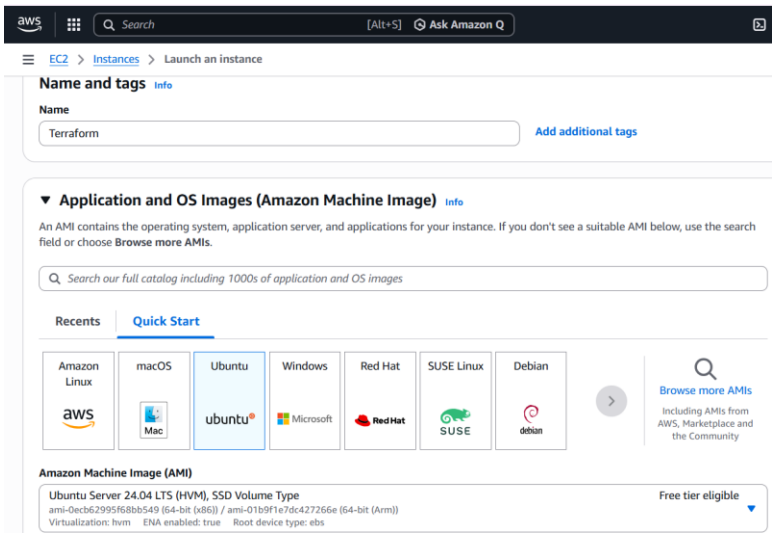
Terraform

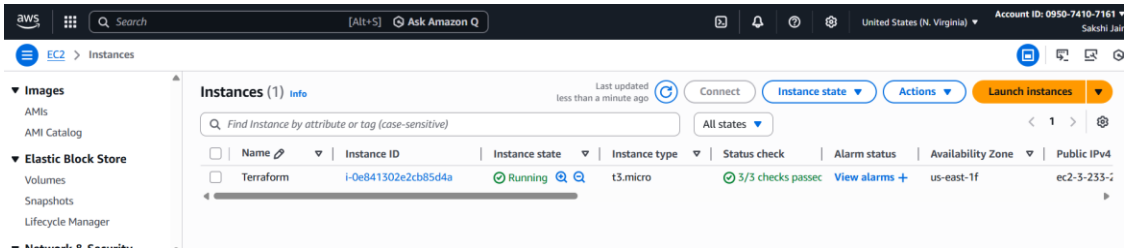
○ Terraform IAC, Information & Installation on AWS

Terraform is an **Infrastructure as Code (IaC) tool** created by **HashiCorp**.

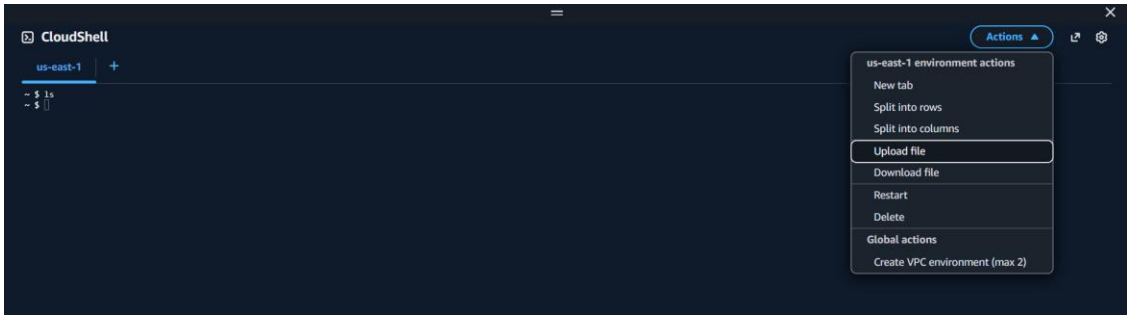
It allows you to **create, update, and delete cloud infrastructure using code**, instead of doing everything manually in the cloud console.

