AUTOMATING MYSQL DATABASE PROVISIONING WITH TERRAFORM

PROBLEM STATEMENT:

Organizations often face challenges in efficiently provisioning MySQL databases within their infrastructure. Manual setups are time-consuming, error-prone, and lack scalability. There is a need for an automated solution that ensures consistency, reliability, and rapid deployment of MySQL databases.

USE-CASE SCENARIO:

Consider a scenario where an organization is expanding its online services and needs to deploy multiple MySQL databases to support growing data requirements. Manual setup would be impractical due to the risk of errors and time constraints. An automated solution using Terraform can provide a streamlined and repeatable process for provisioning MySQL databases across different environments.

TASKS TO BE PERFORMED:

➔ **Define Infrastructure as Code (IaC):** Create Terraform scripts to define the desired state of the MySQL infrastructure, specifying details like database instances, users, and access controls.

➔ **Terraform Initialization:** Initialize the Terraform configuration to set up the working directory and required plugins.

➔ **Terraform Plan:** Review the execution plan generated by Terraform to understand the changes it will make to the infrastructure.

➔ **Terraform Apply:** Apply the defined infrastructure changes to provision MySQL databases automatically.

➔ **Verify and Monitor:** Implement validation steps to ensure the successful deployment of databases and set up monitoring for ongoing management.

****

SOLUTION:

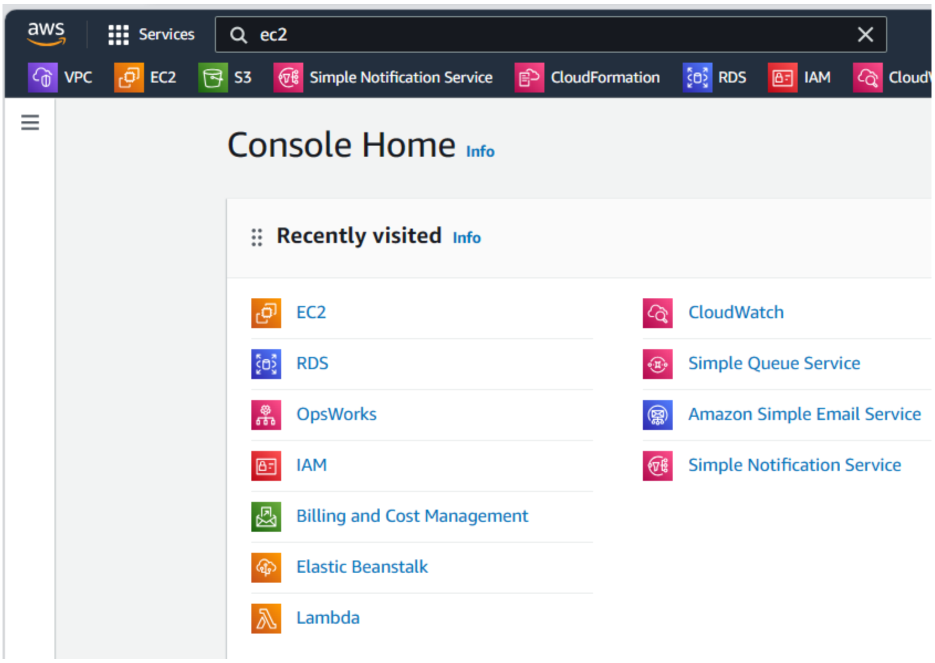
PRE-REQUIREMENTS:

- ➔ **Cloud** : AWS
- ➔ **IAC** : Terraform
- ➔ **Database** : AWS MySQL RDS Database
- ➔ **User** : IAM User

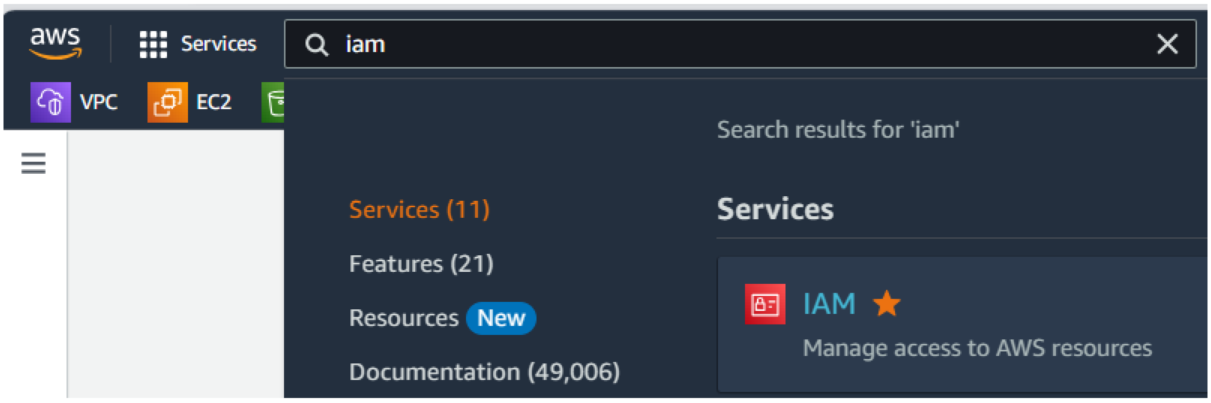
****

STEP:1 – CREATING AN IAM USER & ACCESS_KEY AND SECRET ACCESS KEY

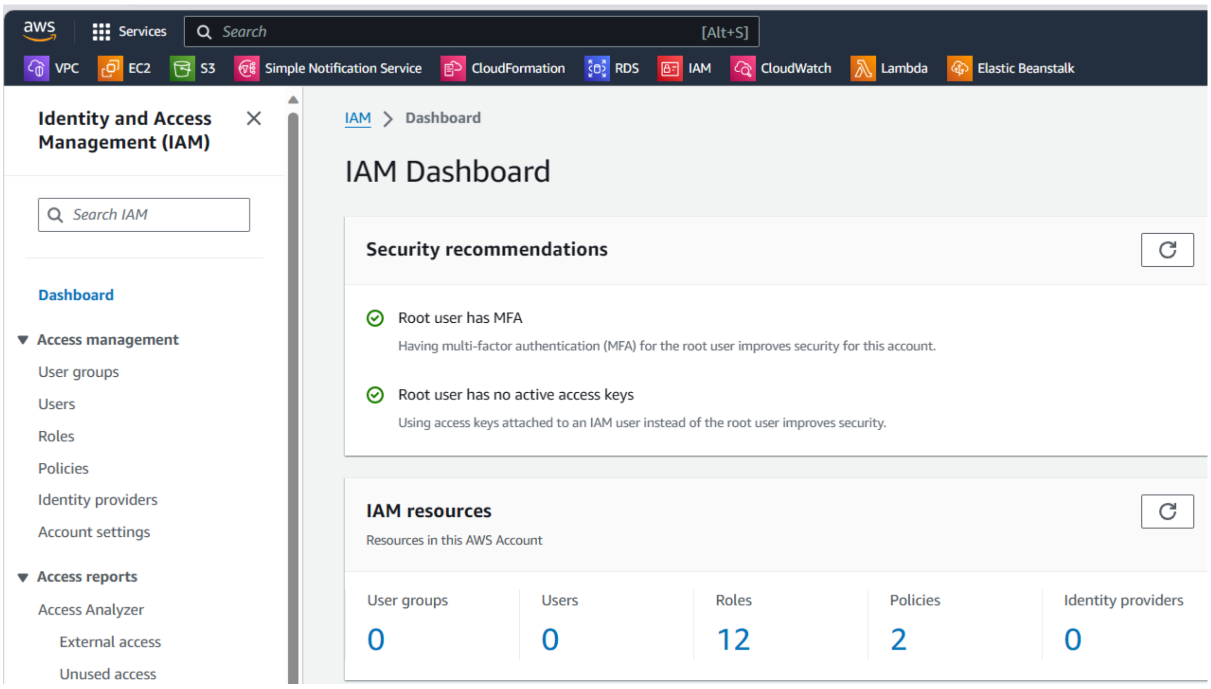
➔ First login into your **AWS Management console:**



➔ Then on the service panel, search **IAM** and click that one:



➔ Then on IAM Management console, we can able to find users click that one:



IAM Dashboard

Security recommendations

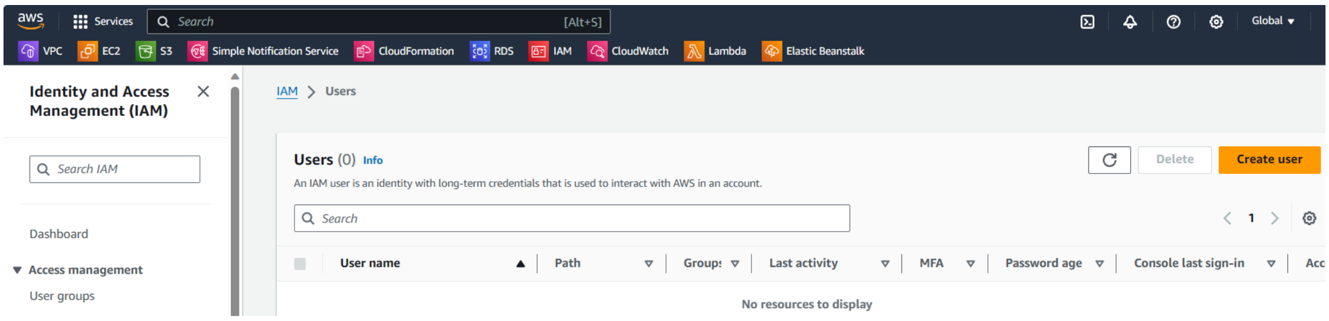
Root user has MFA
Having multi-factor authentication (MFA) for the root user improves security for this account.

Root user has no active access keys
Using access keys attached to an IAM user instead of the root user improves security.

