**Cloud Watch Using Terraform**

## What is cloud watch?

Amazon CloudWatch is a monitoring and observability service offered by AWS (Amazon Web Services) With Amazon CloudWatch, you can track the resources and application performance, collect and monitor log file details, and enable your resources’ alarms and notifications to be triggered on specific events.

**Key Features of CloudWatch:**

* Metrics Collection
* Logs Monitoring
* Alarms
* Dashboards
* Events and Automated Responses
* Application Insights

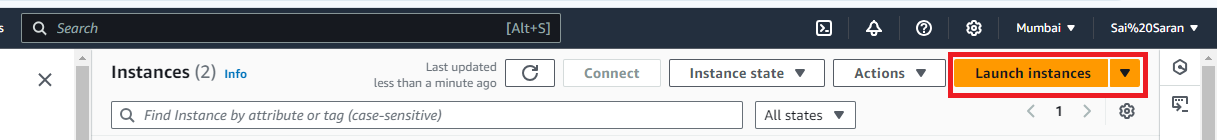
### Use Cases:

* **Monitoring infrastructure** for operational health.
* **Automating responses** to performance changes or failures.
* **Troubleshooting** using logs and traces.
* **Optimizing resource usage** with insights into application performance.

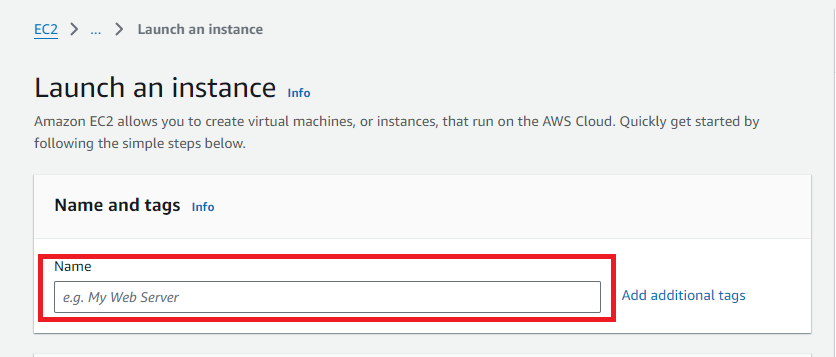
**Steps to create Terraform CloudWatch:**

# Step-1: Create AWS EC2 Instance in AWS Console

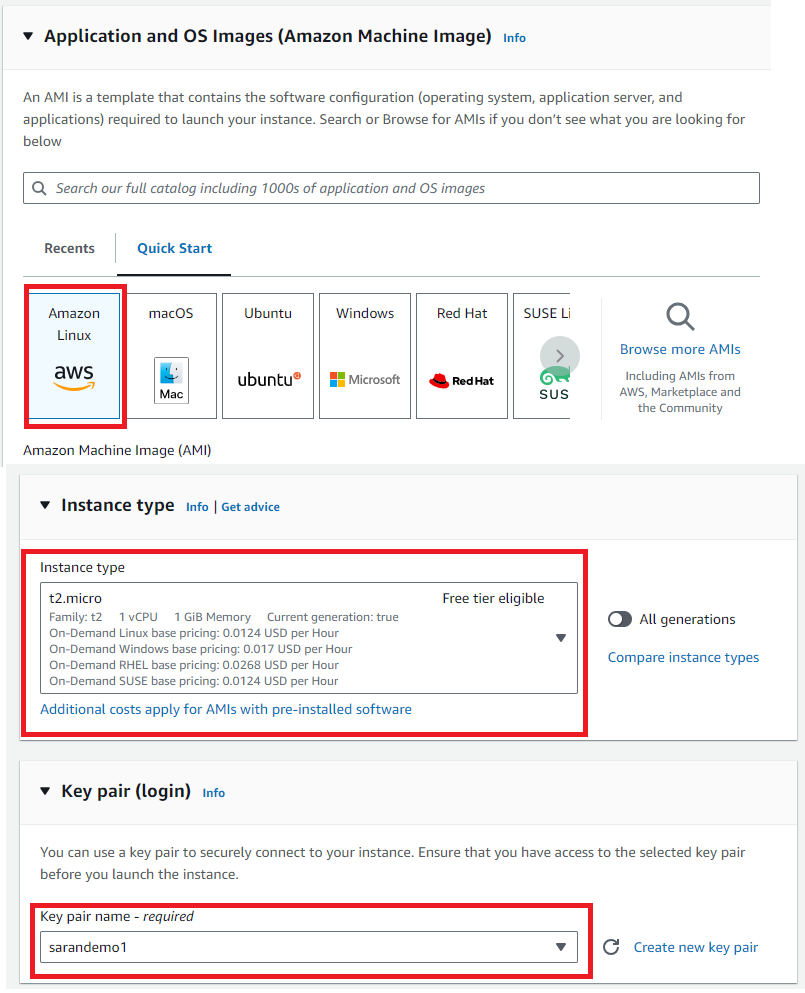
* Go to the AWS console search EC2 Service
* Then Click on launch Instance



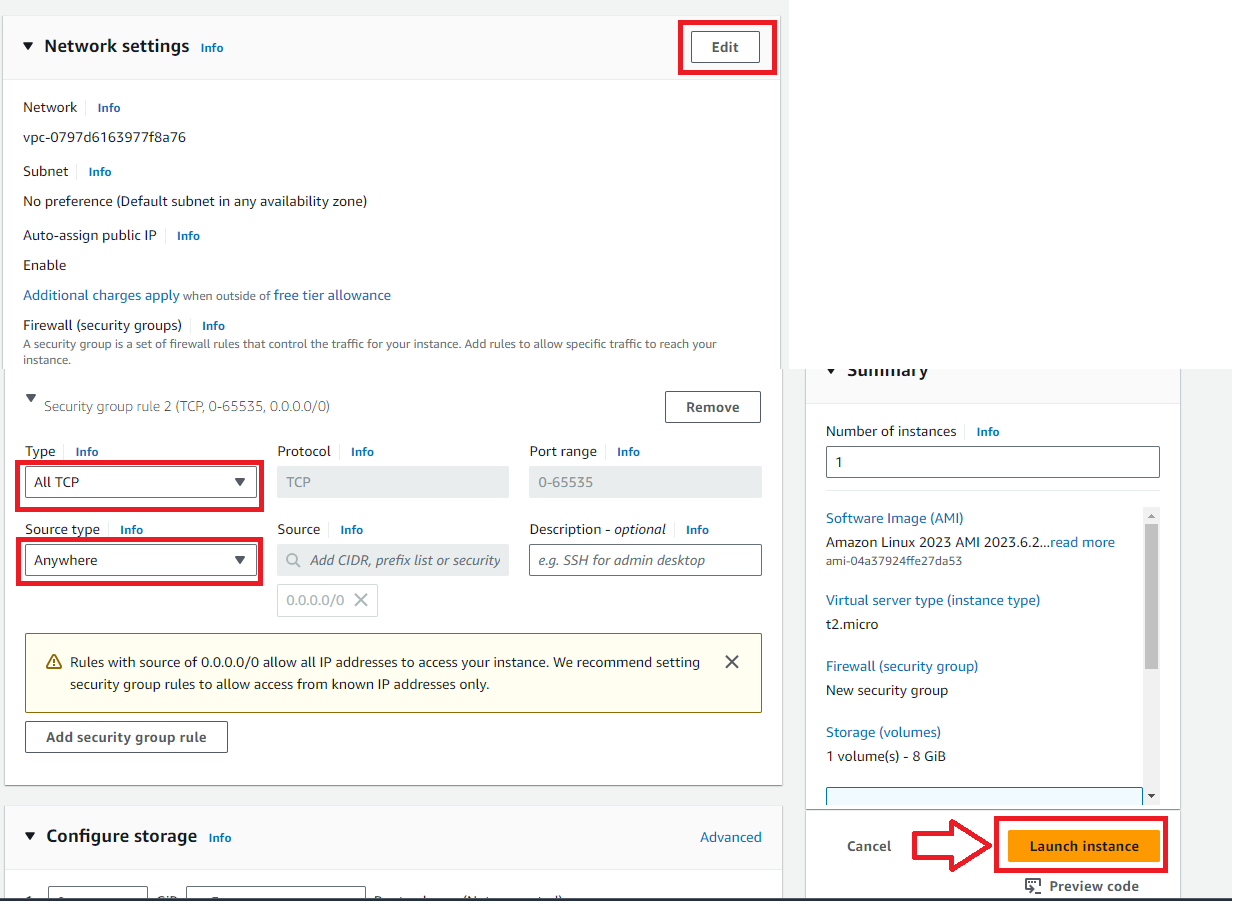
* After click on launch instance provide the name for the Instance



* Then select Amazon Machine Image (AMI)
* After that select Instance type
* Then provide the key pair if existing key pair is not available need to create new key pair by clicking on create new pair

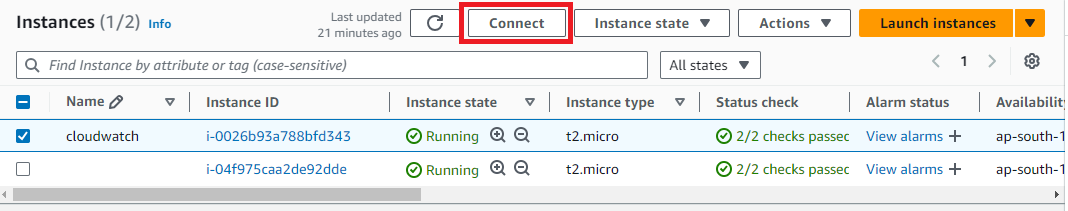


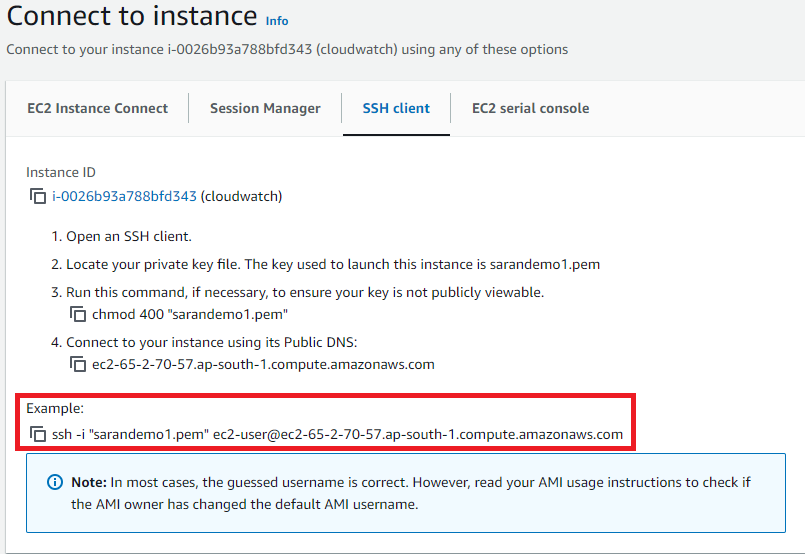
* In Network settings edit network settings and add All TCP
* Then click on Launch Instance



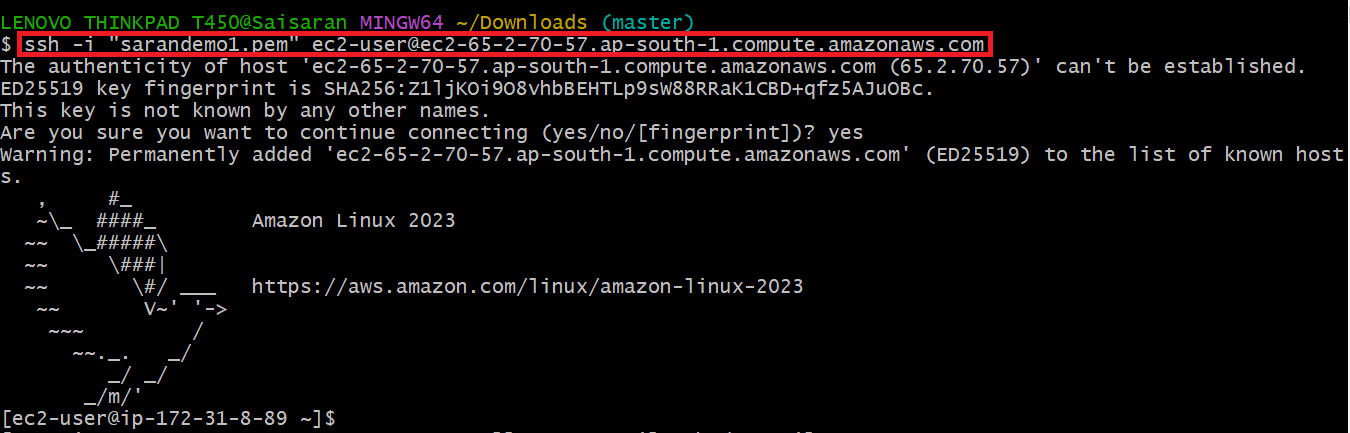
# Step-2: Connect EC2 Instance with Tools

* After creating the Instance need to click on connect on AWS Console
* Then Copy the SSH link





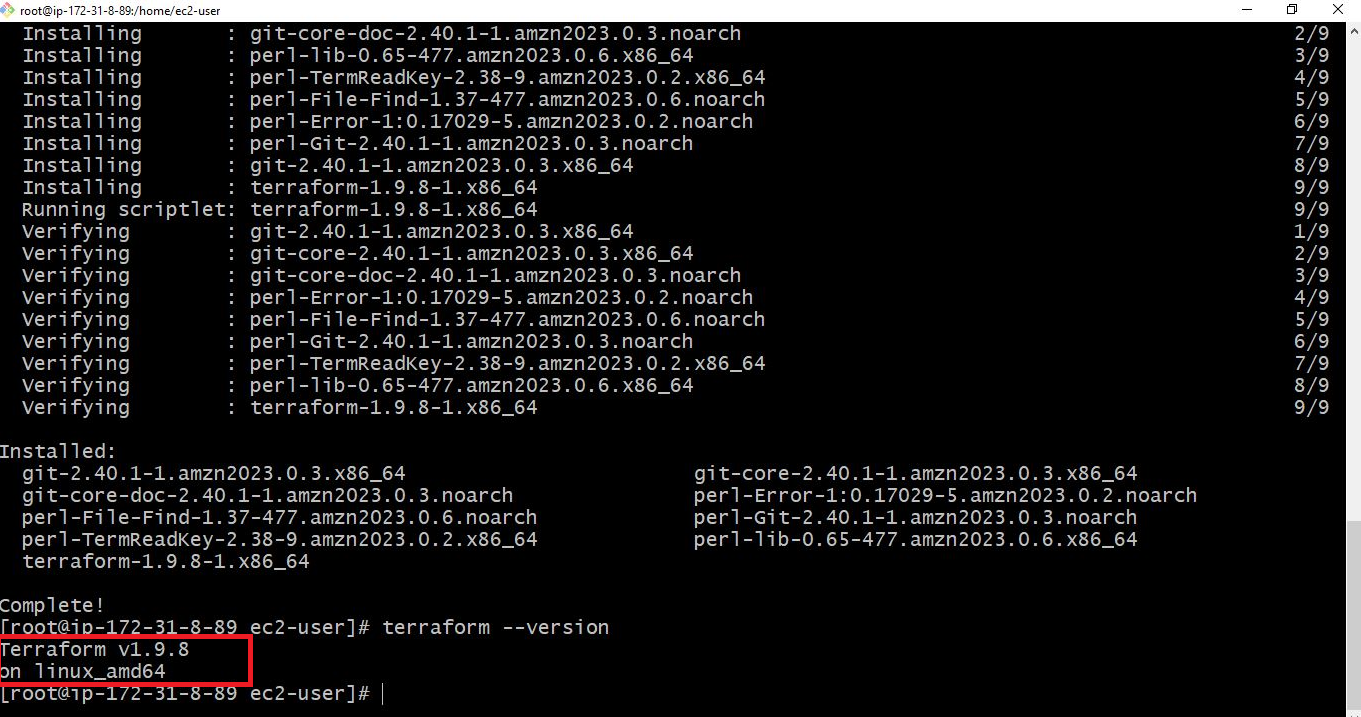
* After copying the SSH link need to connect to GIT,MobaXterm,etc..,
* Here I am connecting with GIT bash for need to got the key pair downloaded path and open GIT bash there and paste the SHH link then click on yes the Linux is connected to tool.



