“Expert Cloud Consulting”

SOP | AWS Infrastructure with Terraform: S3 and DynamoDB Backend Setup

27 Jun 2025

—

Contributed by: Akshata
Approved by: Akshay (In Review)
Expert Cloud Consulting
Office #811, Gera Imperium Rise,
Hinjewadi Phase-II Rd, Pune, India – 411057

AWS Infrastructure with Terraform: S3 and DynamoDB Backend Setup

Expert Cloud Consulting

Enhance Optimise & Scale

ASCP GPUonCLOUD Pvt Ltd

Document Overview

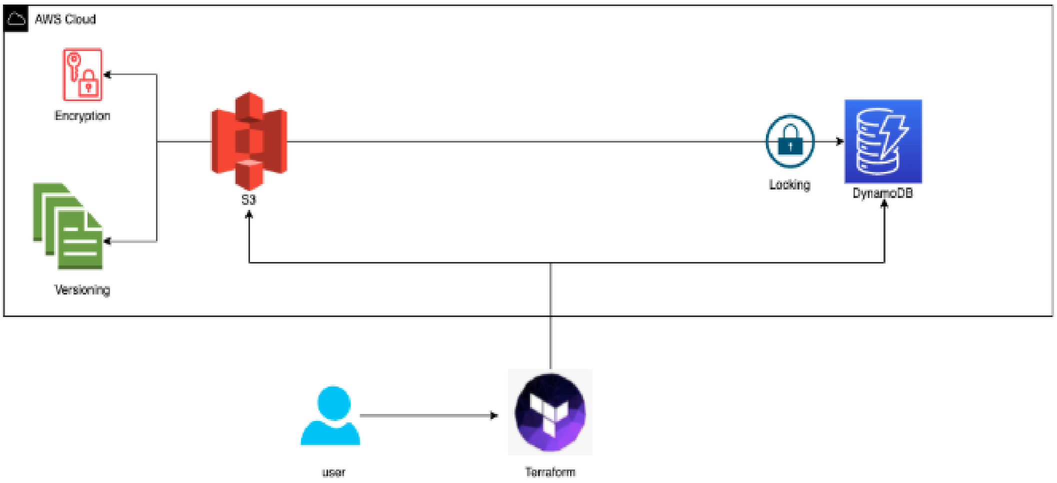
This document provides a detailed walkthrough of setting up a robust and collaborative Terraform backend using AWS S3 and DynamoDB, along with provisioning a complete multi-tier infrastructure on AWS. It demonstrates the practical application of Infrastructure as Code (IaC**)** principles and how centralized state management improves team collaboration and operational safety.

The setup includes multiple stages—from configuring IAM access and Terraform providers to provisioning VPCs, EC2 instances, and an Application Load Balancer—while ensuring the Terraform state is secure, centralized, and version-controlled

Document References

The following resources were referred to during the creation and execution of this Terraform-based infrastructure setup

Date	Document	Filename / Url
27 Jun	Streamlining AWS Infrastructure with Terraform using S3 and DynamoDB	https://www.linkedin.com/pulse/streamlining-aws-infrastructure-terraform-s3-dynamodb-erumal-lvlpc/
27 Jun	Terraform State Remote Storage with S3 and Locking using DynamoDB	https://www.linkedin.com/pulse/terraform-state-remote-storage-s3-locking-dynamodb-oramu-/



Why Use a Terraform Backend?

When you start using Terraform, local state files might suffice for small projects. But as teams grow and infrastructure scales, storing state files locally becomes a bottleneck—or worse, a liability. A remote backend, like AWS S3 with DynamoDB for locking, solves this by enabling

- **Collaboration**: Multiple team members can work on the same infrastructure without conflicts.
- **Security**: State files often contain sensitive data, and S3 offers encryption and access control.
- **Consistency**: DynamoDB ensures state locking and prevents race conditions during deployments.

Setting Up the S3 Bucket

- **Bucket Creation**: Define an S3 bucket resource in Terraform with a unique name (S3 bucket names are globally unique, so get creative or append a random suffix).
- **Versioning**: Enable versioning to maintain a history of state file changes—crucial for rollback scenarios
- **Access Control**: Restrict access with a bucket policy or IAM roles to ensure only authorized users or services can interact with it.

Adding DynamoDB for State Locking

S3 alone handles storage, but it doesn't prevent multiple users from overwriting the state simultaneously. Enter DynamoDB:

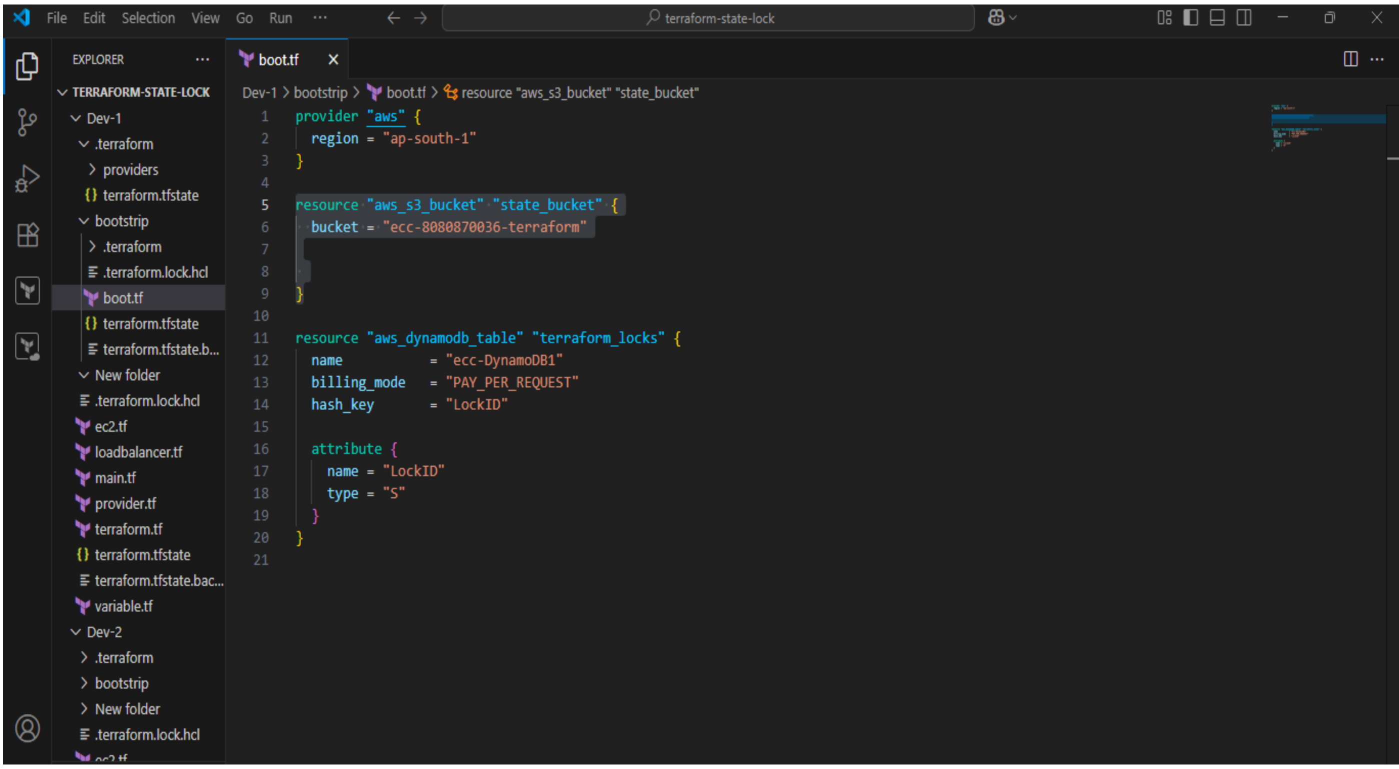
Primary Key: We use a simple key called LockID to track the lock.

Simple Setup: Only one attribute is needed. No complex table design required.

On-Demand Mode: We use PAY_PER_REQUEST so we only pay for what we use. This is great for small teams or less frequent usage

backend.tf

S3.tf



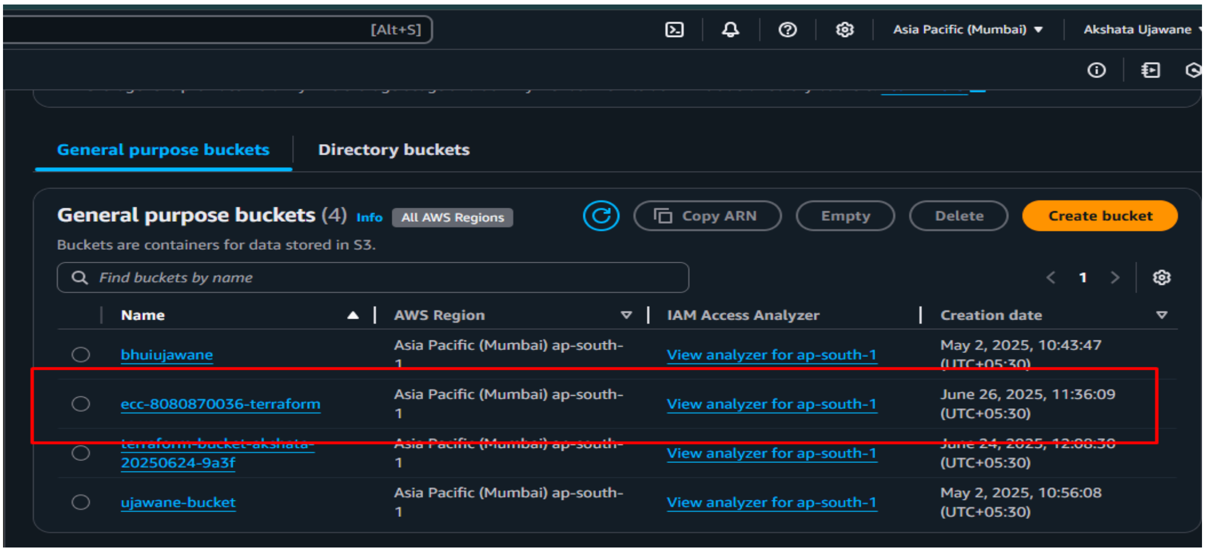
```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_s3_bucket" "state_bucket" {
  bucket = "ecc-8080870036-terraform"


}

resource "aws_dynamodb_table" "terraform_locks" {
  name           = "ecc-DynamoDB1"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}

```

After setting up both the S3 bucket and DynamoDB table**,** you can run **terraform init Terraform plan Terraform apply** to initialize the backend

```
 terraform init
terraform plan
terraform apply -auto-approve
```