IAM resources
Resources in this AWS Account

User groups	Users	Roles	Policies	Identity providers
0	0	12	2	0

➔ Then click create users:

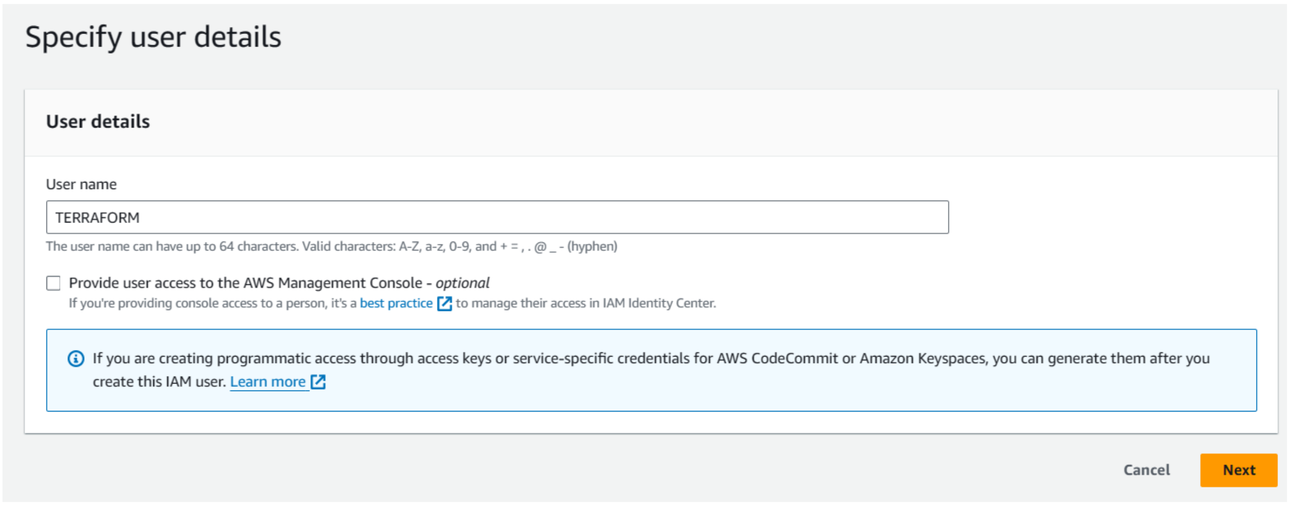


Users (0) Info
An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

	User name	Path	Groups	Last activity	MFA	Password age	Console last sign-in	Acc

No resources to display

➔ Then name the user and click next:



Specify user details

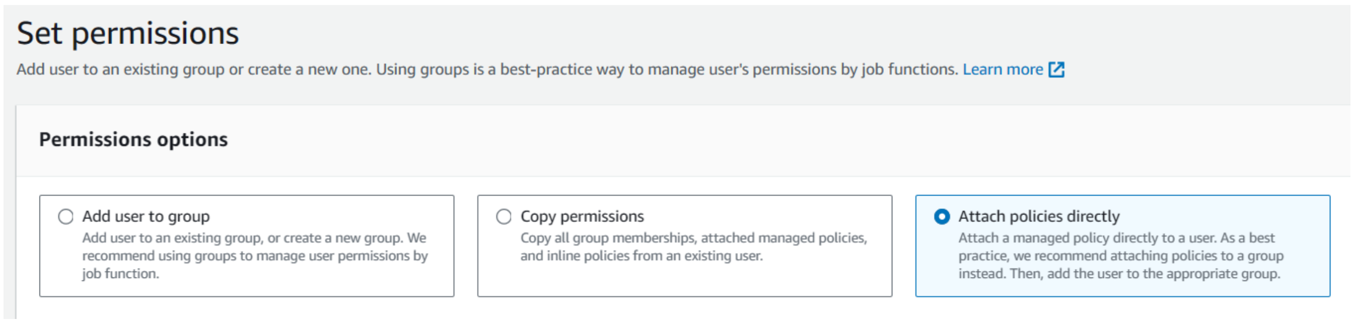
User details

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

➔ Then select the **attach policies directly option:**



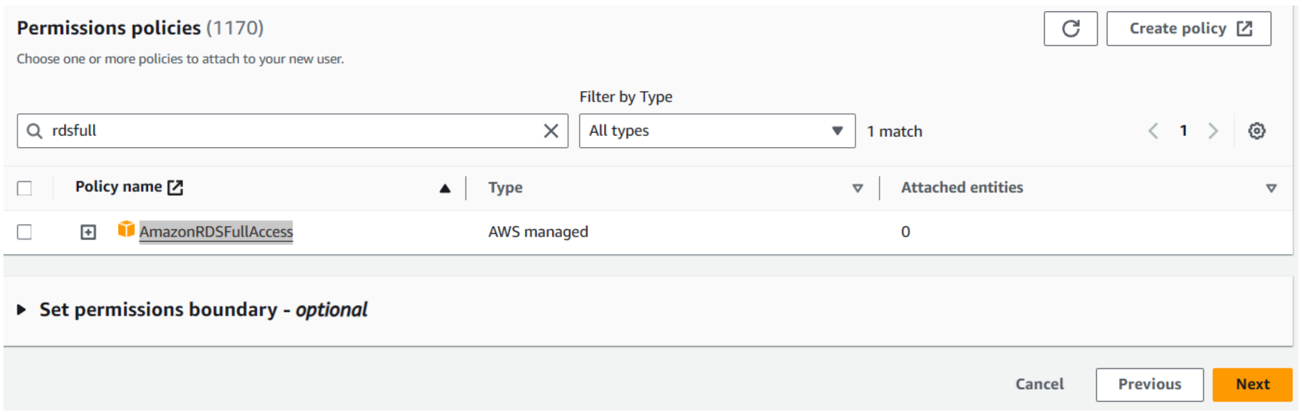
Set permissions

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.

➔ Then add AmazonRDSFullAccess & AmazonEC2FullAccess policy: click next



Permissions policies (1170)

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

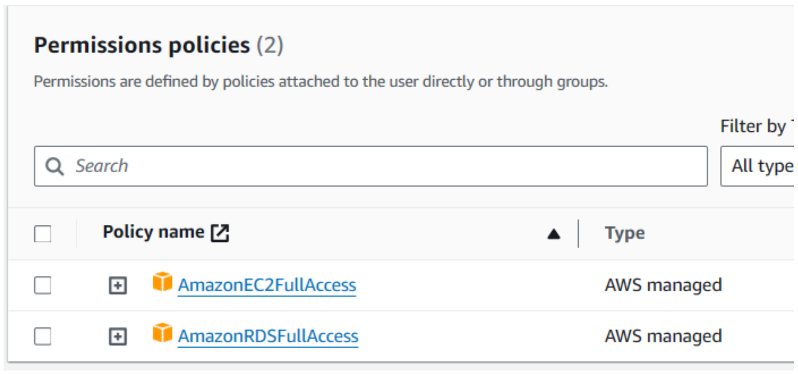
Filter by Type

rdsfull | All types | 1 match

	Policy name	Type	Attached entities
☐	AmazonRDSFullAccess	AWS managed	0

▸ **Set permissions boundary - *optional***

➔ Then just review and create page will appear just review the configurations, **click create user:**



Permissions policies (2)

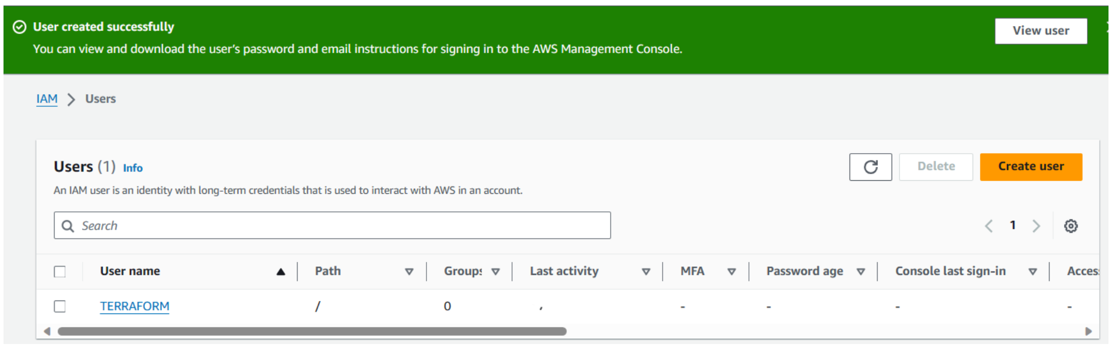
Permissions are defined by policies attached to the user directly or through groups.

Filter by

Search | All type

	Policy name	Type
☐	AmazonEC2FullAccess	AWS managed
☐	AmazonRDSFullAccess	AWS managed

➔ The IAM User has been created successfully.



User created successfully

You can view and download the user's password and email instructions for signing in to the AWS Management Console.

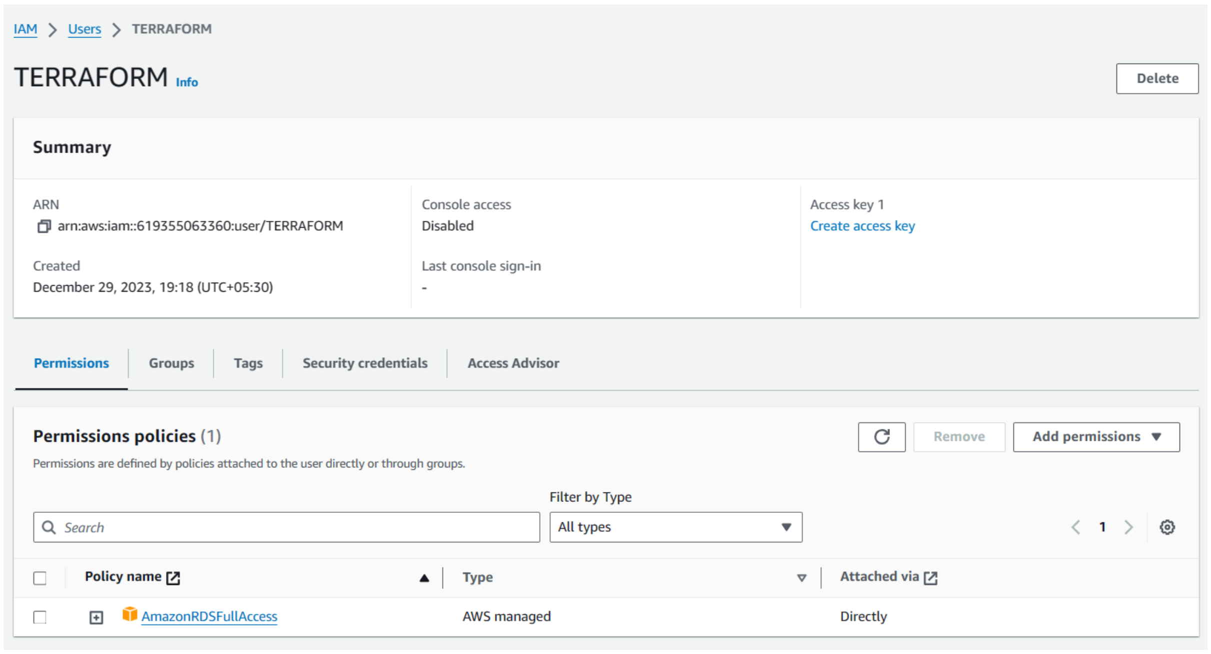
IAM > Users

Users (1) Info

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

	User name	Path	Groups	Last activity	MFA	Password age	Console last sign-in	Acces
☐	TERRAFORM	/	0	,	-	-	-	-

