**Terraform-05**

**1.Watch the Terraform-05 video.**

**2.Execute the script shown in the video.**

Terraform Remote state and state locking:

=========================================

We can store terraform configuration files and state file in github or any other repository but it is not good practise.

We use s3,terraform storage,hashicorp consul to store the state file.

State locking is used to lock the state file so that no two users can execute the state file at the same point of time.

Remote backend and state locking:

================================

We can use s3 as remote backend and dynamo db for state locking.

Create s3 using terraform:

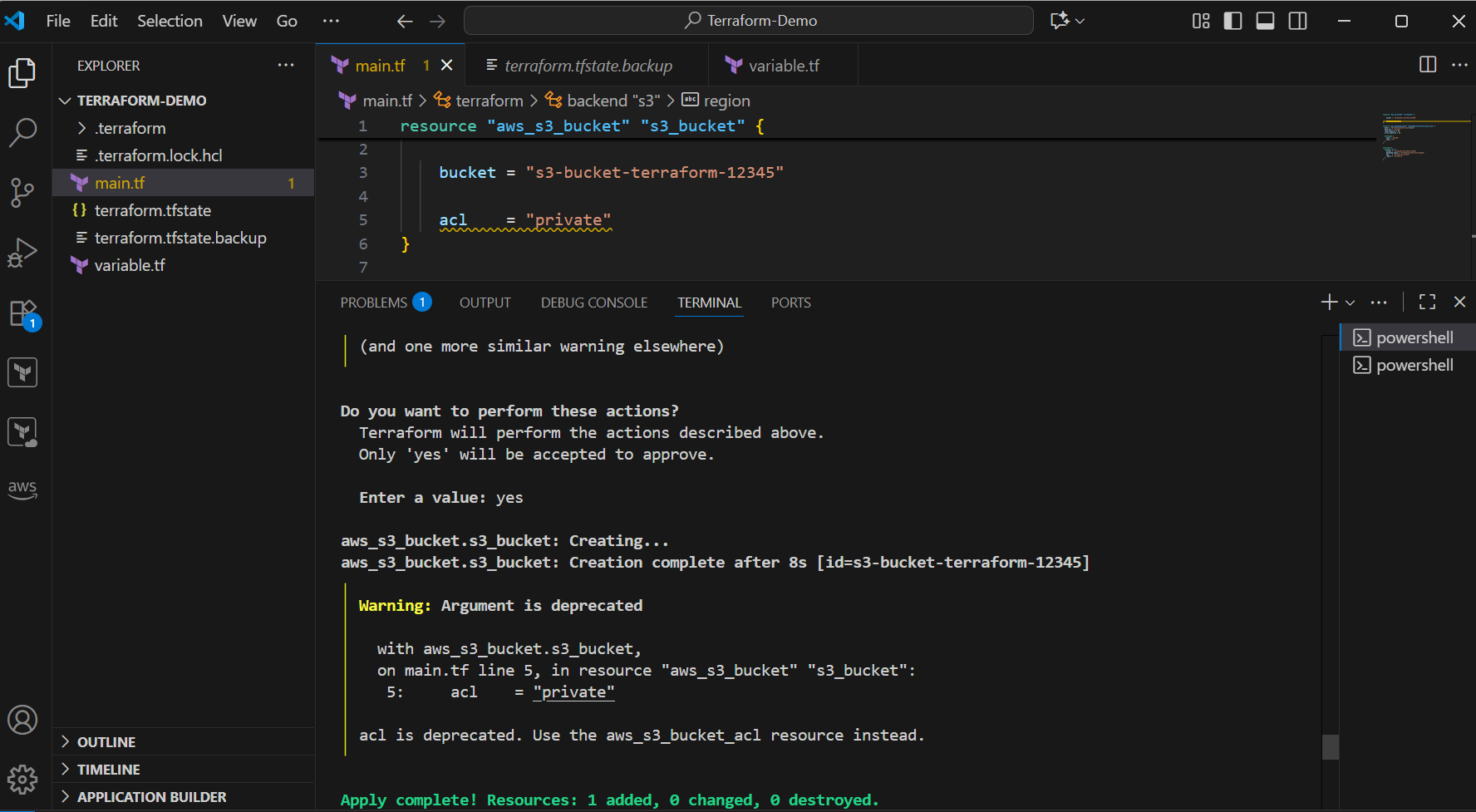
=========================

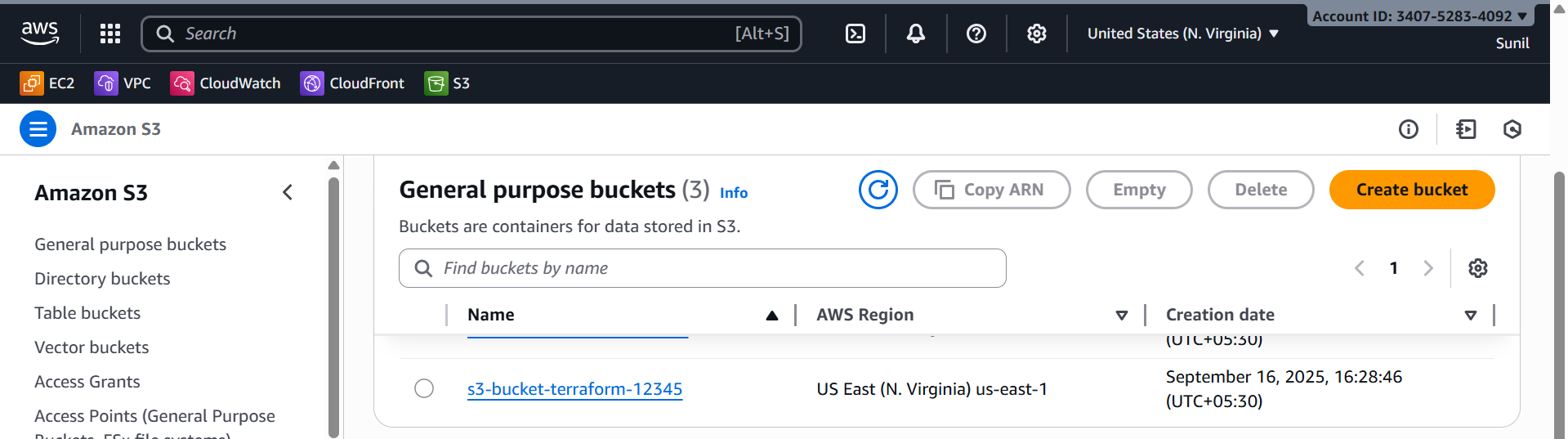
resource "aws\_s3\_bucket" "s3\_bucket" {

