

1) Execute the script shown in video.

terraform provisioners:

Terraform provisioners allow you to execute scripts or actions on a resource after it has been created or destroyed. Provisioners are useful for tasks such as configuring servers, installing packages, or running custom scripts.

Here are the key types of provisioners in Terraform.

File Provisioner : The File provisioner copies files or directories from your local machine to a newly created instance.

This is script I am executed.

```
provider "aws" {
  region      = "us-west-1"
  access_key  = "AKIAW3MD7G7LBGTD56E7"
  secret_key  = "hvPq1uTyzVWzIBx1vfEORKKo5kVEuQ8YBzN80AsS"
}

resource "aws_instance" "Ec2" {
  ami                  = "ami-0175bdd48fdb0973b"
  instance_type       = "t2.micro"
  key_name             = "Ncalifornia"
  vpc_security_group_ids = [aws_security_group.main.id]

  provisioner "file" {
    source      = "C:/Users/ramee/Desktop/Terraform/.terraform/Ncalifornia.pem"
    destination = "/home/ec2-user/Ncalifornia.pem"
  }

  connection {
    type      = "ssh"
    host      = self.public_ip
    user      = "ec2-user"
    private_key = file("${path.module}/Ncalifornia.pem")
    timeout   = "4m"
  }

  tags = {
    Name = "My_EC2_Instance"
  }
}

resource "aws_security_group" "main" {
  name        = "allow_ssh"
  description = "Allow SSH inbound traffic"
```

```

ingress {
  from_port = 22
  to_port   = 22
  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"]
}
}

```

Here this will execute successful.

The screenshot shows a Visual Studio Code editor with a Terraform configuration file named `main.tf`. The configuration defines an `aws_instance` resource named `Ec2` with a `provisioner "file"` block and a `connection` block. The terminal output shows the successful execution of the Terraform command, including the creation of the `aws_instance.Ec2` resource and the application of the configuration.

```

7 resource "aws_instance" "Ec2" {
13   provisioner "file" {
14     source = "C:/Users/ramee/Desktop/Terraform/.terraform/Ncalifornia.pem"
15     destination = "/home/ec2-user/Ncalifornia.pem"
16   }
17
18   connection {
19     type = "ssh"
20     host = self.public_ip
21     user = "ec2-user"
22     private_key = file("${path.module}/Ncalifornia.pem")
23     timeout = "4m"
24   }