# Step-3:Install Terraform in EC2 Instance

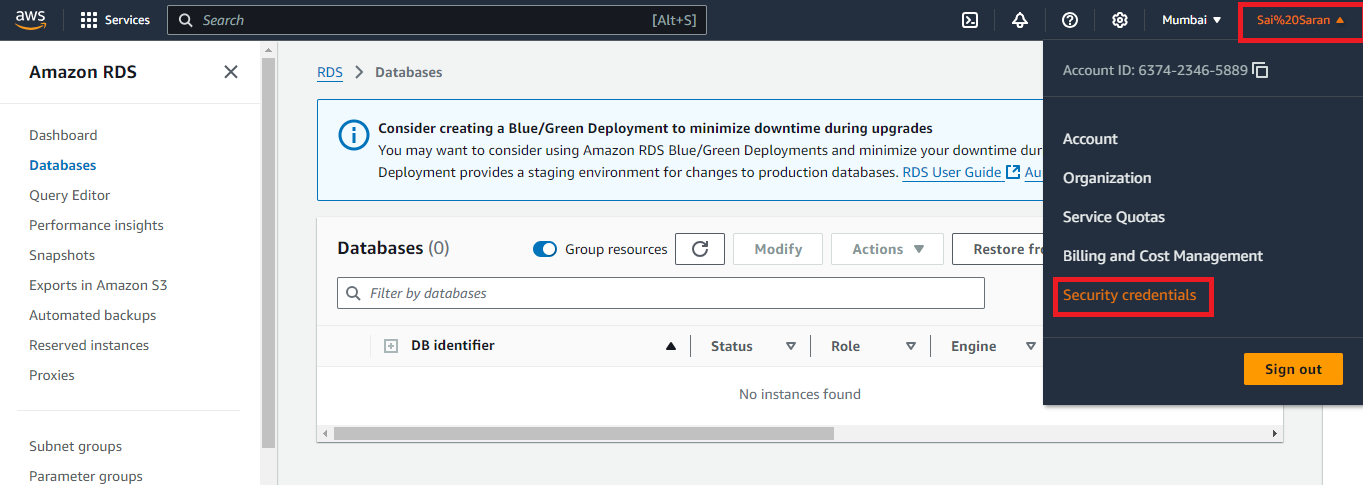
After connecting with Linux need to use below commands

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

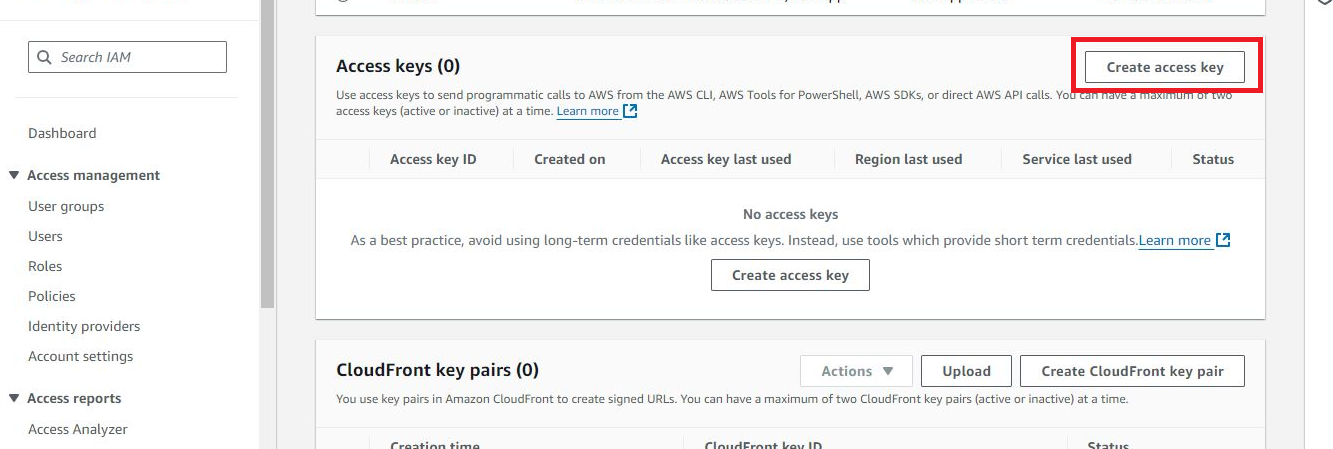


# Step-4:Security credentials creation in AWS Console

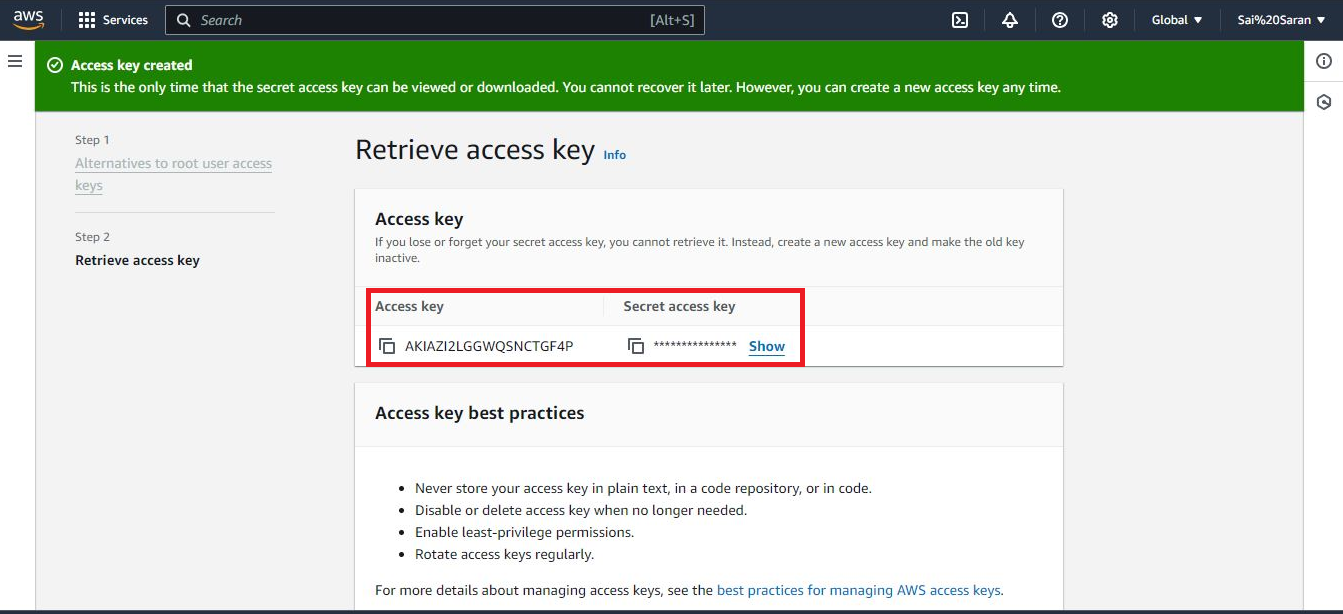
* Open AWS Console on the top right there is profile name in that we can see security credentials



* After opening the security credentials scroll down and check for create access key



* Then Click on next the access was created and we will get access Id and secret key



# Step-5:AWS Configure with Security credentials

After creating the access key in AWS Console then use the command “**aws configure”** to configure the access key with tool

* First provide the access key
* Then give the secret key which was created in AWS Console
* Then provide Region in which we need to reflect the change
* Then provide the output format file type

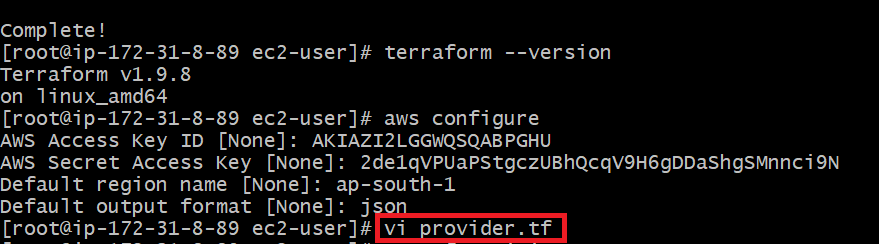


# Step-6: Terraform

* All Terraform files are in the same folder and belong to the same Terraform state file “**.tf” .**
* Make sure to use below commands to avoid unnecessary errors
* **terraform init** command to initialize the Terraform working directory with the AWS plugins
* **terraform validate** to verify your Terraform HCL file
* **terraform plan** to check out the desired changes on every Terraform file creation
* **terraform apply** to create the resources in AWS

# Step-7: AWS Terraform Provider

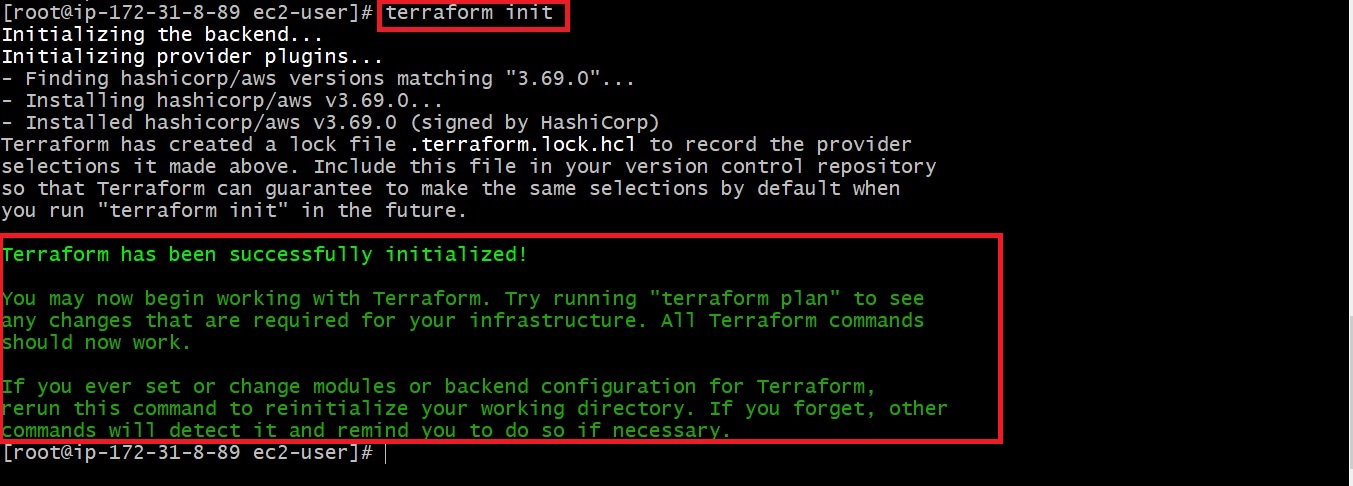
* First create one file in EC2 instance but using any editor like vim,nano,etc…,



* In the file need to enter the below code and exit from file by using **:wq** command
* In the code need to change the region in which we need to do

|  |
| --- |
| terraform {  required\_providers {  aws = {  source = "hashicorp/aws"  version = "3.69.0"  }  }  }  provider "aws" {  profile = "default"  region = "us-east-2"  } |

* After exit from the file need to use **“terraform init”** commandto initialize the Terraform working directory



# Step-8: Create an AWS CloudWatch dashboard

* After initialize terraform with provider details  we will create a dashboard that monitors the Maximum CPU utilization of all running EC2 instances in your AWS account
* For creating dashboard enter below code in the file