bucket = "s3-bucket-terraform-12345"

acl = "private"

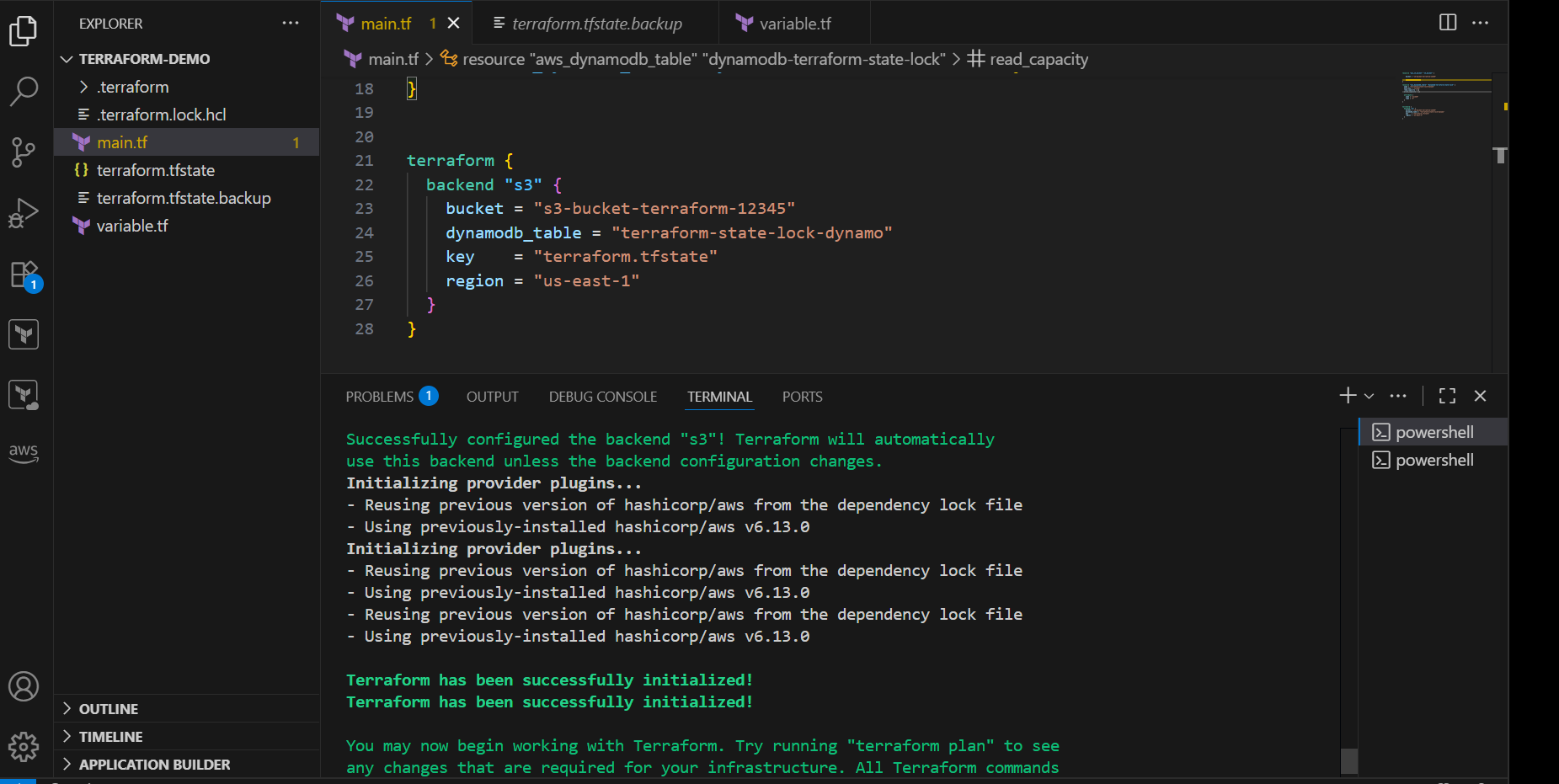
}

****

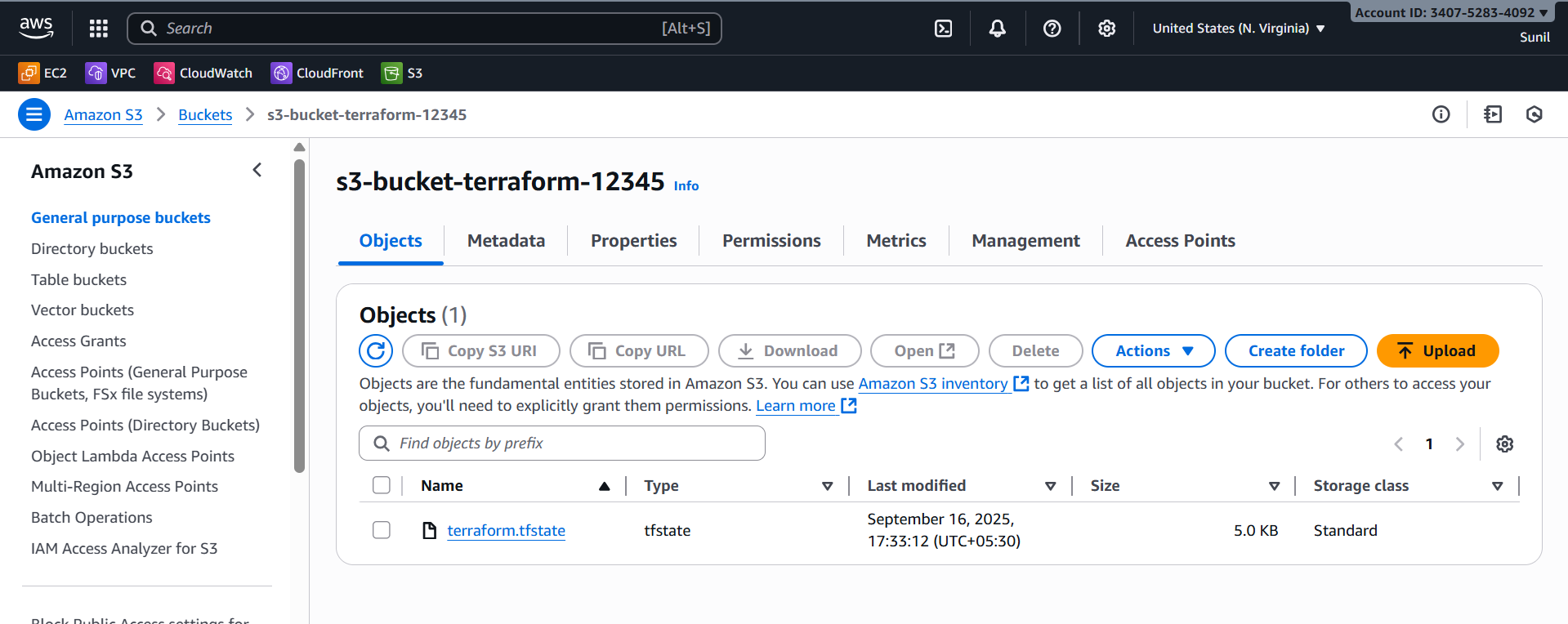