➔ Then click that created user: there we can able to find output **create access key option.** Click that one



IAM > Users > TERRAFORM

TERRAFORM Info

Delete

Summary

ARN	Console access	Access key 1
arn:aws:iam::619355063360:user/TERRAFORM	Disabled	Create access key
Created	Last console sign-in	
December 29, 2023, 19:18 (UTC+05:30)	-	

Permissions | Groups | Tags | Security credentials | Access Advisor

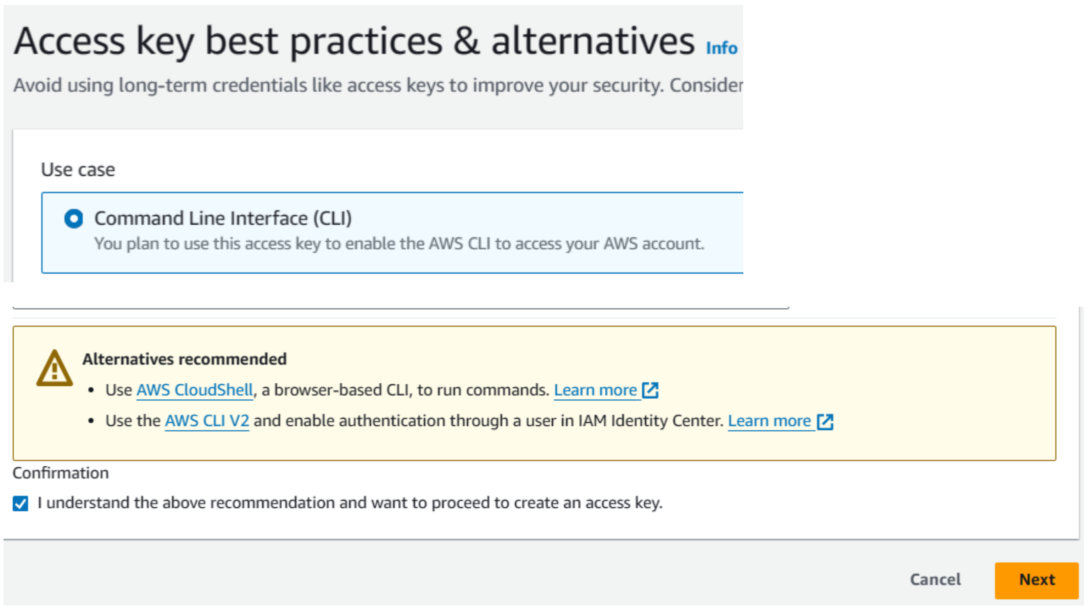
Permissions policies (1)

Permissions are defined by policies attached to the user directly or through groups.

Filter by Type: All types

Policy name	Type	Attached via
AmazonRDSFullAccess	AWS managed	Directly

➔ Then select **command line interface under use-case:** click next



Access key best practices & alternatives Info

Avoid using long-term credentials like access keys to improve your security. Consider

Use case

● Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.

Alternatives recommended

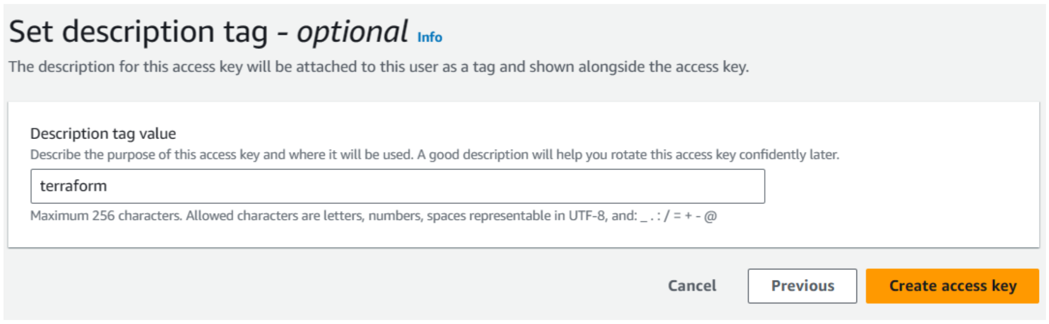
- Use AWS CloudShell, a browser-based CLI, to run commands. Learn more
- Use the AWS CLI V2 and enable authentication through a user in IAM Identity Center. Learn more

Confirmation

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

Cancel | Next

➔ Then description tag optional, **click create access key:**



Set description tag - *optional* Info

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

Description tag value

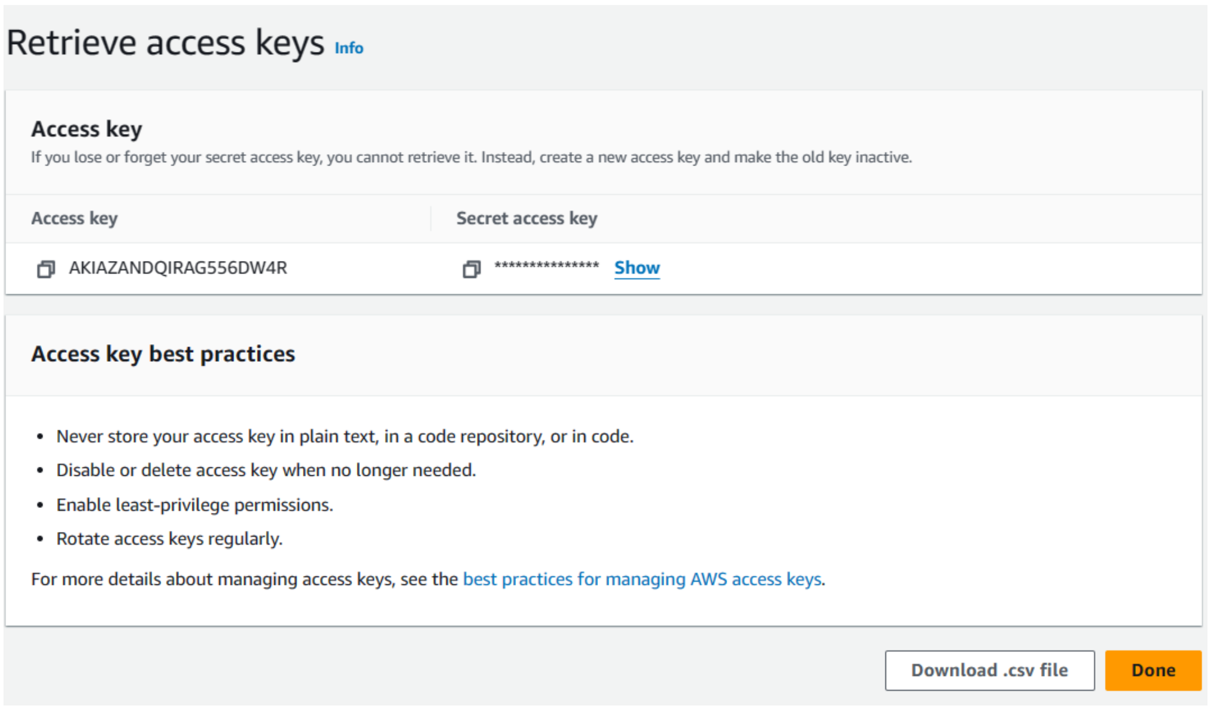
Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.

terraform

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

Cancel | Previous | Create access key

➔ The access key and secret key has been created successfully:



Retrieve access keys Info

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
AKIAZANDQIRAG556DW4R	*************** Show

Access key best practices

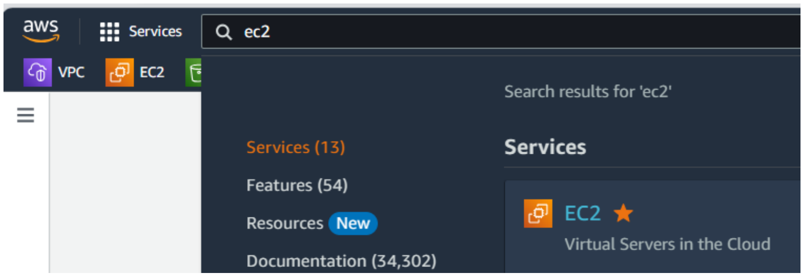
- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the best practices for managing AWS access keys.

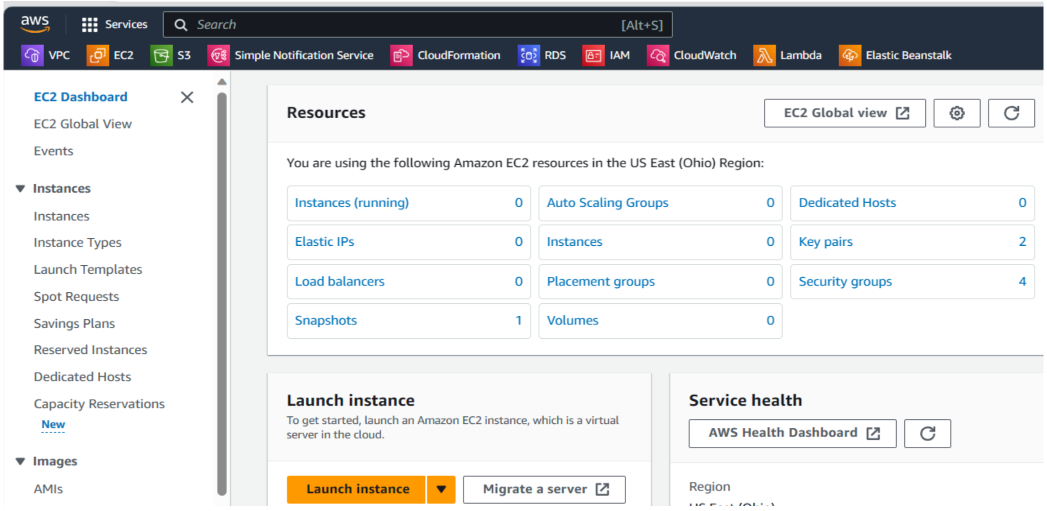
****

STEP:2 – CREATING AN EC2 INSTANCE:

➔ Then search **EC2** on service panel, and click that one:



➔ Then click **launch instance** on EC2 management console:



Resources

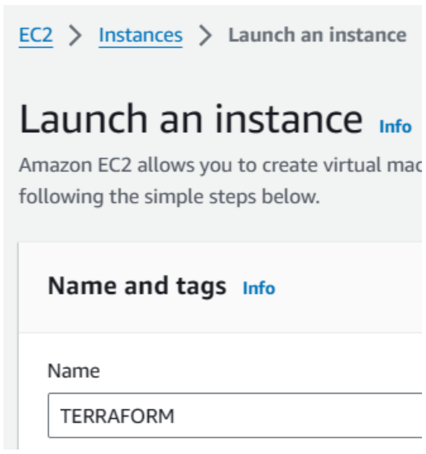
You are using the following Amazon EC2 resources in the US East (Ohio) Region:

Instances (running)	0	Auto Scaling Groups	0	Dedicated Hosts	0
Elastic IPs	0	Instances	0	Key pairs	2
Load balancers	0	Placement groups	0	Security groups	4
Snapshots	1	Volumes	0		

Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

➔ Then **name the instance** according to your preferences:


EC2 > Instances > Launch an instance

Launch an instance Info

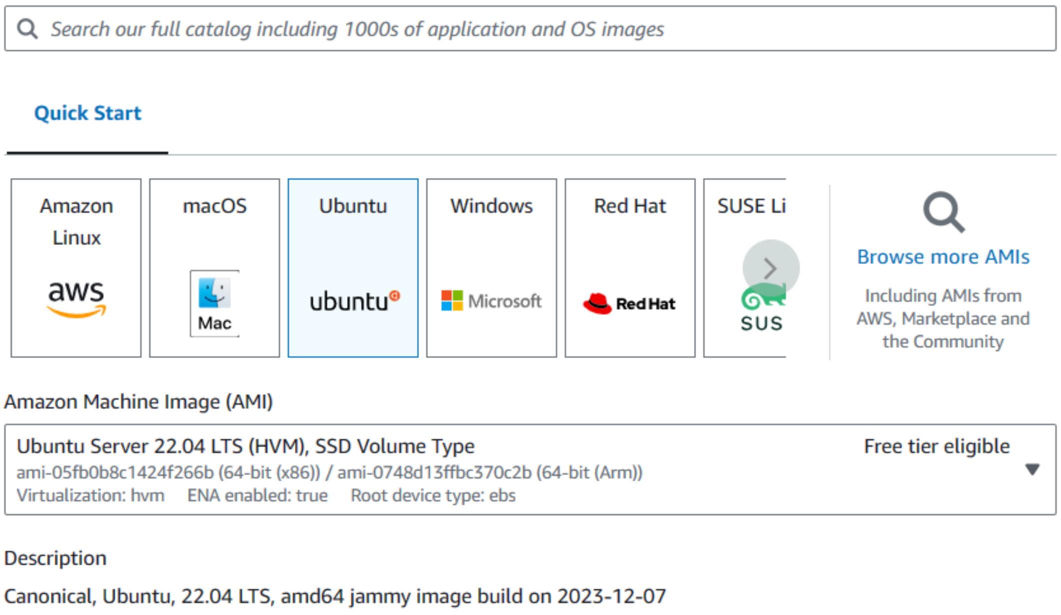
Amazon EC2 allows you to create virtual mac
following the simple steps below.

Name and tags Info

Name

TERRAFORM

➔ Selecting **OS** according to your preferences, here I am using **ubuntu OS:**


Search our full catalog including 1000s of application and OS images
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
SUSE Li
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

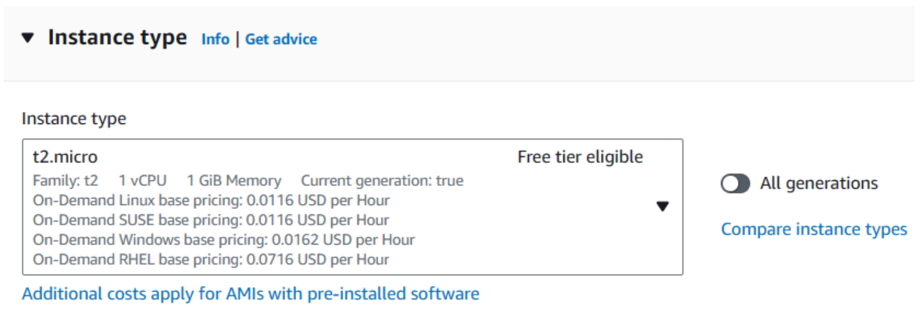
Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type — Free tier eligible
ami-05fb0b8c1424f266b (64-bit (x86)) / ami-0748d13ffbc370c2b (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2023-12-07

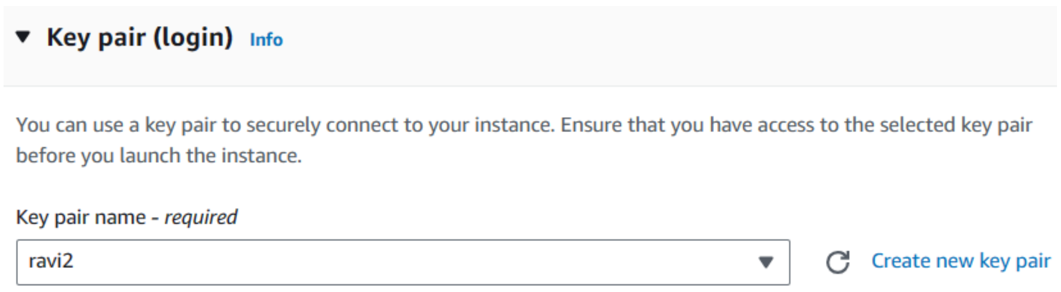
➔ Then select the **instance type:**


Instance type Info | Get advice

Instance type

t2.micro — Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0116 USD per Hour
On-Demand SUSE base pricing: 0.0116 USD per Hour
On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand RHEL base pricing: 0.0716 USD per Hour

➔ Then select the **keypair** according to your wish:


▼ Key pair (login) Info

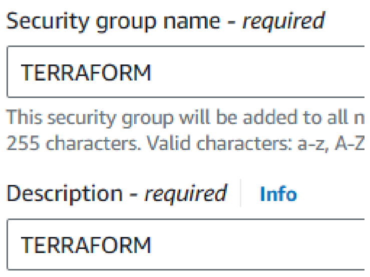
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

ravi2

Create new key pair

➔ Then select the security group or create a new security group:


Security group name - *required*

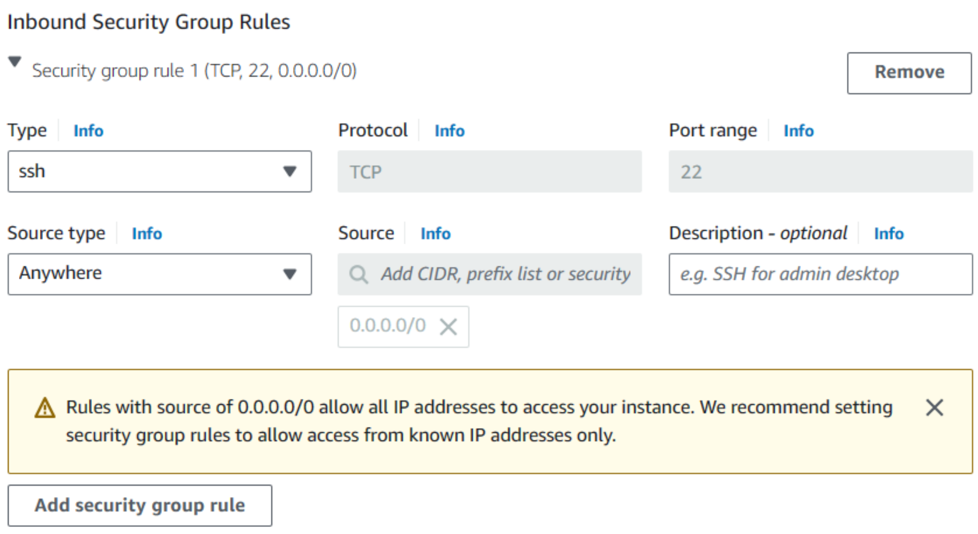
TERRAFORM

This security group will be added to all n
255 characters. Valid characters: a-z, A-Z

Description - *required* Info

TERRAFORM

Security group rules:

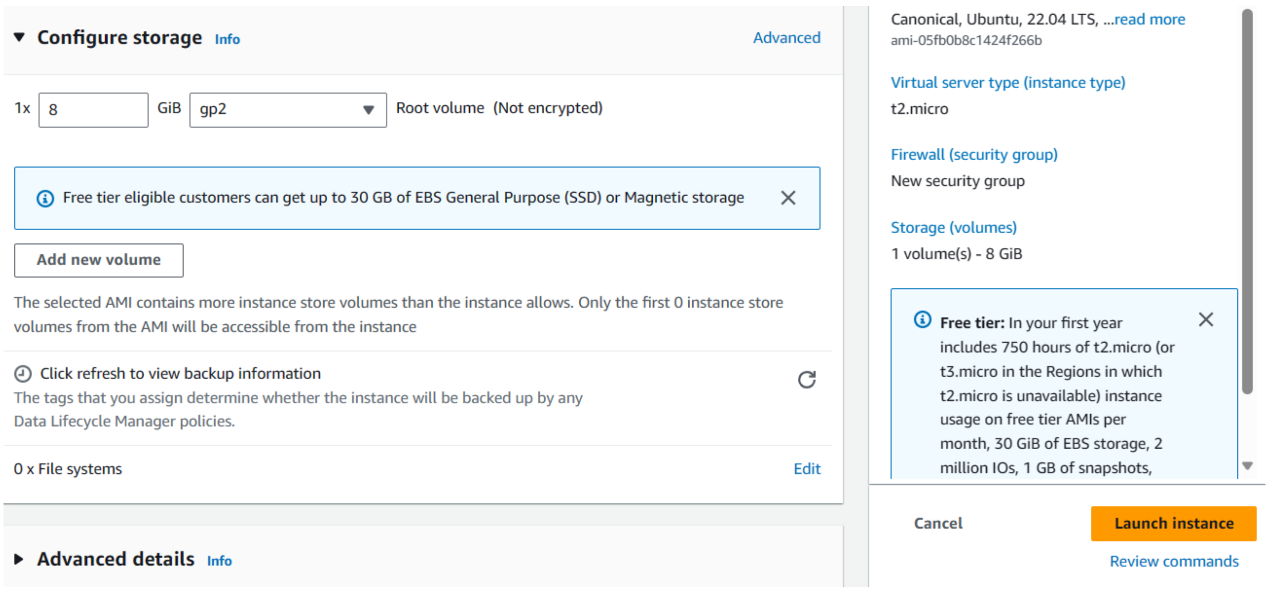

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

Type: ssh	Protocol: TCP	Port range: 22
Source type: Anywhere	Source: Add CIDR, prefix list or security; 0.0.0.0/0	Description - *optional*: e.g. SSH for admin desktop