This successfully created

S3 Bucket

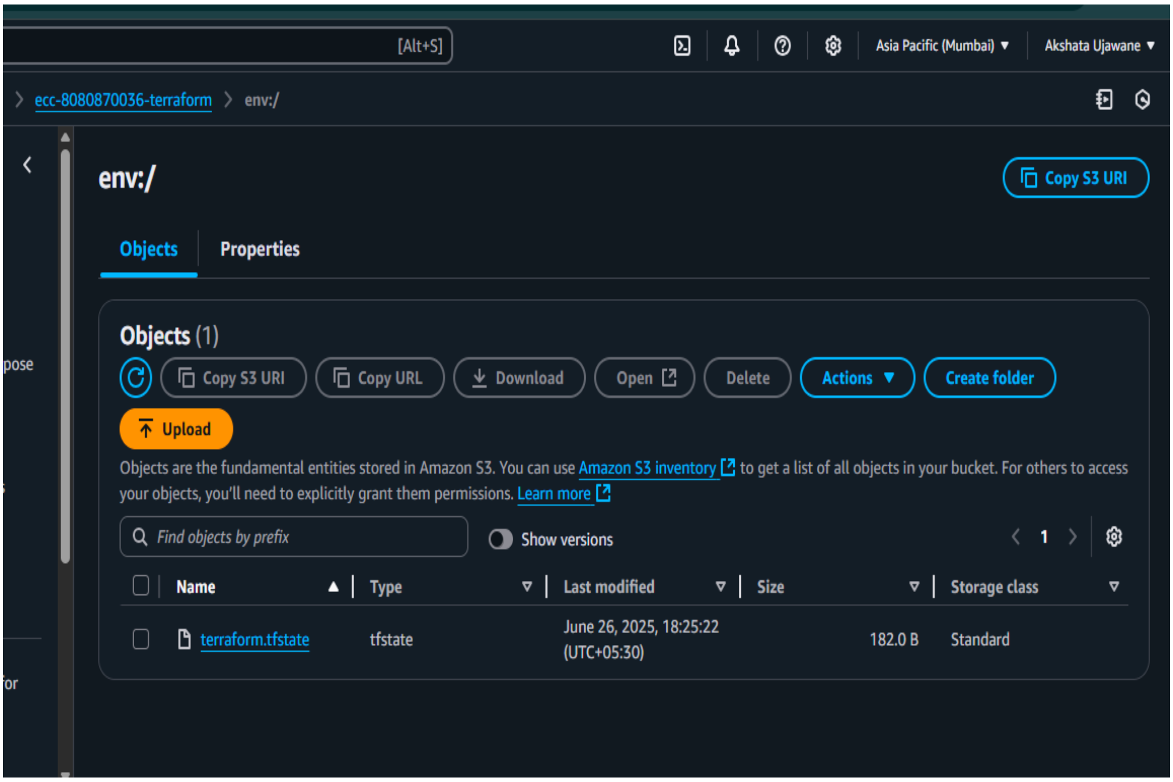


General purpose buckets (4) Info All AWS Regions

Buckets are containers for data stored in S3.

Find buckets by name

Name	AWS Region	IAM Access Analyzer	Creation date
○ bhuiujawane	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1	May 2, 2025, 10:43:47 (UTC+05:30)
○ ecc-8080870036-terraform	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1	June 26, 2025, 11:36:09 (UTC+05:30)
○ terraform-bucket-akshata-20250624-9a3f	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1	June 24, 2025, 12:08:38 (UTC+05:30)
○ ujawane-bucket	Asia Pacific (Mumbai) ap-south-1	View analyzer for ap-south-1	May 2, 2025, 10:56:08 (UTC+05:30)



env:/

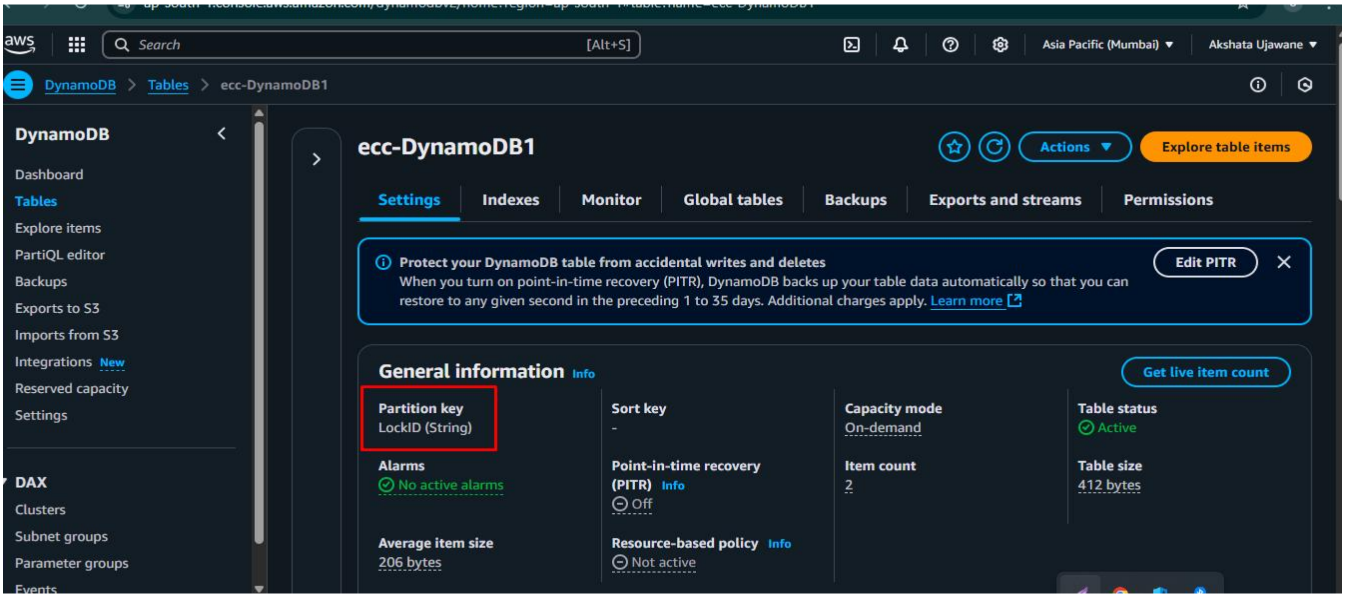
Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

Name	Type	Last modified	Size	Storage class
□ terraform.tfstate	tfstate	June 26, 2025, 18:25:22 (UTC+05:30)	182.0 B	Standard

DynamoDB



ecc-DynamoDB1

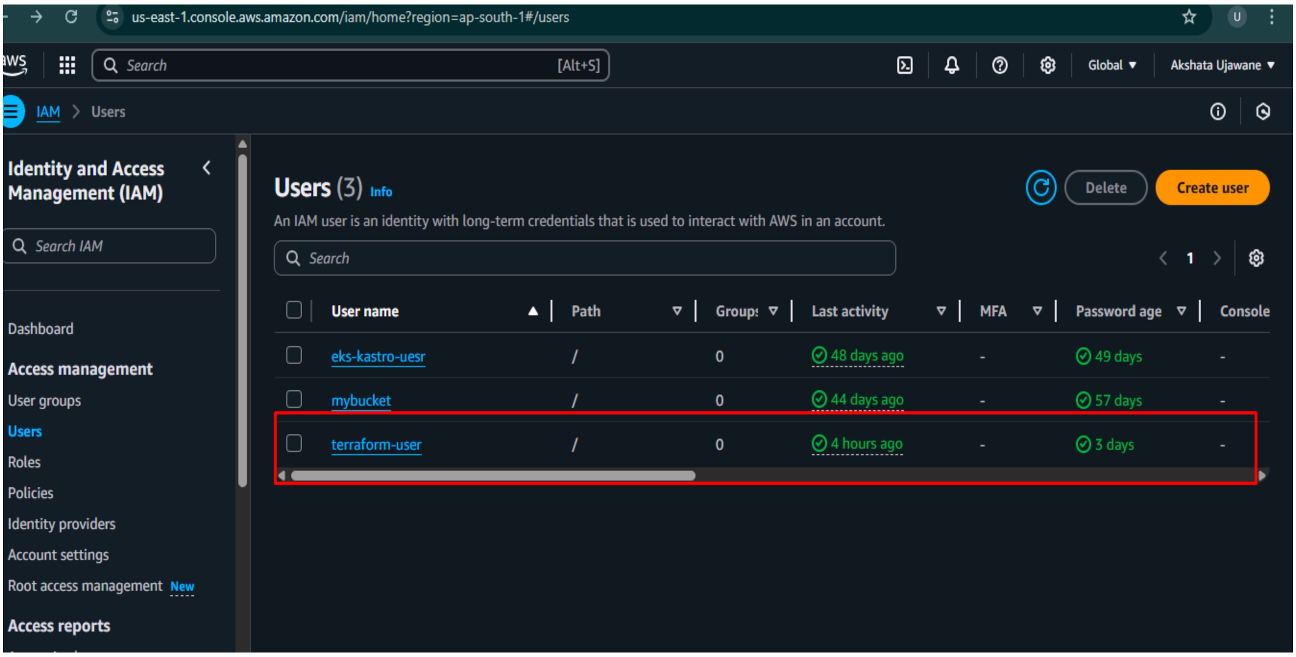
Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 1 to 35 days. Additional charges apply. Learn more

General information Info

Partition key LockID (String)	Sort key -	Capacity mode On-demand	Table status ✓ Active
Alarms ✓ No active alarms	Point-in-time recovery (PITR) Info ⊖ Off	Item count 2	Table size 412 bytes
Average item size 206 bytes	Resource-based policy Info ⊖ Not active		

IAM User Integration

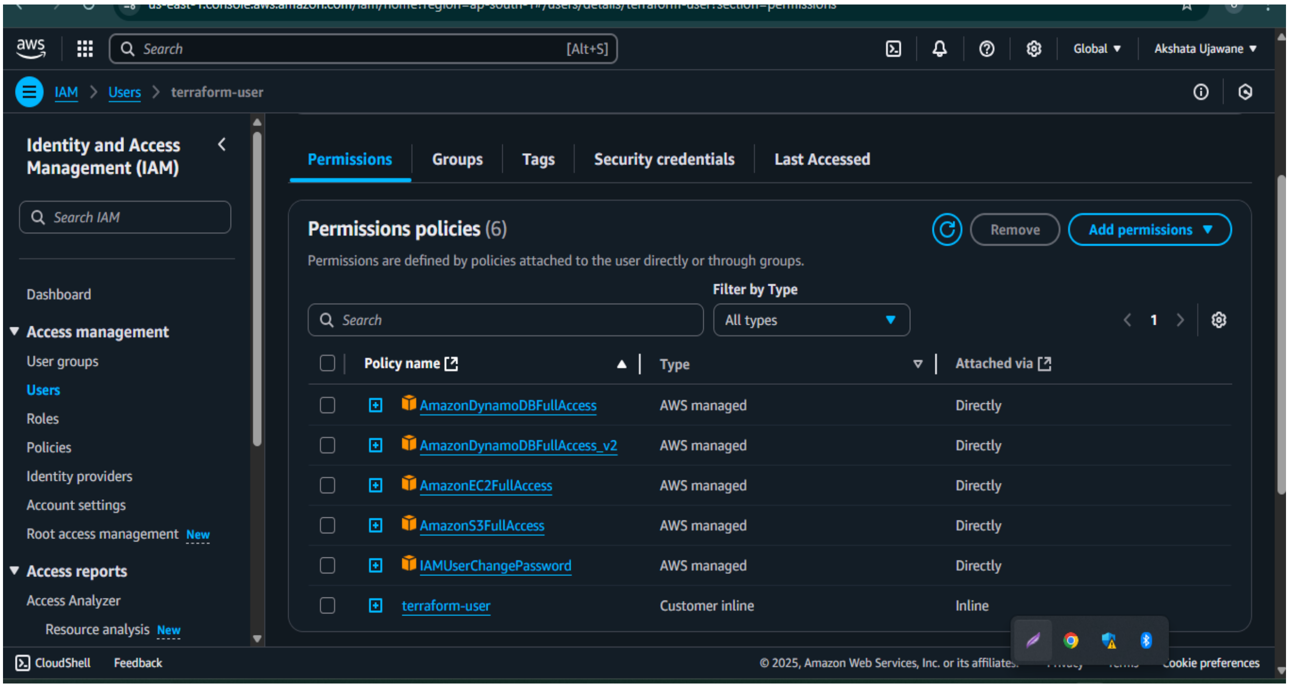
- Create and configure **IAM users** with fine-grained permissions.