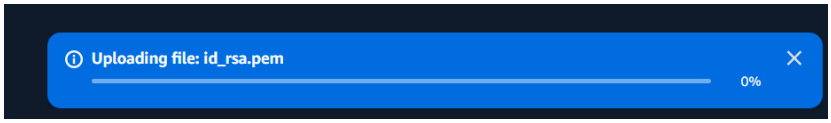
1. Create EC2 Instance



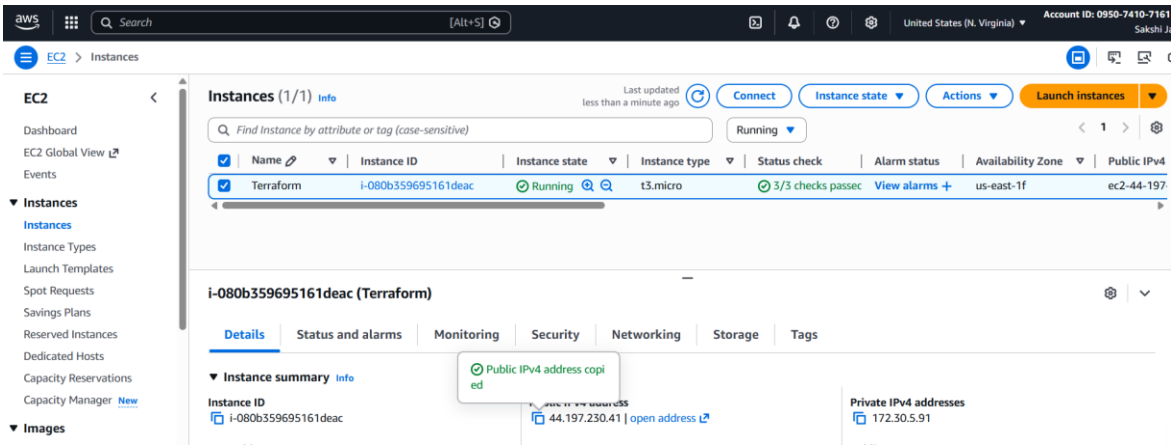


2. Open Cloud shell and upload your public key

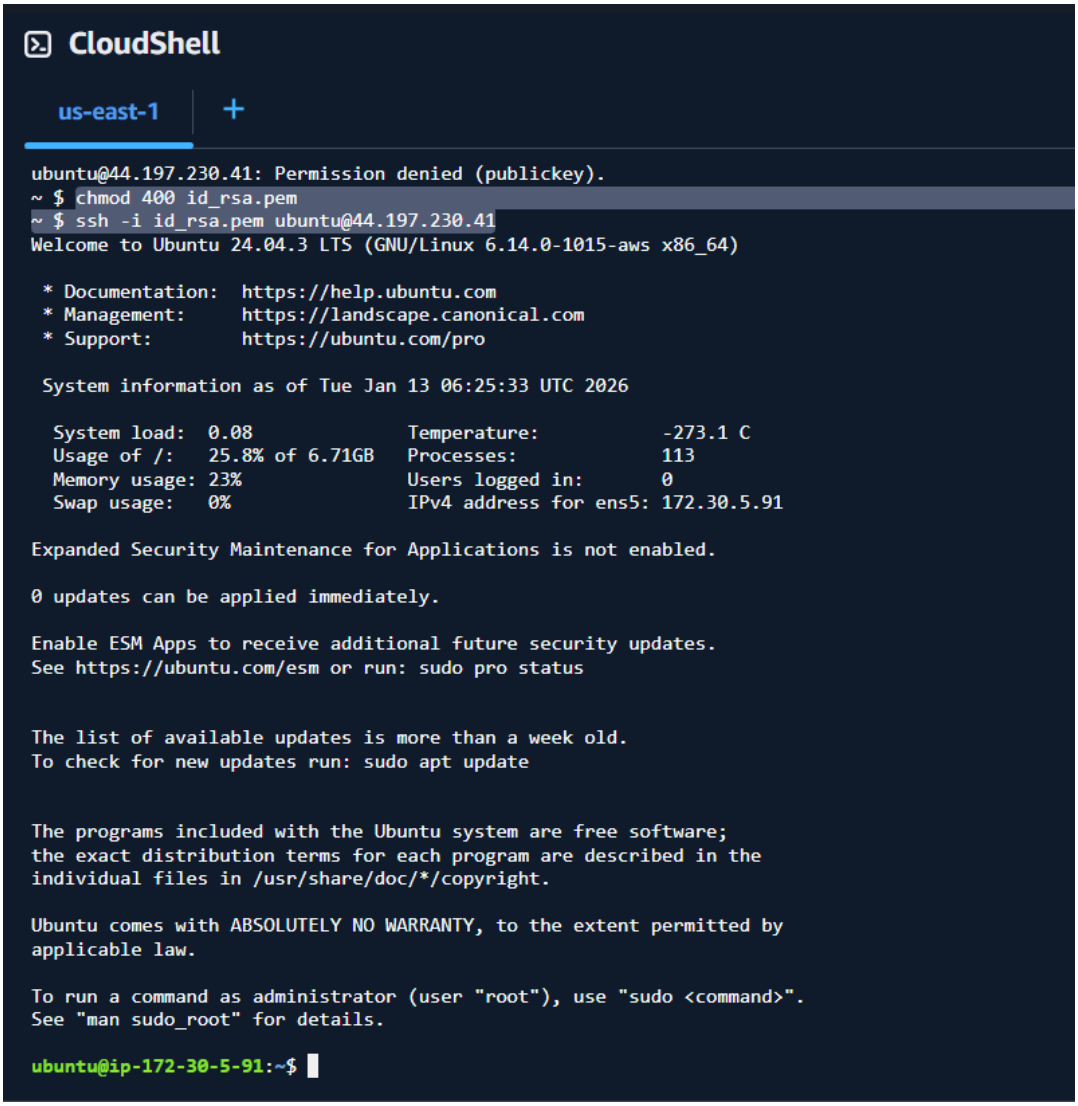




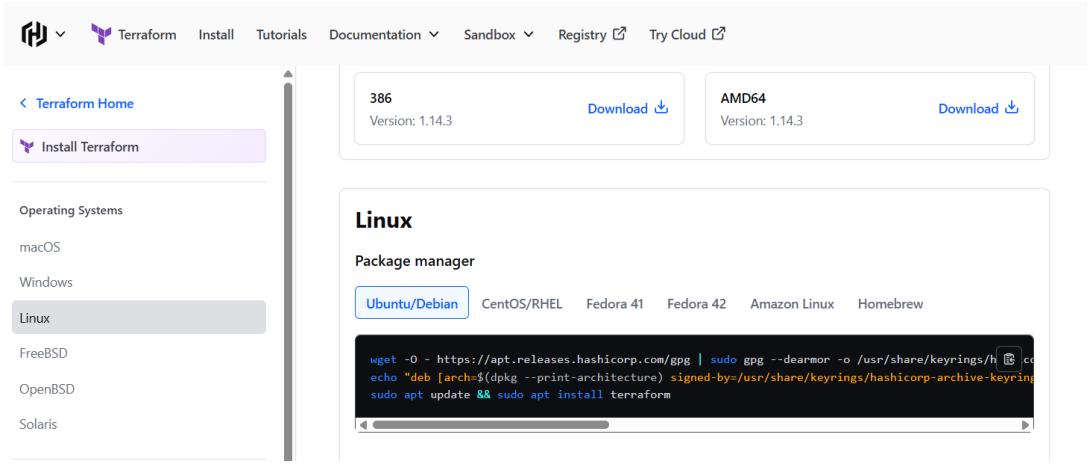
Copy Public IP of you Instance to get access on cloud shell

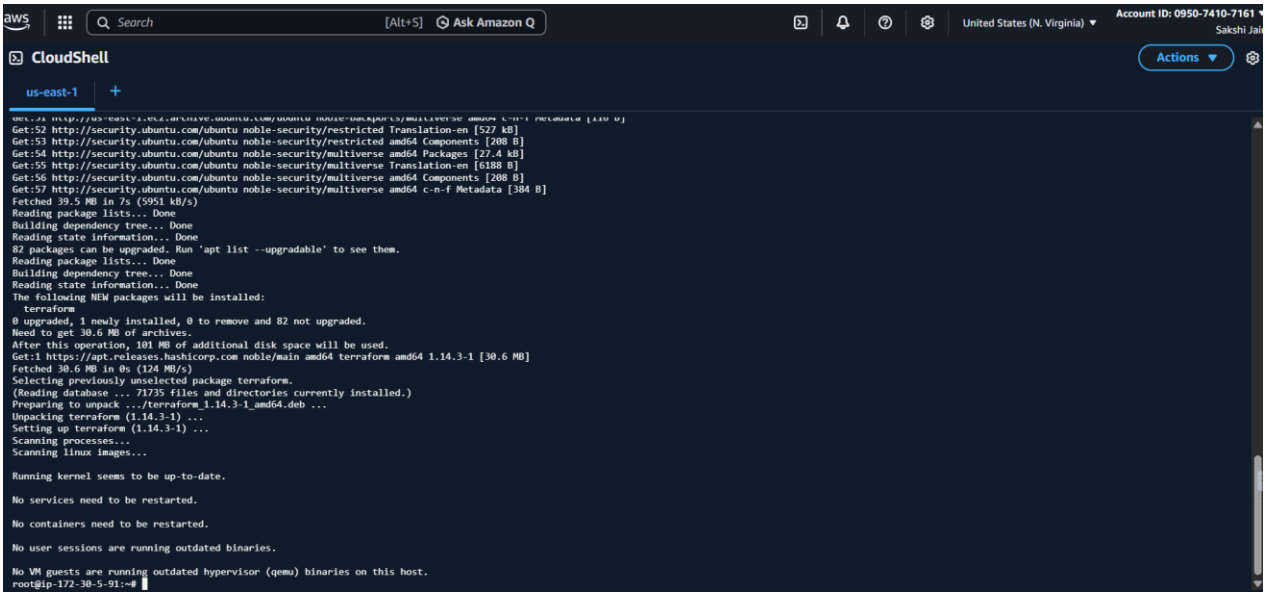


Take access of your ec2



3. Now Got to official site of terraform and copy Linux/ubuntu command to install Terraform





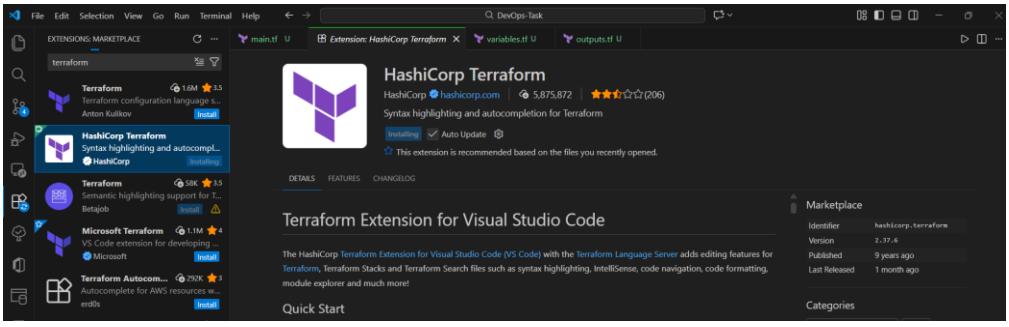
- **write configuration file, terraform life-cycle.**

  Required Visual Studio

  Use this Registry for reference

  Terraform AWS Registry

  Install Terraform Extension in VS code



Basic Terraform File Structure

terraform-project/

```
│
├── main.tf        → main configuration
├── variables.tf   → input values
├── outputs.tf     → output values
```

main.tf (Core File)

This file contains:

1. Provider
2. Resources

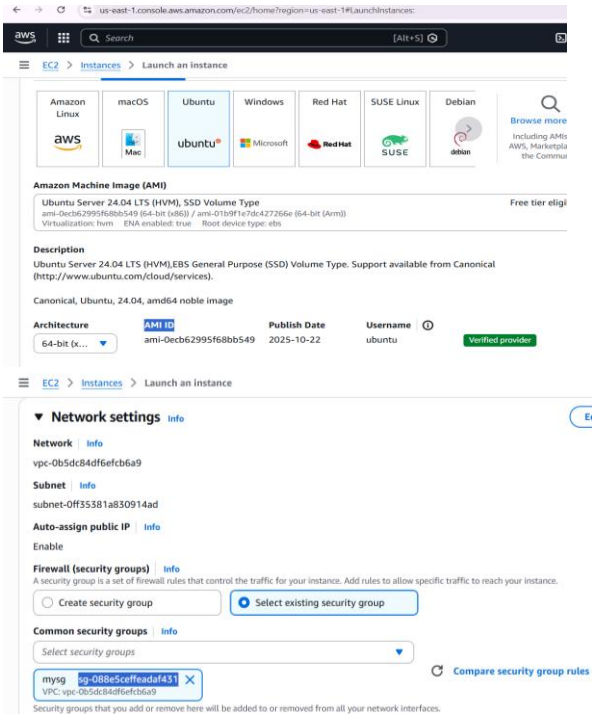
Example: main.tf

```
provider "aws" {
  region = "ap-south-1"
}
```

👉 **Meaning:**

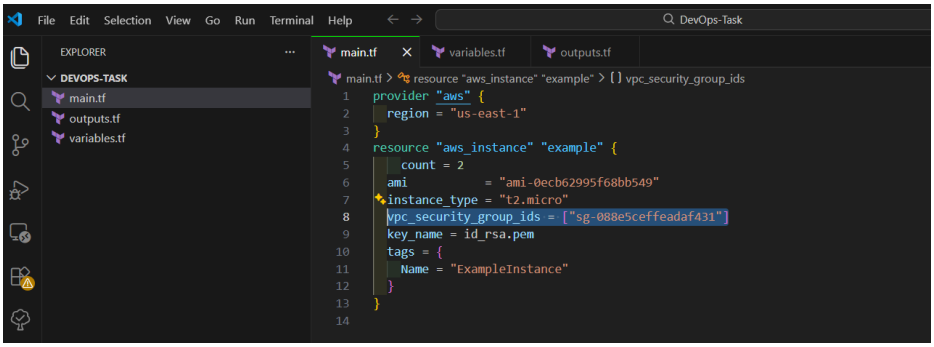
- Use **AWS**
- Region: **Mumbai**

Add a Resource (EC2 Example)



you have to use this ID's in code for Automatically creation

```
resource "aws_instance" "ExampleInstance" {
  ami           = "ami-0ecb62995f68bb549"
  instance_type = "t3.micro"

  subnet_id              = "subnet-0ff35381a830914ad"
  vpc_security_group_ids = ["sg-088e5ceffeadaf431"]

  tags = {
    Name = "ExampleInstance"
  }
```