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

➔ Then for the storage keeping as the default and **click launch instance** on right side:


▼ Configure storage Info Advanced

1x 8 GiB gp2 Root volume (Not encrypted)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

Add new volume

The selected AMI contains more instance store volumes than the instance allows. Only the first 0 instance store volumes from the AMI will be accessible from the instance

Click refresh to view backup information
The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

0 x File systems Edit

▶ Advanced details Info

Canonical, Ubuntu, 22.04 LTS, ...read more
ami-05fb0b8c1424f266b

Virtual server type (instance type)
t2.micro

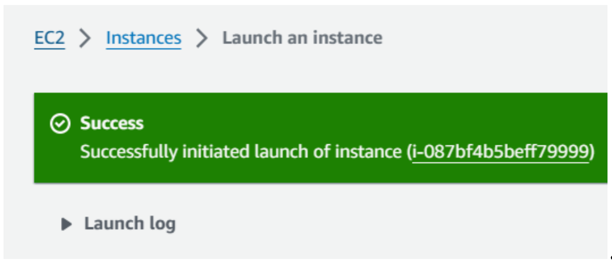
Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots,

Cancel Launch instance
Review commands

➔ The instance launching has been initiated successfully:

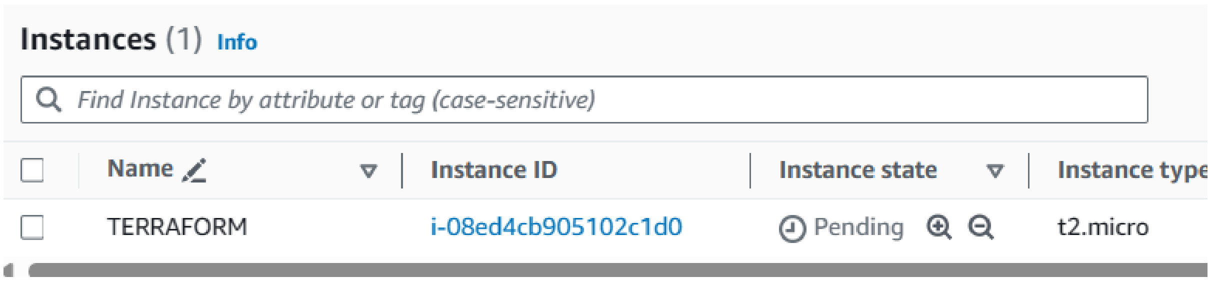

EC2 > Instances > Launch an instance

Success
Successfully initiated launch of instance (i-087bf4b5beff79999)

▶ Launch log

.

➔ The instance has been created successfully:


Instances (1) Info

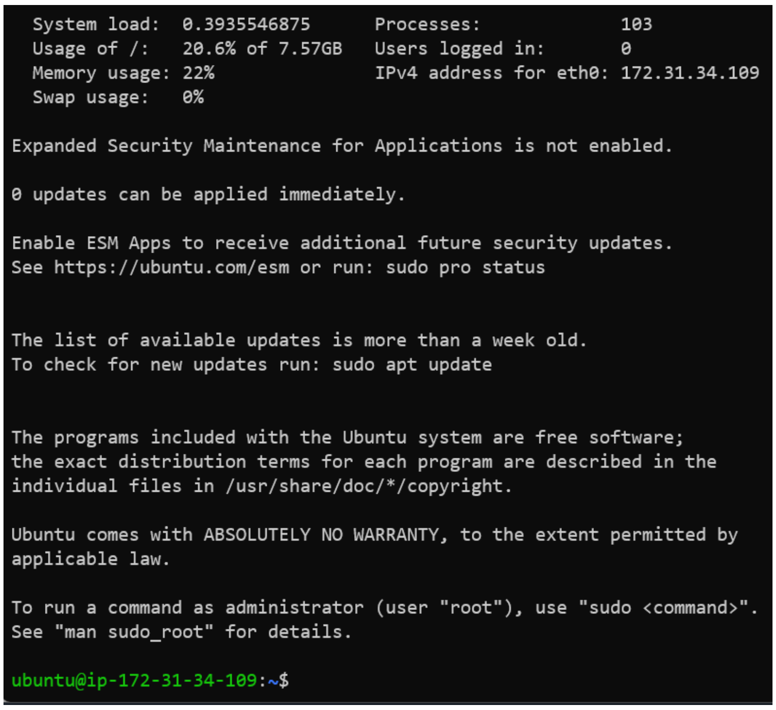
Find Instance by attribute or tag (case-sensitive)

Name	Instance ID	Instance state	Instance type
TERRAFORM	i-08ed4cb905102c1d0	Pending	t2.micro

****

STEP:3 - CONNECTING THE INSTANCE & INSTALLING REQUIRED PACKAGES

➔ Connecting the instance via **EC2 Instance connect** or via **ssh client** method:

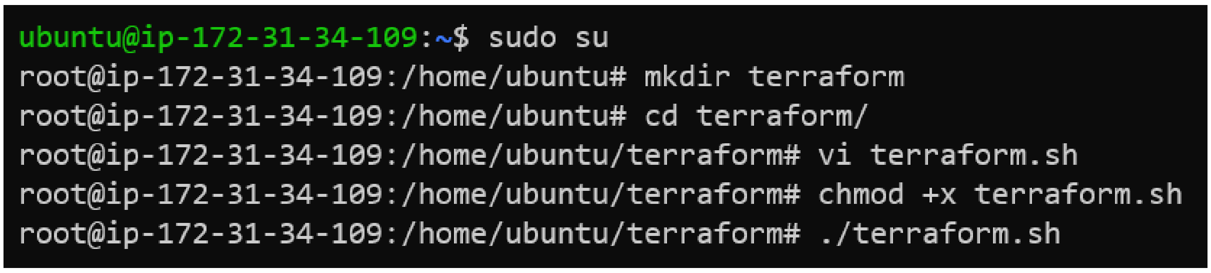

```
  System load:  0.3935546875      Processes:             103
  Usage of /:   20.6% of 7.57GB   Users logged in:       0
  Memory usage: 22%               IPv4 address for eth0: 172.31.34.109
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-34-109:~$
```

➔ Creating a **script to install terraform:**

Script contains:

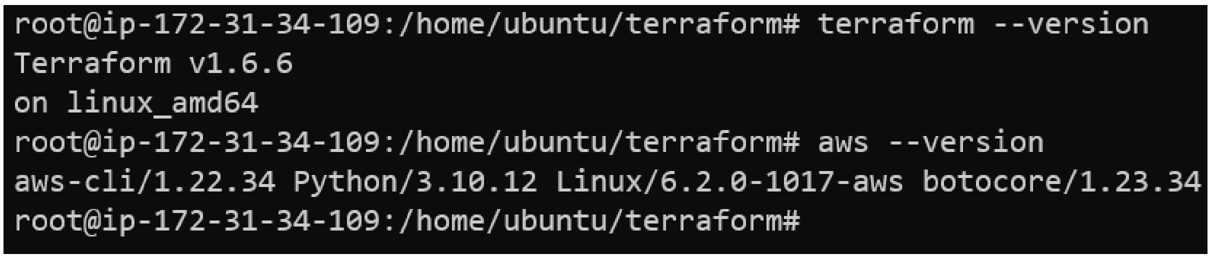
```
#!/bin/bash
#updating the os:
apt-get update

#installing awscli:
apt-get install -y awscli

#installing terraform:
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg -
-dearmor -o /usr/share/keyrings/hashicorp-archive-
keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-
keyring.gpg] https://apt.releases.hashicorp.com
$(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install terraform -y
```

➔ Change the file permission and executing the file:



```
ubuntu@ip-172-31-34-109:~$ sudo su
root@ip-172-31-34-109:/home/ubuntu# mkdir terraform
root@ip-172-31-34-109:/home/ubuntu# cd terraform/
root@ip-172-31-34-109:/home/ubuntu/terraform# vi terraform.sh
root@ip-172-31-34-109:/home/ubuntu/terraform# chmod +x terraform.sh
root@ip-172-31-34-109:/home/ubuntu/terraform# ./terraform.sh
```

➔ Checking packages:



```
root@ip-172-31-34-109:/home/ubuntu/terraform# terraform --version
Terraform v1.6.6
on linux_amd64
root@ip-172-31-34-109:/home/ubuntu/terraform# aws --version
aws-cli/1.22.34 Python/3.10.12 Linux/6.2.0-1017-aws botocore/1.23.34
root@ip-172-31-34-109:/home/ubuntu/terraform#
```

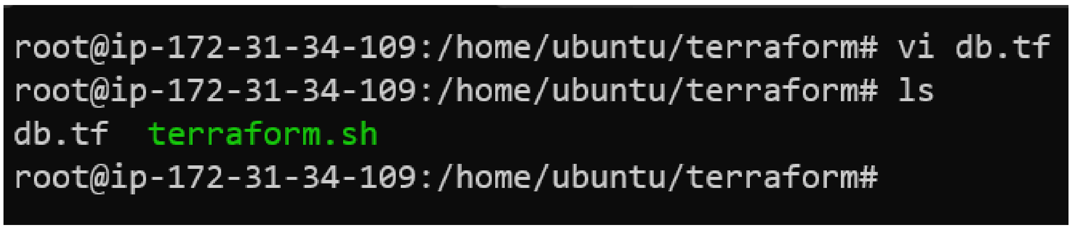
****

STEP:4 - CREATING A TERRAFORM FILE FOR CREATING MYSQL RDS DATABASE

➔ Creating a **terraform script** for creating **MySQL RDS Database:**

Script contains:

```
# 1.Providing aws details:

provider "aws" {
  profile = "default"
  region  = "us-east-2"
}

# 2.Creating security group database:
resource "aws_security_group" "database-sc" {
  name        = "database-sc"
  description = "security group for database"

# Allow mysql database (port 3306) from anywhere
  ingress {
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = ["172.31.0.0/16"]
  }
  # Egress rules (outbound traffic)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "database-sc"
  }
}

#3.Creating mysql database:
resource "aws_db_instance" "database-mysql" {
  identifier                = "terraform-db"
  allocated_storage         = 20
  storage_type              = "gp2"
  db_name                   = "terraformdb"
  engine                    = "mysql"
  engine_version            = "5.7"
  instance_class            = "db.t2.micro"
  username                  = "admin"
  password                  = "nodejs123"
  parameter_group_name      = "default.mysql5.7"
  availability_zone         = "us-east-2c"
  skip_final_snapshot       = true
  vpc_security_group_ids = ["${aws_security_group.database-
sc.id}"]
  tags = {
    Name = "terraform-db"
  }
}
#4. Getting the output of mysql database:
output "rds_endpoint" {
  value = aws_db_instance.database-mysql.endpoint
}
```