```

```

Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

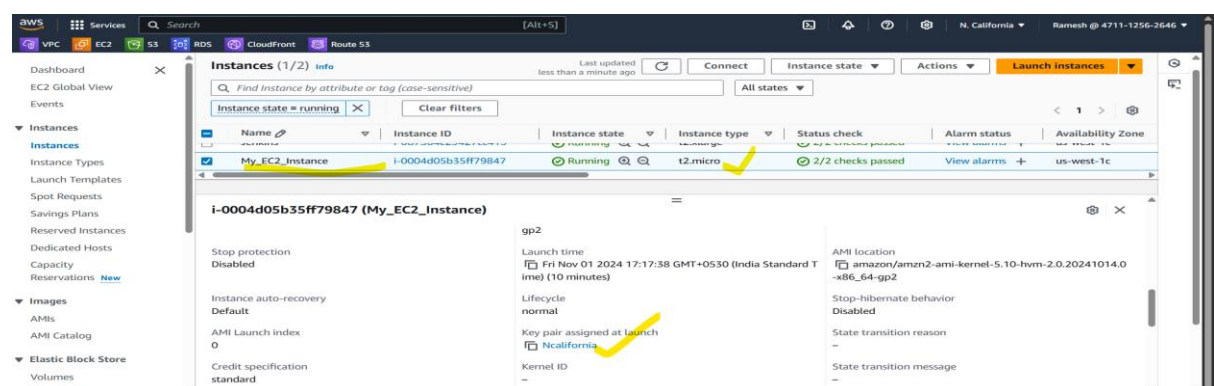
aws_security_group.main: Creating...
aws_security_group.main: Creation complete after 7s [id=sg-0b109b2ac8db009c5] ✓
aws_instance.Ec2: Creating...
aws_instance.Ec2: Still creating... [10s elapsed]
aws_instance.Ec2: Provisioning with 'file'...
aws_instance.Ec2: Still creating... [20s elapsed]
aws_instance.Ec2: Creation complete after 25s [id=i-0004d05b35ff79847] ✓

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
PS C:\Users\ramee\Desktop\Terraform\.terraform>

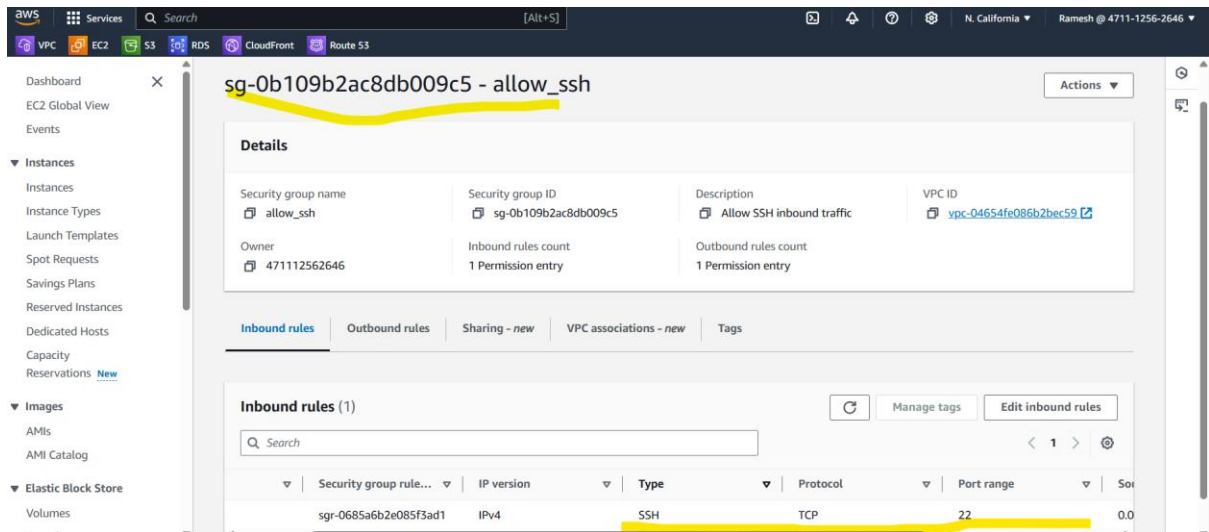
```

Now we check the file is there or not in instance and also we have to check the security groups.

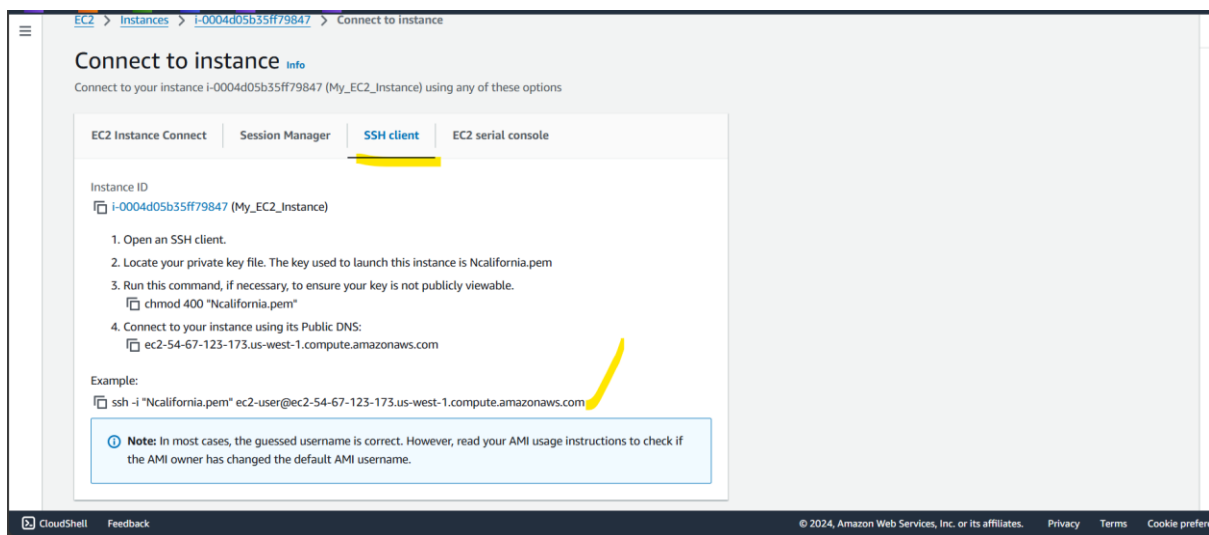
My Ec2 instance is created with Ncalifornia.pem and t2.micro instance type.