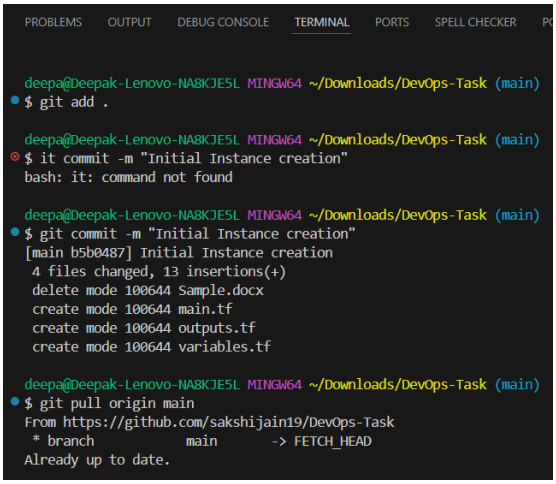



```
}
```

👉 **Meaning:**

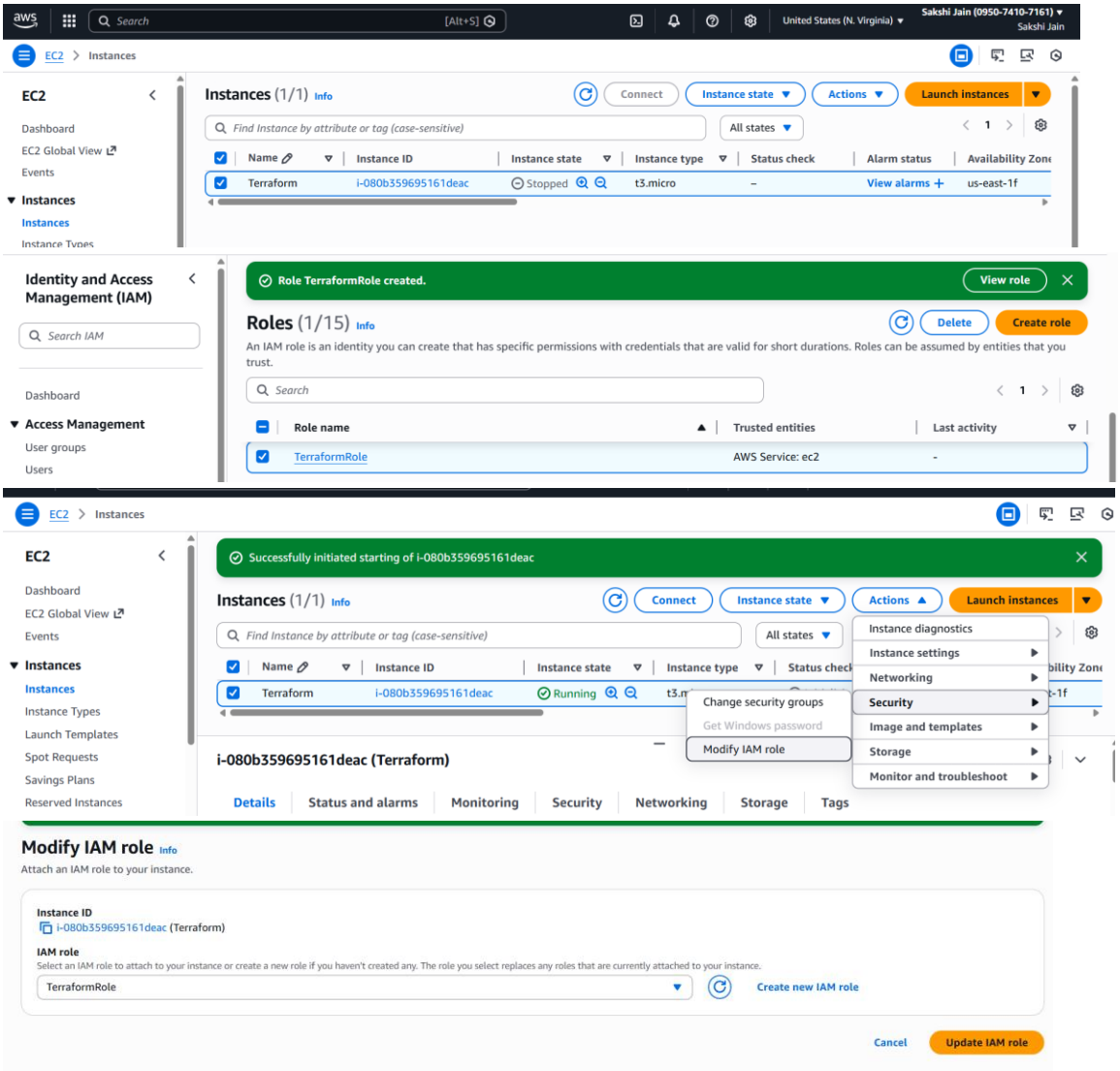
- resource → what to create
- aws_instance → EC2
- my_ec2 → logical name (Terraform uses it)
- ami, instance_type → EC2 settings

Push this files to your Main GitHub Repository



```
deepa@Deepak-Lenovo-NA8KJE5L MINGW64 ~/Downloads/DevOps-Task (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 529 bytes | 264.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/sakshijain19/DevOps-Task.git
   9edf679..b5b0487  main -> main
```

You Need to communicate to AWS for that required one Instance which help you to communicate AWS for that I will create one IAM Role and attach it to this Instance

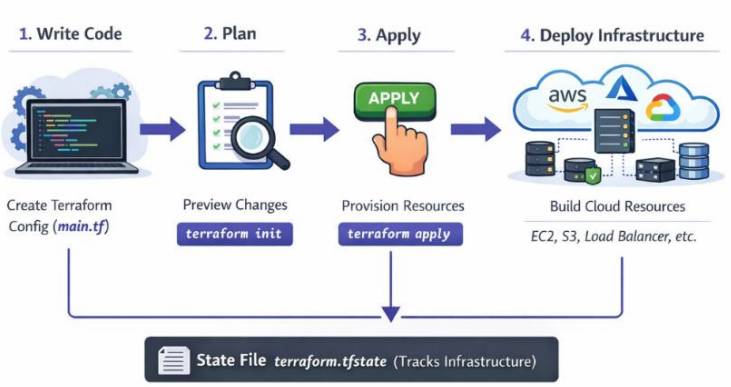


Copy Public IP of Instance and through SSH take remote access of Instance

Clone your Repository in your Instance

```
ubuntu@ip-172-30-5-91:~$ sudo -i
root@ip-172-30-5-91:~# git clone https://github.com/sakshijain19/DevOps-Task.git
Cloning into 'DevOps-Task'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 0), reused 4 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (7/7), 11.83 KiB | 3.94 MiB/s, done.
root@ip-172-30-5-91:~# ls
DevOps-Task  snap
root@ip-172-30-5-91:~# cd DevOps-Task/
root@ip-172-30-5-91:~/DevOps-Task# ls
main.tf  outputs.tf  variables.tf
root@ip-172-30-5-91:~/DevOps-Task#
```

Terraform Workflow/ Lifecycle



What is terraform init? (Very Simple)

terraform init is the **first command** you run in any Terraform project.

Why Do We Need terraform init?

Terraform cannot work until it knows:

- Which **cloud provider** you are using
- Where to store the **state file**
- Which **plugins** are required

terraform init does all this setup automatically

```
root@ip-172-30-5-91:~/DevOps-Task# terraform --version
Terraform v1.14.3
on linux_amd64
root@ip-172-30-5-91:~/DevOps-Task# terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.28.0...
- Installed hashicorp/aws v6.28.0 (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-172-30-5-91:~/DevOps-Task# ls -a
.  ..  .git  .terraform  .terraform.lock.hcl  main.tf  outputs.tf  variables.tf
root@ip-172-30-5-91:~/DevOps-Task#
```

You will get all details about provider

```
root@ip-172-30-5-91:~/DevOps-Task# ls -a
.  ..  .git  .terraform  .terraform.lock.hcl  main.tf  outputs.tf  variables.tf
root@ip-172-30-5-91:~/DevOps-Task# cd .terraform/
root@ip-172-30-5-91:~/DevOps-Task/.terraform# ls
providers
root@ip-172-30-5-91:~/DevOps-Task/.terraform# cd providers/
root@ip-172-30-5-91:~/DevOps-Task/.terraform/providers# ls
registry.terraform.io
root@ip-172-30-5-91:~/DevOps-Task/.terraform/providers# cd registry.terraform.io/
root@ip-172-30-5-91:~/DevOps-Task/.terraform/providers/registry.terraform.io# ls
hashicorp
root@ip-172-30-5-91:~/DevOps-Task/.terraform/providers/registry.terraform.io# cd hashicorp/
root@ip-172-30-5-91:~/DevOps-Task/.terraform/providers/registry.terraform.io/hashicorp# ls
aws
root@ip-172-30-5-91:~/DevOps-Task/.terraform/providers/registry.terraform.io/hashicorp# cd aws/
root@ip-172-30-5-91:~/DevOps-Task/.terraform/providers/registry.terraform.io/hashicorp/aws# ls
6.28.0
root@ip-172-30-5-91:~/DevOps-Task/.terraform/providers/registry.terraform.io/hashicorp/aws#
```

What is .terraform.lock.hcl?

.terraform.lock.hcl is a **lock file** that tells Terraform **which exact provider versions to use**.

Why is .terraform.lock.hcl Important?

Without this file:

- Different team members may use **different provider versions**

- Terraform behavior may change
- Builds may fail unexpectedly

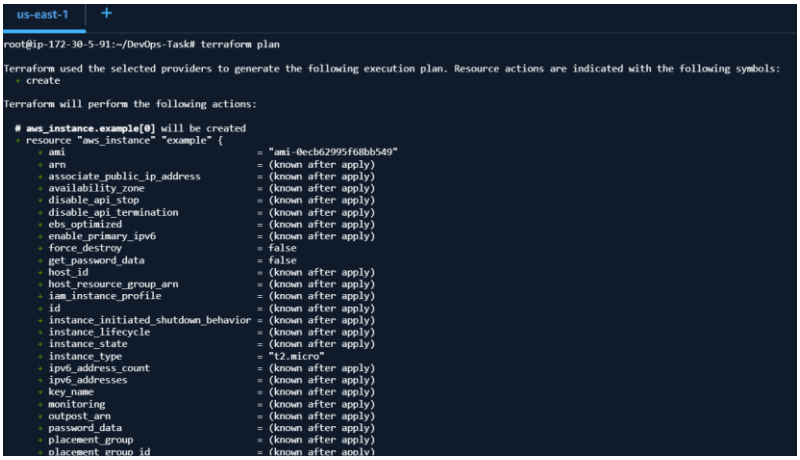
What is terraform plan?

terraform plan shows you **what Terraform is going to do** before it actually makes any changes.