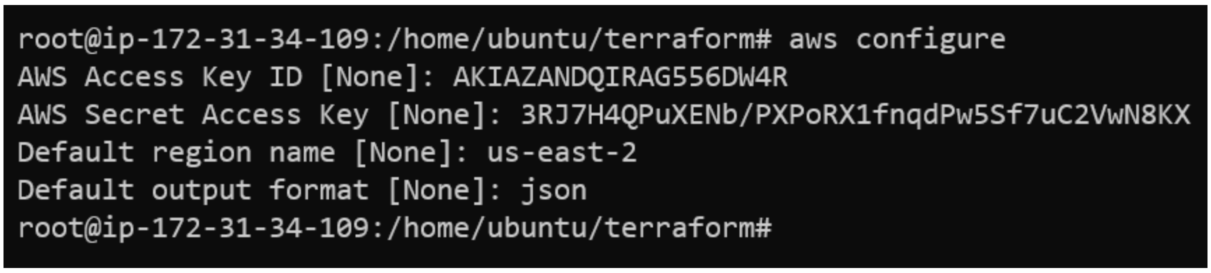


```
root@ip-172-31-34-109:/home/ubuntu/terraform# vi db.tf
root@ip-172-31-34-109:/home/ubuntu/terraform# ls
db.tf  terraform.sh
root@ip-172-31-34-109:/home/ubuntu/terraform#
```

****

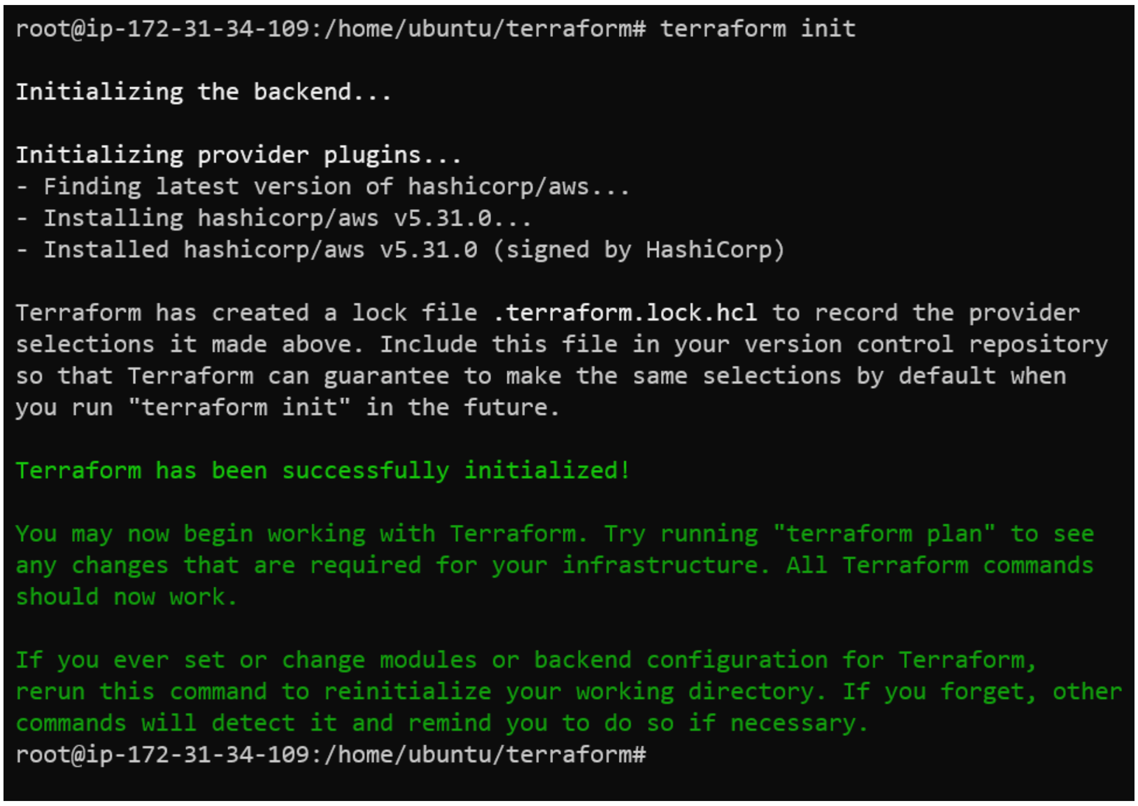
STEP:5 - PROVISIONING THE DATABASE USING TERRAFORM

➔ First, we need to configure our AWS Credentials by using **aws configure** command:



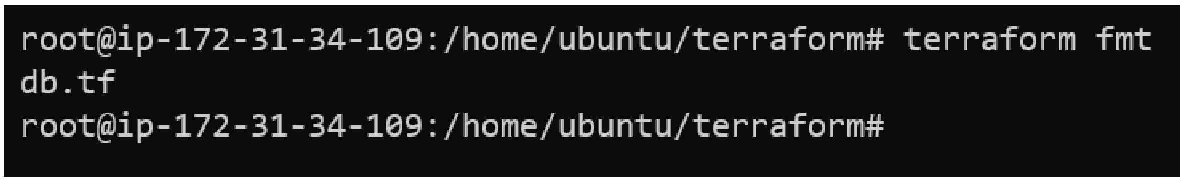
```
root@ip-172-31-34-109:/home/ubuntu/terraform# aws configure
AWS Access Key ID [None]: AKIAZANDQIRAG556DW4R
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-2
Default output format [None]: json
root@ip-172-31-34-109:/home/ubuntu/terraform#
```

➔ Then executing **terraform init** command:

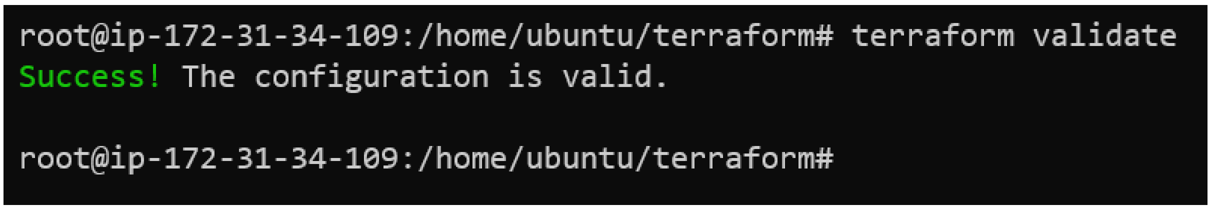


```
root@ip-172-31-34-109:/home/ubuntu/terraform# terraform init

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-172-31-34-109:/home/ubuntu/terraform#
```

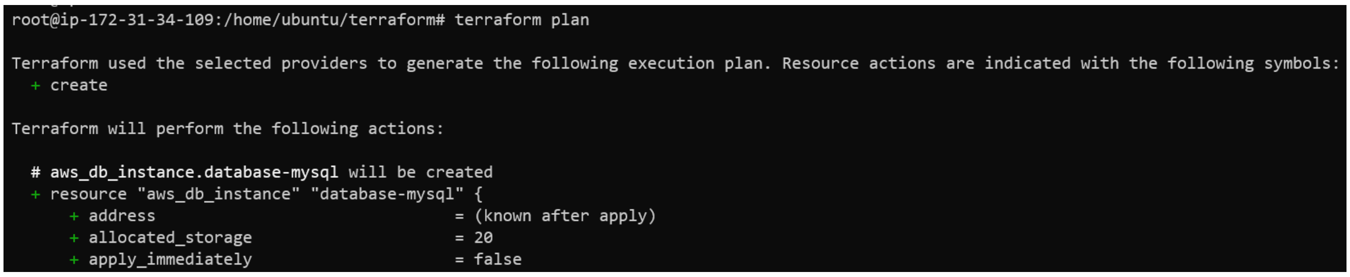
➔ Then executing **terraform fmt** command:


```
root@ip-172-31-34-109:/home/ubuntu/terraform# terraform fmt
db.tf
root@ip-172-31-34-109:/home/ubuntu/terraform#
```

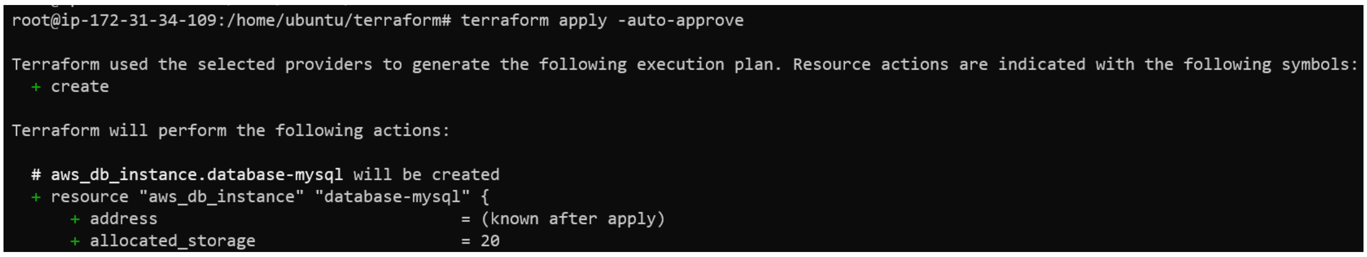
➔ Then executing **terraform validate** command:


```
root@ip-172-31-34-109:/home/ubuntu/terraform# terraform validate
Success! The configuration is valid.

root@ip-172-31-34-109:/home/ubuntu/terraform#
```