My security group is created



To check we have to copy below example.



Now our file is there.

```
PS C:\Users\rname\Downloads> ssh -i "Ncalifornia.pem" ec2-user@ec2-54-67-123-173.us-west-1.compute.amazonaws.com
The authenticity of host 'ec2-54-67-123-173.us-west-1.compute.amazonaws.com (54.67.123.173)' can't be established.
ED25519 public key fingerprint is SHA256:jyn9LXDQzCuMY2qaKL5BoHm20E/AJATzyBGLKhfdQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-67-123-173.us-west-1.compute.amazonaws.com' (ED25519) to the list of known hosts.
```

```
#
#####
#      Amazon Linux 2
#####
#      AL2 End of Life is 2025-06-30.
#####
#
V--1-->
A newer version of Amazon Linux is available!

Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

9 package(s) needed for security, out of 12 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-1-151 ~]$ ls
Ncalifornia.pem
[ec2-user@ip-172-31-1-151 ~]$ |
```

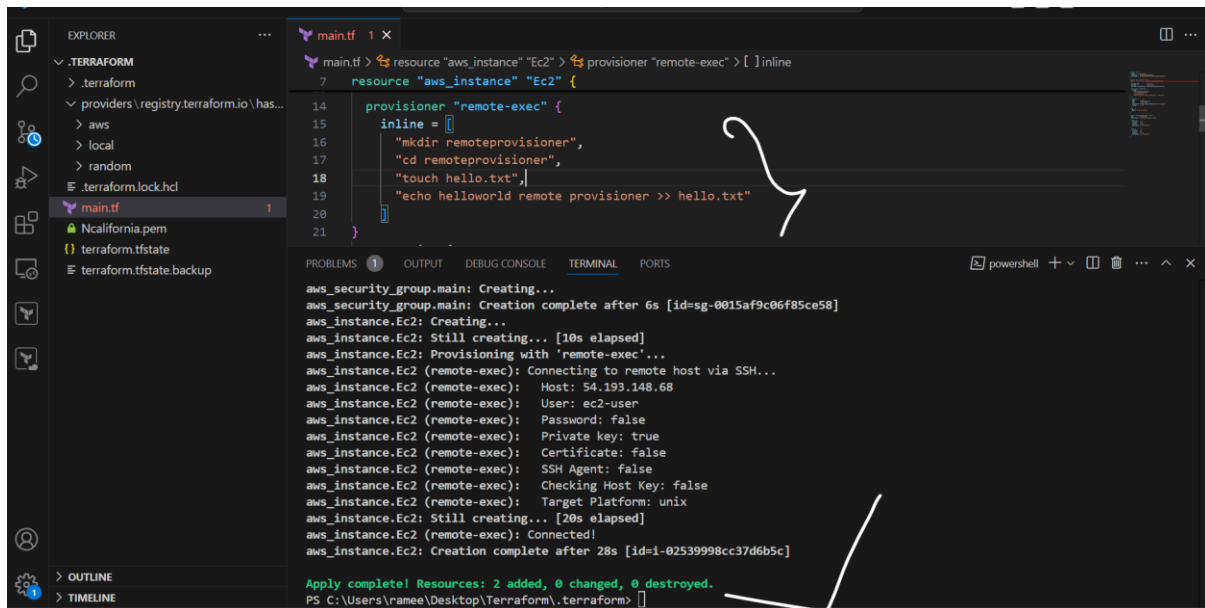
Done.

remote-exec provisioner:

As the name suggests remote-exec it is always going to work on the remote machine. With the help of the remote-exec you can specify the commands of shell scripts that want to execute on the remote machine.