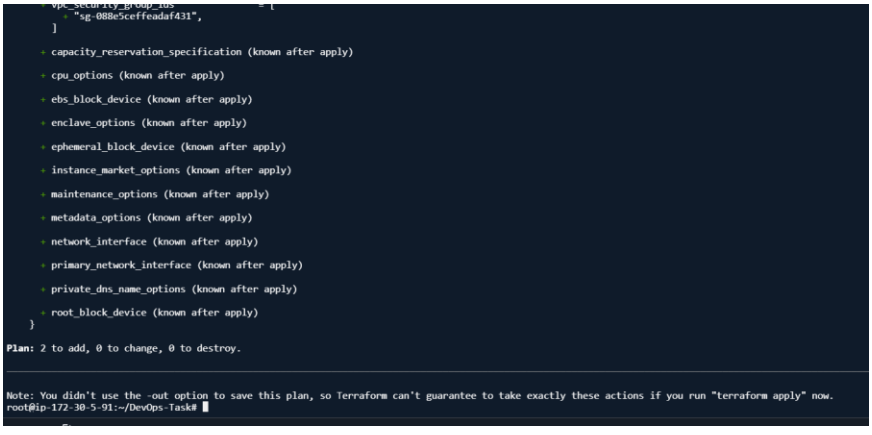
Why Do We Use terraform plan?

It helps you:

- Avoid mistakes
- Understand changes clearly
- Review before applying
- Detect configuration drift



Here your IAM Role is work as credential

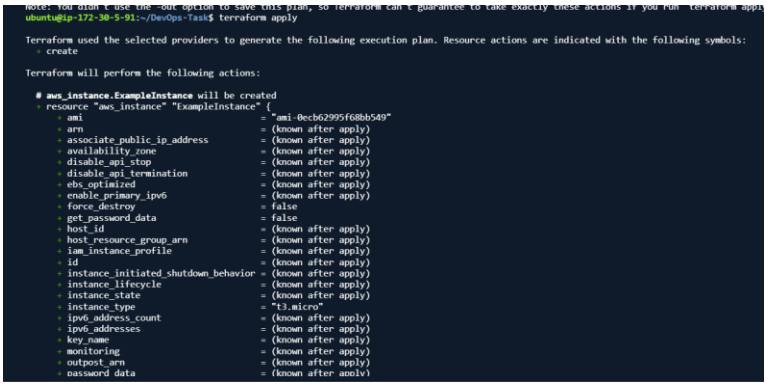


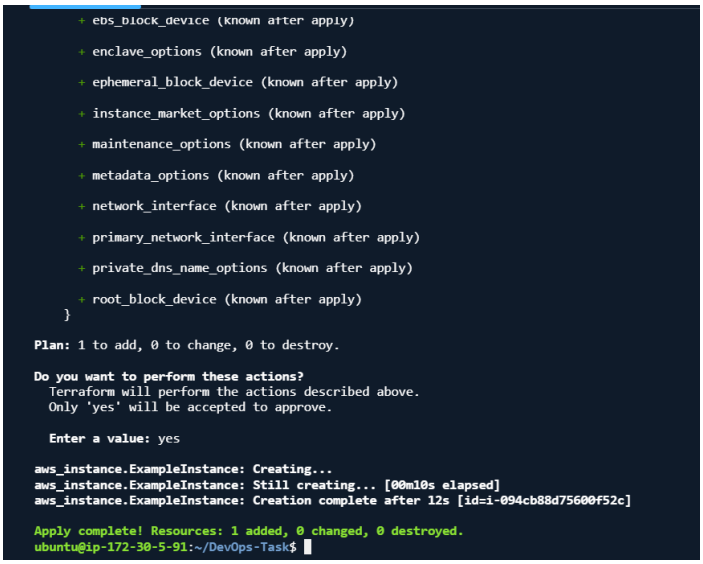
terraform apply —

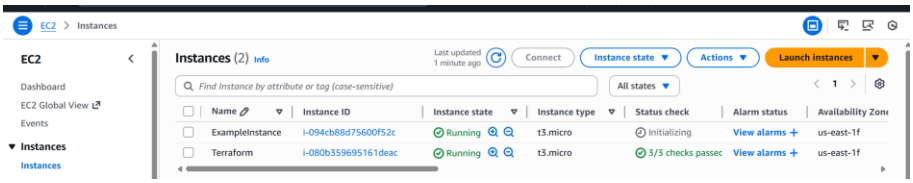
terraform apply is the command that **actually creates or updates your infrastructure**.

1. Reads your .tf files
2. Checks the **state file**
3. Compares with **real cloud resources**
4. Shows a plan (what will change)
5. Asks for confirmation (yes)
6. **Creates / updates / deletes** resources

👉 This is when **real AWS resources are created** (EC2, S3, etc.).







What is terraform.tfstate? (Simple Explanation)

terraform.tfstate is a **state file** that Terraform uses to **remember what infrastructure it has created**.

```
ubuntu@ip-172-30-5-91:~/DevOps-Task$ ls -a
.  ..  .git  .terraform  .terraform.lock.hcl  main.tf  outputs.tf  terraform.tfstate  terraform.tfstate.backup  variables.tf
```

Why Terraform Needs terraform.tfstate

Terraform must know:

- What resources already exist
- Their IDs (EC2 ID, S3 bucket name, IPs)
- Current configuration

terraform destroy —

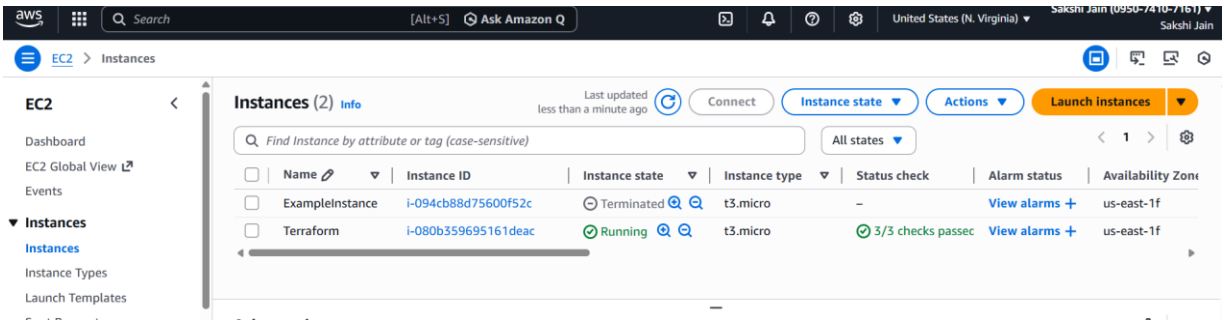
terraform destroy is the command used to **delete everything created by Terraform**.

1. Reads the **state file**
2. Finds all resources it created
3. Shows what will be deleted
4. Asks for confirmation (yes)
5. **Deletes all resources**

👉 Used to avoid **unnecessary cloud billing**

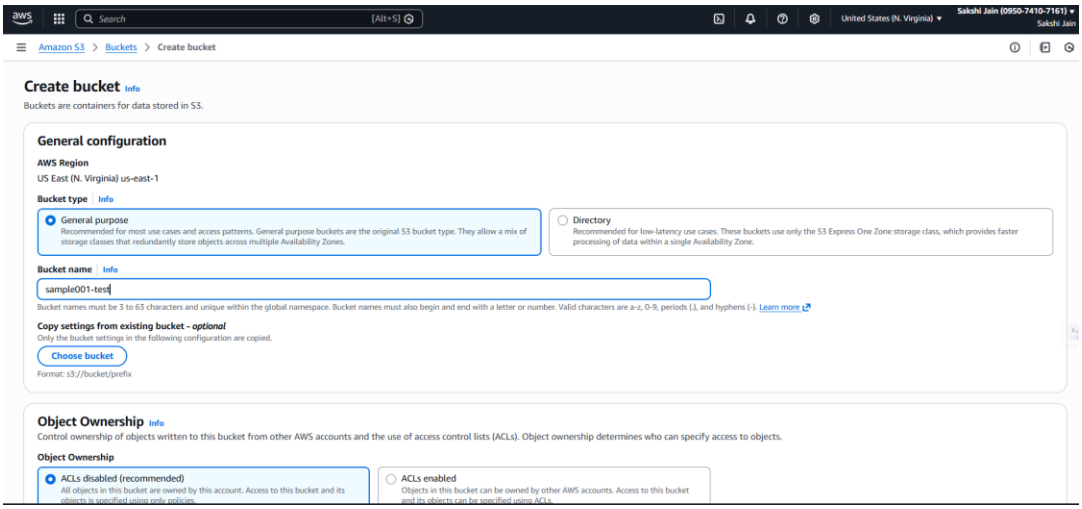
```
ubuntu@ip-172-30-5-91:~/DevOps-Task$ terraform destroy
aws_instance.ExampleInstance: Refreshing state... [id=i-094cb88d75600f52c]

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

Terraform will perform the following actions:

  # aws_instance.ExampleInstance will be destroyed
  - resource "aws_instance" "ExampleInstance" {
      - ami                                  = "ami-0ecb62995f68bb549" -> null
      - arn                                  = "arn:aws:ec2:us-east-1:095074107161:instance/i-094cb88d75600f52c" -> null
      - associate_public_ip_address          = true -> null
      - availability_zone                    = "us-east-1f" -> null
      - disable_api_stop                     = false -> null
      - disable_api_termination              = false -> null
      - ebs_optimized                        = false -> null
      - force_destroy                        = false -> null
      - get_password_data                    = false -> null
      - hibernation                          = false -> null
      - id                                   = "i-094cb88d75600f52c" -> null
      - instance_initiated_shutdown_behavior = "stop" -> null
      - instance_state                       = "running" -> null
      - instance_type                        = "t3.micro" -> null
      - ipv6_address_count                   = 0 -> null
      - ipv6_addresses                       = [] -> null
      - monitoring                           = false -> null
      - placement_partition_number           = 0 -> null
      - primary_network_interface_id         = "eni-0b6cc9167a8a19614" -> null
      - private_dns                          = "ip-172-30-5-44.ec2.internal" -> null
      - private_ip                           = "172.30.5.44" -> null
      - public_dns                           = "ec2-13-220-190-124.compute-1.amazonaws.com" -> null
      - public_ip                            = "13.220.190.124" -> null
```