Users (3) Info

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

□	User name ▲	Path	Group:	Last activity	MFA	Password age	Console
□	eks-kastro-uesr	/	0	✓ 48 days ago	-	✓ 49 days	-
□	mybucket	/	0	✓ 44 days ago	-	✓ 57 days	-
□	terraform-user	/	0	✓ 4 hours ago	-	✓ 3 days	-

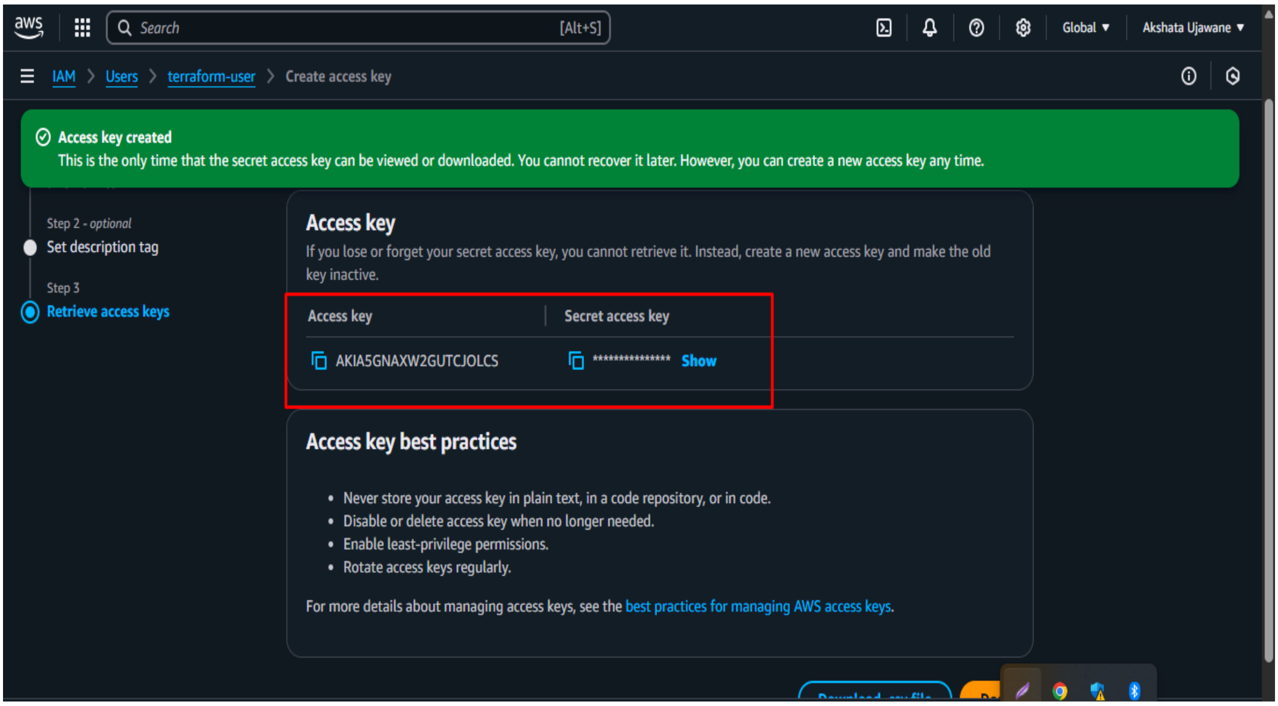


Permissions policies (6)

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

Policy name	Type	Attached via
AmazonDynamoDBFullAccess	AWS managed	Directly
AmazonDynamoDBFullAccess_v2	AWS managed	Directly
AmazonEC2FullAccess	AWS managed	Directly
AmazonS3FullAccess	AWS managed	Directly
IAMUserChangePassword	AWS managed	Directly
terraform-user	Customer inline	Inline

Securely connect Terraform to AWS using **Access Key** and **Secret Key** for automated deployments.



Access key created

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

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
AKIA5GNAXW2GUTCJOLCS	***************

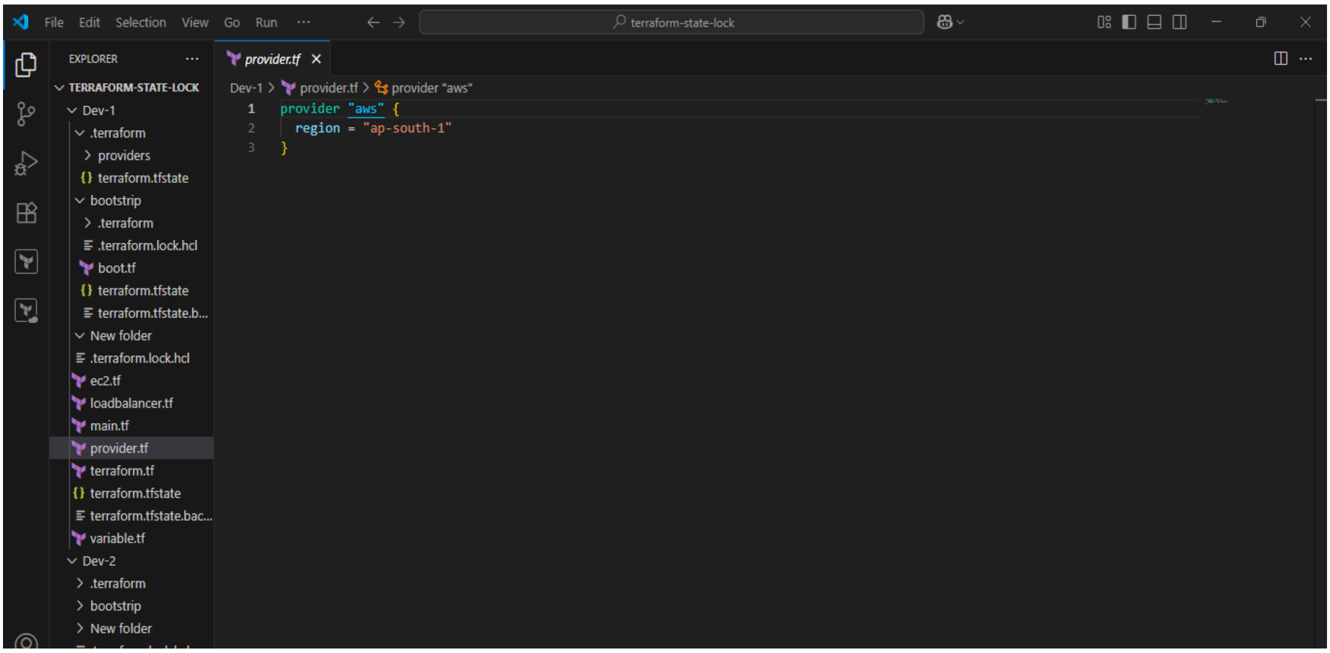
Access key best practices

- 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.

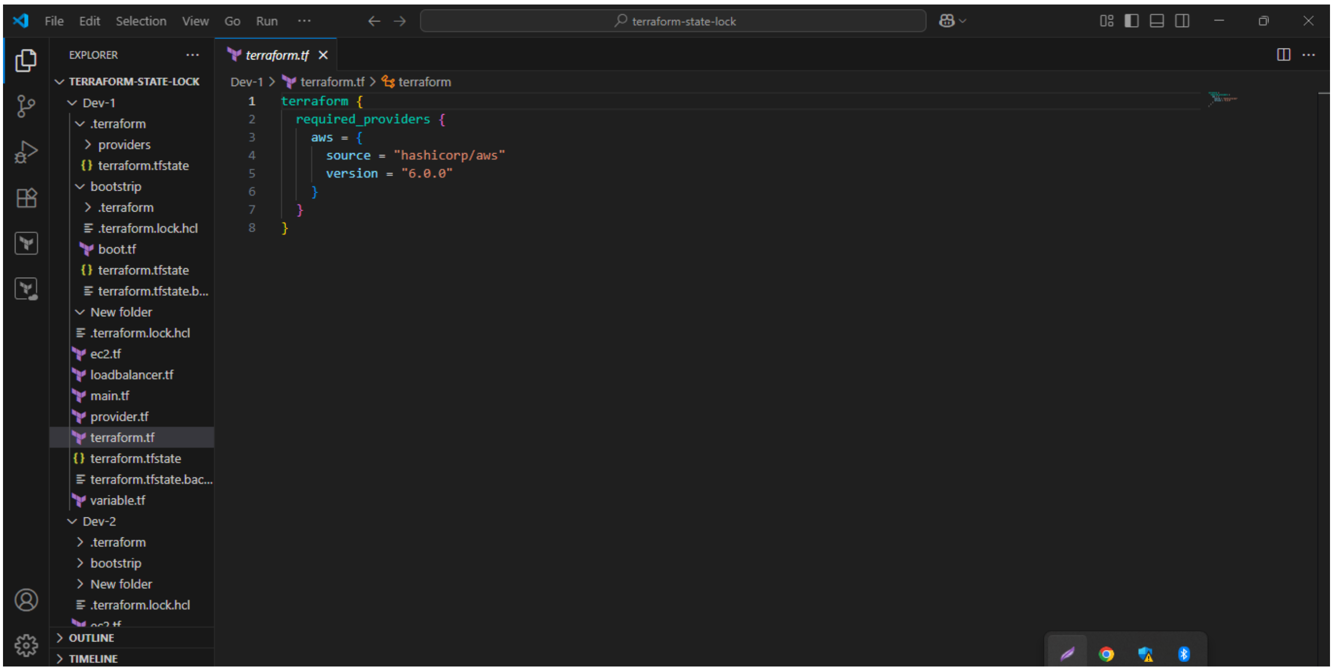
provider.tf:

Specifies AWS provider and region



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

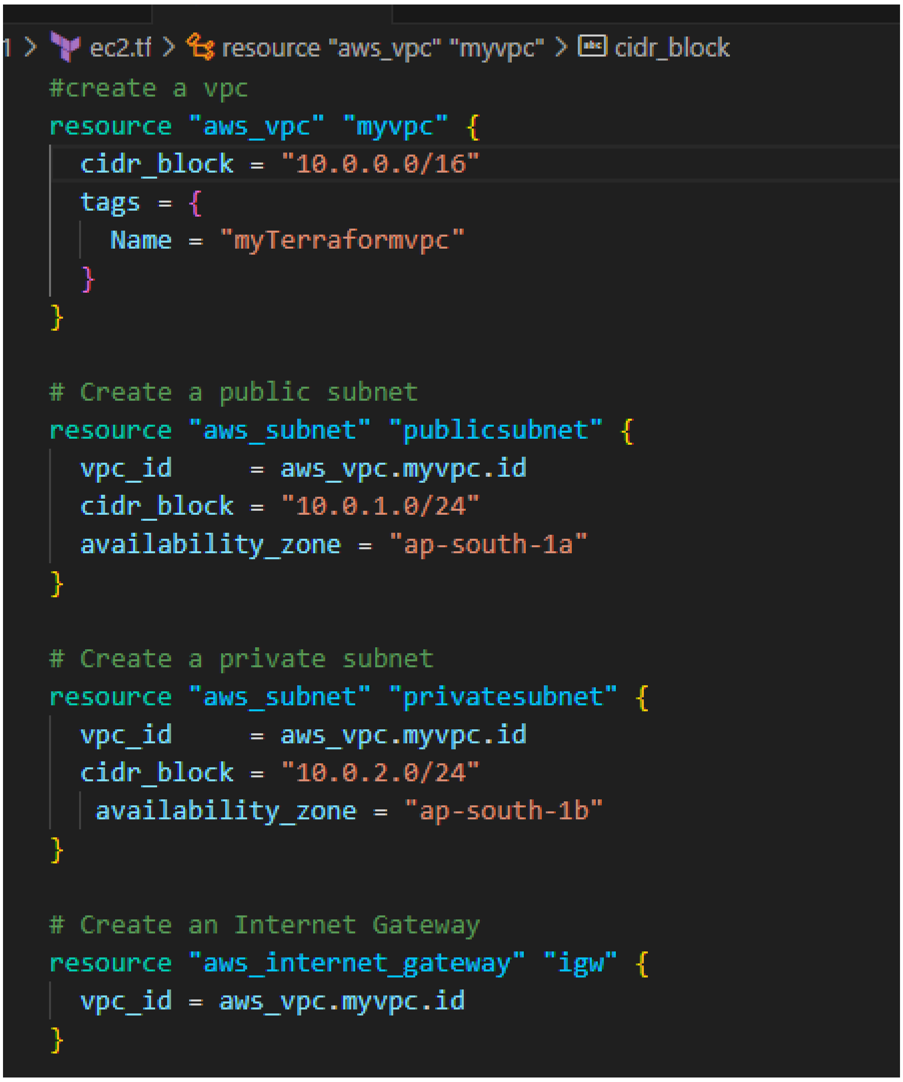
terraform.tf



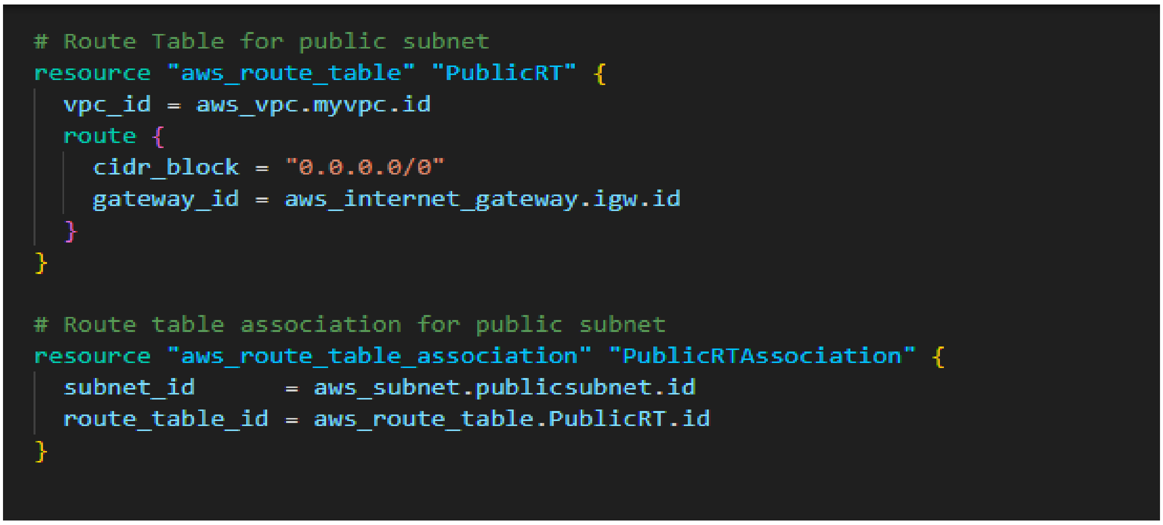
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "6.0.0"
    }
  }
}
```

Created VPC, Subnets, Route Tables, and Internet Gateway

These components form the **network architecture** of your cloud environment.

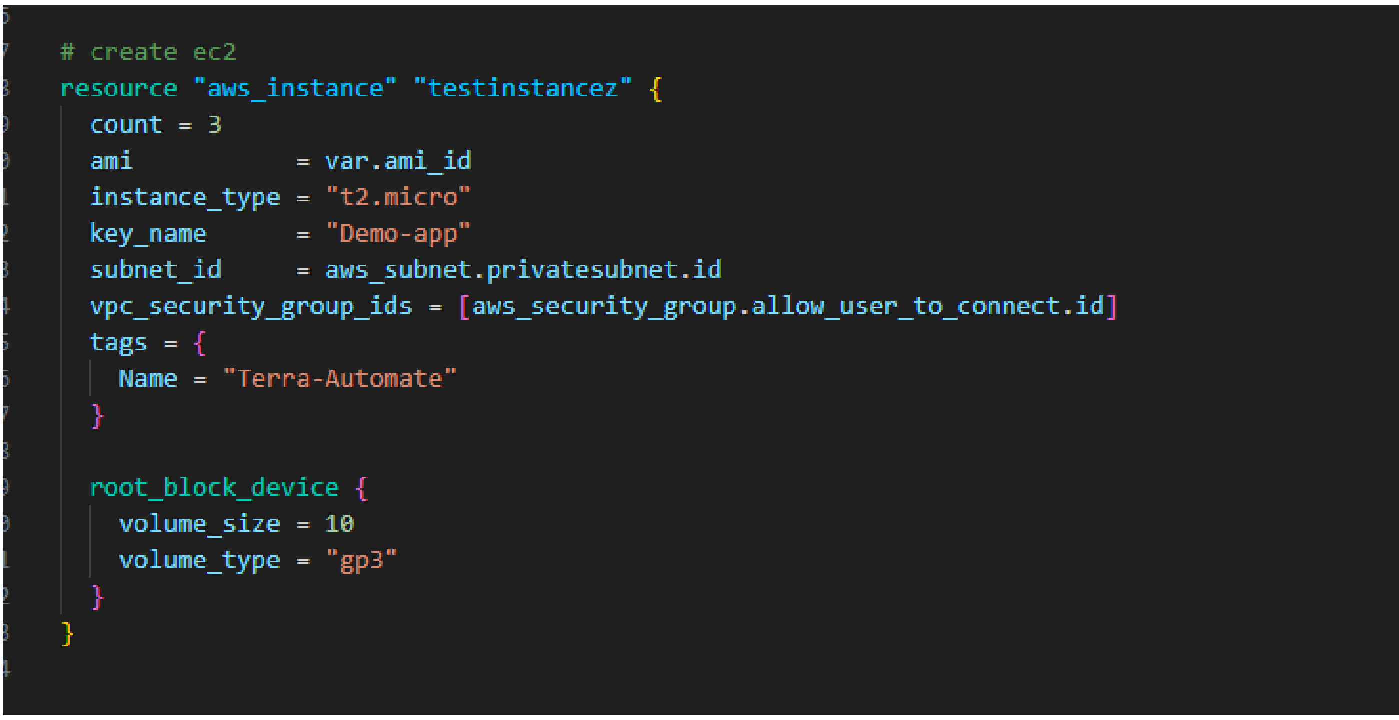


```
1 > ec2.tf > resource "aws_vpc" "myvpc" > cidr_block
#create a vpc
resource "aws_vpc" "myvpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "myTerraformvpc"
  }
}

# Create a public subnet
resource "aws_subnet" "publicsubnet" {
  vpc_id      = aws_vpc.myvpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "ap-south-1a"
}

# Create a private subnet
resource "aws_subnet" "privatesubnet" {
  vpc_id      = aws_vpc.myvpc.id
  cidr_block = "10.0.2.0/24"
   availability_zone = "ap-south-1b"
}

# Create an Internet Gateway
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.myvpc.id
}
```



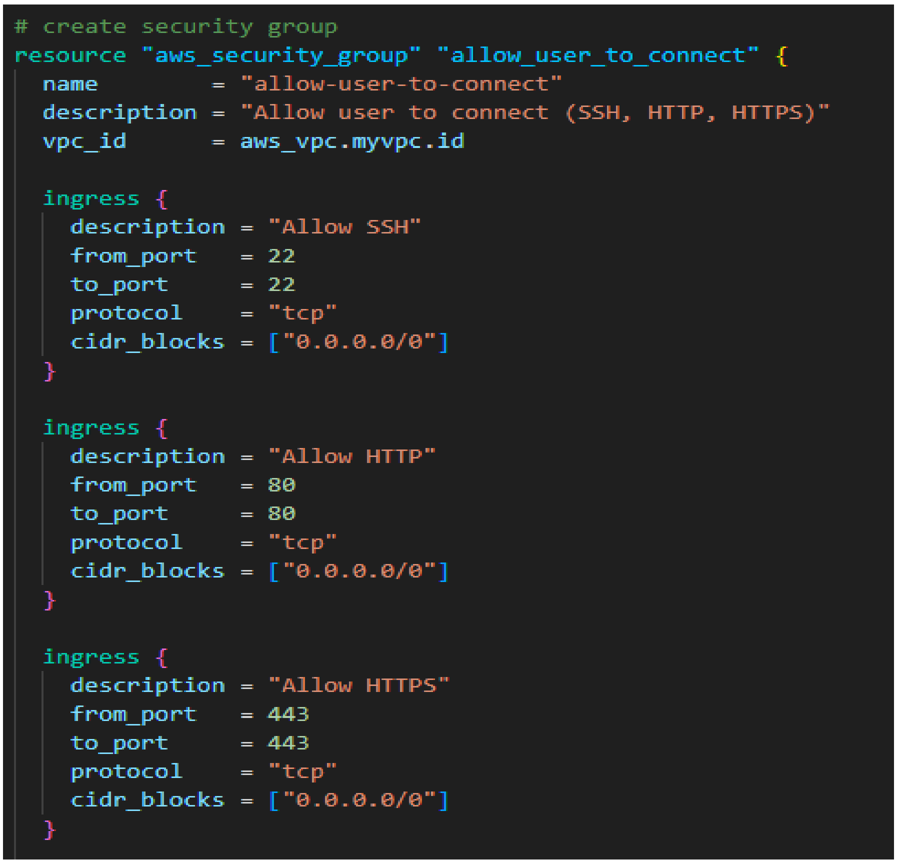
```
# Route Table for public subnet
resource "aws_route_table" "PublicRT" {
  vpc_id = aws_vpc.myvpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }
}

# Route table association for public subnet
resource "aws_route_table_association" "PublicRTAssociation" {
  subnet_id      = aws_subnet.publicsubnet.id
  route_table_id = aws_route_table.PublicRT.id
}
```

Launch EC2 Instances in Private Subnet

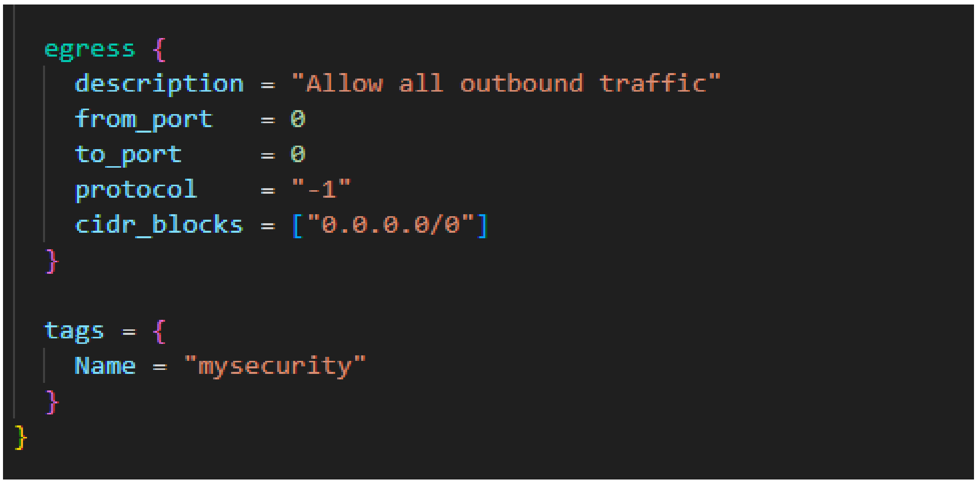