Using the remote-exec provisioner, I am creating a directory. Within this directory, I am creating a hello.txt file and redirecting content into it.

Just here I am changing the script only provisioner block.

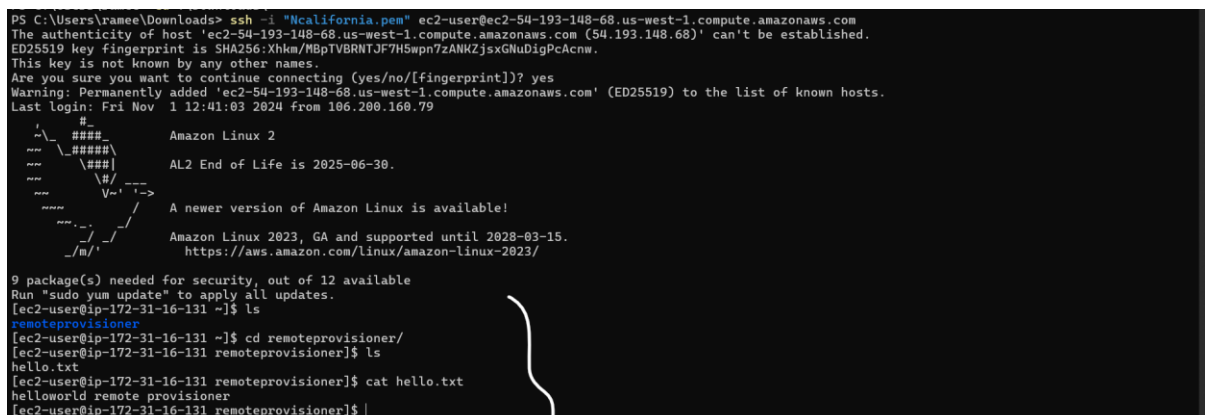


The screenshot shows a VS Code editor with a Terraform configuration file named `main.tf`. The configuration defines an `aws_instance` resource named `Ec2` using the `remote-exec` provisioner. The `inline` block contains the following commands:

```
mkdir remoteprovisioner,
cd remoteprovisioner,
touch hello.txt,
echo helloworld remote provisioner >> hello.txt
```

The terminal output shows the successful execution of the Terraform plan and apply. The `aws_instance.Ec2` resource is created, and the `remote-exec` provisioner is used to connect to the remote host via SSH. The output shows the host IP, user, and the successful execution of the commands defined in the `inline` block.

Now we want to check server.



The screenshot shows a terminal session where an SSH connection is established to an Amazon EC2 instance. The user runs the following commands:

```
ssh -i "Ncalifornia.pem" ec2-user@ec2-54-193-148-68.us-west-1.compute.amazonaws.com
ls
cd remoteprovisioner/
ls
touch hello.txt
echo helloworld remote provisioner >> hello.txt
cat hello.txt
```