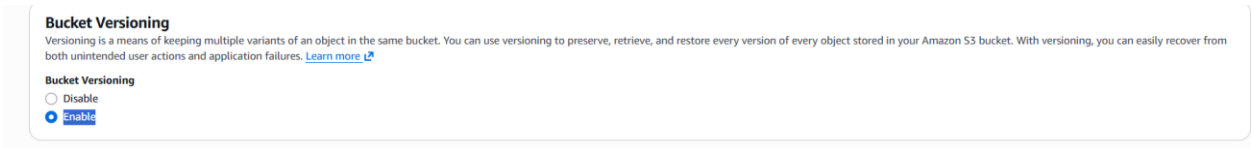
```
us-east-1  +

          - nostname_type                      = "ip-name" -> null
        }

      - root_block_device {
          - delete_on_termination = true -> null
          - device_name           = "/dev/sda1" -> null
          - encrypted             = false -> null
          - iops                  = 3000 -> null
          - tags                  = {} -> null
          - tags_all              = {} -> null
          - throughput            = 125 -> null
          - volume_id             = "vol-06354e00b5497c45a" -> null
          - volume_size           = 8 -> null
          - volume_type           = "gp3" -> null
            # (1 unchanged attribute hidden)
        }
    }

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.ExampleInstance: Destroying... [id=i-094cb88d75600f52c]
aws_instance.ExampleInstance: Still destroying... [id=i-094cb88d75600f52c, 00m10s elapsed]
aws_instance.ExampleInstance: Still destroying... [id=i-094cb88d75600f52c, 00m20s elapsed]
aws_instance.ExampleInstance: Still destroying... [id=i-094cb88d75600f52c, 00m30s elapsed]
aws_instance.ExampleInstance: Still destroying... [id=i-094cb88d75600f52c, 00m40s elapsed]
aws_instance.ExampleInstance: Destruction complete after 50s

Destroy complete! Resources: 1 destroyed.
```

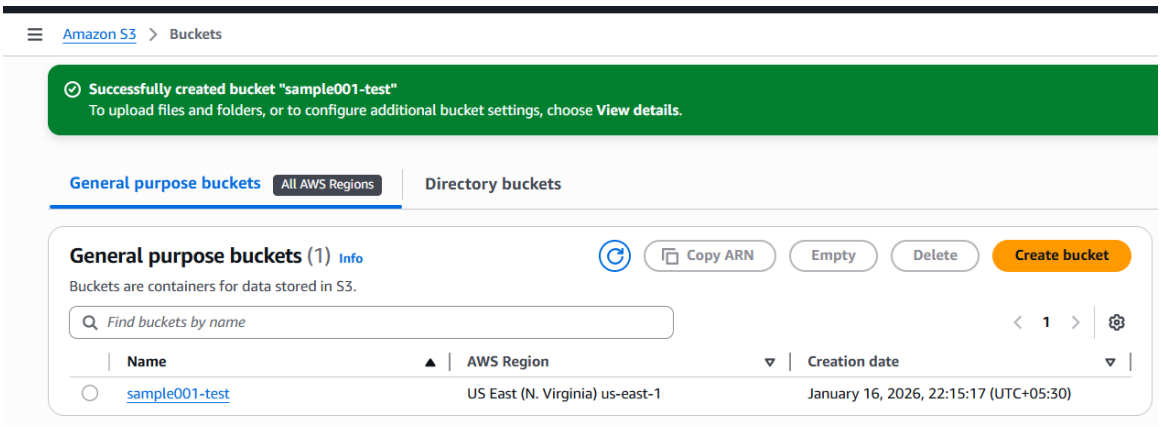


○ Terraform block, set backend, variables and output block

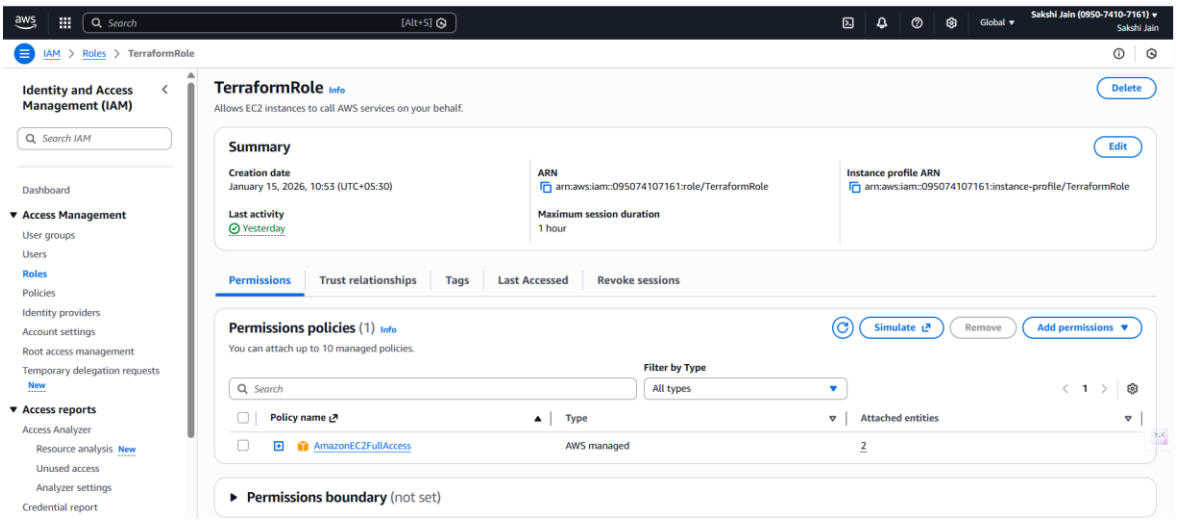
Create S3 Bucket

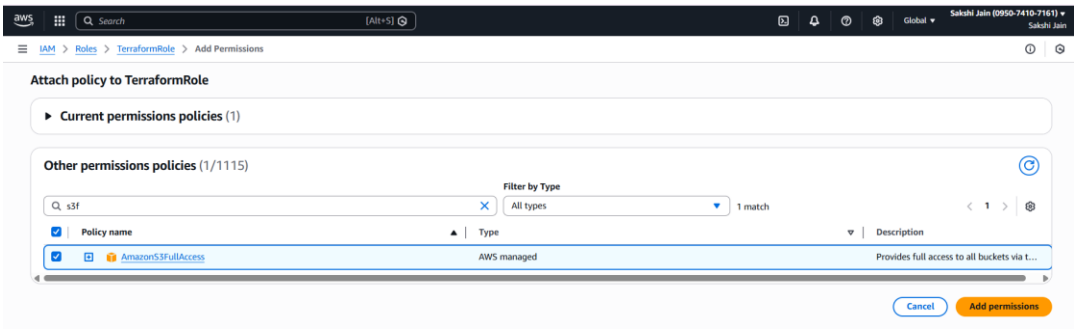


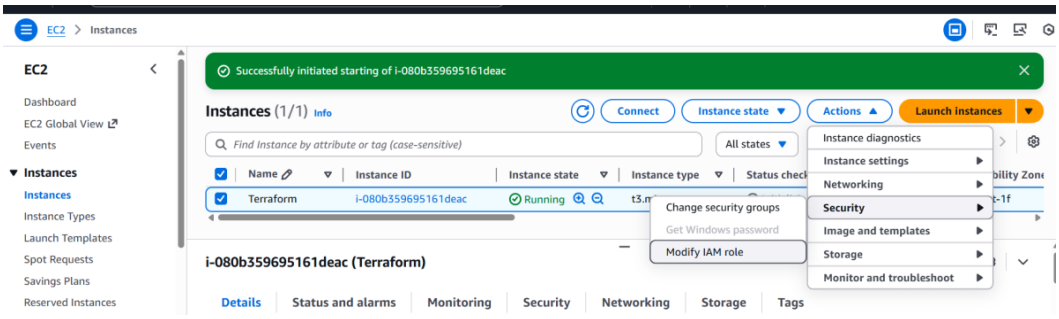


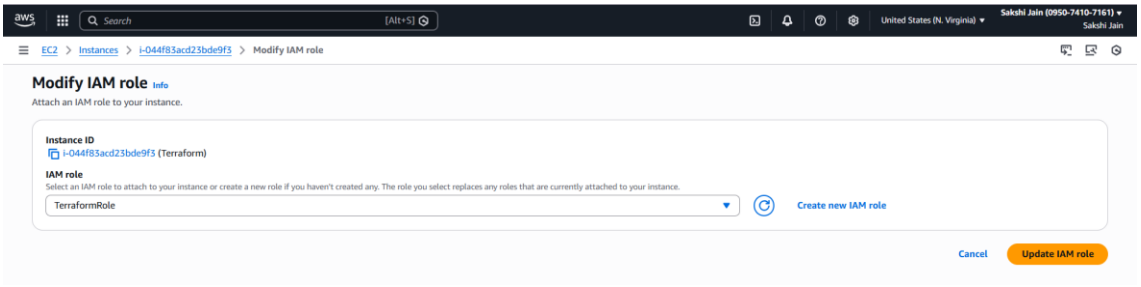


Edit role of your terraform Instance and add s3 permission to it and reattach to Instance