```
# create ec2
resource "aws_instance" "testinstancez" {
  count = 3
  ami           = var.ami_id
  instance_type = "t2.micro"
  key_name      = "Demo-app"
  subnet_id     = aws_subnet.privatesubnet.id
  vpc_security_group_ids = [aws_security_group.allow_user_to_connect.id]
  tags = {
    Name = "Terra-Automate"
  }

  root_block_device {
    volume_size = 10
    volume_type = "gp3"
  }
}
```

Creating Security Group to allow SSH, HTTP, and HTTPS



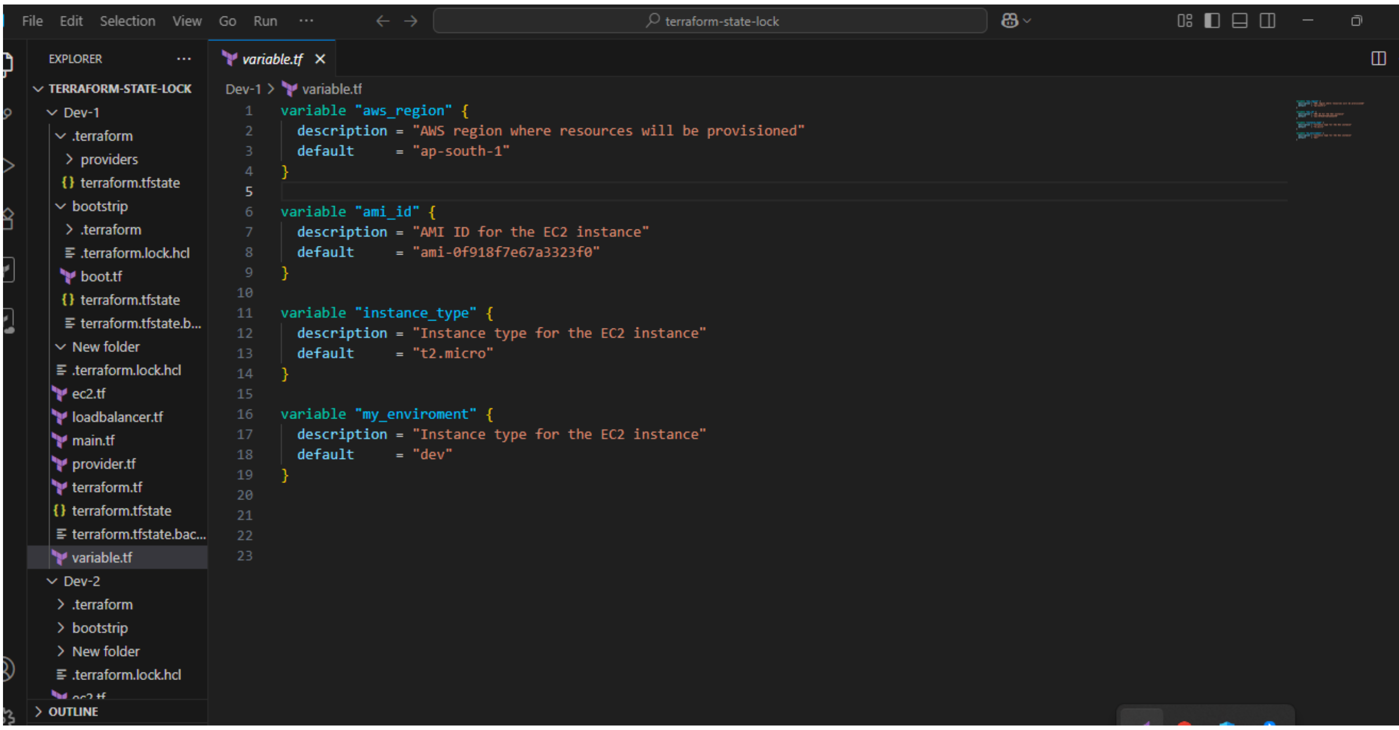
```
# create security group
resource "aws_security_group" "allow_user_to_connect" {
  name        = "allow-user-to-connect"
  description = "Allow user to connect (SSH, HTTP, HTTPS)"
  vpc_id      = aws_vpc.myvpc.id

  ingress {
    description = "Allow SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    description = "Allow HTTP"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    description = "Allow HTTPS"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
```



```
  egress {
    description = "Allow all outbound traffic"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "mysecurity"
  }
}
```

variable.tf

It allows us to define reusable and dynamic input values for Terraform resources. This makes the configuration cleaner and easier to manage**.**



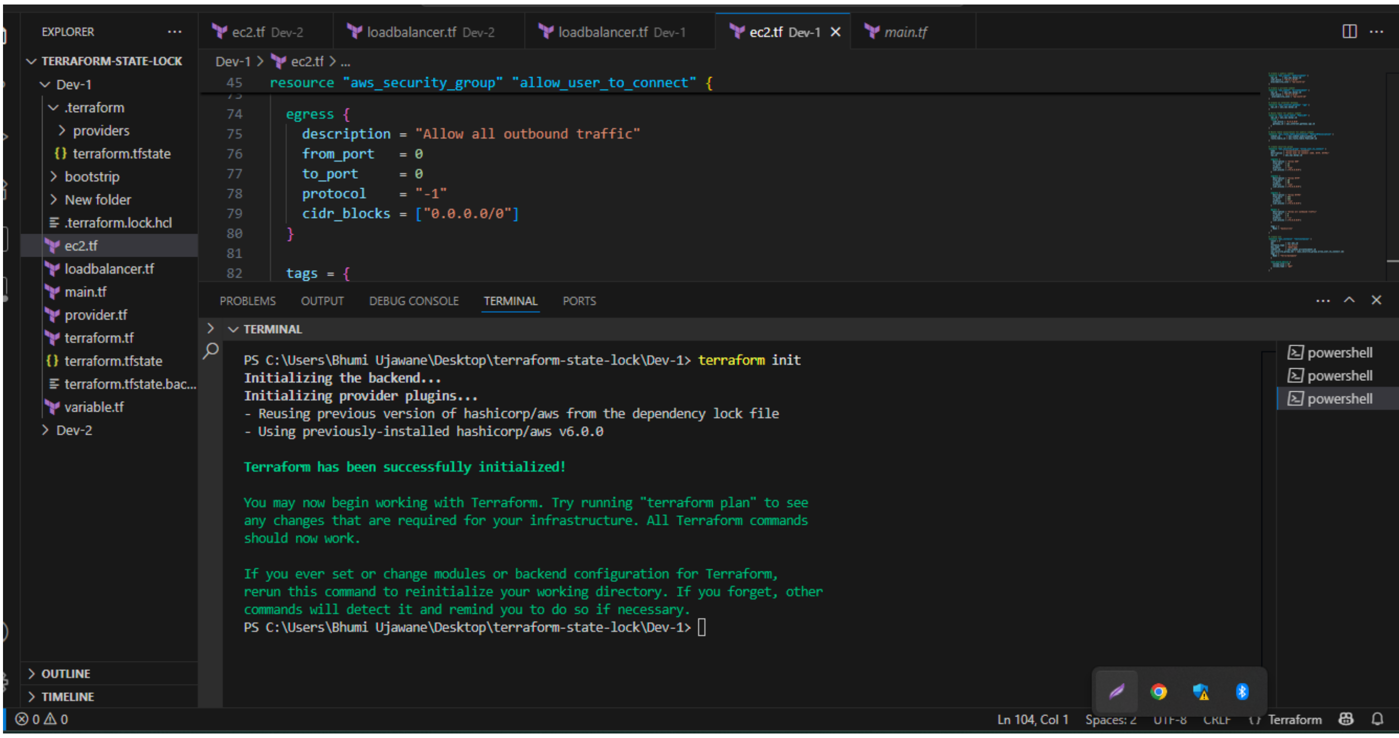
```
variable "aws_region" {
  description = "AWS region where resources will be provisioned"
  default     = "ap-south-1"
}

variable "ami_id" {
  description = "AMI ID for the EC2 instance"
  default     = "ami-0f918f7e67a3323f0"
}

variable "instance_type" {
  description = "Instance type for the EC2 instance"
  default     = "t2.micro"
}

variable "my_enviroment" {
  description = "Instance type for the EC2 instance"
  default     = "dev"
}
```

Dev-1 User: Deploying Infrastructure with Terraform

Once all your .tf files are ready (VPC, EC2, subnets, load balancer, variables, etc.), run the following Terraform commands to deploy your infrastructure

Terraform init

Initializes your Terraform project

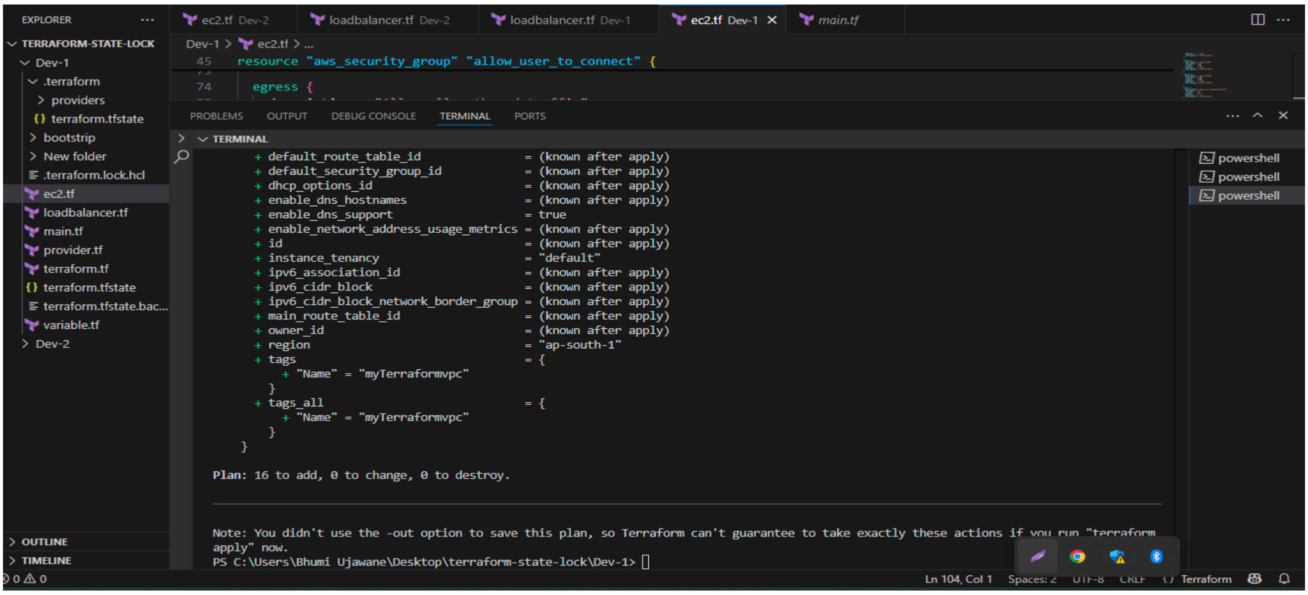


```
resource "aws_security_group" "allow_user_to_connect" {
  egress {
    description = "Allow all outbound traffic"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
```

```
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-1> terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.0.0

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.
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-1> 
```

Terraform Plan

Shows the **execution plan**—what Terraform will create or change. This helps verify that the configuration is correct before applying changes.

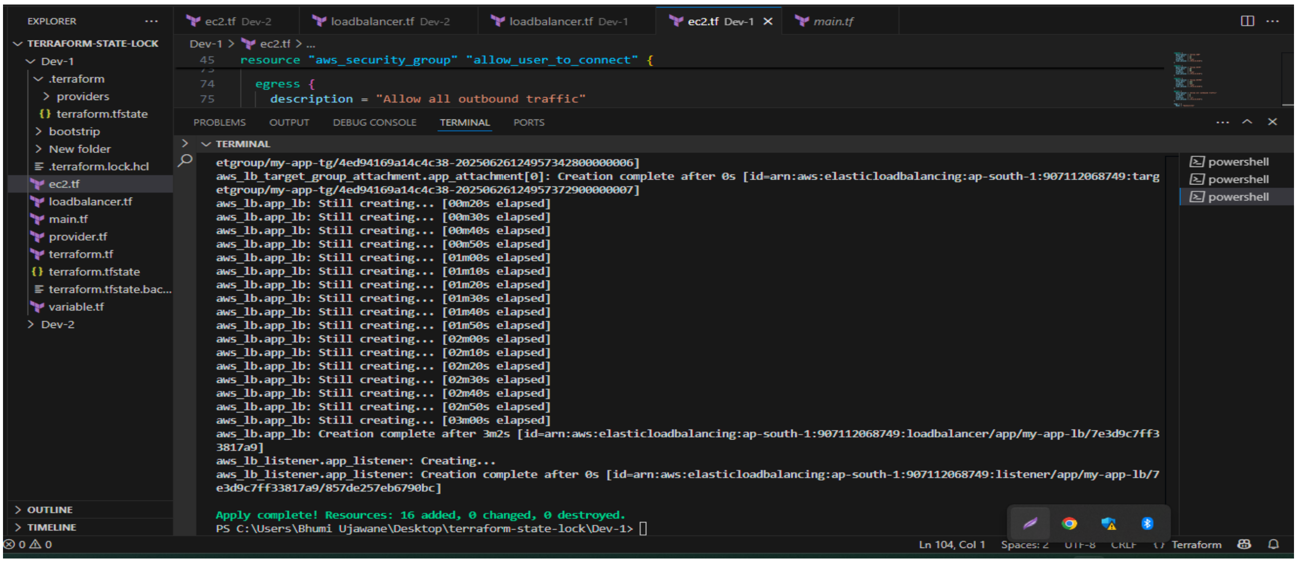


```
resource "aws_security_group" "allow_user_to_connect" {
  egress {
```

```
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + region                               = "ap-south-1"
      + tags                                 = {
          + "Name" = "myTerraformvpc"
        }
      + tags_all                             = {
          + "Name" = "myTerraformvpc"
        }
    }

Plan: 16 to add, 0 to change, 0 to destroy.