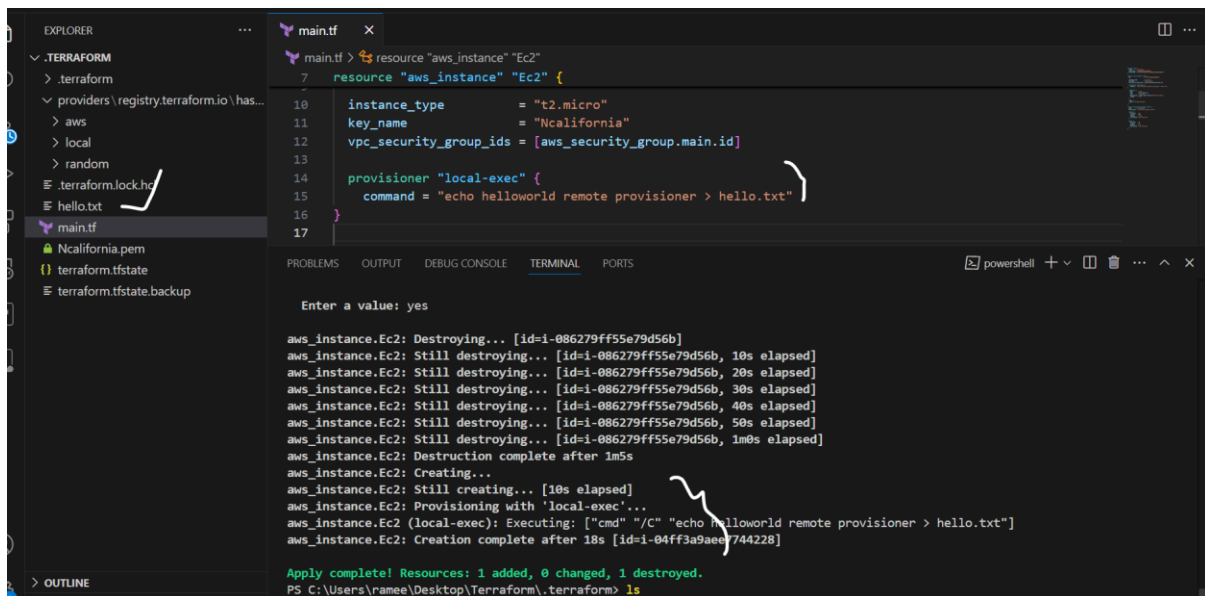
The output shows the successful execution of these commands, confirming that the directory `remoteprovisioner` was created and the file `hello.txt` was created and updated with the content `helloworld remote provisioner`.

Done.

local-exec provisioner:

The next provisioner we are gonna talk about is local-exec provisioner. Basically, this provisioner is used when you want to perform some tasks onto your local machine where you have installed the terraform.

So local-exec provisioner is never used to perform any kind task on the remote machine. It will always be used to perform local operations onto your local machine.

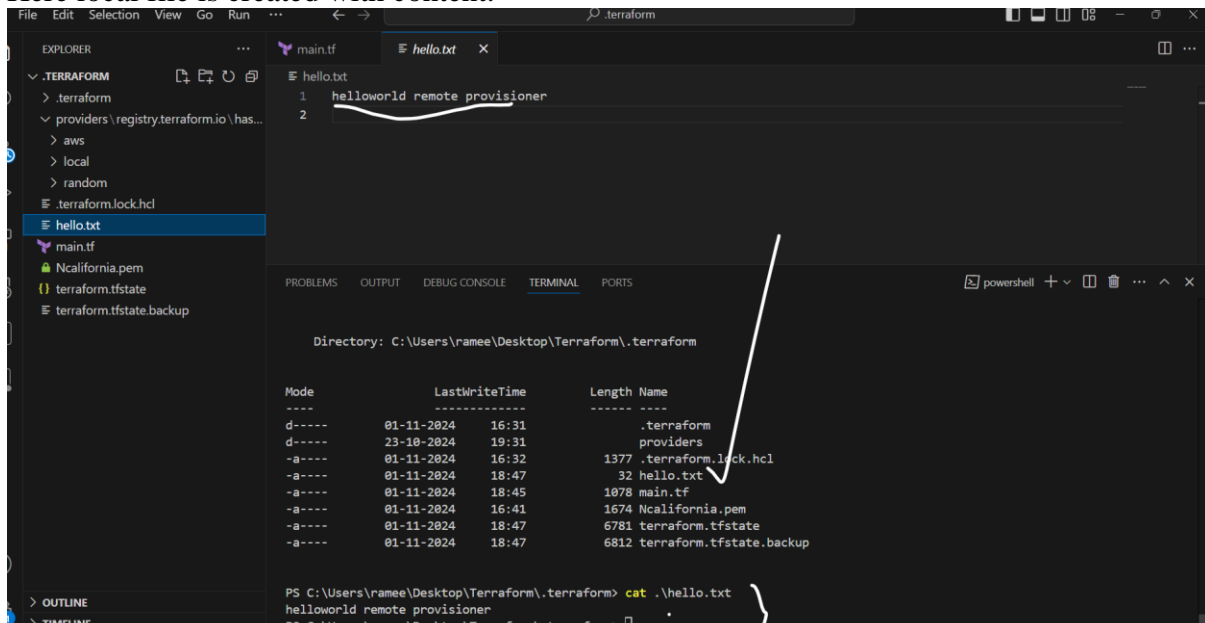


```
main.tf
7 resource "aws_instance" "Ec2" {
8   instance_type     = "t2.micro"
9   key_name           = "Ncalifornia"
10  vpc_security_group_ids = [aws_security_group.main.id]
11
12  provisioner "local-exec" {
13    command = "echo helloworld remote provisioner > hello.txt"
14  }
15 }
16
17
```

```
Enter a value: yes
aws_instance.Ec2: Destroying... [id=i-086279ff55e79d56b, 10s elapsed]
aws_instance.Ec2: Still destroying... [id=i-086279ff55e79d56b, 20s elapsed]
aws_instance.Ec2: Still destroying... [id=i-086279ff55e79d56b, 30s elapsed]
aws_instance.Ec2: Still destroying... [id=i-086279ff55e79d56b, 40s elapsed]
aws_instance.Ec2: Still destroying... [id=i-086279ff55e79d56b, 50s elapsed]
aws_instance.Ec2: Destruction complete after 1m5s
aws_instance.Ec2: Creating...
aws_instance.Ec2: Still creating... [10s elapsed]
aws_instance.Ec2: Provisioning with 'local-exec'...
aws_instance.Ec2 (local-exec): Executing: ["cmd", "/C", "echo helloworld remote provisioner > hello.txt"]
aws_instance.Ec2: Creation complete after 18s [id=i-04ff3a9aee744228]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
PS C:\Users\ramee\Desktop\Terraform\.terraform> ls
```