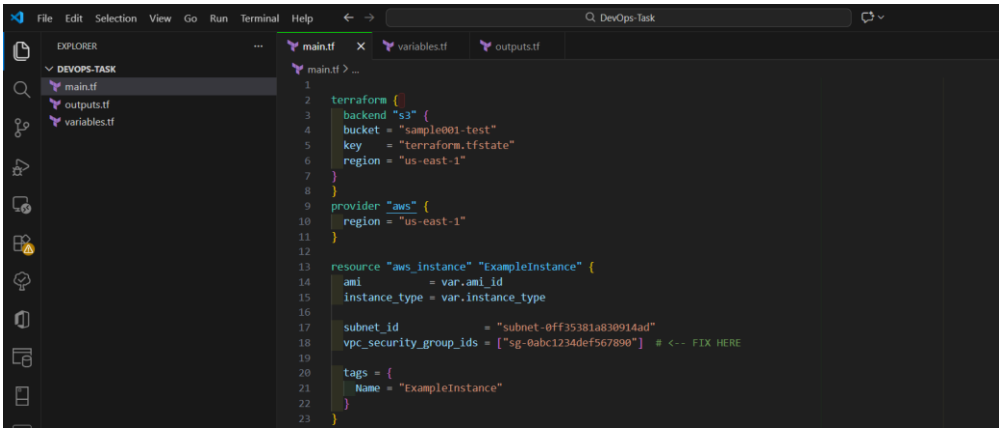


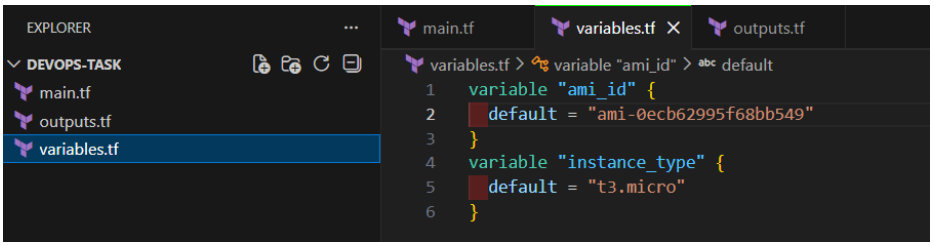


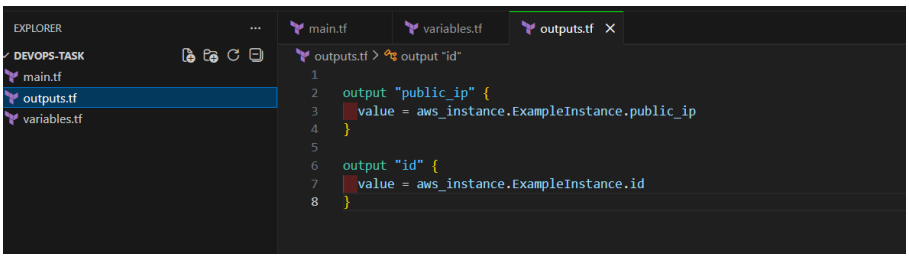




This Backend blog will store tfstate file in the s3



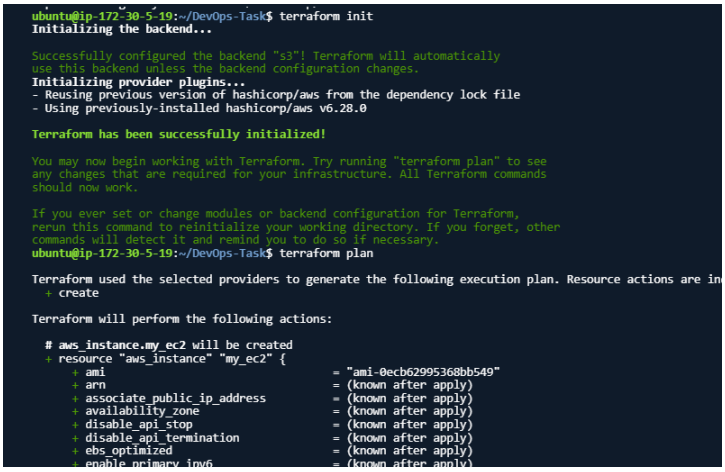


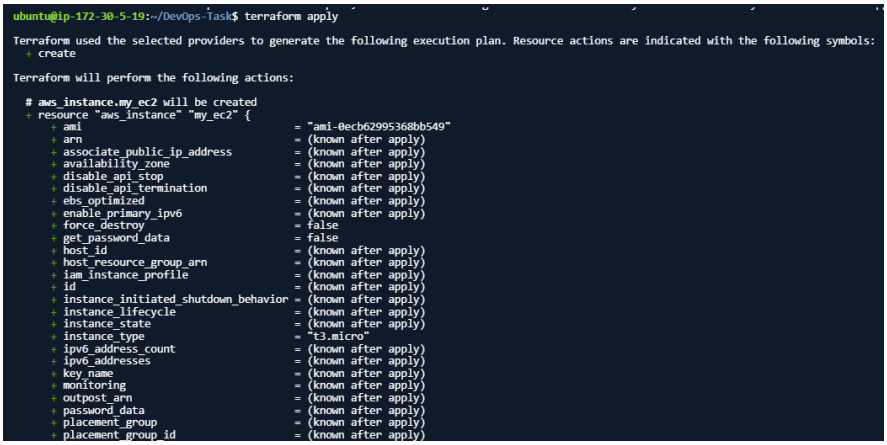


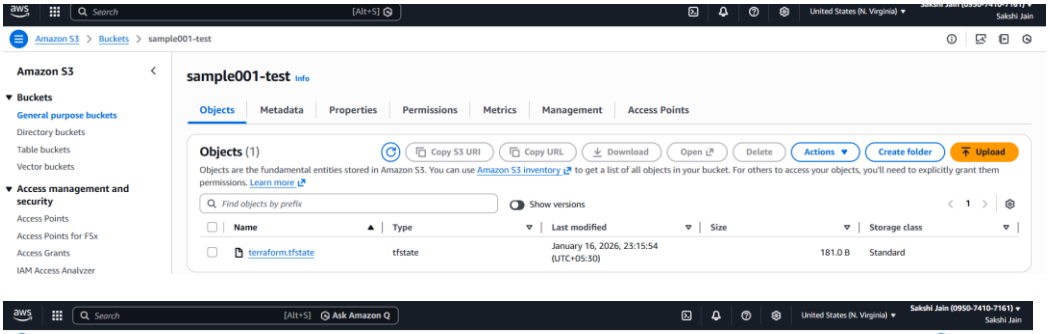

```
deepa@Deepak-Lenovo-NA8KJE5L MINGW64 ~/Downloads/DevOps-Task (main)
$ git add .
warning: in the working copy of 'main.tf', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'outputs.tf', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'variables.tf', LF will be replaced by CRLF the next time Git touches it

deepa@Deepak-Lenovo-NA8KJE5L MINGW64 ~/Downloads/DevOps-Task (main)
$ git commit -m "s3 backend added"
[main 8a88da2] s3 backend added
 3 files changed, 31 insertions(+), 6 deletions(-)

deepa@Deepak-Lenovo-NA8KJE5L MINGW64 ~/Downloads/DevOps-Task (main)
$ git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 751 bytes | 250.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/sakshijain19/DevOps-Task.git
   306046c..8a88da2  main -> main
```

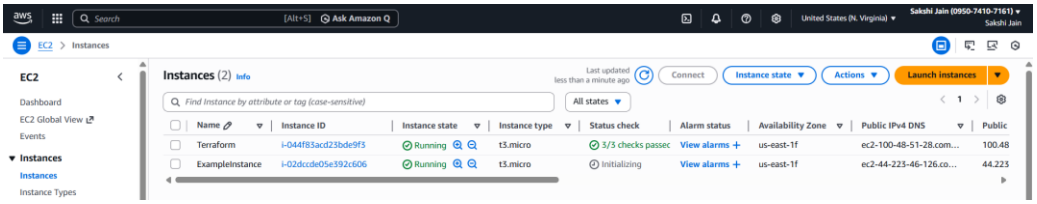
```
ubuntu@ip-172-30-5-19:~/DevOps-Task$ git pull origin
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 8 (delta 1), pack-reused 0 (from 0)
Unpacking objects: 100% (8/8), 1.04 KiB | 530.00 KiB/s, done.
From https://github.com/sakshijain19/DevOps-Task
   306046c..be1c54c  main       -> origin/main
Updating 306046c..be1c54c
Fast-forward
 main.tf      | 23 ++++++++++++++++++------
 outputs.tf   |  8 ++++++++
 variables.tf |  6 ++++++
 3 files changed, 31 insertions(+), 6 deletions(-)
```

Then run terraform init, plan, apply





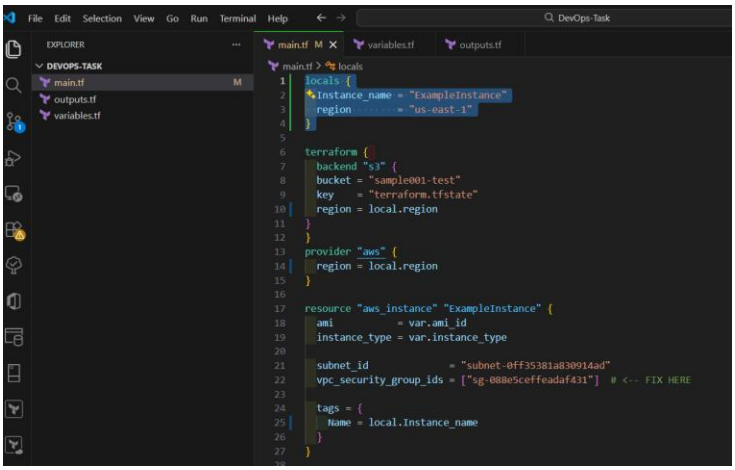




○ Local & data block, modules, and VPC create

What are locals?