****

Create dynamo db using terraform:

================================

****

**Tfstate file in s3**

****

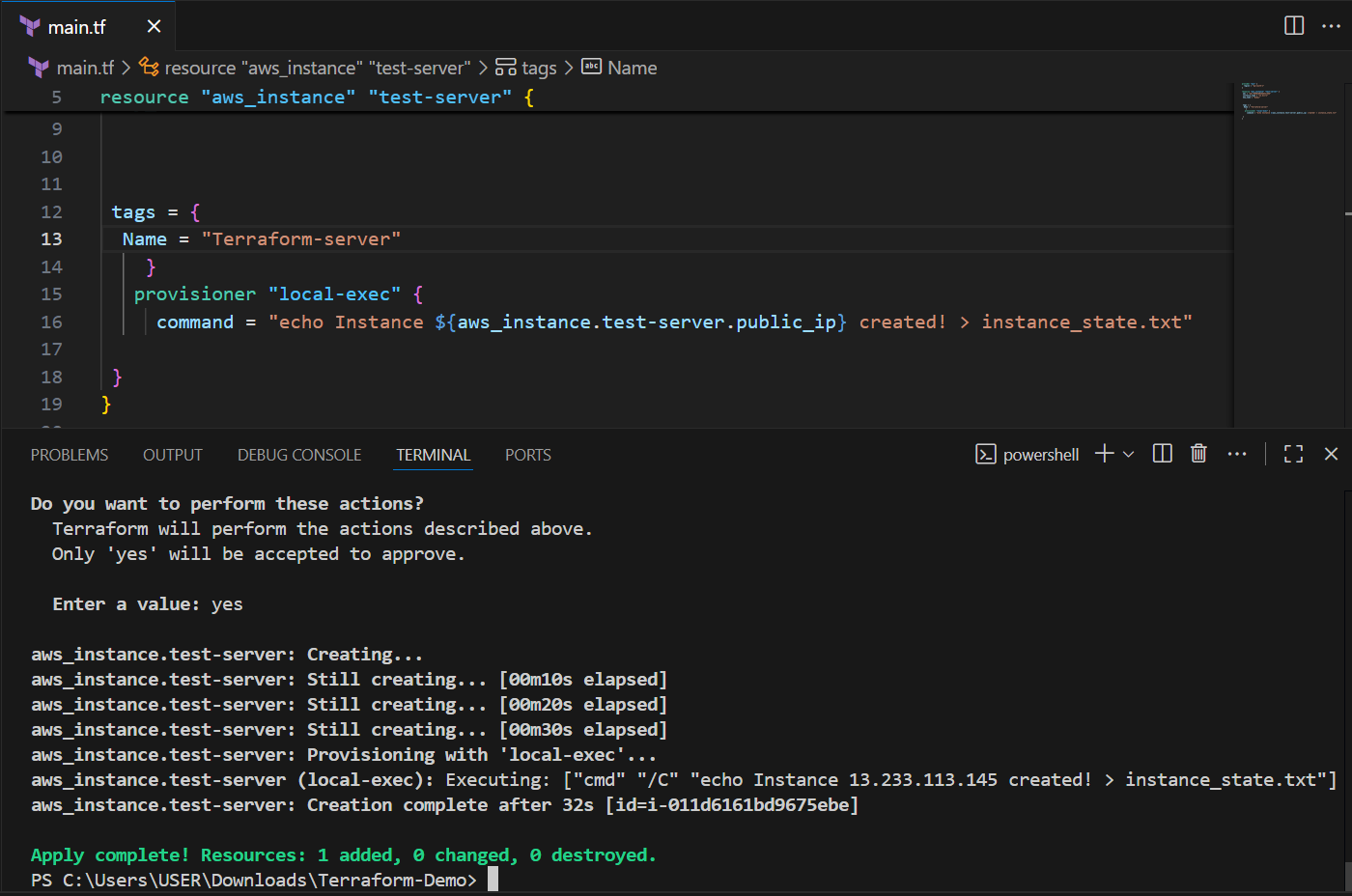
Terraform Provisioners:

=======================

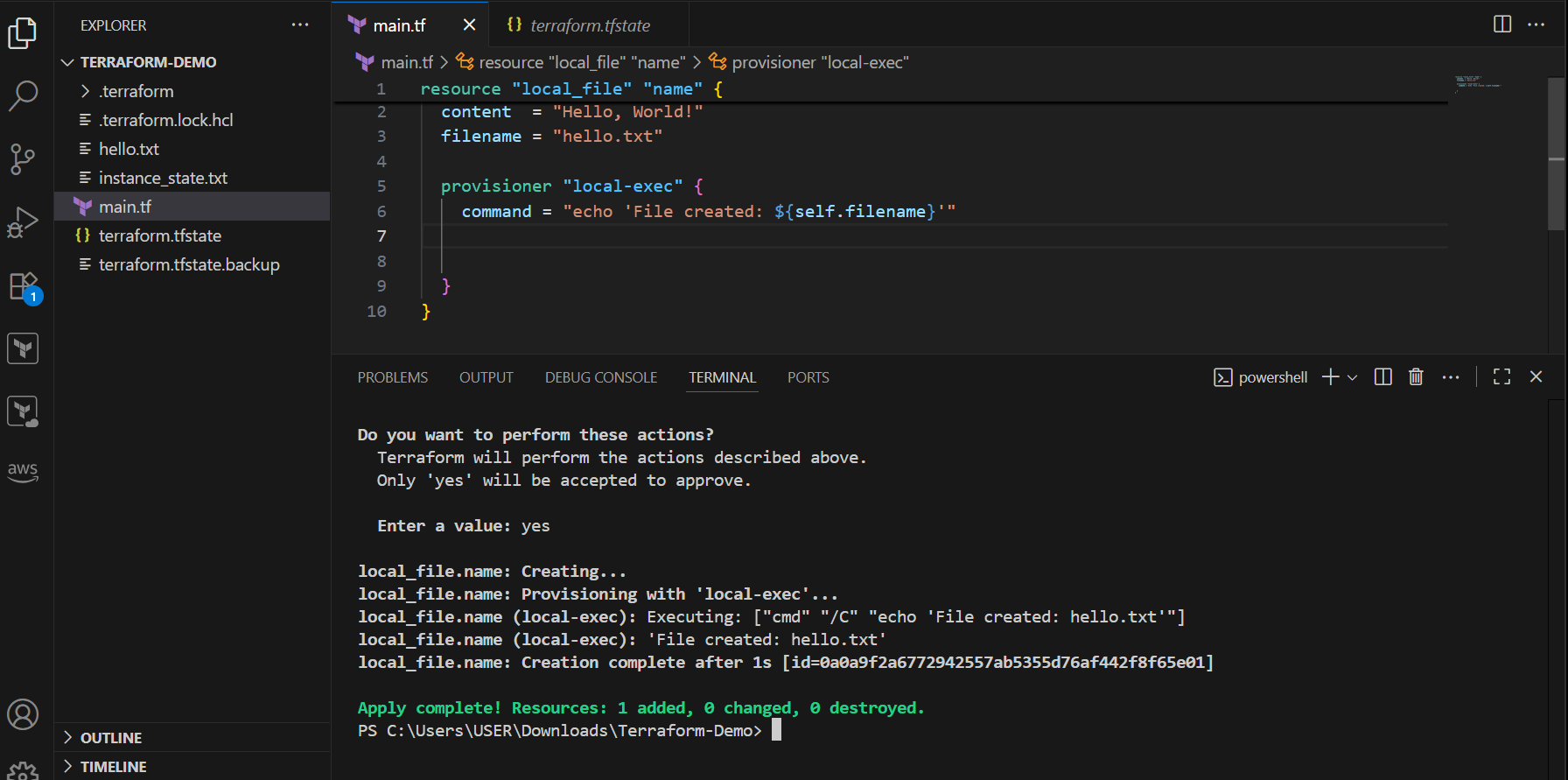
Terraform provisioners allow us to execute command,scripts on remote machines or

local place were terraform is installed.