Here local file is created with content.



```
hello.txt
1 helloworld remote provisioner
2
```

```
Directory: C:\Users\ramee\Desktop\Terraform\.terraform

Mode                LastWriteTime         Length Name
----                -
d-----          01-11-2024   16:31           .terraform
d-----          23-10-2024   19:31         providers
-a-----          01-11-2024   16:32       1377 .terraform.lock.hcl
-a-----          01-11-2024   18:47         32 hello.txt
-a-----          01-11-2024   18:45       1078 main.tf
-a-----          01-11-2024   16:41       1674 Ncalifornia.pem
-a-----          01-11-2024   18:47       6781 terraform.tfstate
-a-----          01-11-2024   18:47       6812 terraform.tfstate.backup

PS C:\Users\ramee\Desktop\Terraform\.terraform> cat .\hello.txt
helloworld remote provisioner
PS C:\Users\ramee\Desktop\Terraform\.terraform>
```

Done.

Terraform provisioner behaviours:

1) Default Behavior:

By default, provisioners run when a resource is created, allowing us to set it up as needed right away.

2) Destroy Behavior:

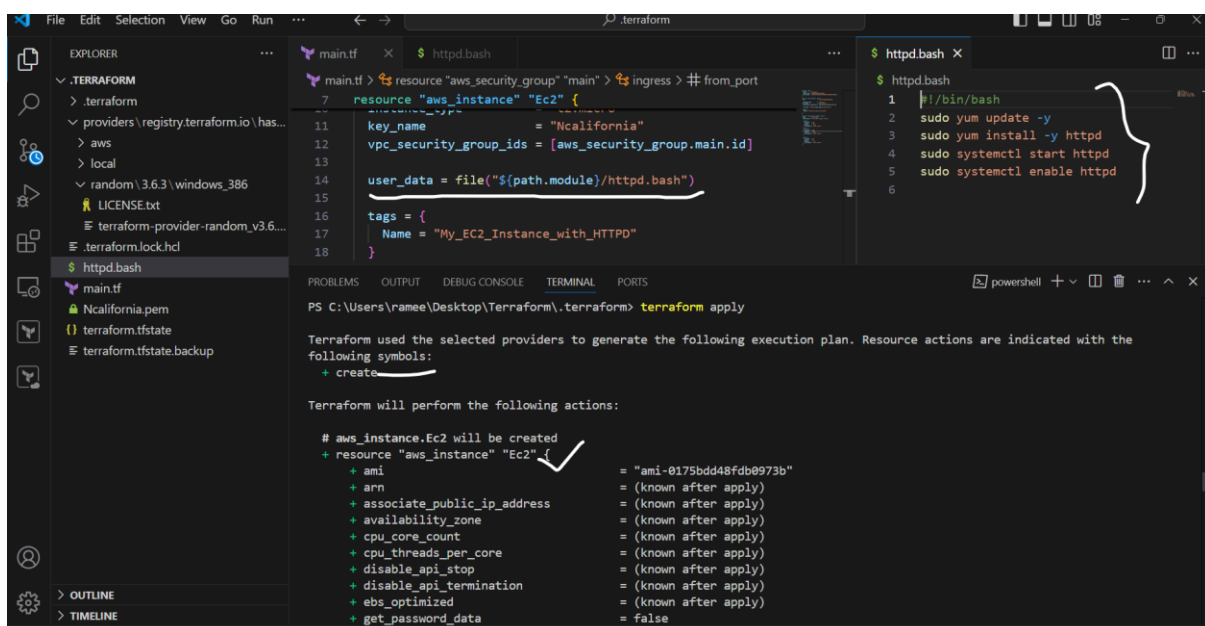
when = destroy allows us to run specific tasks when a resource is being deleted, which helps with cleanup or backup.