`locals` are named values you define **once** and reuse many times in your Terraform code.

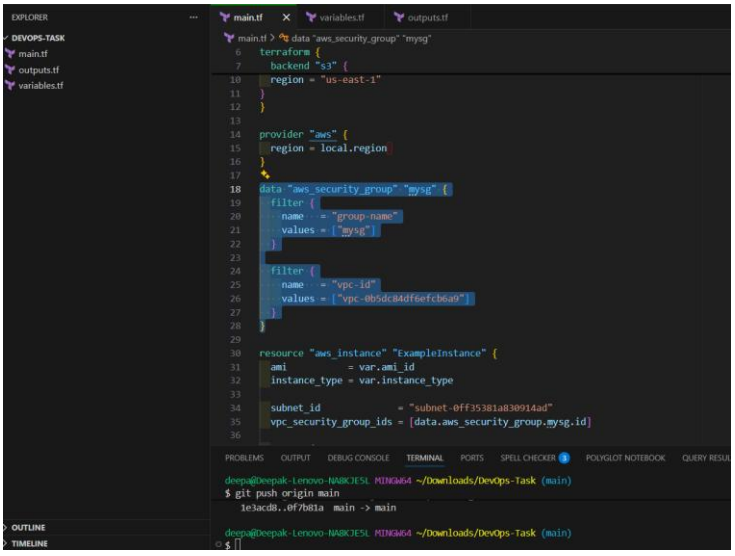


What is a data block?

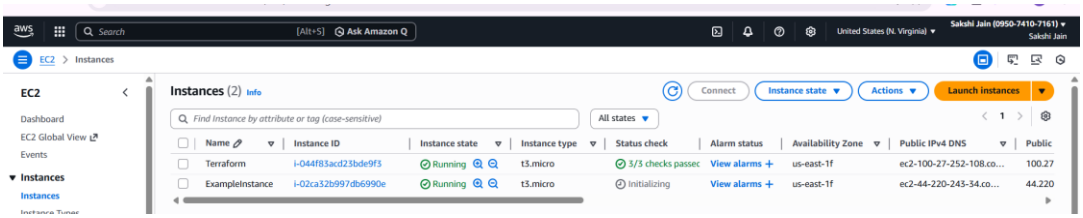
A **data block** is used to **read existing infrastructure** instead of creating new resources.

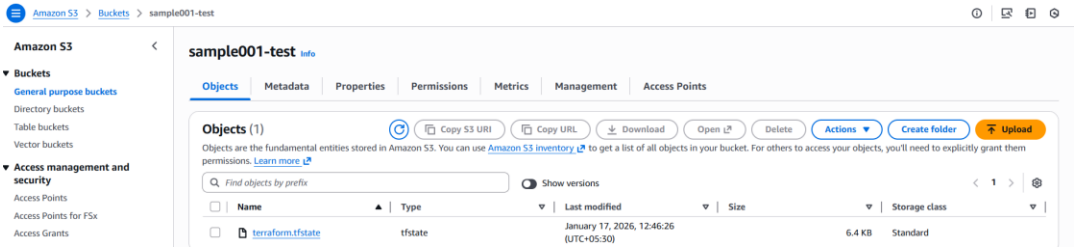
When to use data blocks?

- Use an existing VPC
- Use an existing AMI
- Use an existing Security Group



Using data block



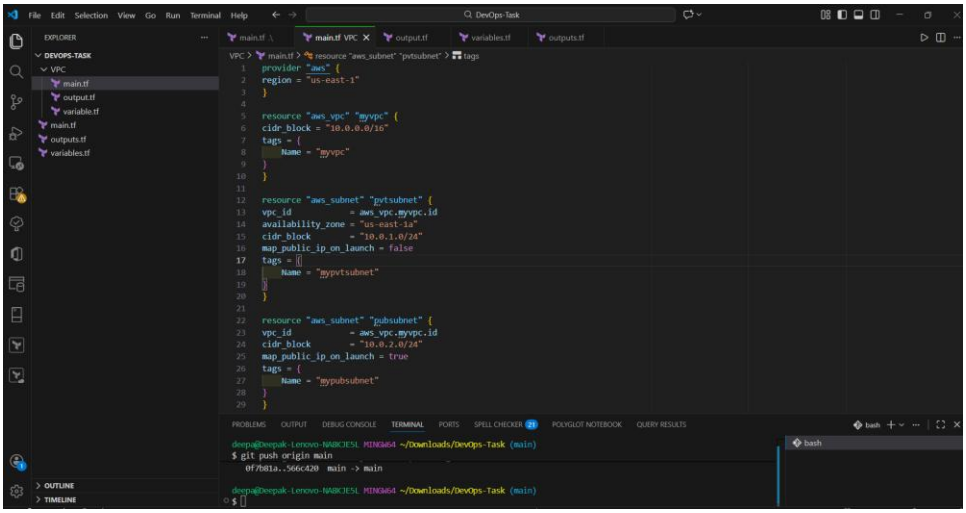


VPC Creation

```
provider "aws" {
region = "us-east-1"
}
resource "aws_vpc" "myvpc" {
cidr_block = "10.0.0.0/16"
tags = {
   Name = "myvpc"
}
}
resource "aws_subnet" "pvtsubnet" {
vpc_id            = aws_vpc.myvpc.id
availability_zone = "us-east-1a"
cidr_block        = "10.0.1.0/24"
map_public_ip_on_launch = false
tags = {
   Name = "mypvtsubnet"
}
}
resource "aws_subnet" "pubsubnet" {
vpc_id            = aws_vpc.myvpc.id
cidr_block        = "10.0.2.0/24"
```

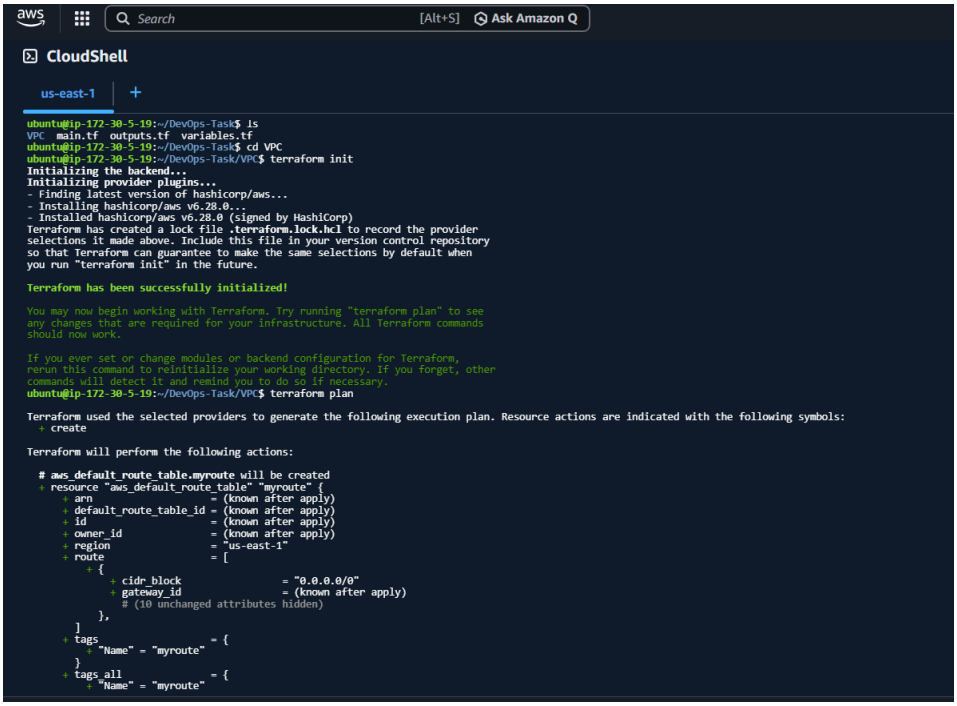
```
map_public_ip_on_launch = true
tags = {
   Name = "mypubsubnet"
}
}
resource "aws_internet_gateway" "myigw" {
vpc_id = aws_vpc.myvpc.id
tags = {
   Name = "myigw"
}
}
resource "aws_default_route_table" "myroute" {
default_route_table_id = aws_vpc.myvpc.default_route_table_id
route {
   cidr_block = "0.0.0.0/0"
   gateway_id = aws_internet_gateway.myigw.id
}
tags = {
   Name = "myroute"
}
}
resource "aws_route_table_association" "routeassoc" {
subnet_id      = aws_subnet.pubsubnet.id
route_table_id = aws_default_route_table.myroute.id
}
```