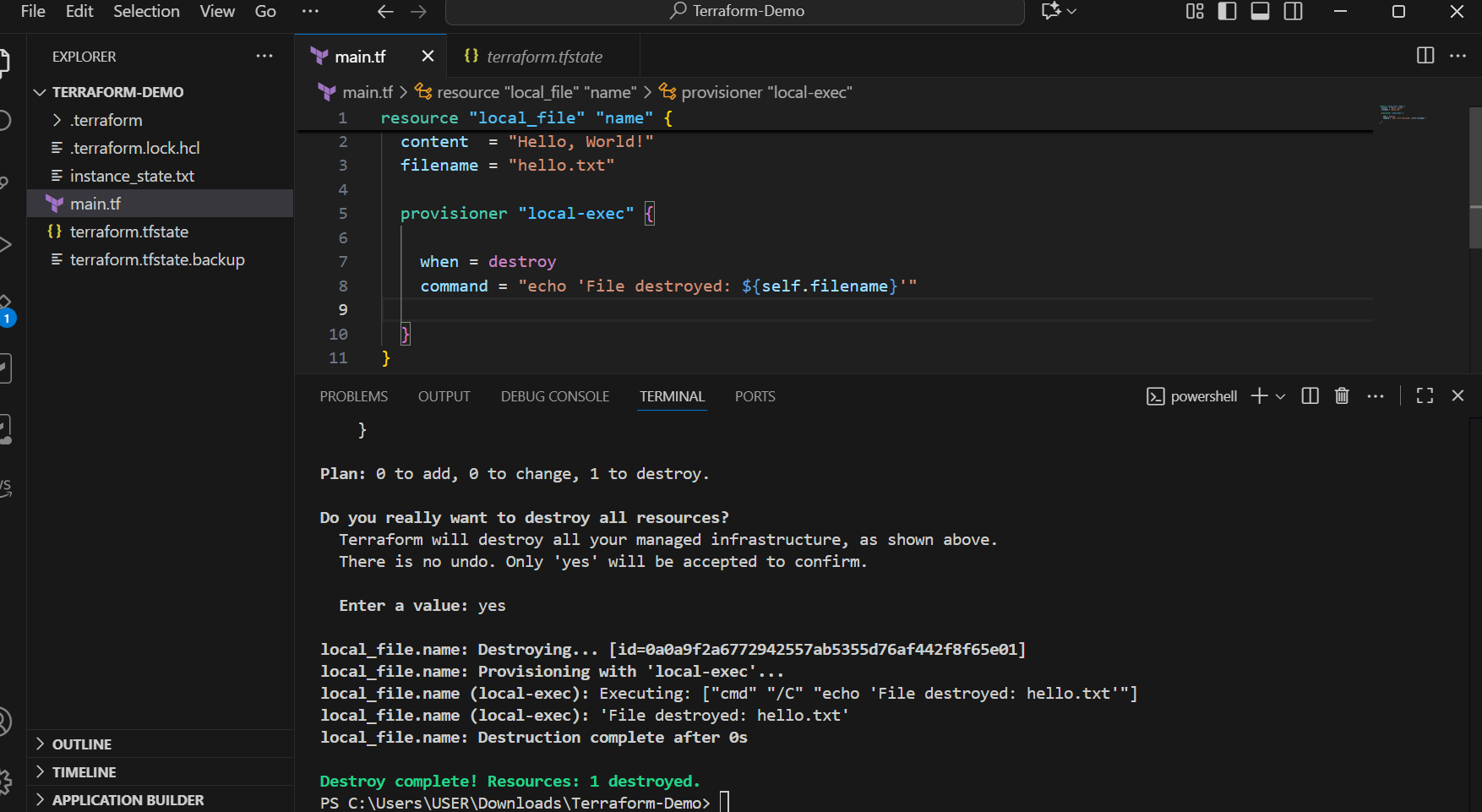
Provisioners will be written inside the reource blocks.

****

**1) Create at the time creating resource (Default)**

****

**2) create at the time of destroyin resource (when = destroy)**

****

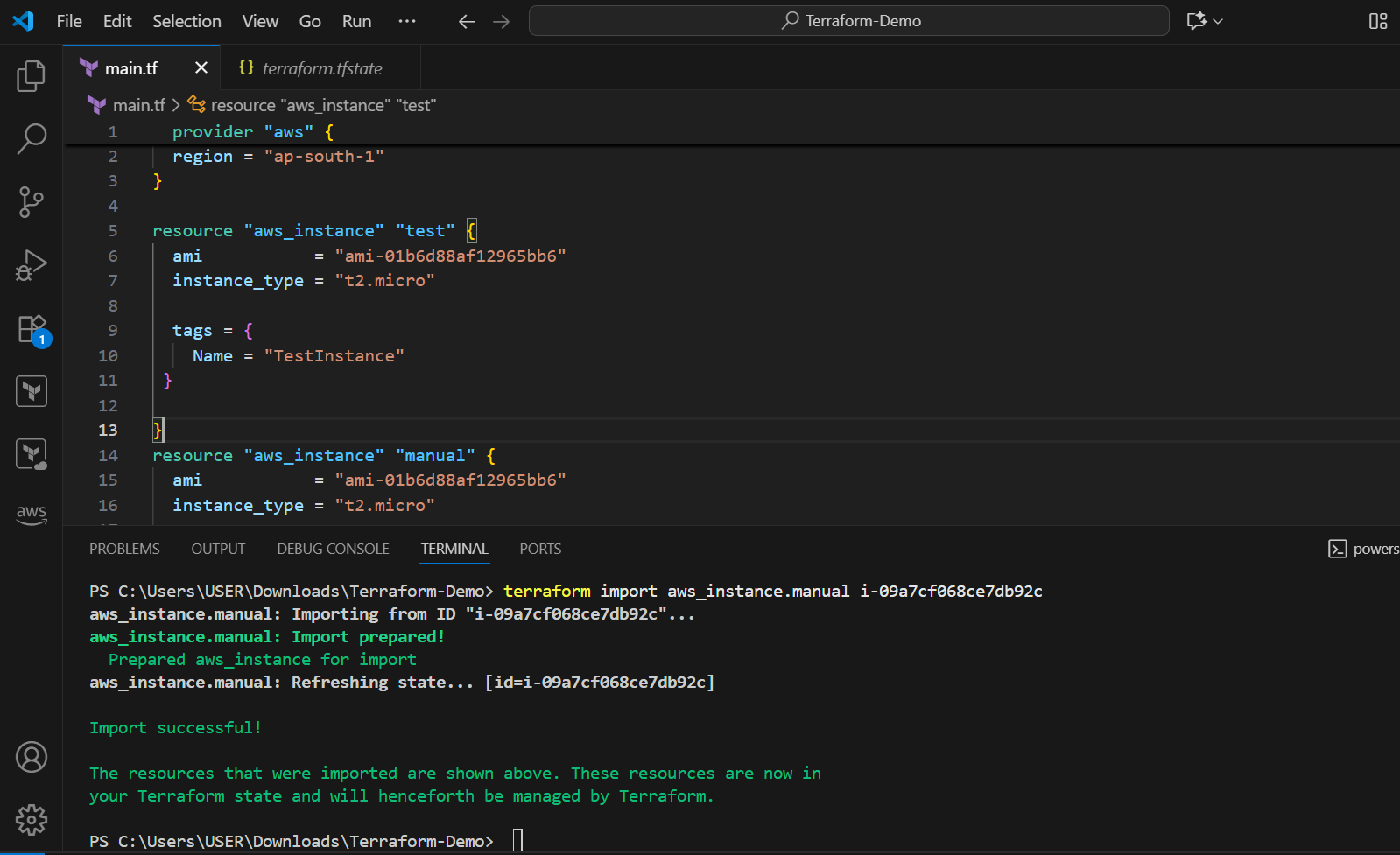
Terraform import:

Terraform import is used to import the existing infrastructure in terraform state file.

Once import is done then we cna be able to create/delete and manage the infrastructure.

In order to import any resource we need to write the resource details in configuration file.

terraform import aws\_instance.<name> instance\_id

****

**3.Create one EC2 instance with httpd installed using a Terraform script.**

**Step-1**

First configure aws to your vscode

Create two files

**Main.tf**

provider "aws" {

  region = var.region

}

resource "aws\_instance" "test-server" {

  ami           = var.ami\_id

  instance\_type = var.instance\_type

  key\_name      = var.key\_name

  tags = {

    Name = "Terraform-server"

  }

  user\_data = <<-EOF

              #!/bin/bash

              yum update -y

              yum install -y httpd

              systemctl enable httpd

              systemctl start httpd

              echo "<h1>Hello from Test-server</h1>" > /var/www/html/index.html

              EOF

}

**Variable.tf**

variable "region" {

  description = "AWS region"

  default     = "ap-south-1"

}

variable "ami\_id" {

  description = "AMI ID for EC2"

  default     = "ami-01b6d88af12965bb6" # Amazon Linux 2 in ap-south-1 (update if needed)

}

variable "instance\_type" {

  description = "EC2 instance type"

  default     = "t2.micro"

}

variable "key\_name" {

  description = "EC2 key pair name for SSH"

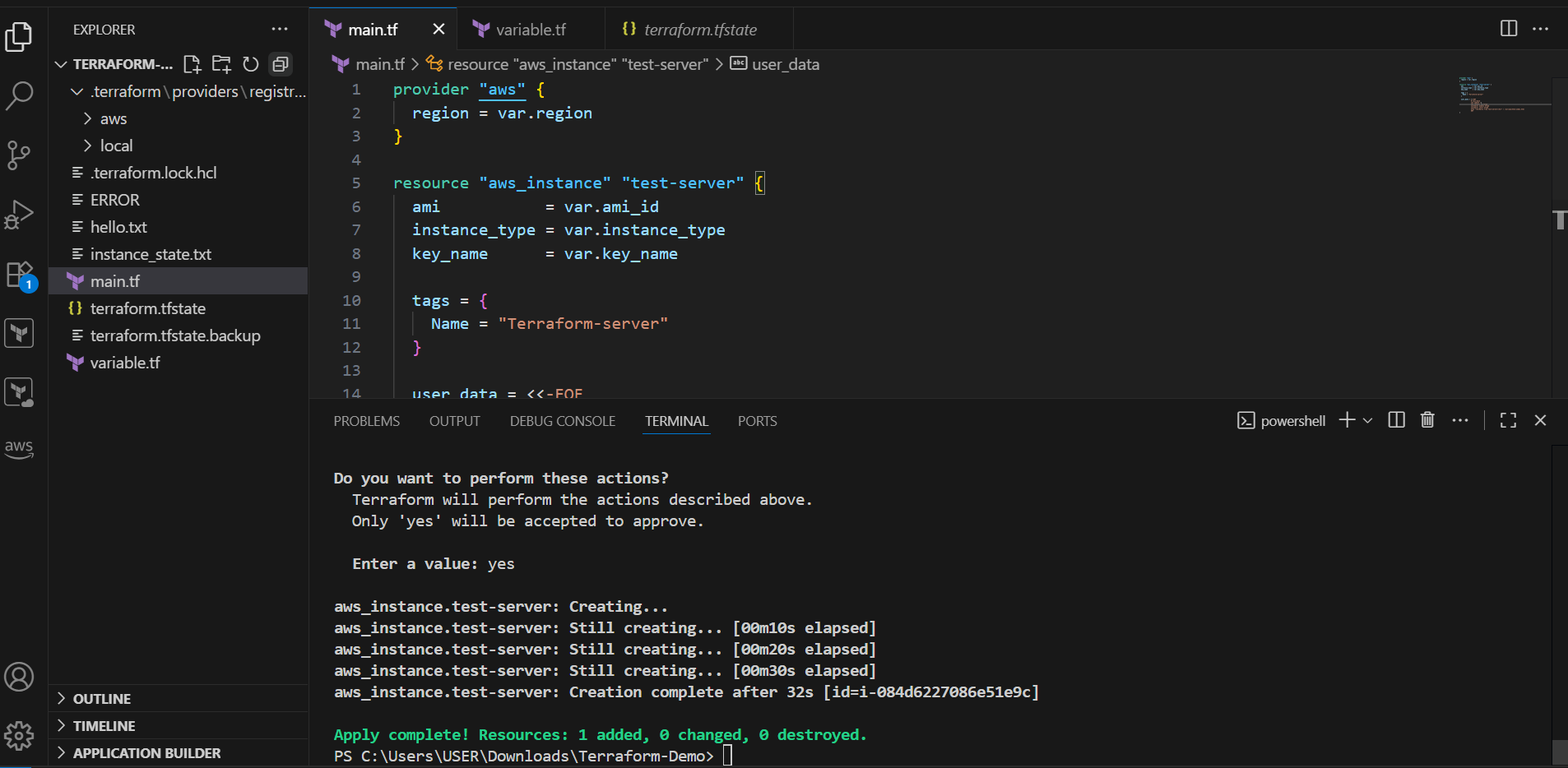
  default     = "jenkins" # Update with your key pair name