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform
apply" now.
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-1> 
```

Terraform Apply

Applies the infrastructure changes. Terraform creates all AWS resources: **VPC, Subnets, Route Tables, Internet Gateway, EC2 Instances, Load Balancer**, etc.

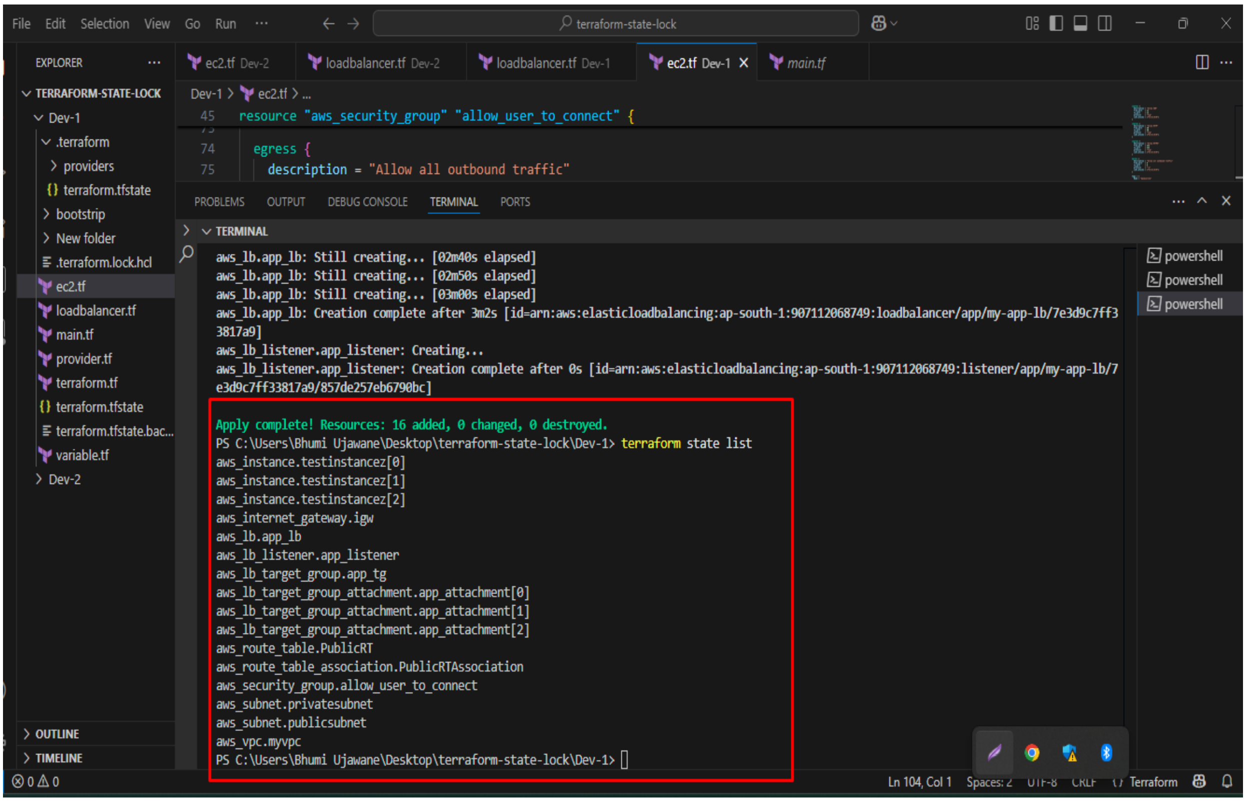


```
resource "aws_security_group" "allow_user_to_connect" {

  egress {
    description = "Allow all outbound traffic"
```

```
etgroup/my-app-tg/4ed94169a14c4c38-20250626124957342800000006]
aws_lb_target_group_attachment.app_attachment[0]: Creation complete after 0s [id=arn:aws:elasticloadbalancing:ap-south-1:907112068749:targ
etgroup/my-app-tg/4ed94169a14c4c38-20250626124957372900000007]
aws_lb.app_lb: Still creating... [00m20s elapsed]
aws_lb.app_lb: Still creating... [00m30s elapsed]
aws_lb.app_lb: Still creating... [00m40s elapsed]
aws_lb.app_lb: Still creating... [00m50s elapsed]
aws_lb.app_lb: Still creating... [01m00s elapsed]
aws_lb.app_lb: Still creating... [01m10s elapsed]
aws_lb.app_lb: Still creating... [01m20s elapsed]
aws_lb.app_lb: Still creating... [01m30s elapsed]
aws_lb.app_lb: Still creating... [01m40s elapsed]
aws_lb.app_lb: Still creating... [01m50s elapsed]
aws_lb.app_lb: Still creating... [02m00s elapsed]
aws_lb.app_lb: Still creating... [02m10s elapsed]
aws_lb.app_lb: Still creating... [02m20s elapsed]
aws_lb.app_lb: Still creating... [02m30s elapsed]
aws_lb.app_lb: Still creating... [02m40s elapsed]
aws_lb.app_lb: Still creating... [02m50s elapsed]
aws_lb.app_lb: Still creating... [03m00s elapsed]
aws_lb.app_lb: Creation complete after 3m2s [id=arn:aws:elasticloadbalancing:ap-south-1:907112068749:loadbalancer/app/my-app-lb/7e3d9c7ff3
3817a9]
aws_lb_listener.app_listener: Creating...
aws_lb_listener.app_listener: Creation complete after 0s [id=arn:aws:elasticloadbalancing:ap-south-1:907112068749:listener/app/my-app-lb/7
e3d9c7ff33817a9/857de257eb6790bc]

Apply complete! Resources: 16 added, 0 changed, 0 destroyed.
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-1> 
```

Terraform state list

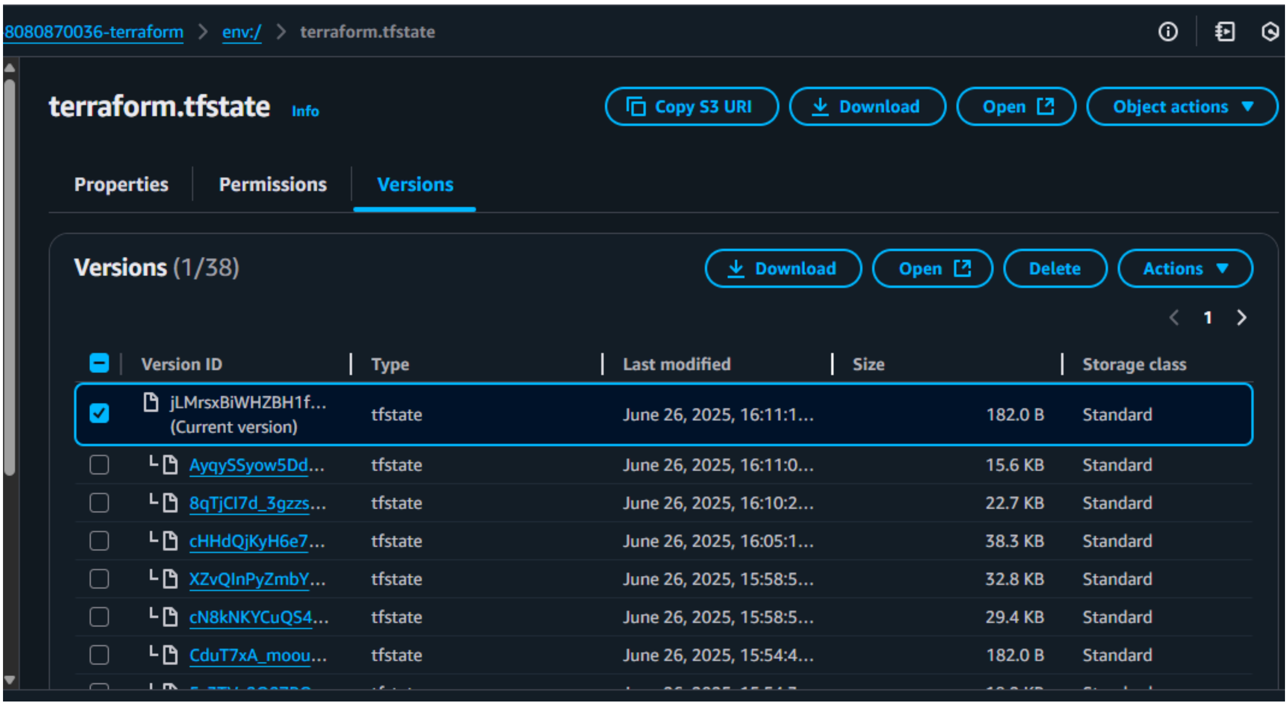
To see what Terraform resources have been created and are currently tracked in your **remote state**



```
resource "aws_security_group" "allow_user_to_connect" {

  egress {
    description = "Allow all outbound traffic"
```

```
aws_lb.app_lb: Still creating... [02m40s elapsed]
aws_lb.app_lb: Still creating... [02m50s elapsed]
aws_lb.app_lb: Still creating... [03m00s elapsed]
aws_lb.app_lb: Creation complete after 3m2s [id=arn:aws:elasticloadbalancing:ap-south-1:907112068749:loadbalancer/app/my-app-lb/7e3d9c7ff3
3817a9]
aws_lb_listener.app_listener: Creating...
aws_lb_listener.app_listener: Creation complete after 0s [id=arn:aws:elasticloadbalancing:ap-south-1:907112068749:listener/app/my-app-lb/7
e3d9c7ff33817a9/857de257eb6790bc]