**resource "aws\_cloudwatch\_dashboard" "EC2\_Dashboard" {**

**dashboard\_name = "EC2-Dashboard"**

**dashboard\_body = <<EOF**

**{**

**"widgets": [**

**{**

**"type": "explorer",**

**"width": 24,**

**"height": 15,**

**"x": 0,**

**"y": 0,**

**"properties": {**

**"metrics": [**

**{**

**"metricName": "CPUUtilization",**

**"resourceType": "AWS::EC2::Instance",**

**"stat": "Maximum"**

**}**

**],**

**"aggregateBy": {**

**"key": "InstanceType",**

**"func": "MAX"**

**},**

**"labels": [**

**{**

**"key": "State",**

**"value": "running"**

**}**

**],**

**"widgetOptions": {**

**"legend": {**

**"position": "bottom"**

**},**

**"view": "timeSeries",**

**"rowsPerPage": 8,**

**"widgetsPerRow": 2**

**},**

**"period": 60,**

**"title": "Running EC2 Instances CPUUtilization"**

**}**

**}**

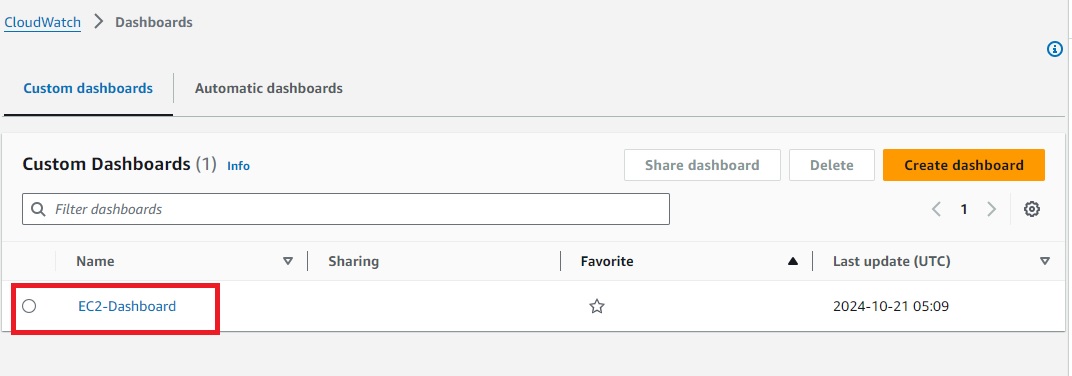
**]**

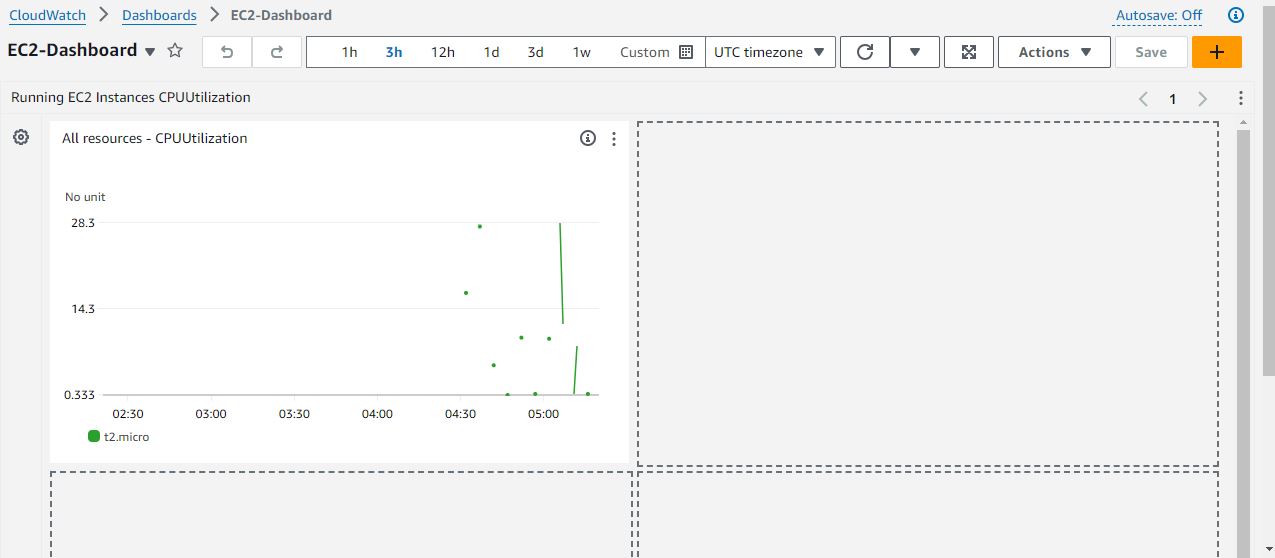
**}**

**EOF**

**}**

* After that run the **terraform plan**  to check out the desired changes on every Terraform file creation
* Then run **terraform apply** to create the resources in AWS
* Then you can log in to the AWS console and access the CloudWatch service dashboard, where you should be able to see the maximum CPU utilization of all the running EC2 instances in your AWS account





### Step-9: Create an AWS CloudWatch metric alarm

### To create an AWS CloudWatch metric alarm

### you can use the [aws\_cloudwatch\_metric\_alarm](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/cloudwatch_metric_alarm) resource and pass the required arguments, such as the alarm\_name, comparison\_operator, and evaluation\_periods

* For creating Cloudwatch metric alarm enter below code in the file

**resource "aws\_cloudwatch\_metric\_alarm" "EC2\_CPU\_Usage\_Alarm" {**

**alarm\_name = "EC2\_CPU\_Usage\_Alarm"**

**comparison\_operator = "GreaterThanOrEqualToThreshold"**

**evaluation\_periods = "2"**

**metric\_name = "CPUUtilization"**

**namespace = "AWS/EC2"**

**period = "60"**

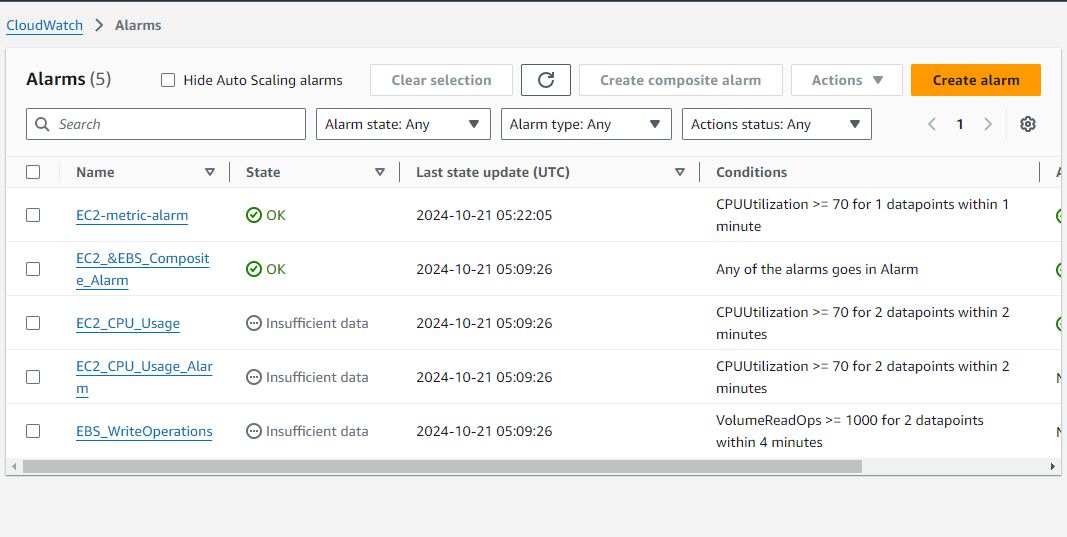
**statistic = "Average"**

**threshold = "70"**

**alarm\_description = "This metric monitors ec2 cpu utilization exceeding 70%"**

**}**

* After that run the **terraform plan**  to check out the desired changes on every Terraform file creation
* Then run **terraform apply** to create the resources in AWS
* Then you can log in to the AWS console and access the CloudWatch service check for all alarms



### Step-10: Create an AWS CloudWatch metric alarm with EC2 Auto-scaling

* You can create an AWS CloudWatch metric alarm that triggers multiple actions depending on the defined CloudWatch metric alarm conditions
* In the following example, we shall create an [AWS CloudWatch metric alarm that monitors the average CPU usage and triggers an AWS AutoScaling](https://hands-on.cloud/managing-aws-auto-scaling-using-terraform/) policy to spin up more EC2 instances in an AWS AutoScaling group, should the average CPU Utilization average at 70 % for one minute
* For creating CloudWatch metric alarm with EC2 Auto-scaling enter below code in the file
* In code need to change Image Id with our instance ami id and change the availability zone with EC2 instance zone

**resource "aws\_launch\_template" "EC2\_Launch\_Template" {**

**name\_prefix = "EC2-Launch-Template"**

**image\_id = "ami-0f540030bb04d884a"**

**instance\_type = "t2.micro"**

**}**

**resource "aws\_autoscaling\_group" "EC2\_AutoScaling\_Group" {**

**availability\_zones = ["us-east-2b"]**

**desired\_capacity = 1**

**max\_size = 5**

**min\_size = 1**

**launch\_template {**

**id = aws\_launch\_template.EC2\_Launch\_Template.id**

**version = "$Latest"**

**}**

**depends\_on = [**

**aws\_launch\_template.EC2\_Launch\_Template,**

**]**

**}**

**resource "aws\_autoscaling\_policy" "EC2\_AutoScaling\_Policy" {**

**name = "EC2-AutoScaling-Policy"**

**scaling\_adjustment = 2**

**adjustment\_type = "ChangeInCapacity"**

**cooldown = 60**

**autoscaling\_group\_name = aws\_autoscaling\_group.EC2\_AutoScaling\_Group.name**

**depends\_on = [**

**aws\_autoscaling\_group.EC2\_AutoScaling\_Group,**

**]**

**}**

**resource "aws\_cloudwatch\_metric\_alarm" "EC2\_metric\_alarm" {**

**alarm\_name = "EC2-metric-alarm"**

**comparison\_operator = "GreaterThanOrEqualToThreshold"**

**evaluation\_periods = "1"**

**metric\_name = "CPUUtilization"**

**namespace = "AWS/EC2"**

**period = "60"**

**statistic = "Average"**

**threshold = "70"**

**depends\_on = [**

**aws\_autoscaling\_group.EC2\_AutoScaling\_Group,**

**]**

**dimensions = {**

**AutoScalingGroupName = aws\_autoscaling\_group.EC2\_AutoScaling\_Group.name**

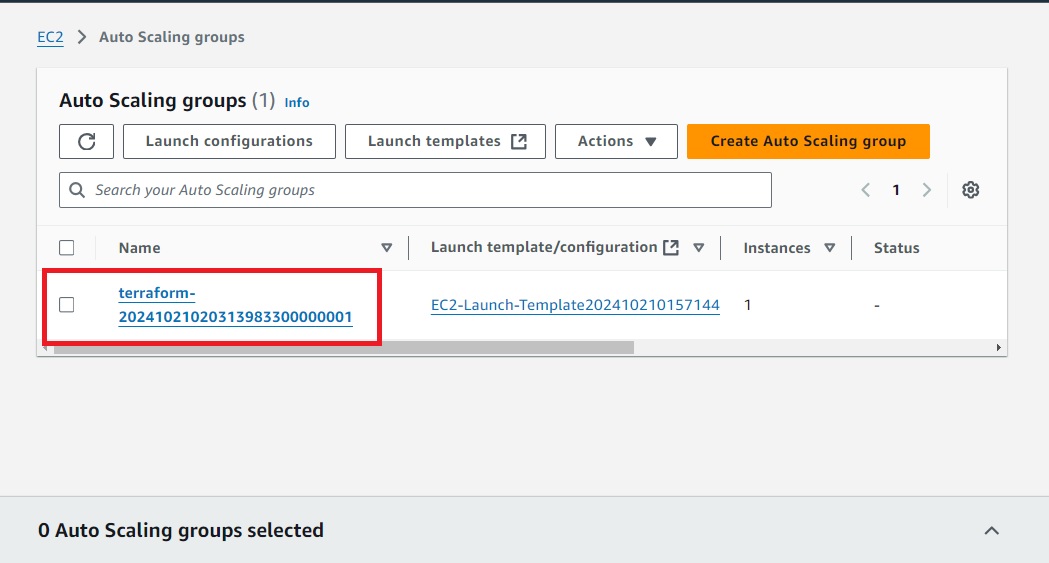
**}**

**alarm\_description = "This metric monitors ec2 cpu utilization"**

**alarm\_actions = [aws\_autoscaling\_policy.EC2\_AutoScaling\_Policy.arn]**

**}**

* After that run the **terraform plan**  to check out the desired changes on every Terraform file creation
* Then run **terraform apply** to create the resources in AWS
* Then you can log in to the AWS console and access the EC2 service check for Autoscaling on left side



### Step-12: Create an AWS CloudWatch composite alarm

* AWS[CloudWatch Composite alarms](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Create_Composite_Alarm.html) are used to monitor two or more alarms to determine the alarm state and take action.
* The most common use of Composite alarms is to reduce alarm notifications by grouping multiple alarms and defining which metrics within’ alarms should trigger the different alarm states
* To create a Composite alarm using Terraform, you can use the [aws\_cloudwatch\_composite\_alarm](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/cloudwatch_composite_alarm) resource and pass the required argument, which is the alarm\_name
* The [aws\_sns\_topic](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/sns_topic) and the [aws\_sns\_topic\_subscription](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/sns_topic_subscription) resources are used to configure notifications
* For creating AWS CloudWatch composite alarm enter below code in the file
* In the code need to change Email Id for those who want to receive the notification regarding monitoring

**resource "aws\_cloudwatch\_composite\_alarm" "EC2\_and\_EBS" {**

**alarm\_description = "Composite alarm that monitors CPUUtilization and EBS Volume Write Operations"**

**alarm\_name = "EC2\_&EBS\_Composite\_Alarm"**

**alarm\_actions = [aws\_sns\_topic.EC2\_and\_EBS\_topic.arn]**

**alarm\_rule = "ALARM(${aws\_cloudwatch\_metric\_alarm.EC2\_CPU\_Usage\_Alarm.alarm\_name}) OR ALARM(${aws\_cloudwatch\_metric\_alarm.EBS\_WriteOperations.alarm\_name})"**