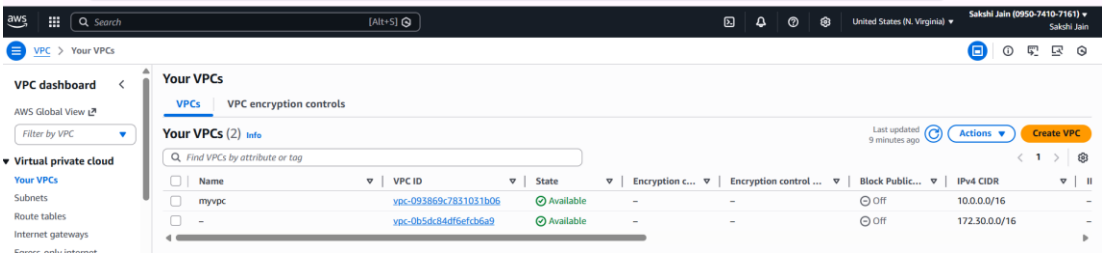
Push it into main GitHub Repository

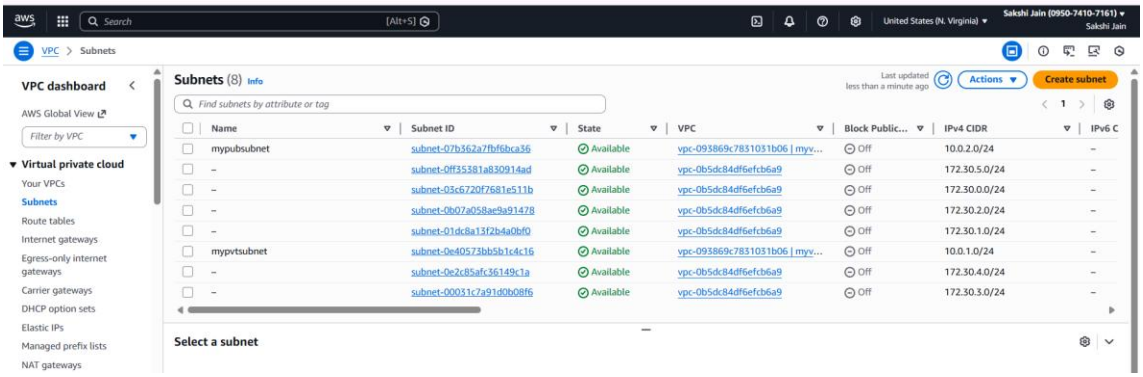


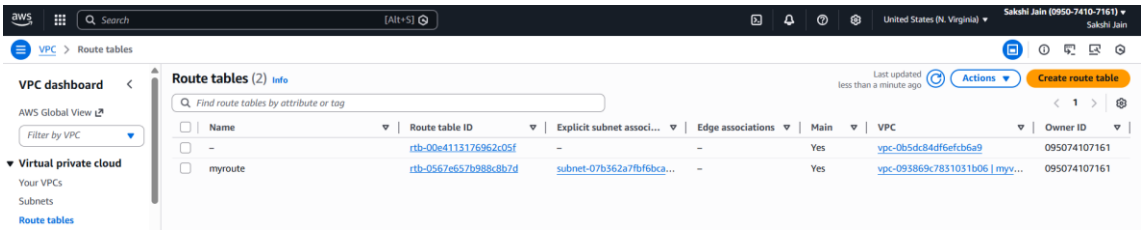
Then run **git pull origin main**

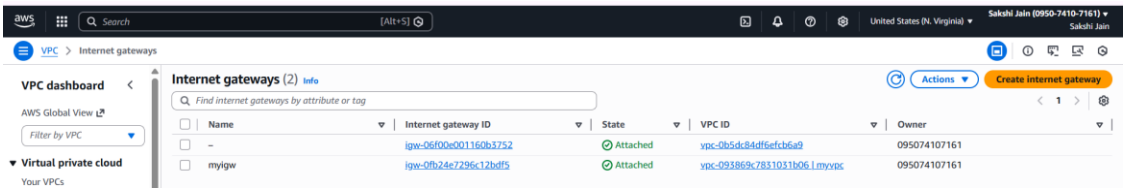
Then run terraform init, plan, apply











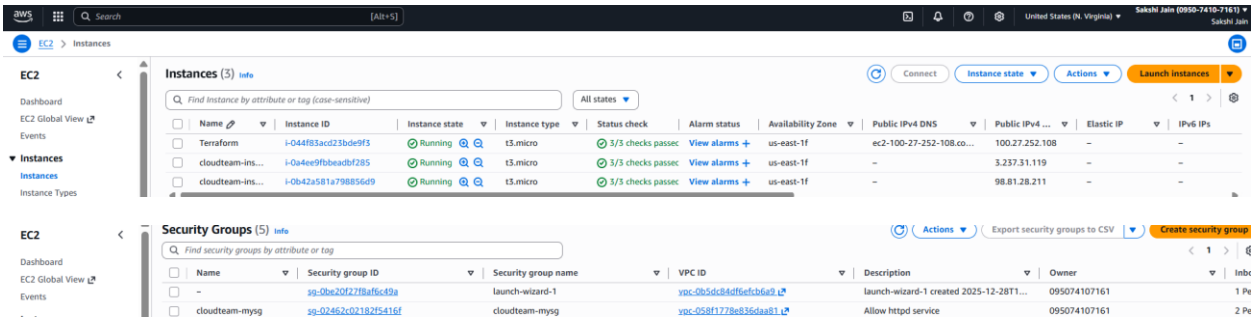
EC2 Instance Creation-inside-VPC

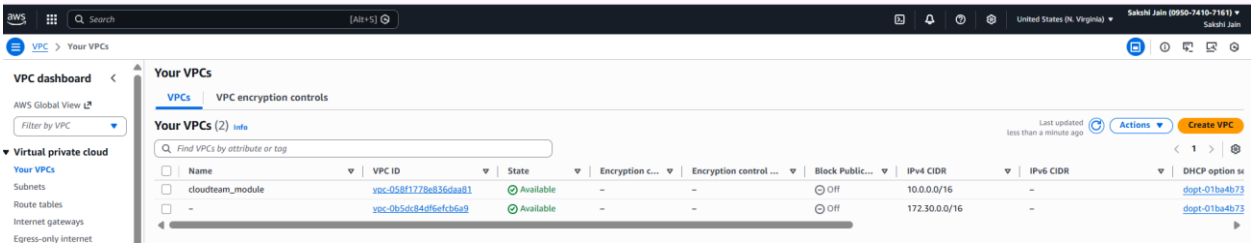
Find code Here

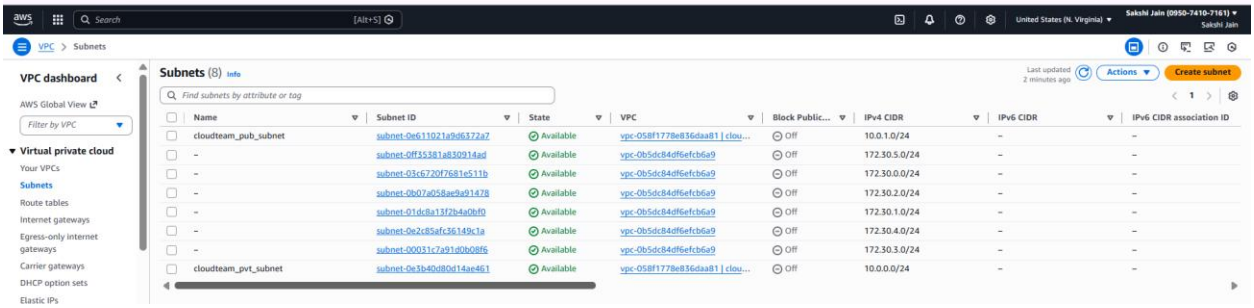
EC2-inside-VPC

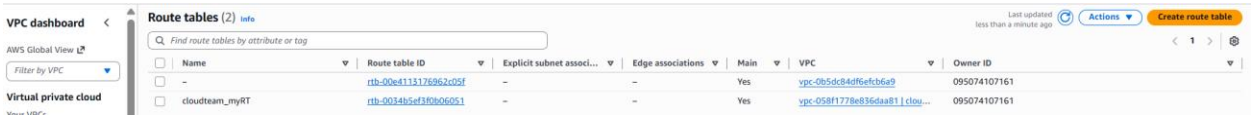
Commands to Run on Cloud shell

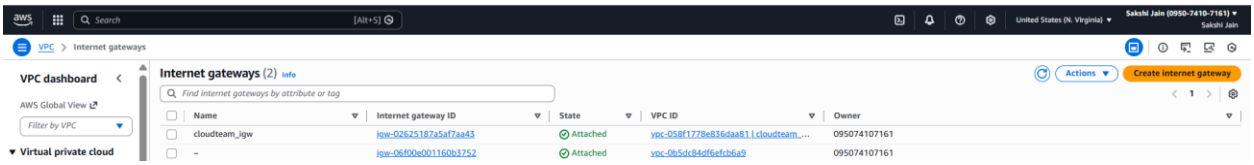
Git pull origin main
terraform init
terraform plan
terraform apply




VPC dashboard
Your VPCs
VPCs
VPC encryption controls
Your VPCs (2) Info
Actions
Create VPC
cloudteam_module
vpc-058f1778e836daa81
10.0.0.0/16
vpc-0b5dc84df6efcb6a9
172.30.0.0/16


VPC dashboard
Subnets (8) Info
Actions
Create subnet
cloudteam_pub_subnet
cloudteam_pvt_subnet


VPC dashboard
Route tables (2) Info
Actions
Create route table
cloudteam_myRT


VPC dashboard
Internet gateways (2) Info
Actions
Create internet gateway
cloudteam_igw