Apply complete! Resources: 16 added, 0 changed, 0 destroyed.
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-1> terraform state list
aws_instance.testinstancez[0]
aws_instance.testinstancez[1]
aws_instance.testinstancez[2]
aws_internet_gateway.igw
aws_lb.app_lb
aws_lb_listener.app_listener
aws_lb_target_group.app_tg
aws_lb_target_group_attachment.app_attachment[0]
aws_lb_target_group_attachment.app_attachment[1]
aws_lb_target_group_attachment.app_attachment[2]
aws_route_table.PublicRT
aws_route_table_association.PublicRTAssociation
aws_security_group.allow_user_to_connect
aws_subnet.privatesubnet
aws_subnet.publicsubnet
aws_vpc.myvpc
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-1> 
```

Now we go to the s3 and dynamo DB to see the state file and locking.



8080870036-terraform > env:/ > terraform.tfstate

terraform.tfstate Info

Versions (1/38)

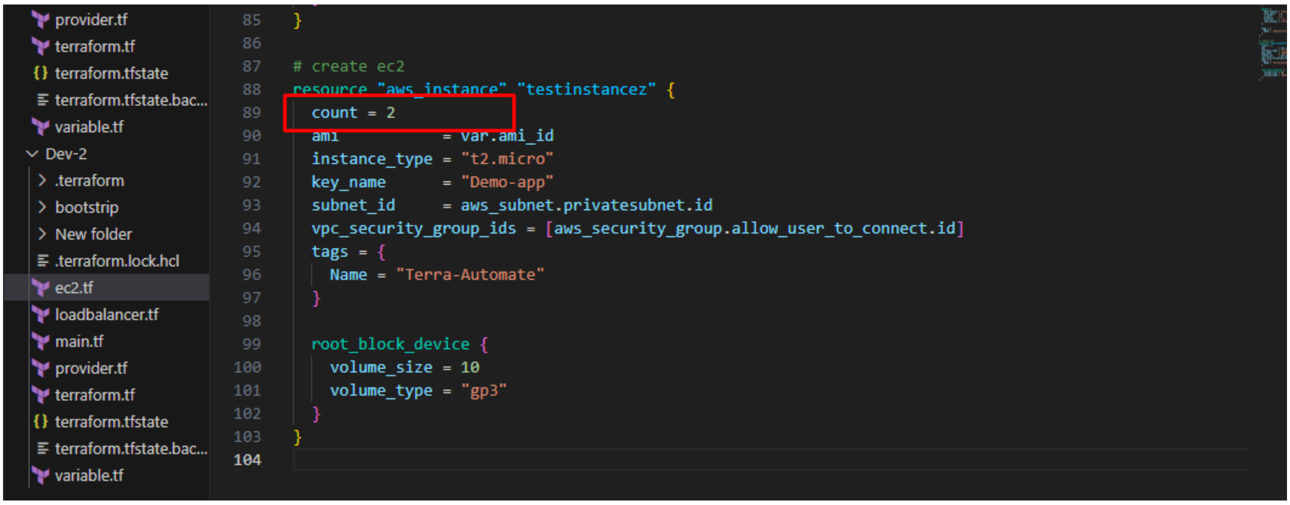
	Version ID	Type	Last modified	Size	Storage class
☑	jLMrsxBiWHZBH1f... (Current version)	tfstate	June 26, 2025, 16:11:1...	182.0 B	Standard
□	AyqySSyow5Dd...	tfstate	June 26, 2025, 16:11:0...	15.6 KB	Standard
□	8qTjCl7d_3gzzs...	tfstate	June 26, 2025, 16:10:2...	22.7 KB	Standard
□	cHHdQjKyH6e7...	tfstate	June 26, 2025, 16:05:1...	38.3 KB	Standard
□	XZvQInPyZmbY...	tfstate	June 26, 2025, 15:58:5...	32.8 KB	Standard
□	cN8kNKYCuQS4...	tfstate	June 26, 2025, 15:58:5...	29.4 KB	Standard
□	CduT7xA_moou...	tfstate	June 26, 2025, 15:54:4...	182.0 B	Standard

Dev-2 User: Testing Remote State Locking with Terraform

To simulate a real-world team collaboration scenario, a second user (**Dev-2**) was introduced. Dev-2 worked in a **separate folder** but used the **same backend** configuration pointing to the shared S3 bucket and DynamoDB table.

Copied all Terraform configuration files from Dev-1 to Dev-2.

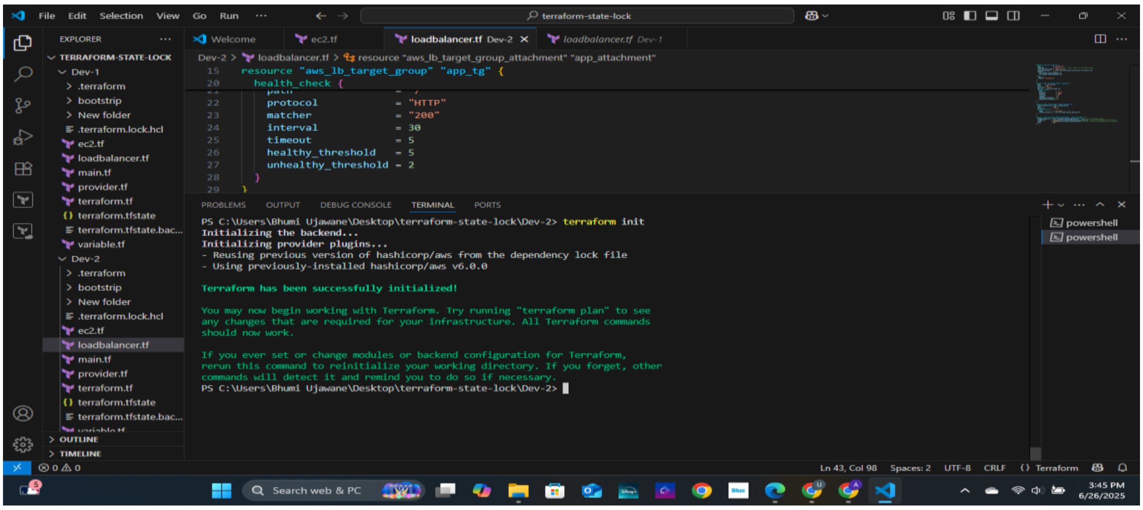
Made a small change (e.g., increased EC2 count to 2).



```
provider.tf
terraform.tf
terraform.tfstate
terraform.tfstate.bac...
variable.tf
Dev-2
  .terraform
  bootstrip
  New folder
  .terraform.lock.hcl
  ec2.tf
  loadbalancer.tf
  main.tf
  provider.tf
  terraform.tf
  terraform.tfstate
  terraform.tfstate.bac...
  variable.tf
```

```
}

# create ec2
resource "aws_instance" "testinstancez" {
  count = 2
  ami           = var.ami_id
  instance_type = "t2.micro"
  key_name      = "Demo-app"
  subnet_id     = aws_subnet.privatesubnet.id
  vpc_security_group_ids = [aws_security_group.allow_user_to_connect.id]
  tags = {
    Name = "Terra-Automate"
  }

  root_block_device {
    volume_size = 10
    volume_type = "gp3"
  }
}
```

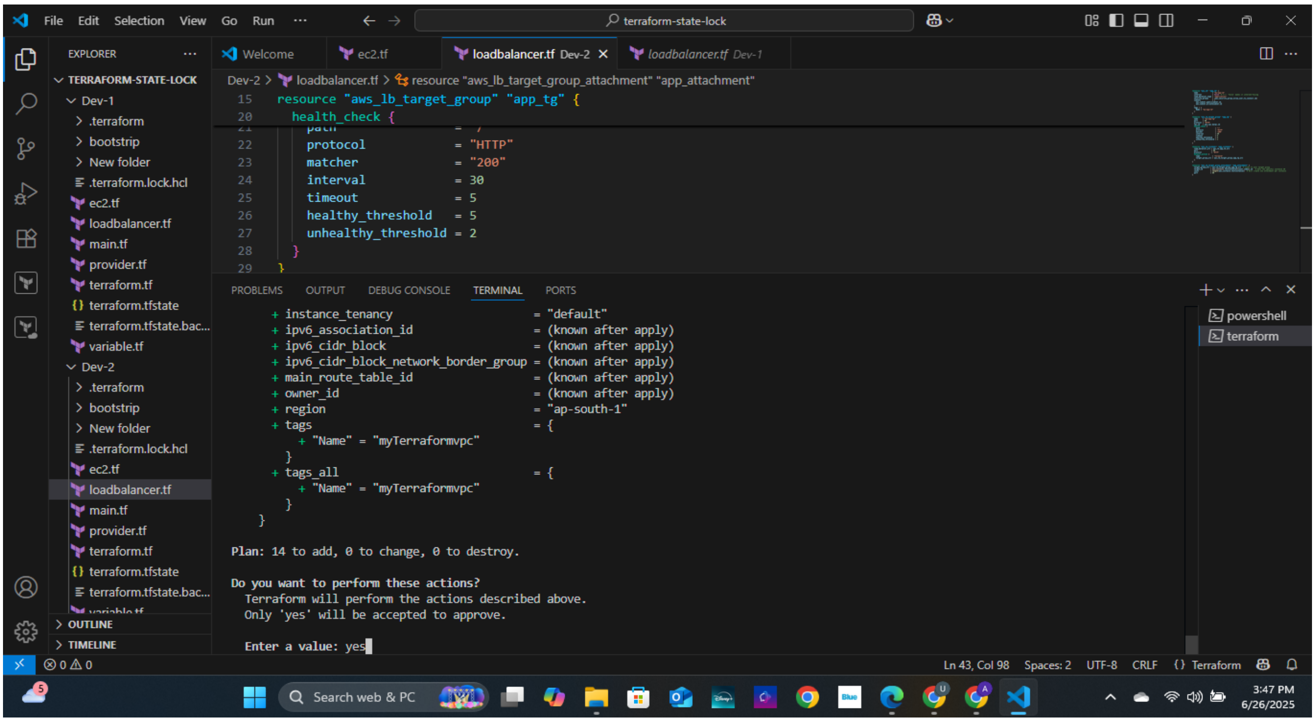
Terraform init **-Dev-2**



```
resource "aws_lb_target_group" "app_tg" {
  health_check {
    protocol            = "HTTP"
    matcher             = "200"
    interval            = 30
    timeout             = 5
    healthy_threshold   = 5
    unhealthy_threshold = 2
  }
}
```

```
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-2> terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.0.0

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.
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-2>
```

Terraform plan **-Dev -2**

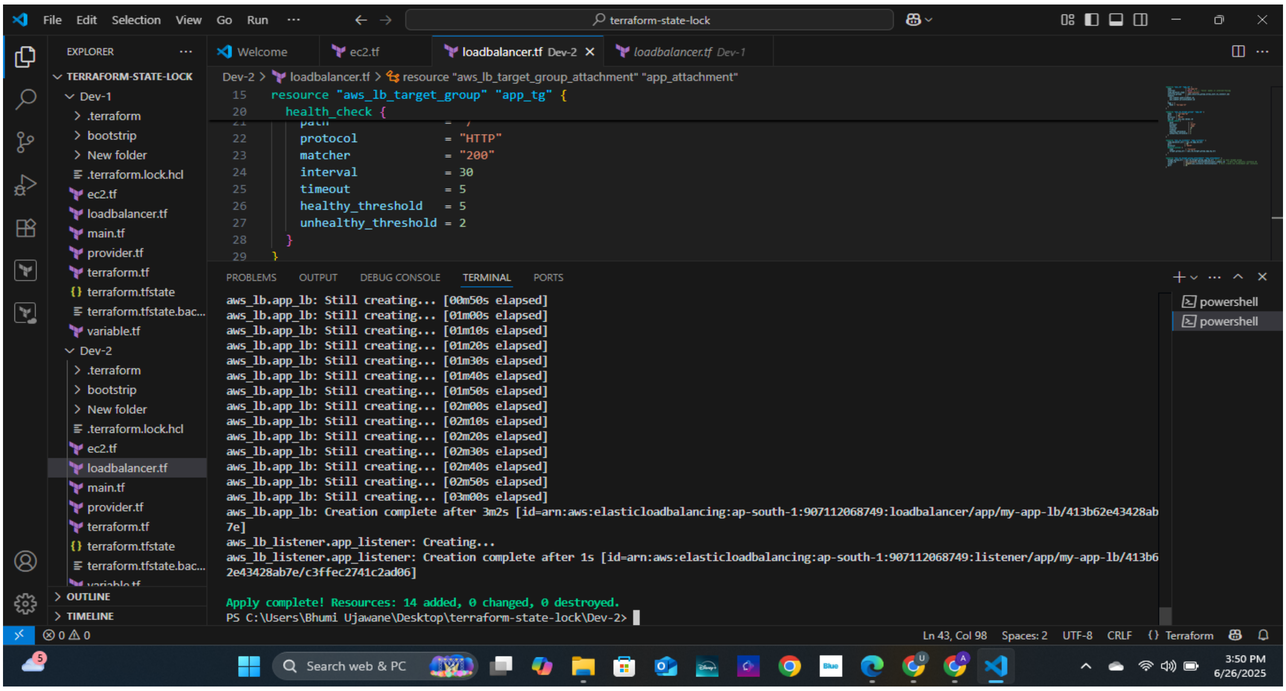


```
resource "aws_lb_target_group" "app_tg" {
  health_check {
    protocol            = "HTTP"
    matcher             = "200"
    interval            = 30
    timeout             = 5
    healthy_threshold   = 5
    unhealthy_threshold = 2
  }
}
```

```
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + region                               = "ap-south-1"
      + tags                                 = {
          + "Name" = "myTerraformvpc"
        }
      + tags_all                             = {
          + "Name" = "myTerraformvpc"
        }
    }

Plan: 14 to add, 0 to change, 0 to destroy.