**depends\_on = [**

**aws\_cloudwatch\_metric\_alarm.EC2\_CPU\_Usage\_Alarm,**

**aws\_cloudwatch\_metric\_alarm.EBS\_WriteOperations,**

**aws\_sns\_topic.EC2\_and\_EBS\_topic,**

**aws\_sns\_topic\_subscription.EC2\_and\_EBS\_Subscription**

**]**

**}**

**resource "aws\_cloudwatch\_metric\_alarm" "EC2\_CPU\_Usage\_Alarm" {**

**alarm\_name = "EC2\_CPU\_Usage\_Alarm"**

**comparison\_operator = "GreaterThanOrEqualToThreshold"**

**evaluation\_periods = "2"**

**metric\_name = "CPUUtilization"**

**namespace = "AWS/EC2"**

**period = "60"**

**statistic = "Average"**

**threshold = "70"**

**alarm\_description = "This metric monitors ec2 cpu utilization exceeding 70%"**

**}**

**resource "aws\_cloudwatch\_metric\_alarm" "EBS\_WriteOperations" {**

**alarm\_name = "EBS\_WriteOperations"**

**comparison\_operator = "GreaterThanOrEqualToThreshold"**

**evaluation\_periods = "2"**

**metric\_name = "VolumeReadOps"**

**namespace = "AWS/EC2"**

**period = "120"**

**statistic = "Average"**

**threshold = "1000"**

**alarm\_description = "This monitors the average read operations on EBS Volumes in a specified period of time"**

**}**

**resource "aws\_sns\_topic" "EC2\_and\_EBS\_topic" {**

**name = "EC2\_and\_EBS\_topic"**

**}**

**resource "aws\_sns\_topic\_subscription" "EC2\_and\_EBS\_Subscription" {**

**topic\_arn = aws\_sns\_topic.EC2\_and\_EBS\_topic.arn**

**protocol = "email"**

**endpoint = "kelvingalabuzi@gmail.com"**

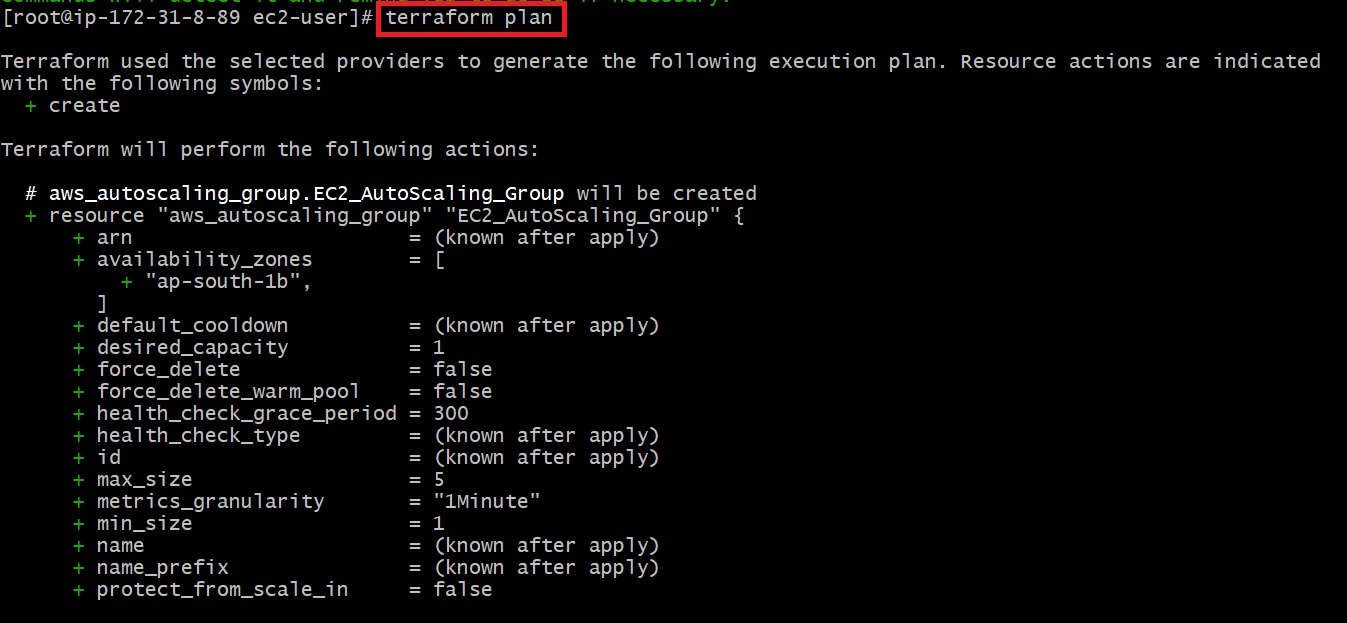
**depends\_on = [**

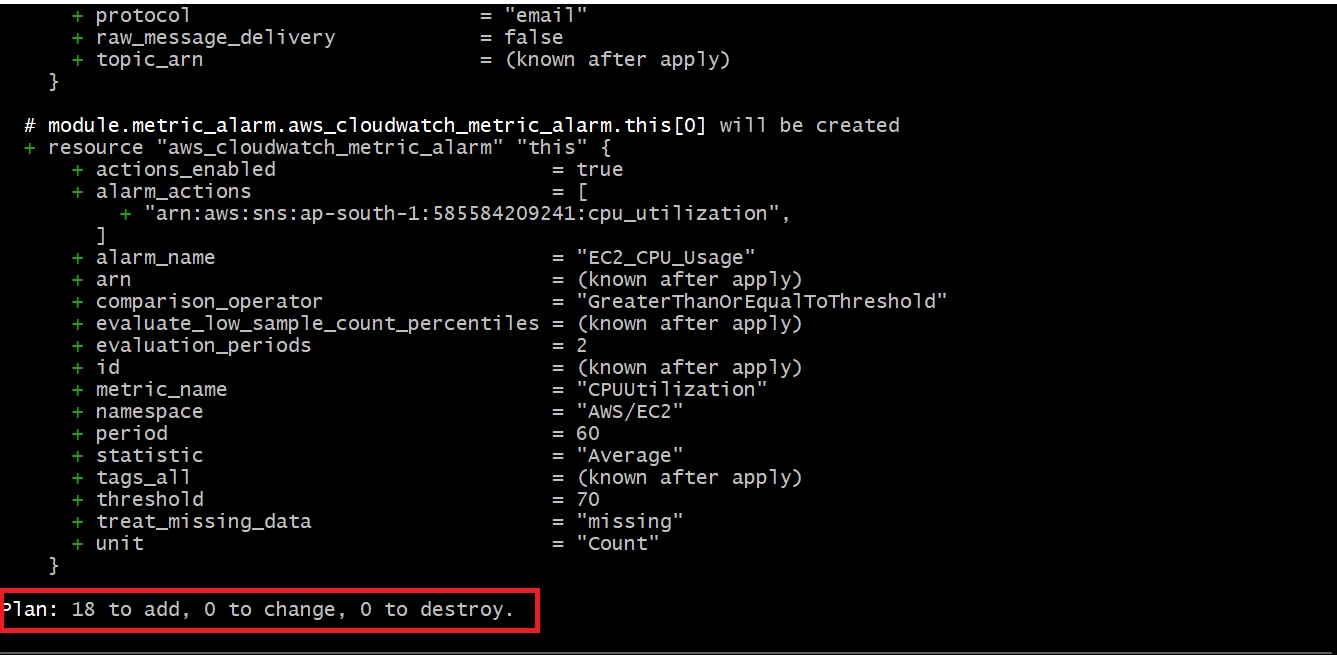
**aws\_sns\_topic.EC2\_and\_EBS\_topic**

**]**

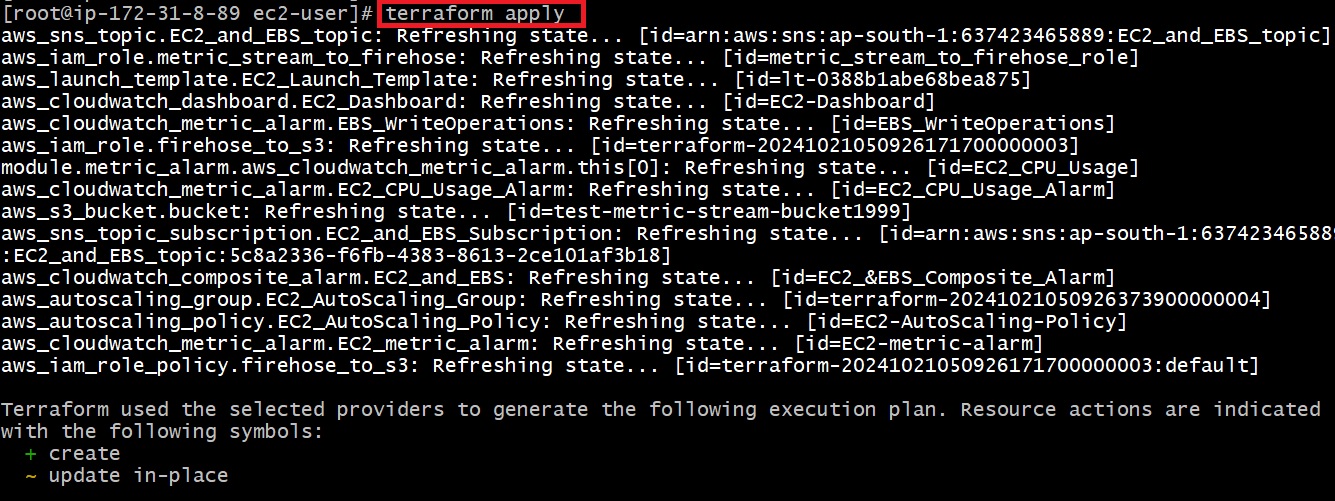
**}**

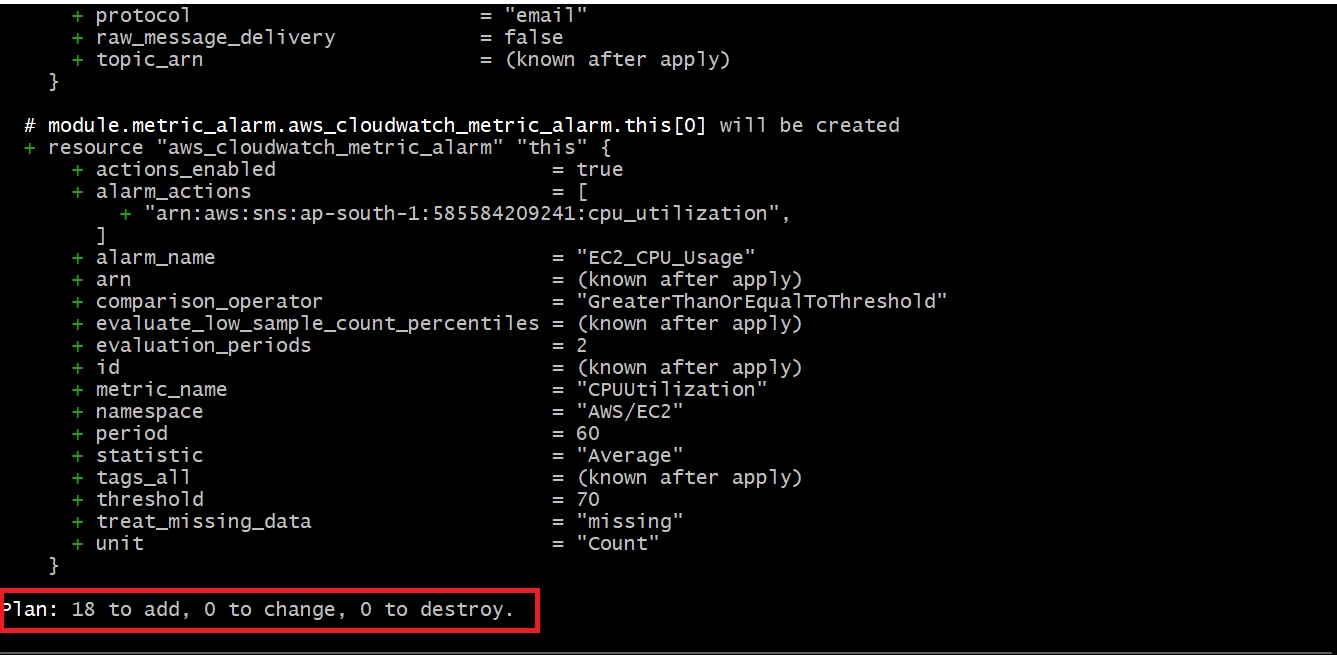
* After that run the **terraform plan**  to check out the desired changes on every Terraform file creation



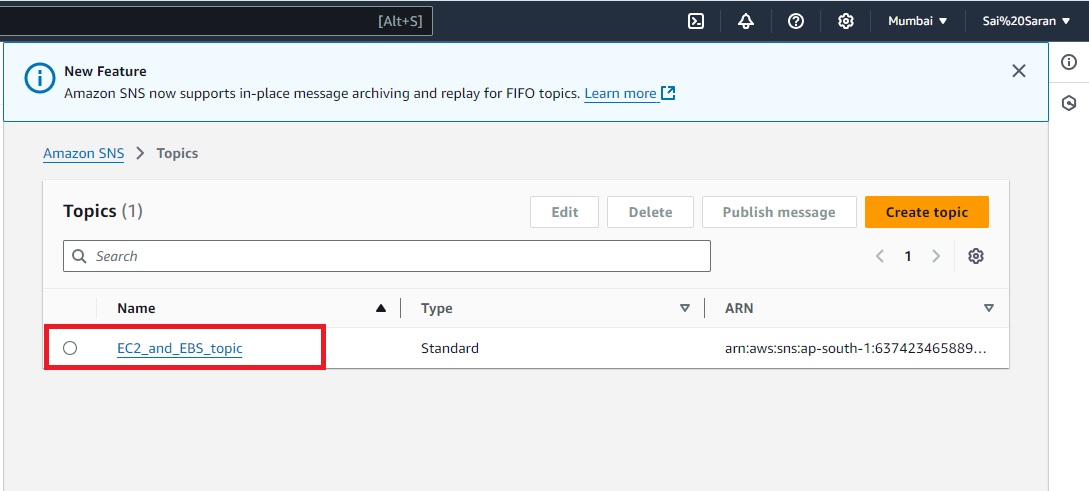


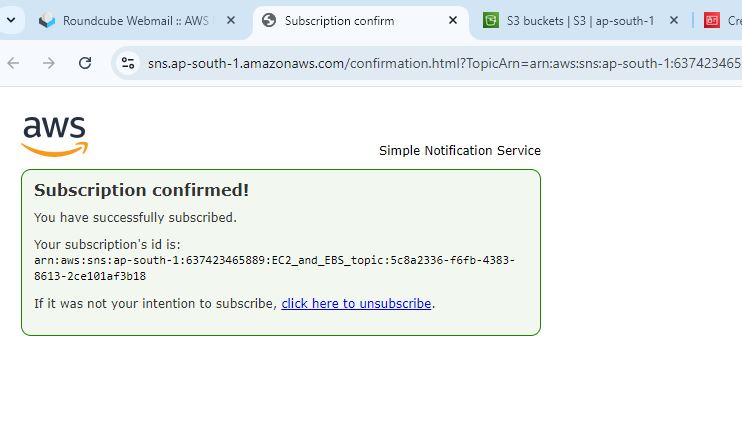
* Then run **terraform apply** to create the resources in AWS





* Then you can log in to the AWS console and access the SNS and check the Emails for notification.





**AWS DATABASE**

**What is Database:-**

A database is an electronically stored, systematic collection of data. It can contain any type of data, including words, numbers, images, videos, and files. You can use software called a database management system (DBMS) to store, retrieve, and edit data.

**What is Relational Database:-**

* A relational database (RDB) is a type of database that organizes data into tables, rows, and columns to establish relationships between data points
* The relational model is a standard way of representing and querying data

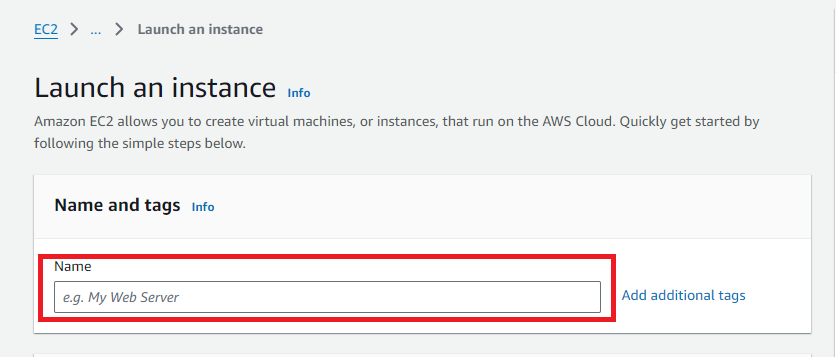
**What is AWS Database :-**

Aws Database is a collection of databases that offer a high-performance, secure, and reliable foundation for data-driven applications and generative AI solutions

**Steps to create RDS by using terraform:-**

**Steps to create Terraform RDS:**

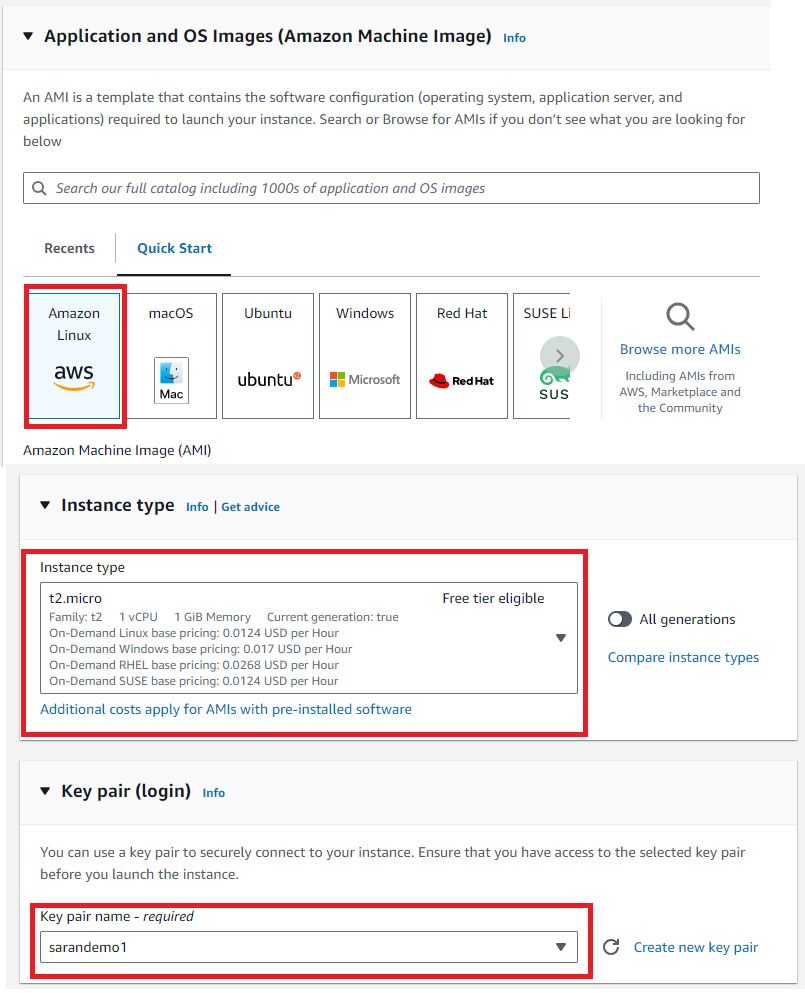
**Step-1: Create AWS EC2 Instance in AWS Console**