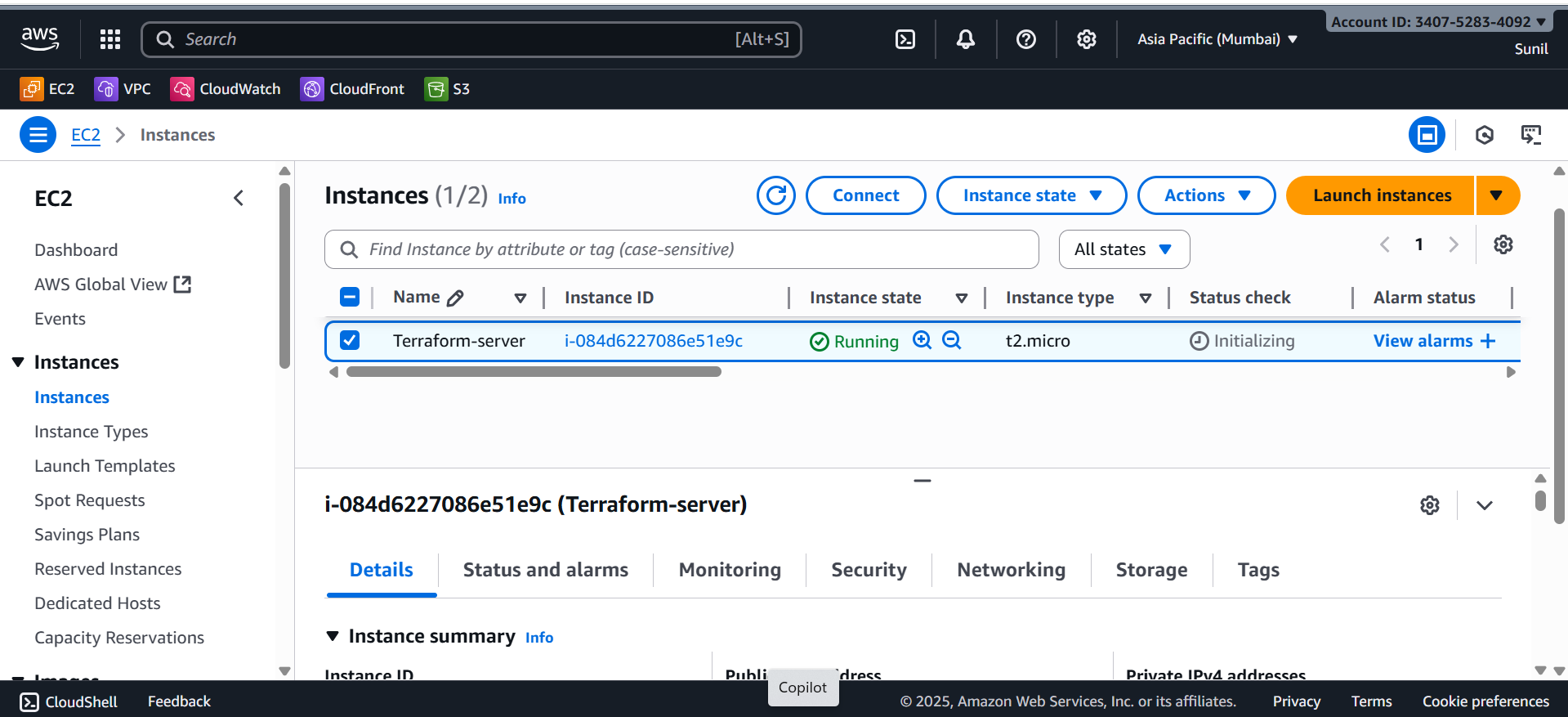
}

Save the main.tf and variables.tf

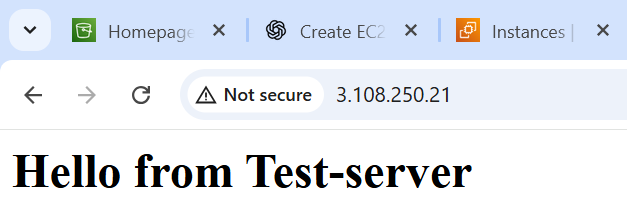
**terraform init**

**terraform apply -auto-approve**

****

****

Open http://<public-ip> in your browser — you should see **Hello from Test-server**

****

**4.Set up S3 as backend for task 3.**

**In main.tf**

provider "aws" {

  region = "ap-south-1"

