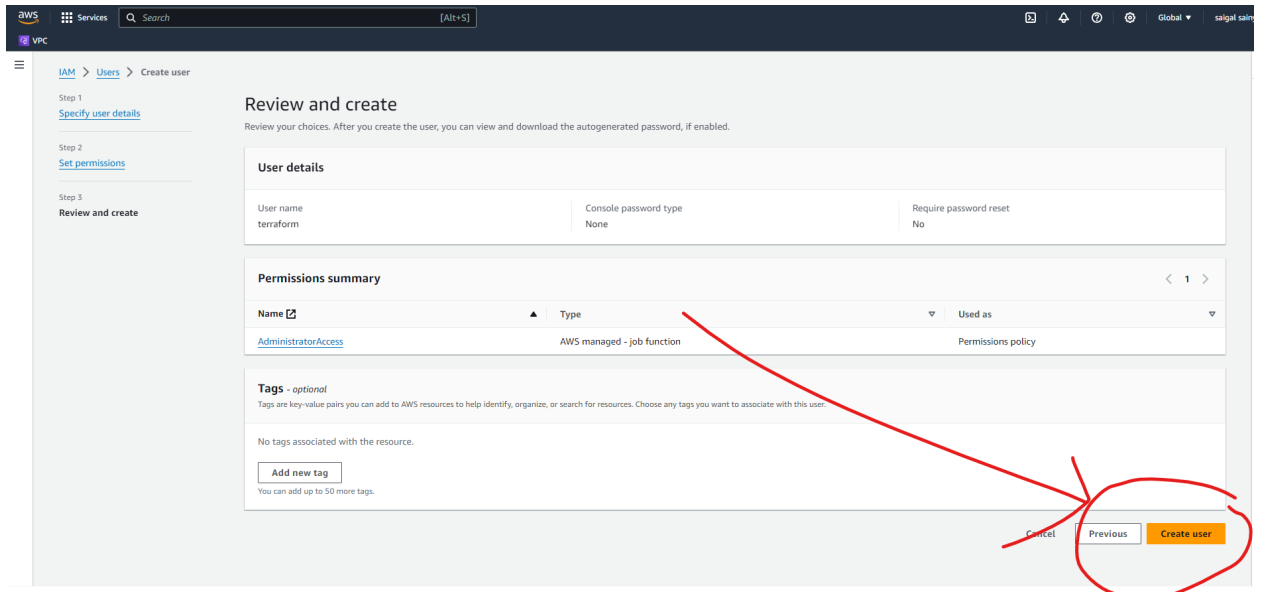
0

0

0

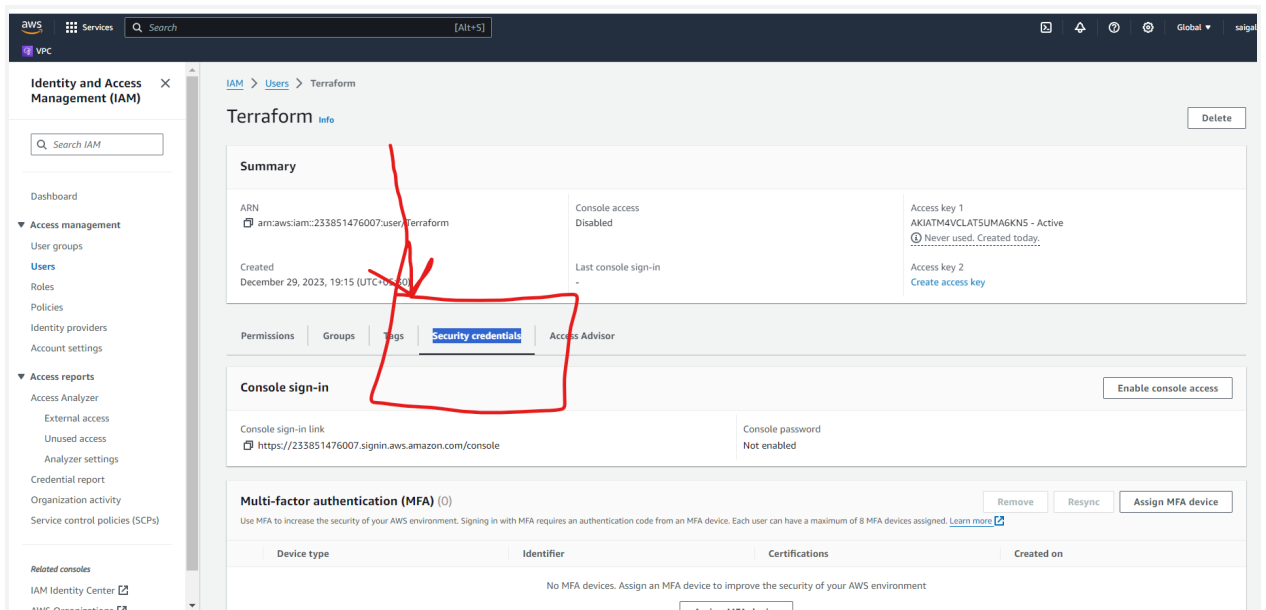
Create policy

- Create user by clicking 'create user'