* Go to the AWS console search EC2 Service
* Then Click on launch Instance
* After click on launch instance provide the name for the Instance

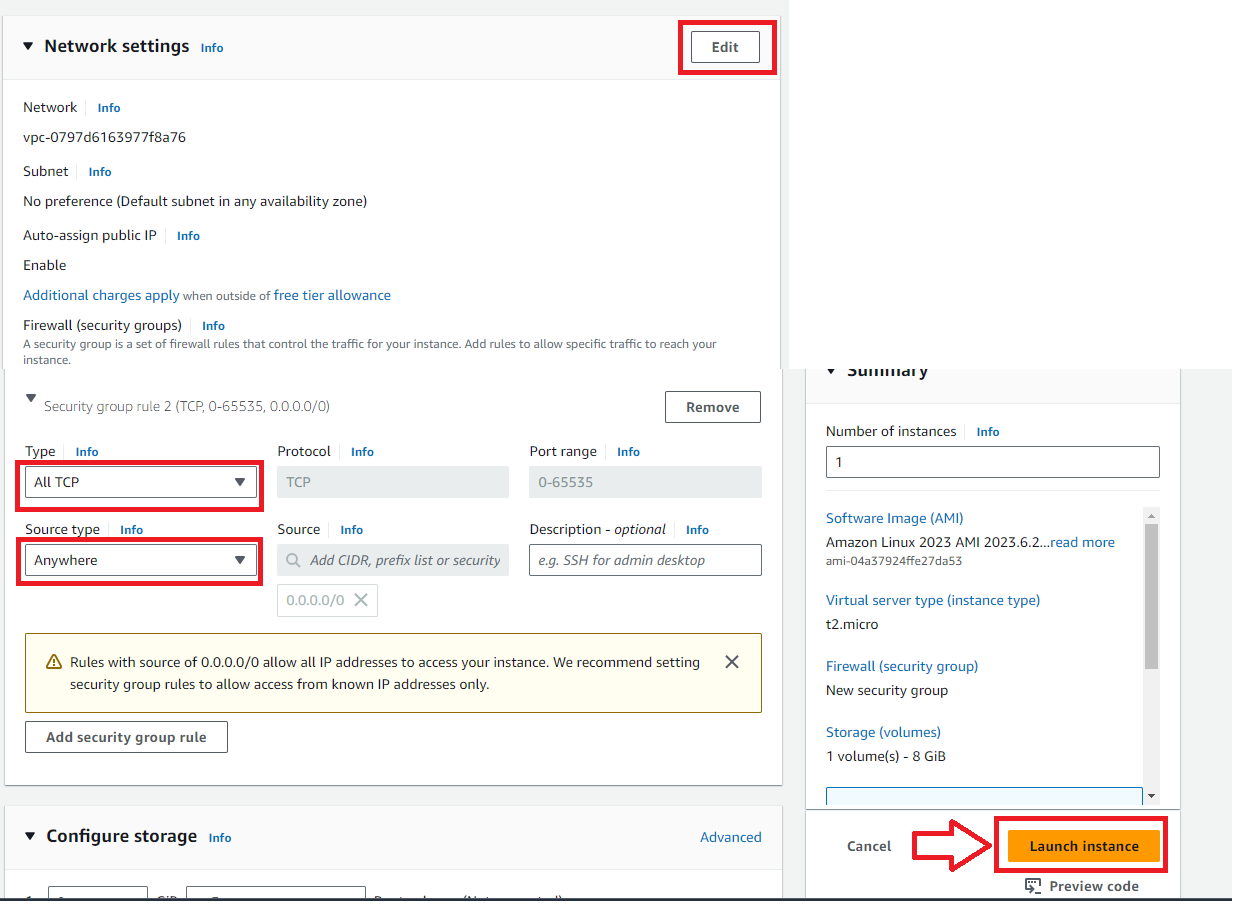
\* Then select Amazon Machine Image (AMI)

\* After that select Instance type

Then provide the key pair if existing key pair is not available need to create new key pair by

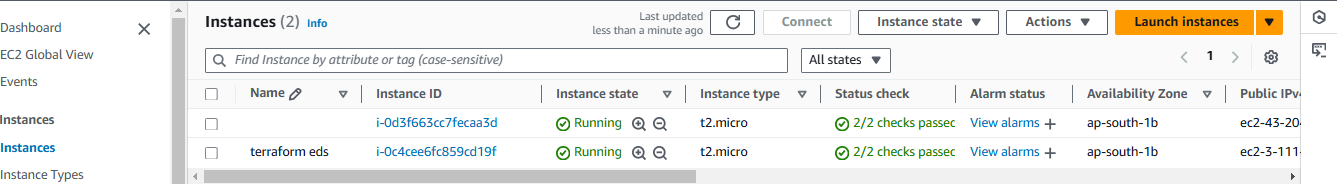


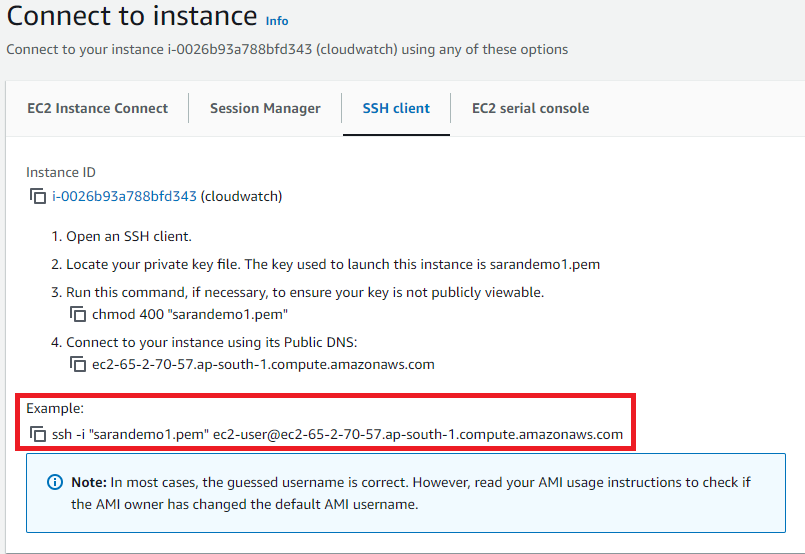
* In Network settings edit network settings and add All TCP
* Then click on
* Launch Instance



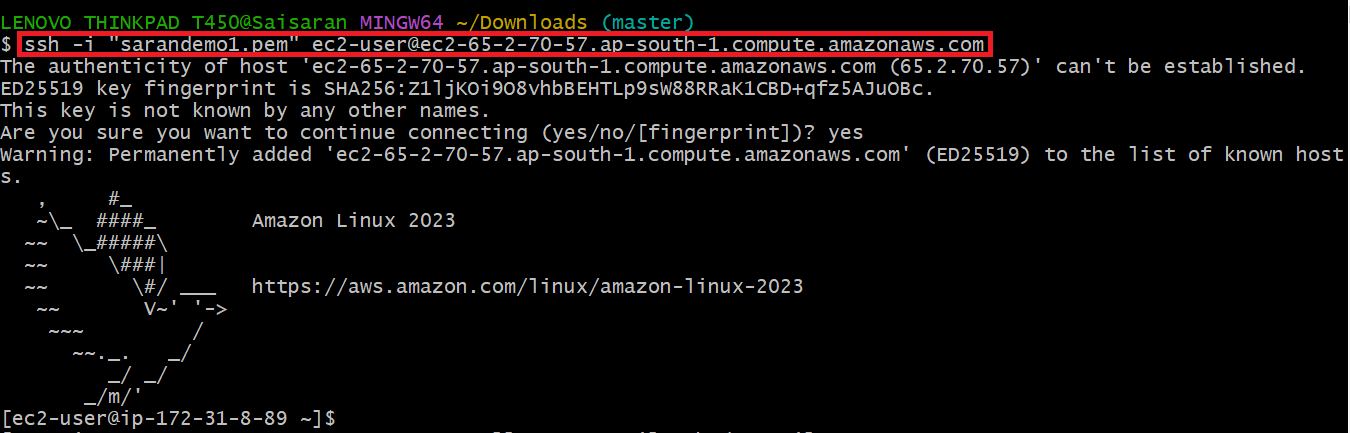
**Step-2: Connect EC2 Instance with Tools**

* After creating the Instance need to click on connect on AWS Console
* Then Copy the SSH link



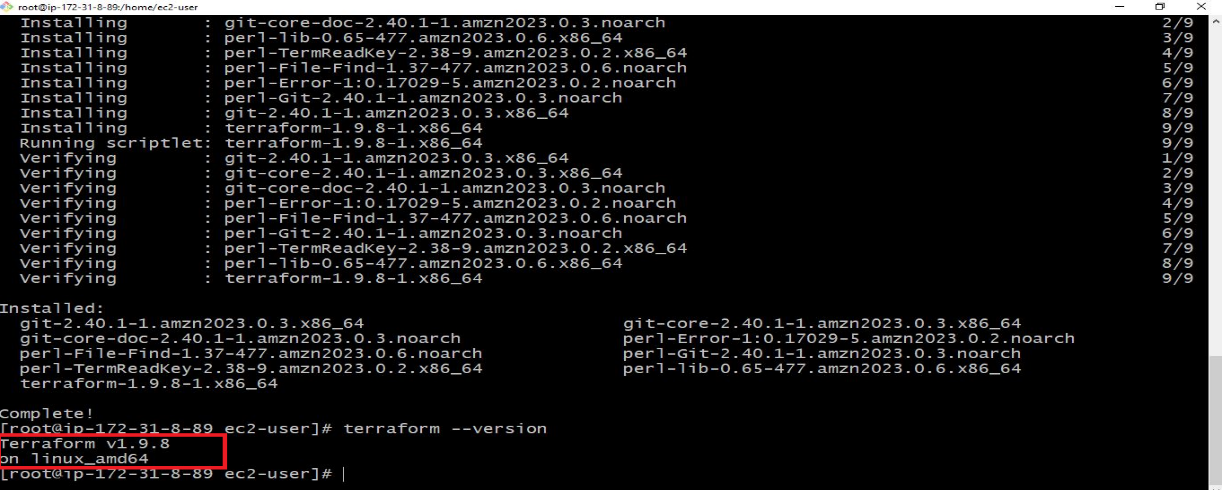


* After copying the SSH link need to connect GIT,MobaXterm,etc..,
* Here I am connecting with GIT bash for need to got the key pair downloaded path and open GIT bash there and paste the SHH link then click on yes the Linux is connected to tool.

**Step-3:Install Terraform in EC2 Instance**

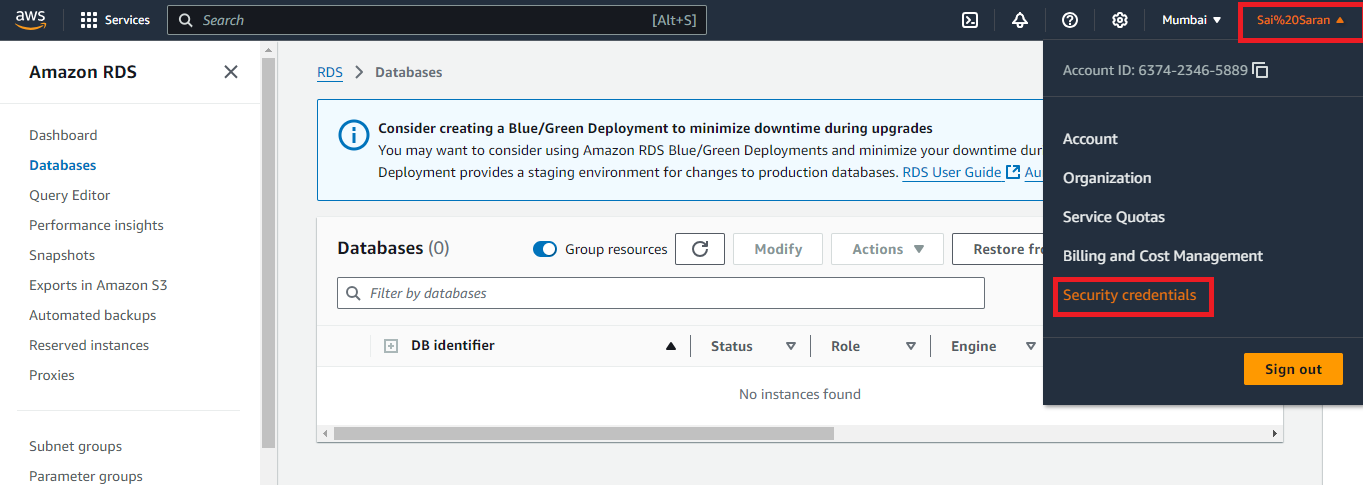
After connecting with Linux need to use below commands