3) on_failure = fail --> to create the resource if the script gets failed. (But terraform will mark the resource as tainted)

4) On_fail = continue --> to create the resource and ignore the changes.

2) Create one ec2 instance with httpd installed using terraform script.

- Terraform configuration file (main.tf) uses the httpd.bash script to automate the setup of an EC2 instance with Apache HTTPD installed.
- The user_data = file("\${path.module}/httpd.bash") line in main.tf calls httpd.bash as a user data script, which is executed on instance launch to install and start the HTTP server, making the instance ready to serve HTTP requests automatically.



The screenshot displays the Visual Studio Code interface with a Terraform project. The Explorer pane on the left shows the file structure, including `main.tf` and `httpd.bash`. The main editor shows the `main.tf` file with the following configuration:

```
resource "aws_instance" "Ec2" {  
  key_name       = "Ncalifornia"  
  vpc_security_group_ids = [aws_security_group.main.id]  
  user_data      = file("${path.module}/httpd.bash")  
  tags = {  
    Name = "My_EC2_Instance_with_HTTPD"  
  }  
}
```

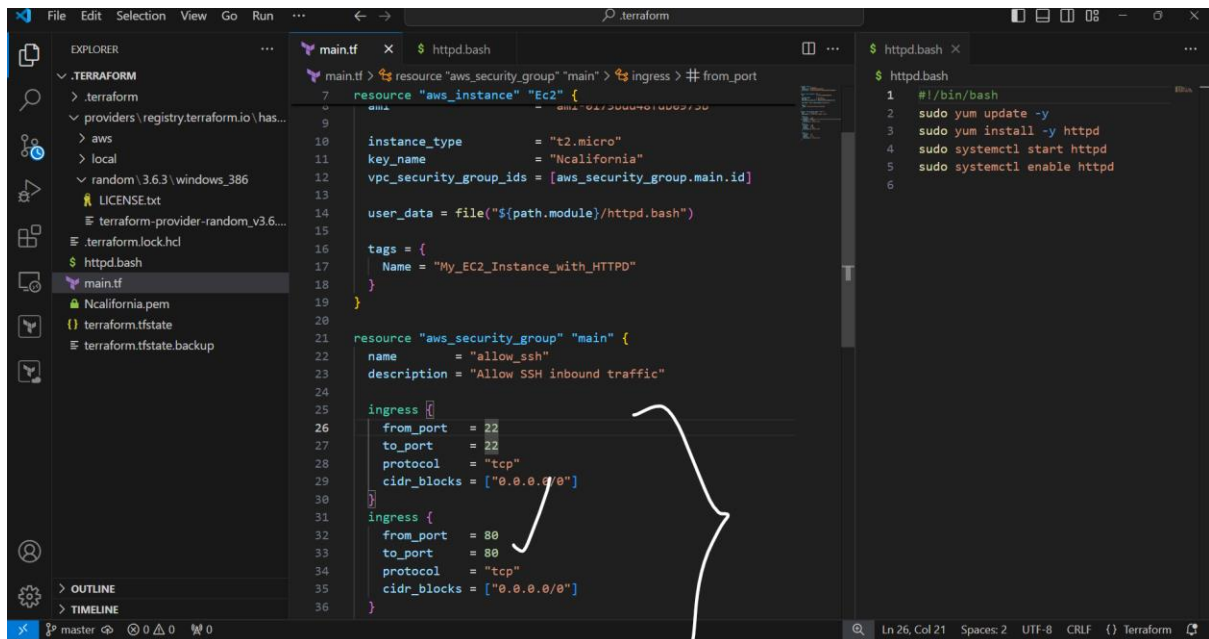
The `httpd.bash` script is shown in a separate window, containing the following commands:

```
#!/bin/bash  
1 sudo yum update -y  
2 sudo yum install -y httpd  
3 sudo systemctl start httpd  
4 sudo systemctl enable httpd
```