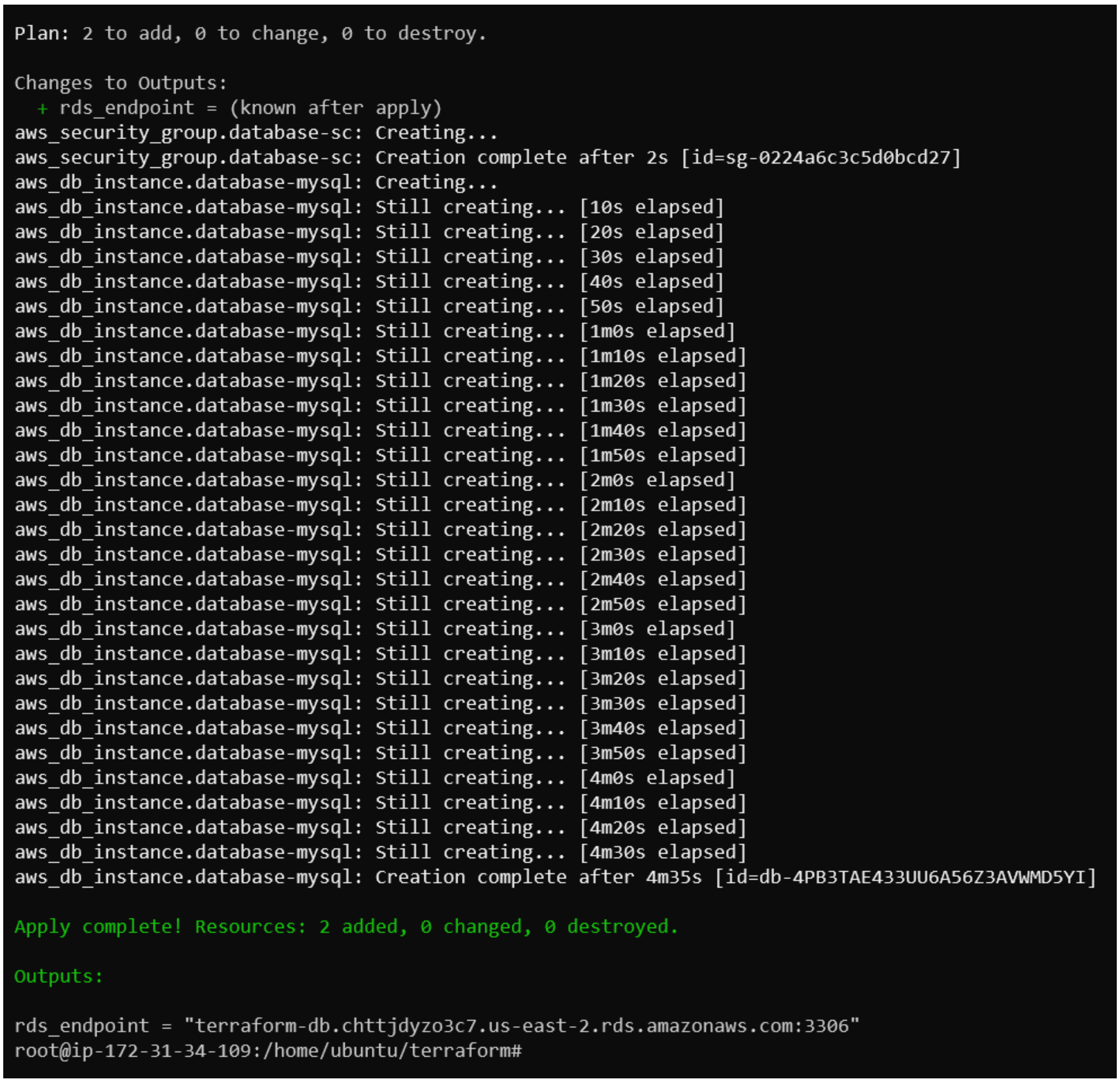
➔ Then executing **terraform plan** command:


```
root@ip-172-31-34-109:/home/ubuntu/terraform# 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_db_instance.database-mysql will be created
  + resource "aws_db_instance" "database-mysql" {
      + address                              = (known after apply)
      + allocated_storage                    = 20
      + apply_immediately                    = false
```

➔ Then executing **terraform apply -auto-approve** command:

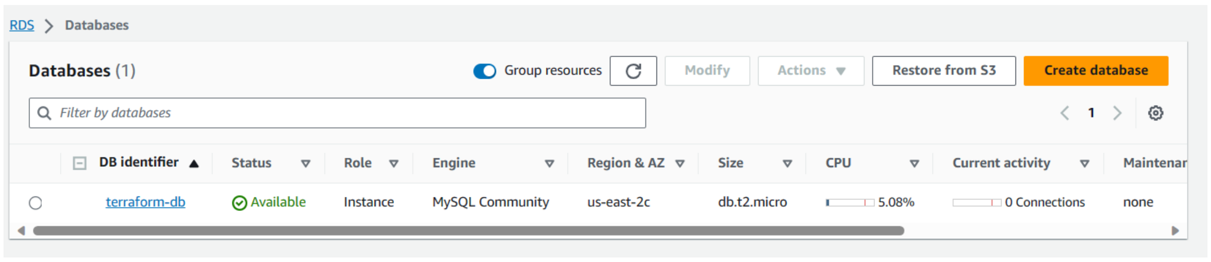

```
root@ip-172-31-34-109:/home/ubuntu/terraform# terraform apply -auto-approve

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_db_instance.database-mysql will be created
  + resource "aws_db_instance" "database-mysql" {
      + address                              = (known after apply)
      + allocated_storage                    = 20
```

Terraform output:



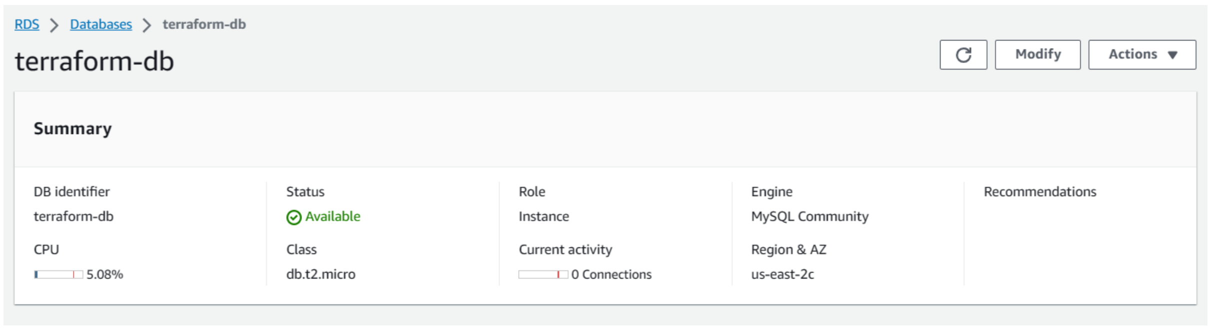
```
Plan: 2 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + rds_endpoint = (known after apply)
aws_security_group.database-sc: Creating...
aws_security_group.database-sc: Creation complete after 2s [id=sg-0224a6c3c5d0bcd27]
aws_db_instance.database-mysql: Creating...
aws_db_instance.database-mysql: Still creating... [10s elapsed]
aws_db_instance.database-mysql: Still creating... [20s elapsed]
aws_db_instance.database-mysql: Still creating... [30s elapsed]
aws_db_instance.database-mysql: Still creating... [40s elapsed]
aws_db_instance.database-mysql: Still creating... [50s elapsed]
aws_db_instance.database-mysql: Still creating... [1m0s elapsed]
aws_db_instance.database-mysql: Still creating... [1m10s elapsed]
aws_db_instance.database-mysql: Still creating... [1m20s elapsed]
aws_db_instance.database-mysql: Still creating... [1m30s elapsed]
aws_db_instance.database-mysql: Still creating... [1m40s elapsed]
aws_db_instance.database-mysql: Still creating... [1m50s elapsed]
aws_db_instance.database-mysql: Still creating... [2m0s elapsed]
aws_db_instance.database-mysql: Still creating... [2m10s elapsed]
aws_db_instance.database-mysql: Still creating... [2m20s elapsed]
aws_db_instance.database-mysql: Still creating... [2m30s elapsed]
aws_db_instance.database-mysql: Still creating... [2m40s elapsed]
aws_db_instance.database-mysql: Still creating... [2m50s elapsed]
aws_db_instance.database-mysql: Still creating... [3m0s elapsed]
aws_db_instance.database-mysql: Still creating... [3m10s elapsed]
aws_db_instance.database-mysql: Still creating... [3m20s elapsed]
aws_db_instance.database-mysql: Still creating... [3m30s elapsed]
aws_db_instance.database-mysql: Still creating... [3m40s elapsed]
aws_db_instance.database-mysql: Still creating... [3m50s elapsed]
aws_db_instance.database-mysql: Still creating... [4m0s elapsed]
aws_db_instance.database-mysql: Still creating... [4m10s elapsed]
aws_db_instance.database-mysql: Still creating... [4m20s elapsed]
aws_db_instance.database-mysql: Still creating... [4m30s elapsed]
aws_db_instance.database-mysql: Creation complete after 4m35s [id=db-4PB3TAE433UU6A56Z3AVWMD5YI]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

rds_endpoint = "terraform-db.chttjdyzo3c7.us-east-2.rds.amazonaws.com:3306"
root@ip-172-31-34-109:/home/ubuntu/terraform#
```

➔ Checking the console:



Databases (1)

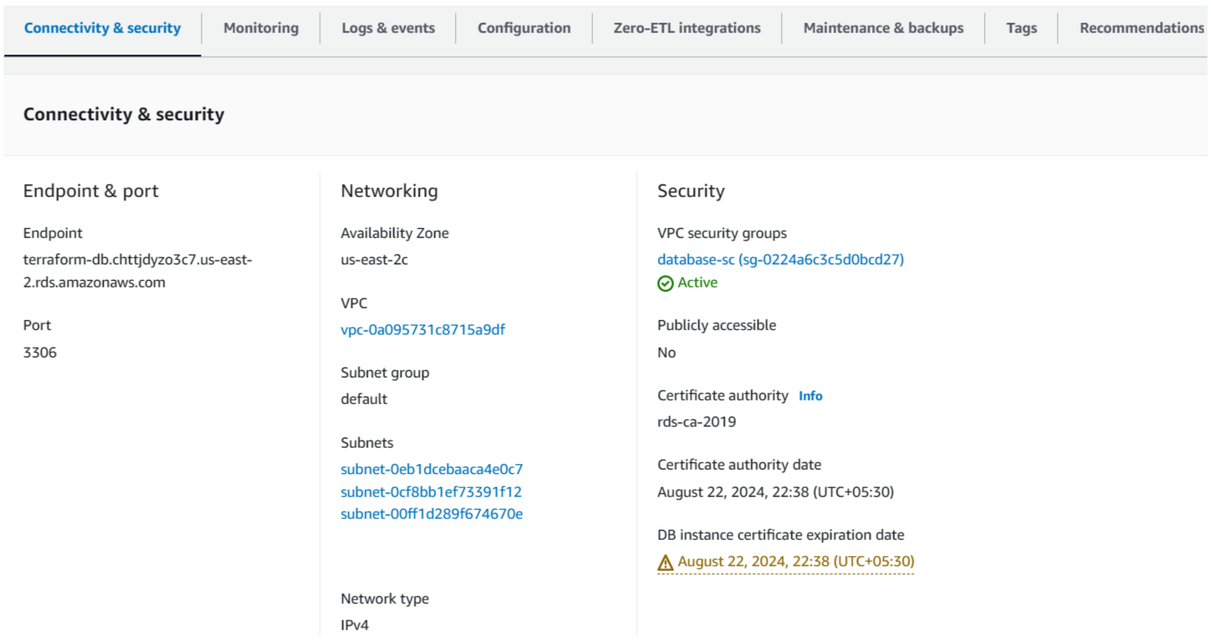
DB identifier	Status	Role	Engine	Region & AZ	Size	CPU	Current activity	Maintenar
terraform-db	Available	Instance	MySQL Community	us-east-2c	db.t2.micro	5.08%	0 Connections	none



terraform-db

Summary

DB identifier	Status	Role	Engine	Recommendations
terraform-db	Available	Instance	MySQL Community	
CPU	Class	Current activity	Region & AZ	
5.08%	db.t2.micro	0 Connections	us-east-2c	