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

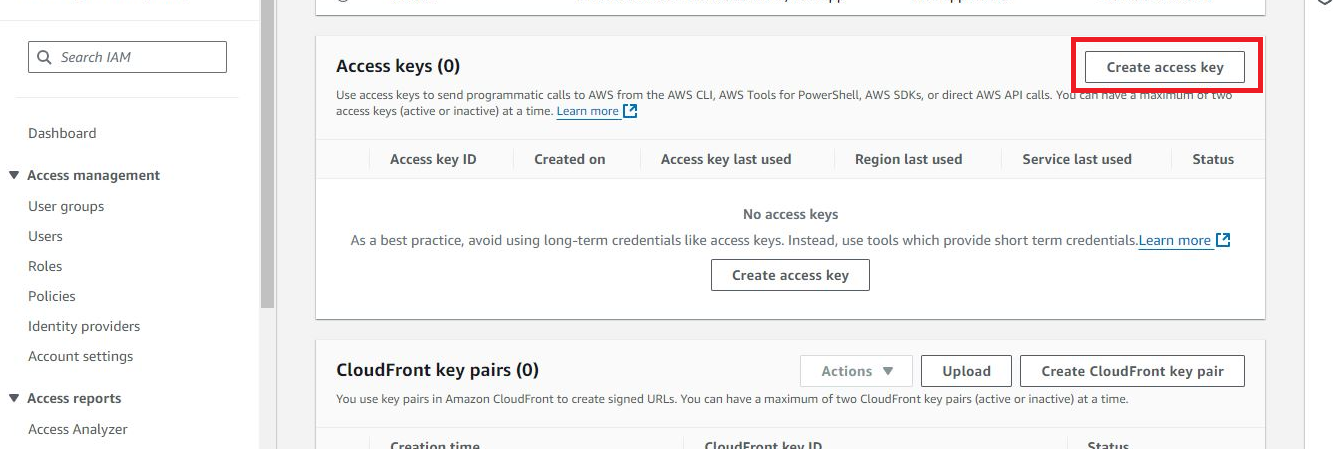


**Step-4: Security credentials creation in AWS Console**

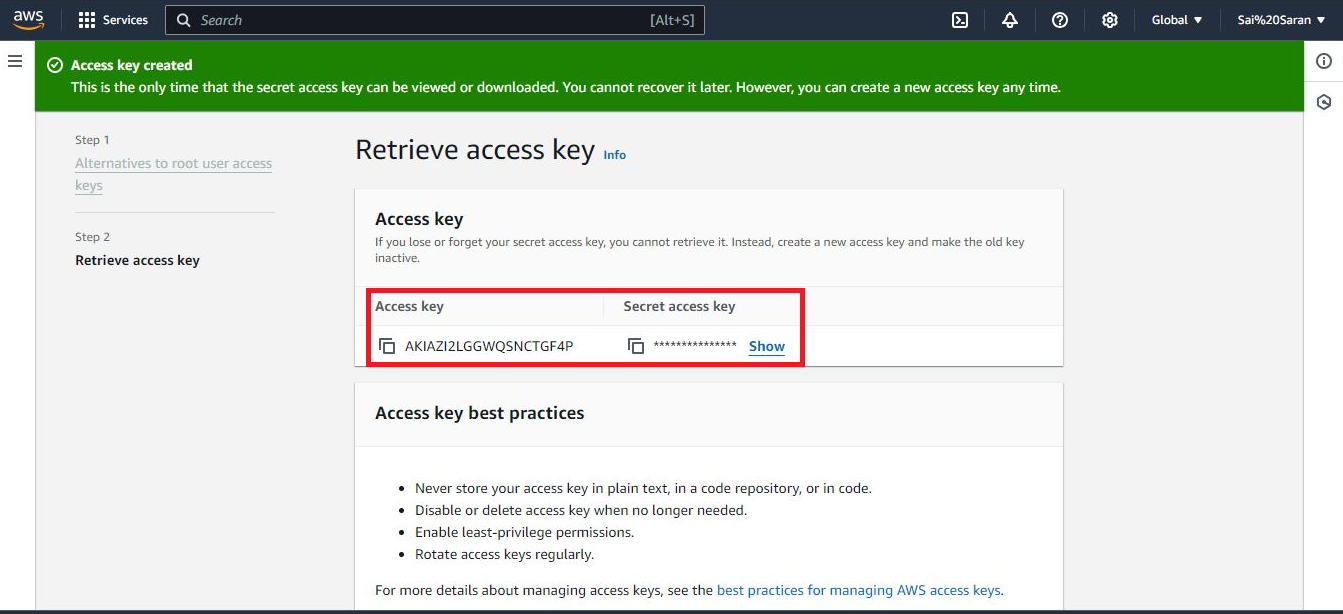
Open AWS Console on the top right there is profile name in that we can see security credentials



After opening the security credentials scroll down and check for create access key



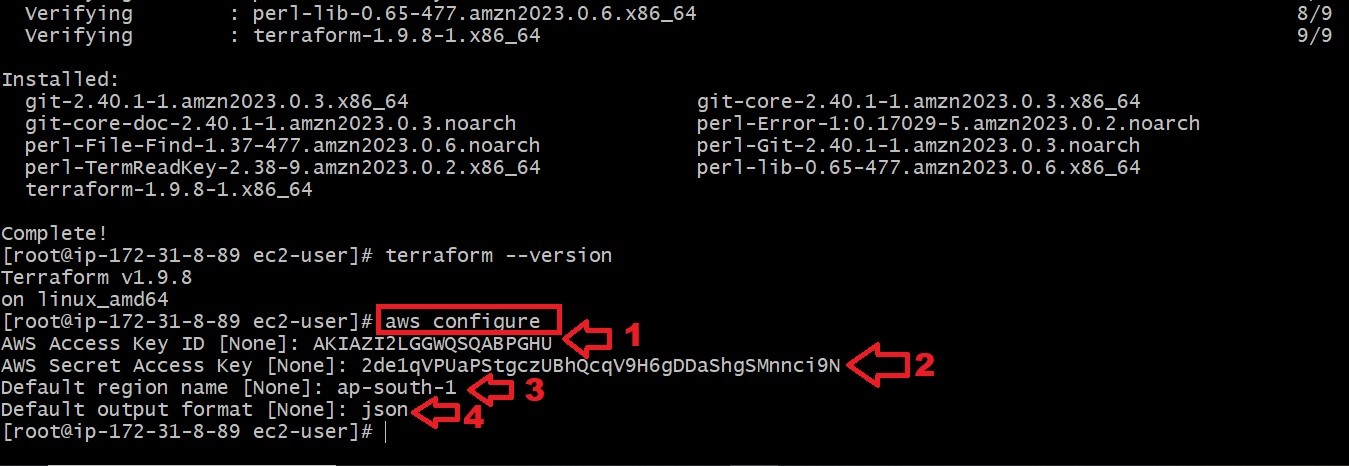
* Then Click on next the access was created and we will get access Id and secret key



**Step-5:AWS Configure with Security credentials**

After creating the access key in AWS Console then use the command “**aws configure”** to configure the access key with tool

* First provide the access key
* Then give the secret key which was created in AWS Console
* Then provide Region in which we need to reflect the change
* Then provide the output format file type



**Step-6: Terraform**

* All Terraform files are in the same folder and belong to the same Terraform state file “**.tf” .**
* Make sure to use below commands to avoid unnecessary errors
* **terraform init** command to initialize the Terraform working directory with the AWS plugins
* **terraform validate** to verify your Terraform HCL file
* **terraform plan** to check out the desired changes on every Terraform file creation
* **terraform apply** to create the resources in AWS
* clicking on create new pair

**Step-7: Create a file by using vim editor vim rds.tf**

**\*After entering into vim editor enter ‘I’ to enter into insert mode and write below code provider "aws" {**

**region = "us-east-1"**

**}**

**resource "aws\_db\_instance" "myinstance12" {**

**engine = "mysql"**

**identifier = "myrdsinstance1"**

**allocated\_storage = 20**

**#db\_name = mydb**

**engine\_version = "8.0.39"**

**instance\_class = "db.t3.micro"**

**username = "myrdsuser"**

**password = "myrdspassword"**

**parameter\_group\_name = "default.mysql8.0"**

**skip\_final\_snapshot = true**

**}**

**resource "aws\_vpc" "demo-vpc" {**

**cidr\_block = "10.0.0.0/16"**

**tags = {**

**Name = "demo-vpc"**

**}**

**}**

**resource "aws\_internet\_gateway" "demo-igw" {**

**vpc\_id = aws\_vpc.demo-vpc.id**

**tags = {**

**Name = "demo-vpc-IGW"**

**}**

**}**

**resource "aws\_subnet" "private-subnet-1" {**

**vpc\_id = aws\_vpc.demo-vpc.id**

**cidr\_block = "10.0.1.0/24"**

**availability\_zone = "us-east-1a"**

**tags = {**

**Name = "private-subnet-1"**

**}**

**}**

**resource "aws\_subnet" "private-subnet-2" {**

**vpc\_id = aws\_vpc.demo-vpc.id**

**cidr\_block = "10.0.2.0/24"**

**availability\_zone = "us-east-1c"**

**tags = {**

**Name = "private-subnet-2"**

**}**

**}**

**resource "aws\_subnet" "public-subnet-1" {**

**vpc\_id = aws\_vpc.demo-vpc.id**

**cidr\_block = "10.0.3.0/24"**

**availability\_zone = "us-east-1a"**

**tags = {**

**Name = "public-subnet-1"**

**}**

**}**

**resource "aws\_subnet" "public-subnet-2" {**

**vpc\_id = aws\_vpc.demo-vpc.id**

**cidr\_block = "10.0.4.0/24"**

**availability\_zone = "us-east-1c"**

**tags = {**

**Name = "public-subnet-2"**

**}**

**}**

**resource "aws\_route\_table" "public-route-table" {**

**vpc\_id = aws\_vpc.demo-vpc.id**

**tags = {**

**Name = "public-route-table"**

**}**

**}**

**resource "aws\_route" "public-route" {**

**route\_table\_id = aws\_route\_table.public-route-table.id**

**destination\_cidr\_block = "0.0.0.0/0"**

**gateway\_id = aws\_internet\_gateway.demo-igw.id**

**}**

**resource "aws\_route\_table\_association" "public-subnet-1-association" {**

**subnet\_id = aws\_subnet.public-subnet-1.id**

**route\_table\_id = aws\_route\_table.public-route-table.id**

**}**

**resource "aws\_route\_table\_association" "public-subnet-2-association" {**

**subnet\_id = aws\_subnet.public-subnet-2.id**

**route\_table\_id = aws\_route\_table.public-route-table.id**

**}**

**resource "aws\_eip" "nat-eip" {**

**vpc = true**

**tags = {**

**Name = "nat-eip"**

**}**

**}**

**resource "aws\_nat\_gateway" "nat-gateway" {**

**allocation\_id = aws\_eip.nat-eip.id**

**subnet\_id = aws\_subnet.public-subnet-1.id**

**tags = {**

**Name = "nat-gateway"**

**}**

**}**

**resource "aws\_security\_group" "secgroup" {**

**name = "secgroup"**

**description = "awssecuritygroup"**

**vpc\_id = aws\_vpc.demo-vpc.id**

**ingress {**

**from\_port = 3306**

**to\_port = 3306**

**protocol = "tcp"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**egress {**

**from\_port =0**

**to\_port = 0**

**protocol ="-1"**

**cidr\_blocks = ["0.0.0.0/0"]**

**}**

**tags = {**

**name ="secgroup"**

**}**

**}**

**resource "aws\_network\_acl" "pub\_az1\_nacl" {**

**vpc\_id = aws\_vpc.demo-vpc.id**

**subnet\_ids = [ aws\_subnet.public-subnet-1.id]**

**tags = {**

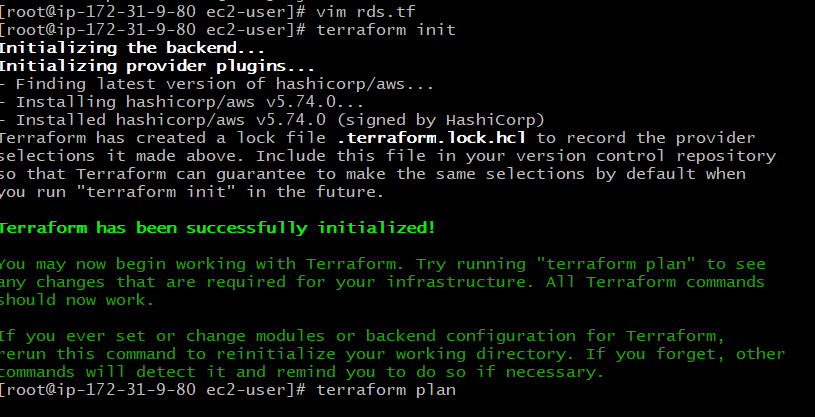
**Name = "Public-AZ1-NACL"**

**}**

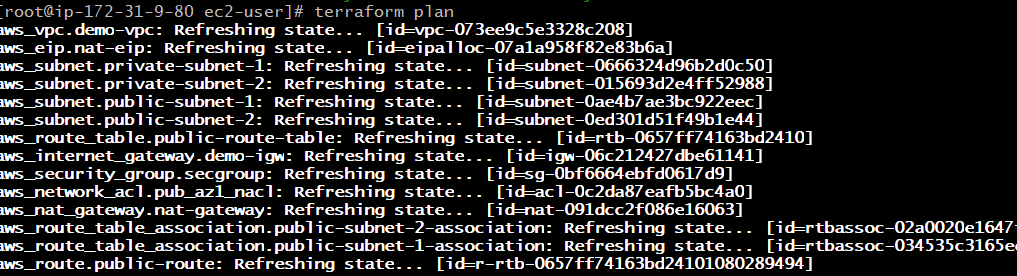
**}**

**\*come to normal mode and gave ‘WQ’ to save &exit from vim editor**

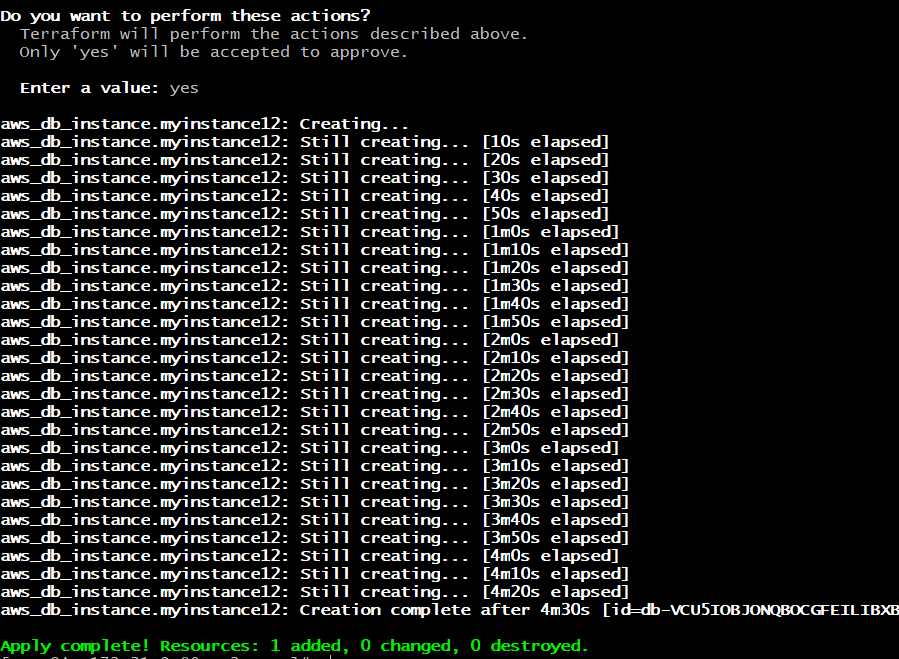
**terraform init** command to initialize the Terraform working directory with the AWS plugins



After that run the **terraform plan**  to check out the desired changes on every Terraform file creation