}

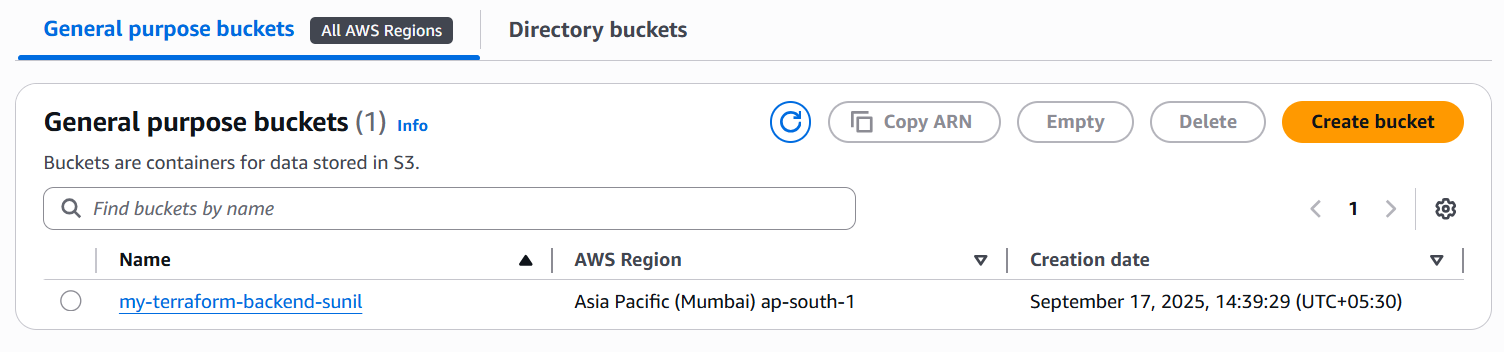
# Create S3 bucket for backend

resource "aws\_s3\_bucket" "s3\_bucket" {

  bucket = "my-terraform-backend-sunil"

}

**Save and apply**

****

**5.Set up DynamoDB locking for task 3.**

# Create DynamoDB table for state locking

resource "aws\_dynamodb\_table" "dynamodb\_terraform\_state\_lock" {

  name           = "terraform-state-lock-dynamo"

  hash\_key       = "LockID"

  read\_capacity  = 20

  write\_capacity = 20

  attribute {

    name = "LockID"

    type = "S"

  }

}

terraform {

  backend "s3" {

    bucket         = "my-terraform-backend-sunil"   # same bucket you created

    key            = "terraform.tfstate"            # path inside bucket

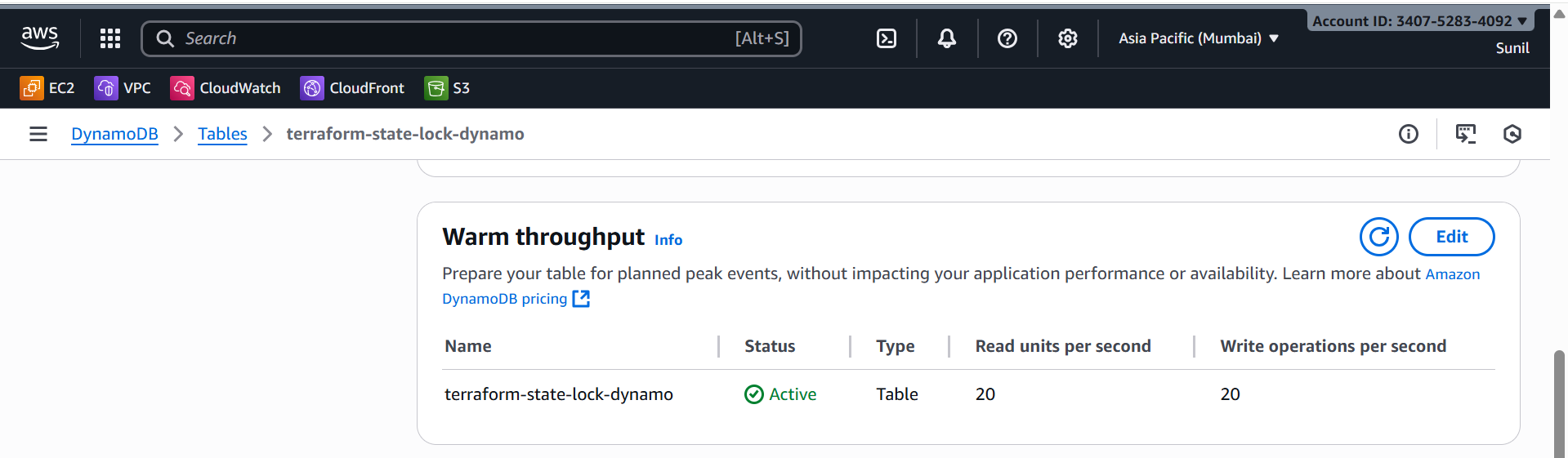
    region         = "ap-south-1"

    dynamodb\_table = "terraform-state-lock-dynamo"  # locking

    encrypt        = true

  }

}

****

**Default Behavior**

* By default, Terraform stores the state file (terraform.tfstate) **locally** in your project folder.
* This state file is Terraform’s “memory” of what resources exist in AWS (or any cloud).
* Problem: If multiple people run Terraform, or if you switch machines, you risk conflicts or losing the state.

## Creating Remote Backend Resources

You first created:

1. **S3 Bucket (my-terraform-backend-sunil)** → This is where Terraform will store the terraform.tfstate file.
2. **DynamoDB Table (terraform-state-lock-dynamo)** → This prevents two people from running Terraform at the same time (state locking).

At this point, you just had infrastructure ready, but Terraform was **still using local state**.

## Configuring Terraform Backend

Then you added the **backend block**:

terraform {

backend "s3" {

bucket = "my-terraform-backend-sunil"

key = "terraform.tfstate"

region = "ap-south-1"

dynamodb\_table = "terraform-state-lock-dynamo"

encrypt = true

}

}

What this means:

* bucket: tells Terraform where in S3 to save the state file.
* key: the filename inside the bucket (terraform.tfstate).
* region: where the bucket exists.
* dynamodb\_table: enables **locking** so two people can’t run terraform apply simultaneously.
* encrypt: ensures your state file is encrypted at rest.

## Reinitializing with terraform init -migrate-state

* When you ran terraform init -migrate-state, Terraform **moved your local state** into S3.
* From now on, Terraform always pulls the latest state from **S3** before planning/applying changes.
* DynamoDB is consulted first to create a **lock entry**, so no one else can apply at the same time.