Connectivity & security

Endpoint & port

Endpoint
terraform-db.chttjdyzo3c7.us-east-2.rds.amazonaws.com

Port
3306

Networking

Availability Zone
us-east-2c

VPC
vpc-0a095731c8715a9df

Subnet group
default

Subnets
subnet-0eb1dcebaaca4e0c7
subnet-0cf8bb1ef73391f12
subnet-00ff1d289f674670e

Network type
IPv4

Security

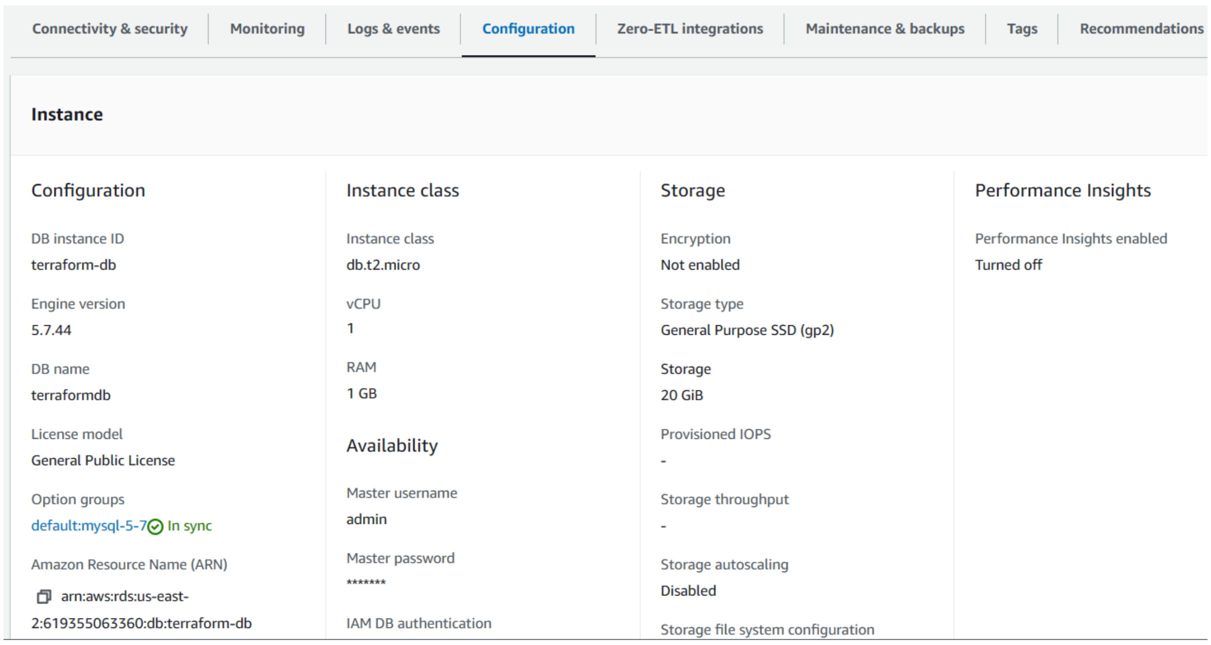
VPC security groups
database-sc (sg-0224a6c3c5d0bcd27)
Active

Publicly accessible
No

Certificate authority Info
rds-ca-2019

Certificate authority date
August 22, 2024, 22:38 (UTC+05:30)

DB instance certificate expiration date
August 22, 2024, 22:38 (UTC+05:30)



Instance

Configuration

DB instance ID
terraform-db

Engine version
5.7.44

DB name
terraformdb

License model
General Public License

Option groups
default:mysql-5-7 In sync

Amazon Resource Name (ARN)
arn:aws:rds:us-east-2:619355063360:db:terraform-db

Instance class

Instance class
db.t2.micro

vCPU
1

RAM
1 GB

Availability

Master username
admin

Master password
****

IAM DB authentication

Storage

Encryption
Not enabled

Storage type
General Purpose SSD (gp2)

Storage
20 GiB

Provisioned IOPS
-

Storage throughput
-

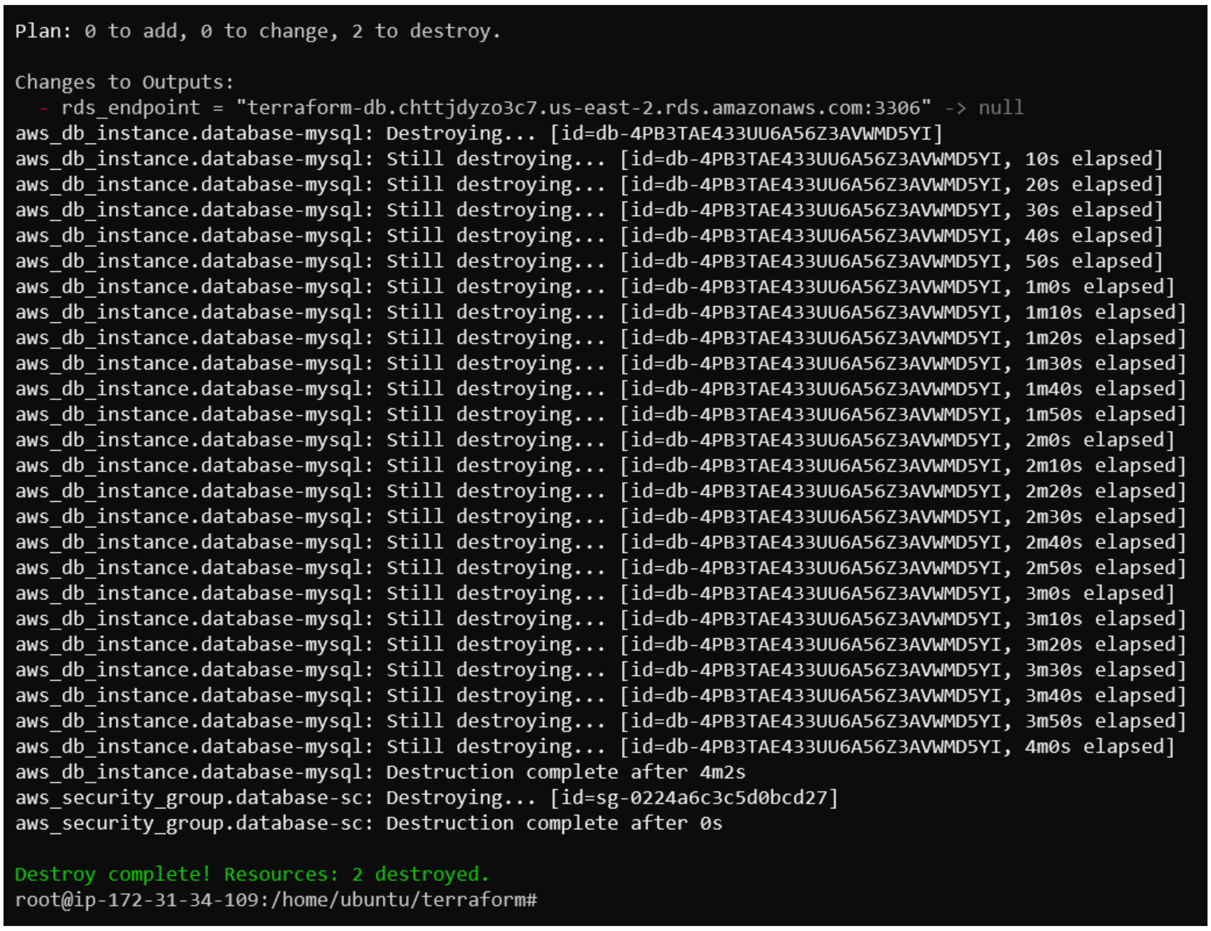
Storage autoscaling
Disabled

Storage file system configuration

Performance Insights

Performance Insights enabled
Turned off

➔ Then executing **terraform destroy -auto-approve** command: for deleting the database:



```
Plan: 0 to add, 0 to change, 2 to destroy.

Changes to Outputs:
  - rds_endpoint = "terraform-db.chttjdyzo3c7.us-east-2.rds.amazonaws.com:3306" -> null
aws_db_instance.database-mysql: Destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 10s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 20s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 30s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 40s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 50s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 1m0s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 1m10s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 1m20s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 1m30s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 1m40s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 1m50s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 2m0s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 2m10s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 2m20s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 2m30s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 2m40s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 2m50s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 3m0s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 3m10s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 3m20s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 3m30s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 3m40s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 3m50s elapsed]
aws_db_instance.database-mysql: Still destroying... [id=db-4PB3TAE433UU6A56Z3AVWMD5YI, 4m0s elapsed]
aws_db_instance.database-mysql: Destruction complete after 4m2s
aws_security_group.database-sc: Destroying... [id=sg-0224a6c3c5d0bcd27]
aws_security_group.database-sc: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
root@ip-172-31-34-109:/home/ubuntu/terraform#
```

****

BENEFITS OF DOING ABOVE TASK:

➔ **Consistency:** Terraform ensures that the infrastructure is provisioned consistently every time, reducing the chance of configuration drift.

➔ **Scalability:** Easily scale the number of MySQL databases up or down based on demand by adjusting the Terraform configuration.

➔ **Time Efficiency:** Automation speeds up the deployment process, allowing teams to focus on more critical tasks rather than manual provisioning.

➔ **Repeatability:** The IaC approach allows for easy replication of the infrastructure setup in different environments, promoting a consistent development, testing, and production workflow.

➔ **Error Reduction:** With Terraform handling the provisioning process, human errors are minimized, leading to a more reliable and stable database environment.

****

All the above files used in this task has been uploaded in this GitHub Repository: https://github.com/Ravivarman16/automating-mysql-database-provisioning-with-terraform.git

****