Then run **terraform apply** to create the resources in AWS



Then you can log in to the AWS console and check database is created or not



VPC: Amazon VPC or Amazon Virtual Private Cloud

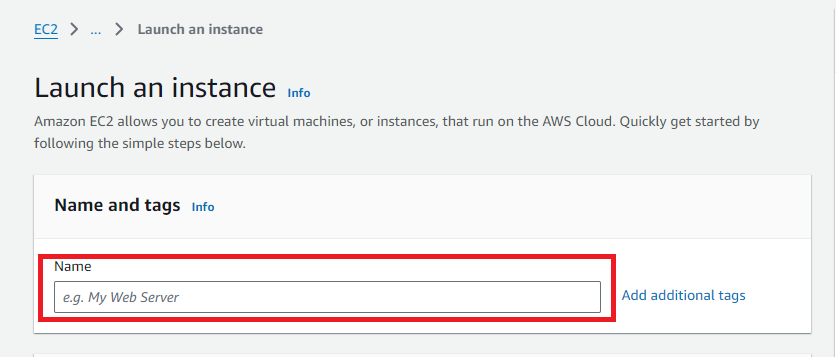
**VPC**: Amazon VPC or Amazon Virtual Private Cloud is a service that allows its users to launch their virtual machines in a protected as well as isolated virtual environment defined by them. You have complete control over your VPC, from creation to customization and even deletion

**Steps to create vpc by using terraform:-**

**Steps to create Terraform vpc:**

**Step-1: Create AWS EC2 Instance in AWS Console**

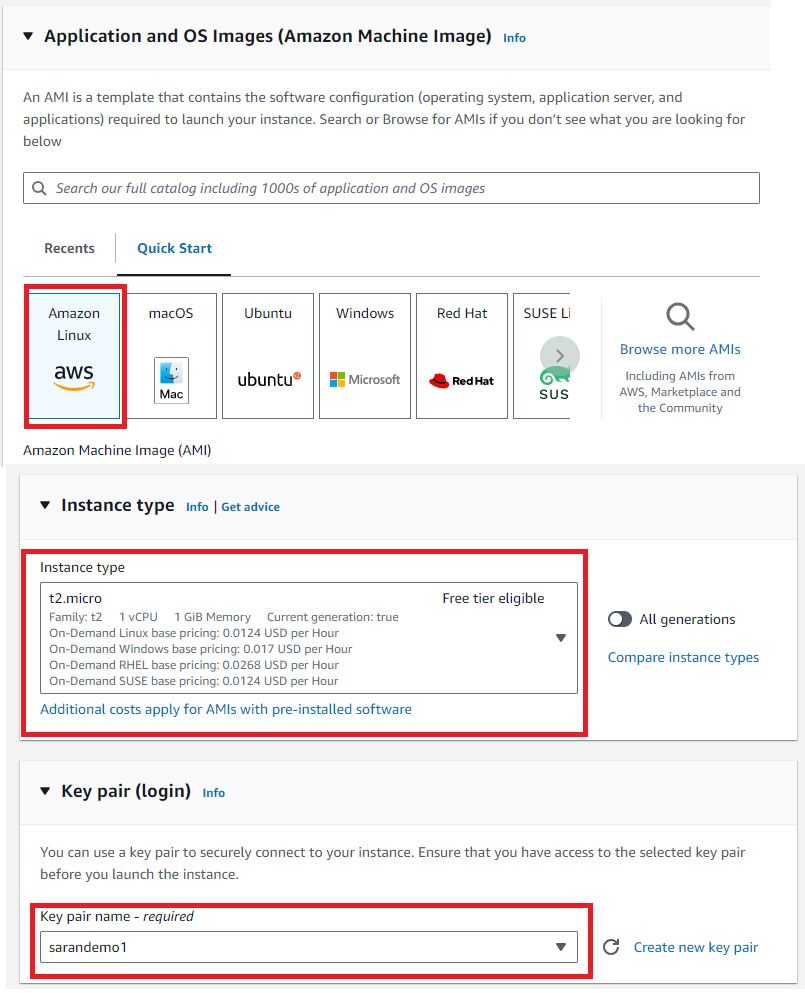
* Go to the AWS console search EC2 Service
* Then Click on launch Instance
* After click on launch instance provide the name for the Instance



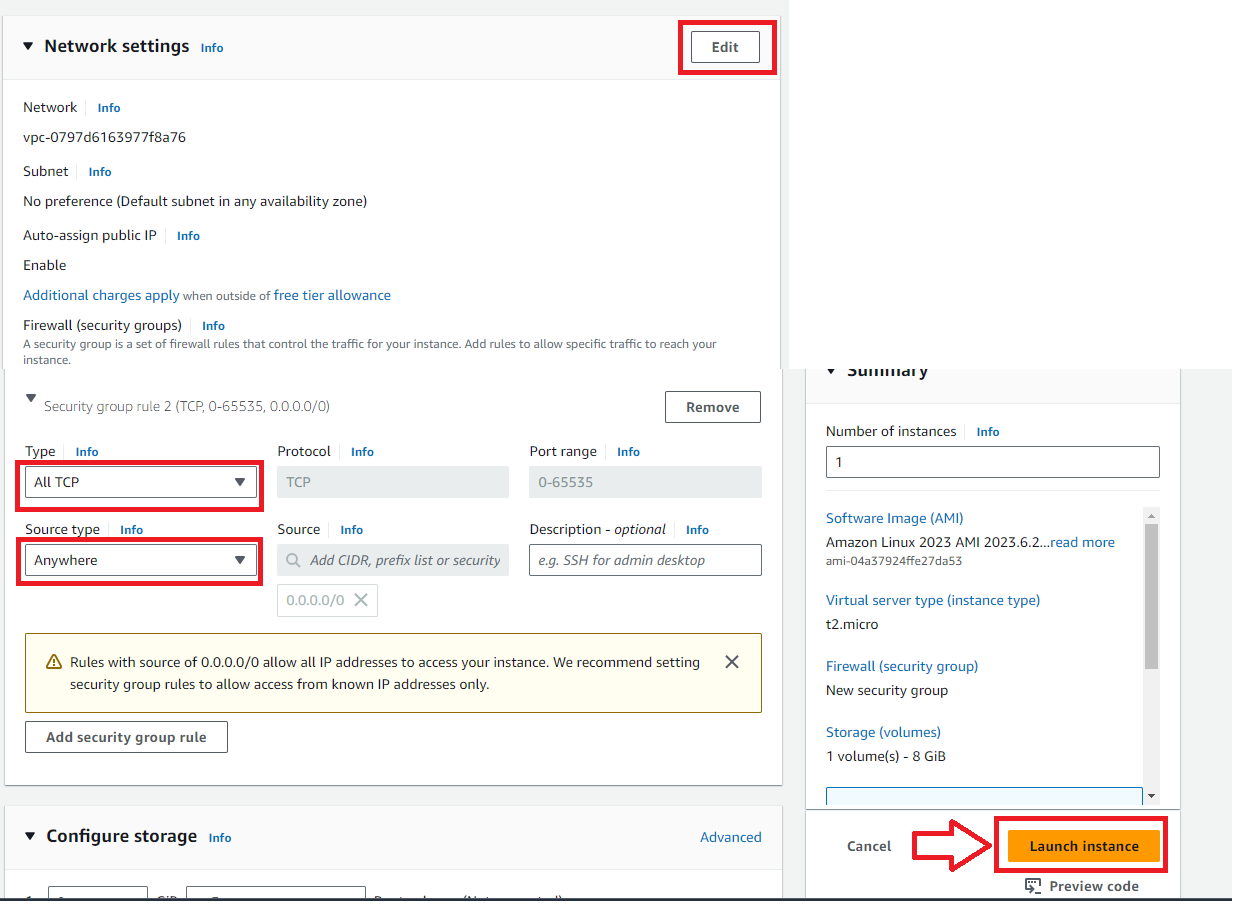
\* Then select Amazon Machine Image (AMI)

\* After that select Instance type

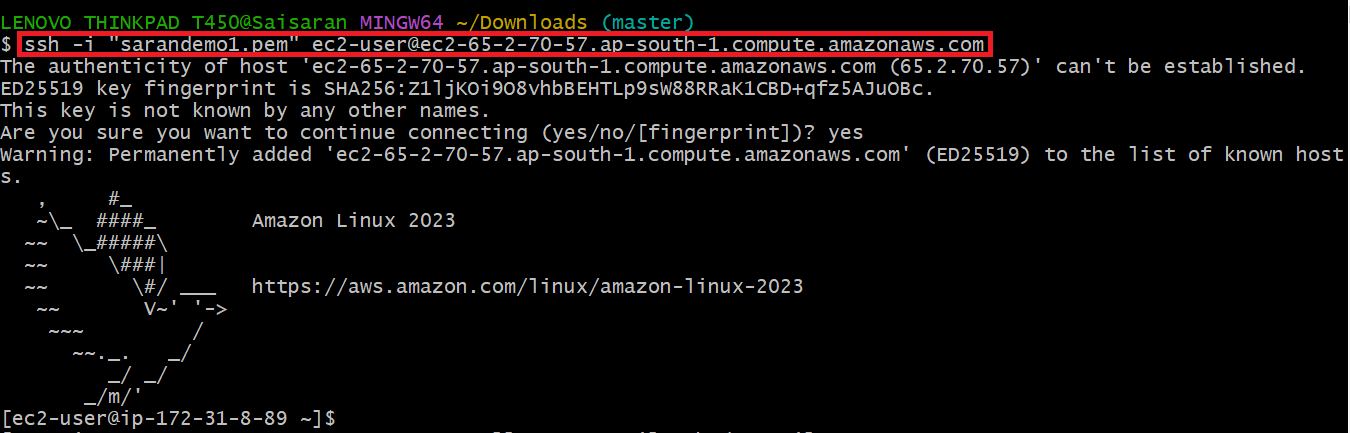
Then provide the key pair if existing key pair is not available need to create new key pair by



* In Network settings edit network settings and add All TCP
* Then click on
* Launch Instance



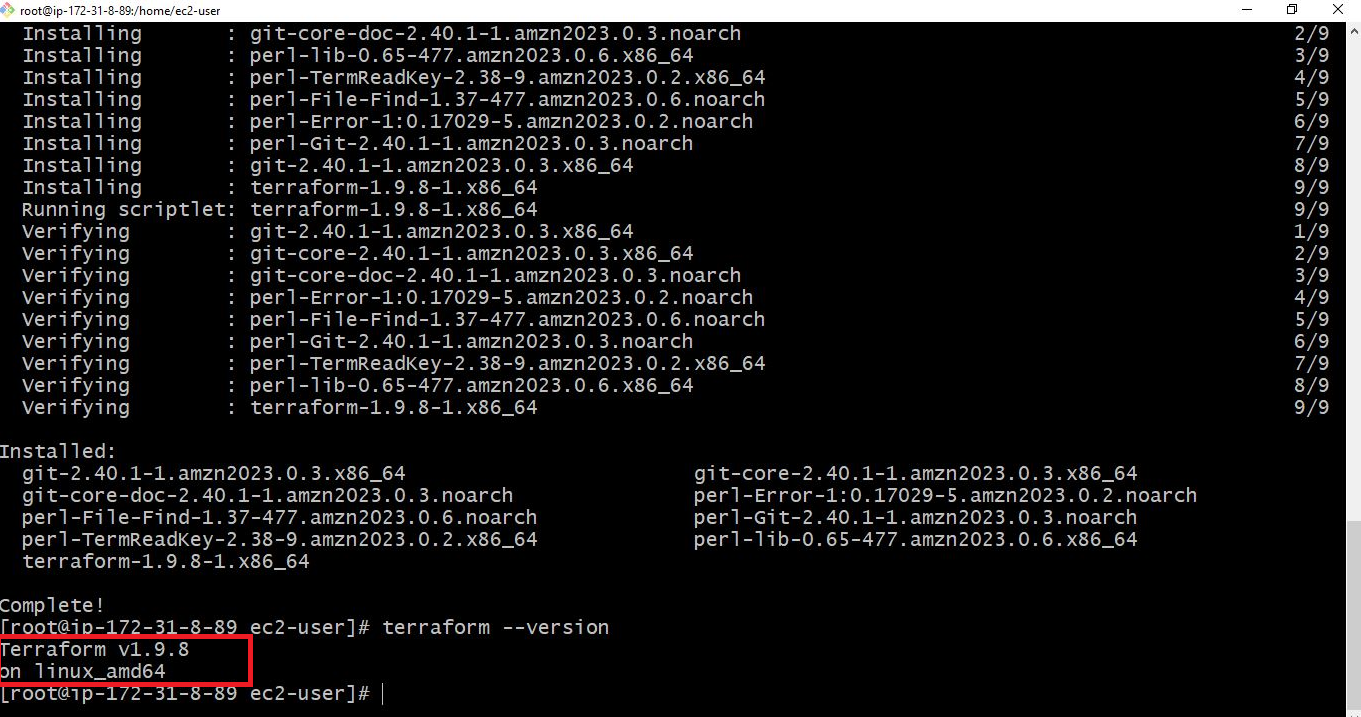
* After copying the SSH link need to connect GIT,MobaXterm,etc..,
* Here I am connecting with GIT bash for need to got the key pair downloaded path and open GIT bash there and paste the SHH link then click on yes the Linux is connected to tool.