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

  Enter a value: yes
```

If Dev-1 is already applying changes, Dev-2 will see a **state lock error**. This proves that **state locking works** and prevents conflicting updates.

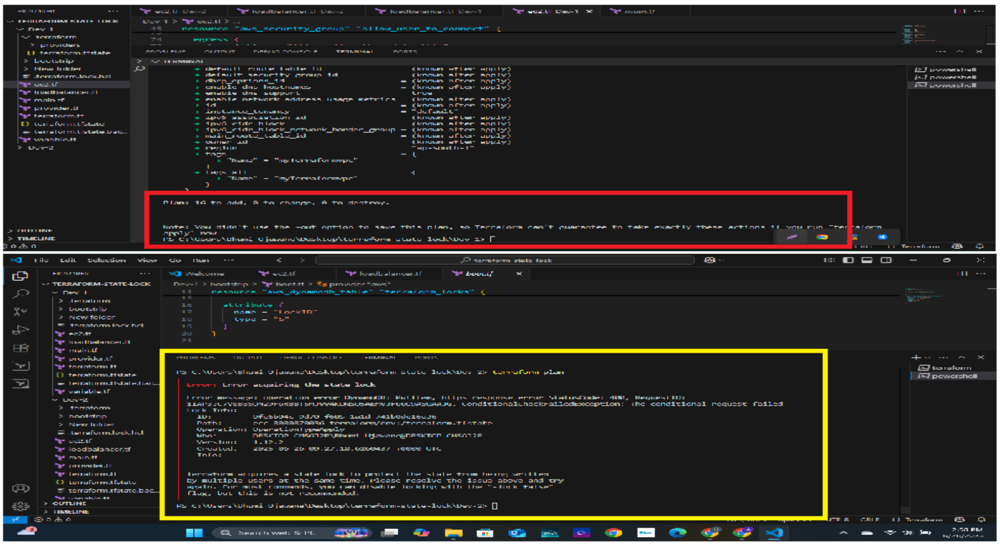
Terraform apply   - **Dev2**



```
resource "aws_lb_target_group" "app_tg" {
  health_check {
    path                = "/"
    protocol            = "HTTP"
    matcher             = "200"
    interval            = 30
    timeout             = 5
    healthy_threshold   = 5
    unhealthy_threshold = 2
  }
}
```

```
aws_lb.app_lb: Still creating... [00m50s elapsed]
aws_lb.app_lb: Still creating... [01m00s elapsed]
aws_lb.app_lb: Still creating... [01m10s elapsed]
aws_lb.app_lb: Still creating... [01m20s elapsed]
aws_lb.app_lb: Still creating... [01m30s elapsed]
aws_lb.app_lb: Still creating... [01m40s elapsed]
aws_lb.app_lb: Still creating... [01m50s elapsed]
aws_lb.app_lb: Still creating... [02m00s elapsed]
aws_lb.app_lb: Still creating... [02m10s elapsed]
aws_lb.app_lb: Still creating... [02m20s elapsed]
aws_lb.app_lb: Still creating... [02m30s elapsed]
aws_lb.app_lb: Still creating... [02m40s elapsed]
aws_lb.app_lb: Still creating... [02m50s elapsed]
aws_lb.app_lb: Still creating... [03m00s elapsed]
aws_lb.app_lb: Creation complete after 3m2s [id=arn:aws:elasticloadbalancing:ap-south-1:907112068749:loadbalancer/app/my-app-lb/413b62e43428ab
7e]
aws_lb_listener.app_listener: Creating...
aws_lb_listener.app_listener: Creation complete after 1s [id=arn:aws:elasticloadbalancing:ap-south-1:907112068749:listener/app/my-app-lb/413b6
2e43428ab7e/c3ffec2741c2ad06]

Apply complete! Resources: 14 added, 0 changed, 0 destroyed.
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-2>
```

Now we will simulate two different users are trying to access the terraform state file and for one user he will be able to access the state file and for others, it will be locked.

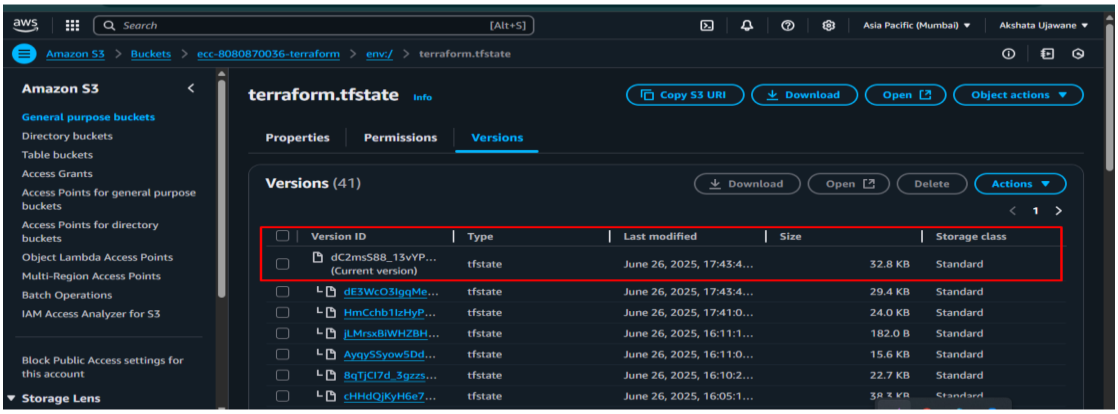


```
Plan: 16 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform
apply" now.
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-1>
```

```
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-2> terraform plan
│ Error: Error acquiring the state lock
│
│ Error message: operation error DynamoDB: PutItem, https response error StatusCode: 400, RequestID:
│ [illegible]. ConditionalCheckFailedException: The conditional request failed
│ Lock Info:
│   ID:        [illegible]
│   Path:      [illegible]/terraform.tfstate
│   Operation: OperationTypeApply
│   Who:       DESKTOP-[illegible]\Bhumi Ujawane@DESKTOP-[illegible]
│   Version:   1.12.2
│   Created:   2025-06-26 [illegible] +0000 UTC
│   Info:
│
│
│ Terraform acquires a state lock to protect the state from being written
│ by multiple users at the same time. Please resolve the issue above and try
│ again. For most commands, you can disable locking with the "-lock=false"
│ flag, but this is not recommended.
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-2>
```

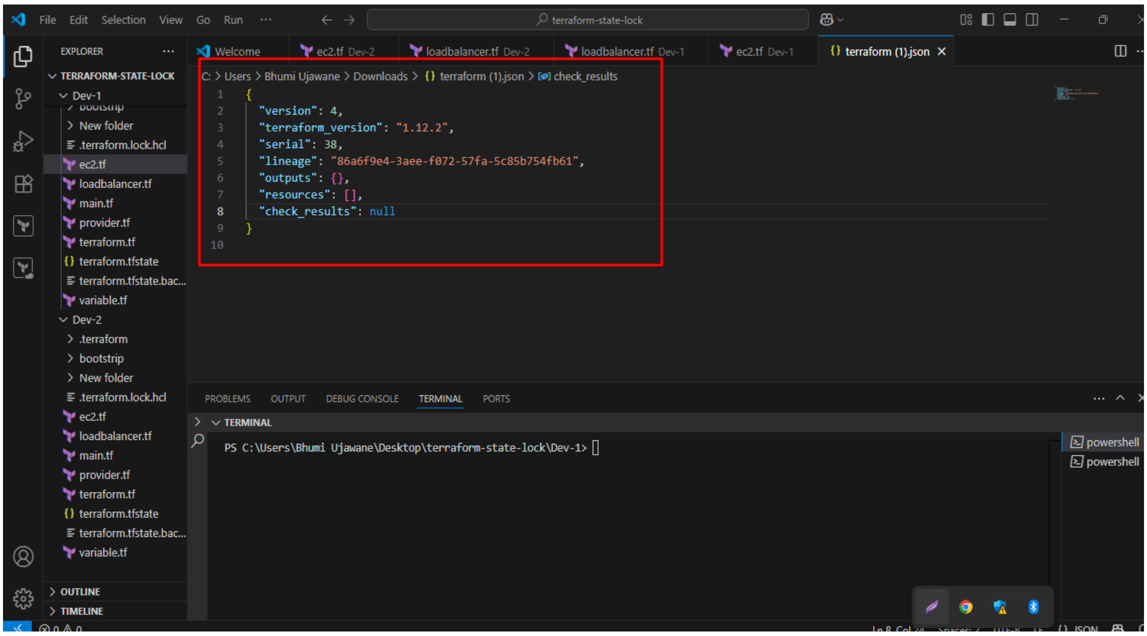
As we can see, in our case the user 1 ran command first, the command worked, and it succeeded. However, user 2 was not allowed to run because the state was locked.

Now we go to the s3 and dynamo DB to see the state file and locking.



Version ID	Type	Last modified	Size	Storage class
dC2msS88_13vYP... (Current version)	tfstate	June 26, 2025, 17:43:4...	32.8 KB	Standard
dE3WcO3IgqMe...	tfstate	June 26, 2025, 17:43:4...	29.4 KB	Standard
HmCchb1IzHyP...	tfstate	June 26, 2025, 17:41:0...	24.0 KB	Standard
jLMrsxBiWHZBH...	tfstate	June 26, 2025, 16:11:1...	182.0 B	Standard
AyqySSyow5Dd...	tfstate	June 26, 2025, 16:11:0...	15.6 KB	Standard
8qTjCl7d_3gzzs...	tfstate	June 26, 2025, 16:10:2...	22.7 KB	Standard
cHHdQjKyH6e7...	tfstate	June 26, 2025, 16:05:1...	38.3 KB	Standard

the S3 bucket started to take versions of the state



```json
{
  "version": 4,
  "terraform_version": "1.12.2",
  "serial": 38,
  "lineage": "86a6f9e4-3aee-f072-57fa-5c85b754fb61",
  "outputs": {},
  "resources": [],
  "check_results": null
}
```

```
PS C:\Users\Bhumi Ujawane\Desktop\terraform-state-lock\Dev-1>
```

In the end, don’t forget to run the terraform destroy command. Also, as we use the prevent destroy argument while creating the S3 bucket, the S3 bucket, and DynamoDB table will not be deleted. So you can delete them from the AWS console.