User created

6. Create secret key for user  
→ click on user and go into **Security credentials**



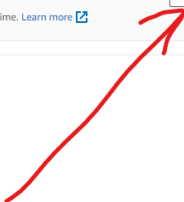
7. Scroll down and click 'create access key'

**Access keys (1)**

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](#)

<b>AKIATM4VCLAT5UMA6KN5</b>	Status Active
Description -	Created 14 minutes ago
Last used None	Last used service N/A
Last used region N/A	

Actions



## 8. Select first option and check documentation then click 'save'

**Access key best practices & alternatives**

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

Use case

- ☒ **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.
- ☐ Application running outside AWS  
You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.
- ☐ Other  
Your use case is not listed here.

**Alternatives recommended**

- Use [AWS CloudShell](#), a browser-based CLI, to run commands. [Learn more](#)
- Use the [AWS CLI V2](#) and enable authentication through a user in IAM Identity Center. [Learn more](#)

Confirmation

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

Cancel Next

**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.

**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.

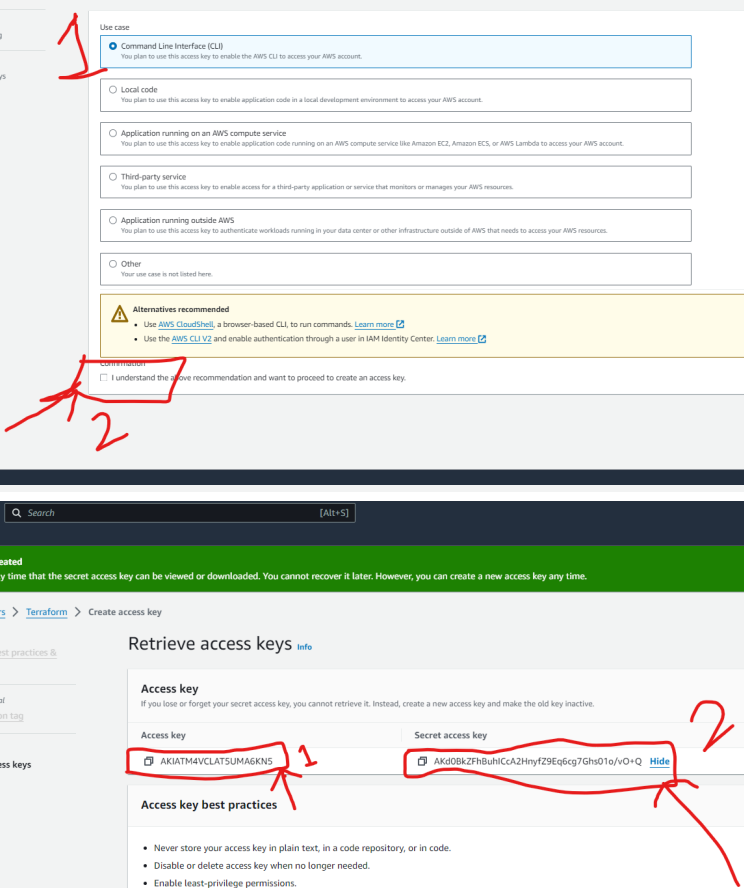
Access key	Secret access key
AKIATM4VCLAT5UMA6KN5	AKJ0BKZFhBuHCCa2HnyfZ9Eq6cg7Ghs0ta/vO+Q <a href="#">Hide</a>

**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



'Access key' and 'secret key' created

## Run commands for create environment

1. Run this command for install plugins for create environment

→terraform init

```
[root@ip-172-31-44-69 ~]# terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- 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.
[root@ip-172-31-44-69 ~]#
```

2. Run this command is used to see the changes that will take place on the infrastructure.

→terraform plan

```
[root@ip-172-31-44-69 ~]# 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.ec2_instance will be created
+ resource "aws_instance" "ec2_instance" {
  + ami                        = "ami-03f4878755434977f"
  + 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_lifecycle         = (known after apply)
  + instance_state             = (known after apply)
  + instance_type              = "t2.micro"
  + ipv6_address_count         = (known after apply)
  + ipv6_addresses             = (known after apply)
  + key_name                   = "terraform"
  + monitoring                 = (known after apply)
  + outpost_arn                = (known after apply)
  + password_data              = (known after apply)
  + placement_group            = (known after apply)
}
```