**Step-3:Install Terraform in EC2 Instance**

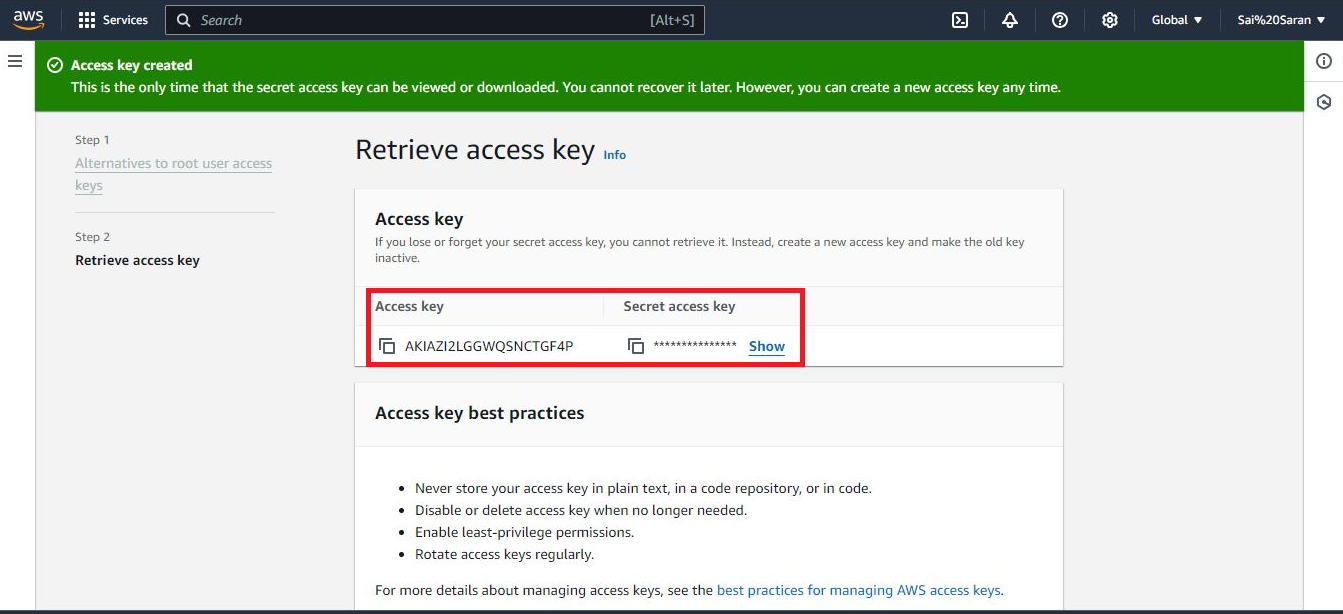
After connecting with Linux need to use below commands

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



**Step-4:Security credentials creation in AWS Console**

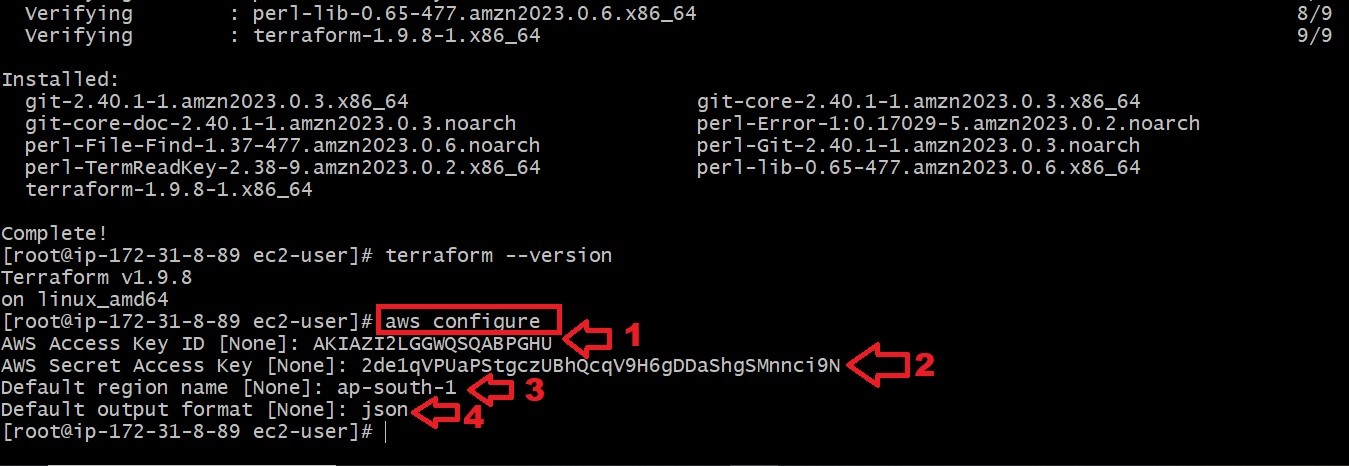
* Open AWS Console on the top right there is profile name in that we can see security credentials



**Step-5:AWS Configure with Security credentials**

After creating the access key in AWS Console then use the command “**aws configure”** to configure the access key with tool

* First provide the access key
* Then give the secret key which was created in AWS Console
* Then provide Region in which we need to reflect the change
* Then provide the output format file type



**Step-6: Terraform**

* All Terraform files are in the same folder and belong to the same Terraform state file “**.tf” .**
* Make sure to use below commands to avoid unnecessary errors
* **terraform init** command to initialize the Terraform working directory with the AWS plugins
* **terraform validate** to verify your Terraform HCL file
* **terraform plan** to check out the desired changes on every Terraform file creation
* **terraform apply** to create the resources in AWS
* clicking on create new pair

**Step-7: Create a file by using vim editor vim vpc.tf**

**Create vpc by using below** **code**

**code**

provider "aws" {

  region = "eu-west-1"

}

resource "aws\_vpc" "demo-vpc" {

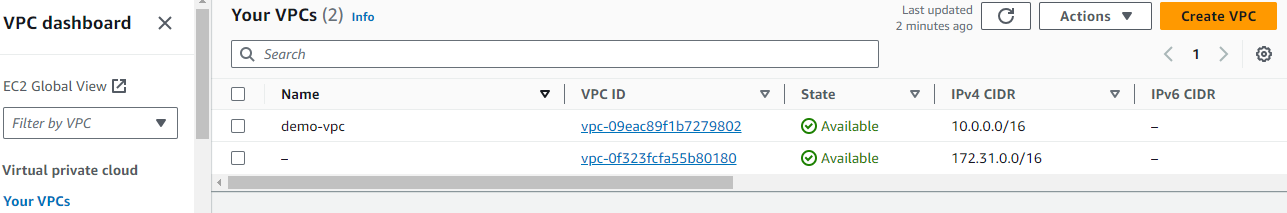
  cidr\_block = "10.0.0.0/16"

  tags = {

    Name = "demo-vpc"

  }

}



**Internet Gateway**

An internet gateway is a logical connection between the vpc and internet. It allows communication between resources within the vpc and the internet. Each vpc has only one IGW and supports both IPv4 and IPv6 traffic.

**STEP-6 Create** **internet gateway by using below code**

**CODE**:

resource "aws\_internet\_gateway" "demo-igw" {

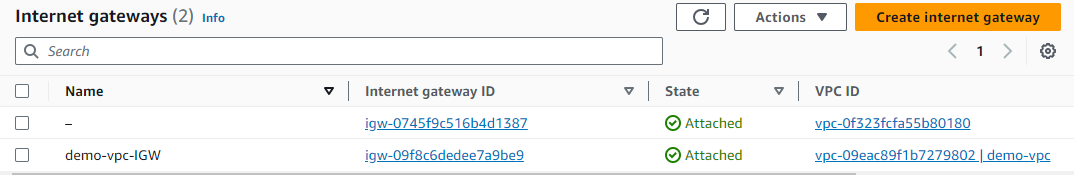
  vpc\_id = aws\_vpc.demo-vpc.id

  tags = {

    Name = "demo-vpc-IGW"

    }

}



SUBNETS:

* A subnet is a range of IP addresses in your VPC.
* You launch AWS resources, such as Amazon EC2 instances, into your subnets.
* Subnets are regional resources.
* Each subnet defines a range of IPv4 addresses.
* Traffic to and from instances can be controlled with network firewall rules.

**STEP-7 Create subnets by using below code**

**CODE:**

       resource "aws\_subnet" "private-subnet-1" {

      vpc\_id     = aws\_vpc.demo-vpc.id

      cidr\_block = "10.0.1.0/24"

      availability\_zone = "eu-west-1a"

      tags = {

      Name = "private-subnet-1"

}

       }

      resource "aws\_subnet" "private-subnet-2" {

      vpc\_id     = aws\_vpc.demo-vpc.id

      cidr\_block = "10.0.2.0/24"

      availability\_zone = "eu-west-1b"

      tags = {

      Name = "private-subnet-2"

         }

      }

       resource "aws\_subnet" "public-subnet-1" {

      vpc\_id     = aws\_vpc.demo-vpc.id

      cidr\_block = "10.0.3.0/24"

       availability\_zone = "eu-west-1a"

      tags = {

       Name = "public-subnet-1"

             }

        }

        resource "aws\_subnet" "public-subnet-2" {

       vpc\_id     = aws\_vpc.demo-vpc.id

       cidr\_block = "10.0.4.0/24"

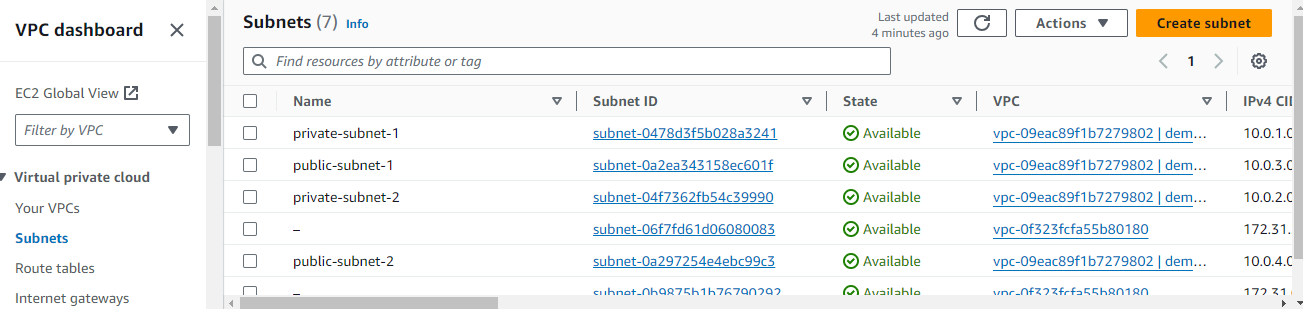
       availability\_zone = "eu-west-1b"

       tags = {

       Name = "public-subnet-2"

            }

        }

****

**ROUTE TABLE**

In AWS, a route table is a set of rules that determines where neytwork traffic is directed. Each subnet in your aws virtual private cloud is associated with a route table traffis flow between subnets. The route tables includes details like the route table ID and the ID of its associated vpc.

**STEP-8 Create Route Tables by using below code**

**CODE**

   resource "aws\_route\_table" "public-route-table" {

   vpc\_id = aws\_vpc.demo-vpc.id

   tags = {

    Name = "public-route-table"

           }

      }

    resource "aws\_route" "public-route" {

    route\_table\_id         = aws\_route\_table.public-route-table.id

    destination\_cidr\_block = "0.0.0.0/0"

    gateway\_id             = aws\_internet\_gateway.demo-igw.id

             }

     resource "aws\_route\_table\_association" "public-subnet-1-association" {

     subnet\_id      = aws\_subnet.public-subnet-1.id

     route\_table\_id = aws\_route\_table.public-route-table.id

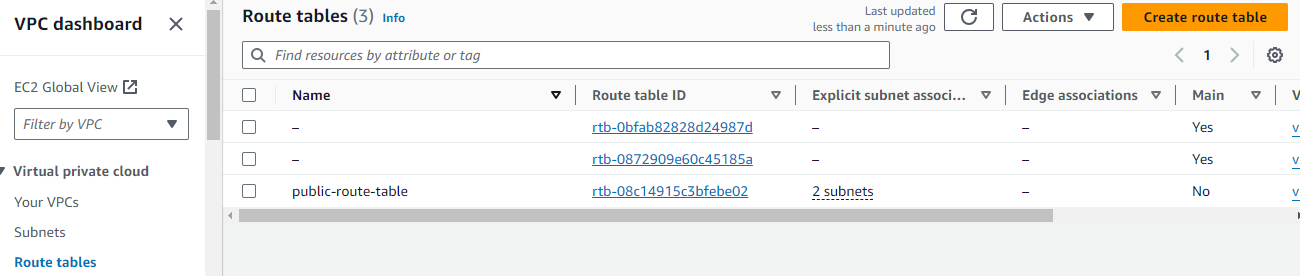
           }

      resource "aws\_route\_table\_association" "public-subnet-2-association" {

      subnet\_id      = aws\_subnet.public-subnet-2.id

     route\_table\_id = aws\_route\_table.public-route-table.id

            }

****

**NAT GATEWAY**

AWS NAT Gateway – stands for Network Address Translation. It is a managed AWS service that is scaled based on your usage. NAT Gateway will help you to access the internet which instances are configured in the private subnet but without proper routing, no one can access that instance from outside

**STEP-9 Create NAT GATEWAY by using below code**

**CODE:**

 resource "aws\_nat\_gateway" "nat-gateway" {

  allocation\_id = aws\_eip.nat-eip.id

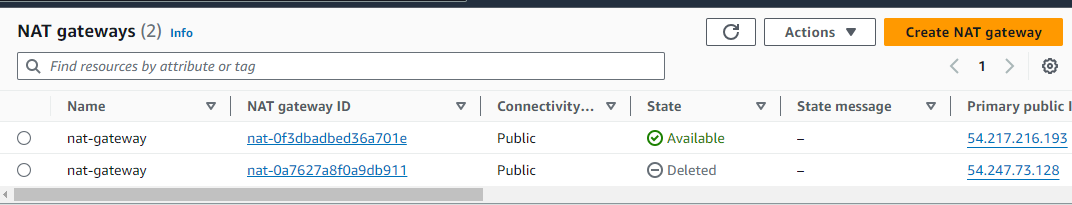
  subnet\_id     = aws\_subnet.public-subnet-1.id

  tags = {

   Name = "nat-gateway"

          }

     }



**Security group**

Security group, which functions as a virtual firewall to regulate the inbound and outgoing traffic for AWS EC2 instances or other AWS resources in a VPC. We shall go over a security group’s definition and formation in this article.

**STEP-10 Create Security group by using below code**

**CODE:**

    resource "aws\_security\_group" "secgroup" {

    name = "secgroup"

    description = "awssecuritygroup"

    vpc\_id = aws\_vpc.demo-vpc.id

       ingress {

        from\_port = 0

        to\_port = 65535

        protocol = "tcp"

        cidr\_blocks = ["0.0.0.0/0"]

       }

       egress {

          from\_port =0

          to\_port = 65535

          protocol ="tcp"

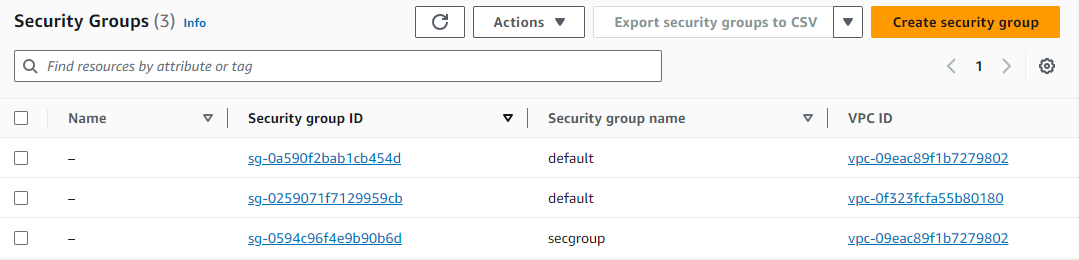
          cidr\_blocks = ["0.0.0.0/0"]

       }

    tags = {

        name ="secgroup"

    }

****

**PROCESS:**

Created vpc with region eu-west-1 with IP adddress 10.0.0.0/16 and named demo vpc.

Created a internet gateway for demo vpc.

Created 4 subnets; 2 private, 2 public –

* Private subnet-1, cidr 10.0.1.0/24, availability zone eu-west-1a
* Private subnet-2, cidr 10.0.2.0/24, availability zone eu-west-1b
* Public subnet-1, cidr  10.0.3.0/24, availability zone eu-west-1a
* Public subnet-1, cidr  10.0.4.0/24, availability zone eu-west-1b

Created route table for vpc named as public route table, connect to internet gateway through routes with cidr 0.0.0.0/0 and attaching public subnets to the route table.

Created NAT Gateway for the public subnet and named as nat gateway.

Created security group for vpc with inbound and outbound rules as port from port 0 to port 65535 and cidr blocks 0.0.0.0/0.

  At the end we have allowed internet access to public subnets using IGW to private subnets using NGW.

**CONCLUSION:**

VPC provides a secure and flexible way to deploy applications in the cloud.

**THANKYOU**