3. Run this command for apply script and create infrastructure  
→ terraform apply

```
[root@ip-172-31-44-69 ~]# terraform apply

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.ec2_instance will be created
+ resource "aws_instance" "ec2_instance" {
+   ami                        = "ami-03f4878755434977f"
+   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_lifecycle         = (known after apply)
+   instance_state            = (known after apply)
+   instance_type              = "t2.micro"
+   ipv6_address_count         = (known after apply)
+   ipv6_addresses            = (known after apply)
+   key_name                   = "terraform"
+   monitoring                 = (known after apply)
+   outpost_arn               = (known after apply)
+   password_data              = (known after apply)
+   placement_group            = (known after apply)
+   ipv6_address_count         = (known after apply)
+   ipv6_addresses            = (known after apply)
+   key_name                   = "terraform"
+   monitoring                 = (known after apply)
+   outpost_arn               = (known after apply)
+   password_data              = (known after apply)
+   placement_group            = (known after apply)
+   placement_partition_number = (known after apply)
+   primary_network_interface_id = (known after apply)
+   private_dns                = (known after apply)
+   private_ip                 = (known after apply)
+   public_dns                 = (known after apply)
+   public_ip                  = (known after apply)
+   secondary_private_ips      = (known after apply)
+   security_groups             = (known after apply)
+   source_dest_check           = true
+   spot_instance_request_id    = (known after apply)
+   subnet_id                   = (known after apply)
+   tags_all                    = (known after apply)
+   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.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: 
```

Type 'yes'

```
aws
Services
Search [Alt+S]

VPC

+ source_dest_check           = true
+ spot_instance_request_id    = (known after apply)
+ subnet_id                   = (known after apply)
+ tags_all                    = (known after apply)
+ 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.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value yes

aws_instance.ec2_instance: Creating...
aws_instance.ec2_instance: Still creating... [10s elapsed]
aws_instance.ec2_instance: Still creating... [20s elapsed]
aws_instance.ec2_instance: Still creating... [30s elapsed]
aws_instance.ec2_instance: Creation complete after 32s [id=i-0f0aca040d91a8dff]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
[root@ip-172-31-44-69 ~]#
```

EC2 created

Instances (1/2) [Info](#)

Connect

Instance state ▾

Actions ▾

Launch instances

Find Instance by attribute or tag (case-sensitive)

< 1 >

<div></div>	Name <div></div>	Instance ID	Instance state <div></div>	Instance type <div></div>	Status check	Alarm sta
<div></div>		i-0f0aca040d91a8dff	<div>Running</div>	t2.micro	<div>2/2 checks passed</div>	No alarms
<div></div>	terraform	i-0bd23769baa2e5a73	<div>Running</div>	t2.micro	<div>2/2 checks passed</div>	No alarms

## EC2-Deletion

## Run below command for delete EC2 via terraform script  
→terraform destroy



```
[root@ip-172-31-44-69 ~]# terraform destroy
aws_instance.ec2_instance: Refreshing state... [id=i-0f0aca040d91a8dff]

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

Terraform will perform the following actions:

# aws_instance.ec2_instance will be destroyed
- resource "aws_instance" "ec2_instance" {
  - ami                                = "ami-0a0f1259dd1c90938" -> null
  - arn                                = "arn:aws:ec2:ap-south-1:233851476007:instance/i-0f0aca040d91a8dff" -> null
  - associate_public_ip_address       = true -> null
  - availability_zone                 = "ap-south-1b" -> null
  - cpu_core_count                     = 1 -> null
  - cpu_threads_per_core              = 1 -> null
  - disable_api_stop                   = false -> null
  - disable_api_termination           = false -> null
  - ebs_optimized                     = false -> null
  - get_password_data                 = false -> null
  - hibernation                       = false -> null
  - id                                = "i-0f0aca040d91a8dff" -> null
  - instance_initiated_shutdown_behavior = "stop" -> null
  - instance_state                    = "running" -> null
  - instance_type                     = "t2.micro" -> null
  - volume_id                         = "vol-0814eec0040f106b8" -> null
  - volume_size                       = 8 -> null
  - volume_type                       = "gp3" -> null
}
```

Type 'yes'

```

- device_name      = "/dev/xvda" -> null
- encrypted        = false -> null
- iops              = 3000 -> null
- tags             = {} -> null
- throughput       = 125 -> null
- volume_id        = "vol-0814eec0040f106b8" -> null
- volume_size      = 8 -> null
- volume_type      = "gp3" -> 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

aws_instance.ec2_instance: Destroying... [id=i-0f0aca040d91a8dff]
aws_instance.ec2_instance: Still destroying... [id=i-0f0aca040d91a8dff, 10s elapsed]
aws_instance.ec2_instance: Still destroying... [id=i-0f0aca040d91a8dff, 20s elapsed]
aws_instance.ec2_instance: Destruction complete after 29s

Destroy complete! Resources: 1 destroyed.
[root@ip-172-31-44-69 ~]#
```