The Output pane at the bottom shows the execution of `terraform apply`. It indicates that Terraform will perform the following actions:

```
# aws_instance.Ec2 will be created  
+ resource "aws_instance" "Ec2" {  
  + ami              = "ami-0175bdd48fdb0973b"  
  + arn              = (known after apply)  
  + associate_public_ip_address = (known after apply)  
  + availability_zone = (known after apply)  
  + cpu_core_count   = (known after apply)  
  + cpu_threads_per_core = (known after apply)  
  + disable_api_stop  = (known after apply)  
  + disable_api_termination = (known after apply)  
  + ebs_optimized     = (known after apply)  
  + get_password_data = false
```

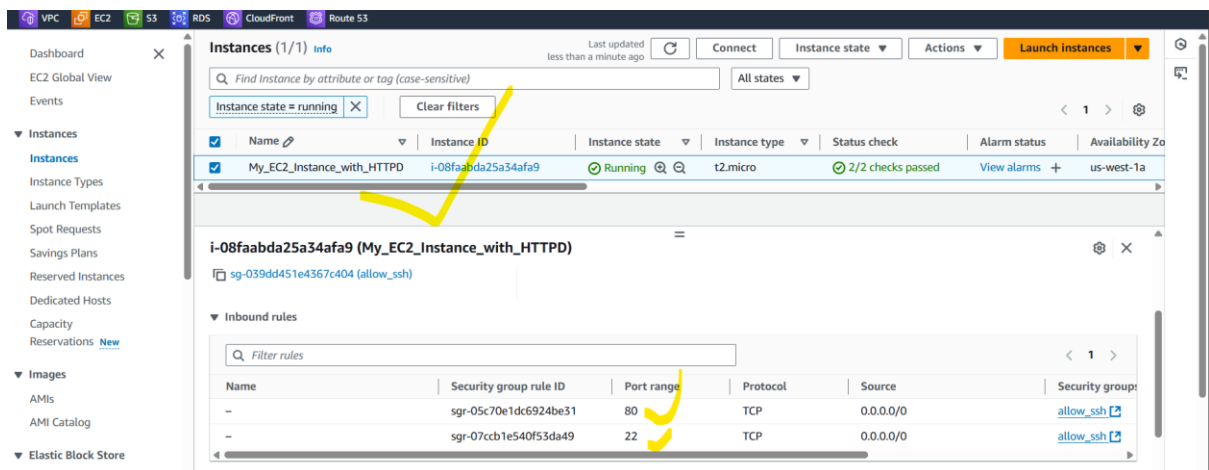
Here I am given the inbound rules port 22 and 80.



```
main.tf
7 resource "aws_instance" "ec2" {
8   # ...
9   instance_type = "t2.micro"
10  key_name      = "Ncalifornia"
11  vpc_security_group_ids = [aws_security_group.main.id]
12
13  user_data = file("${path.module}/httpd.bash")
14
15  tags = {
16    Name = "My_EC2_Instance_with_HTTPD"
17  }
18 }
19
20 resource "aws_security_group" "main" {
21   name        = "allow_ssh"
22   description = "Allow SSH inbound traffic"
23
24   ingress {
25     from_port = 22
26     to_port   = 22
27     protocol  = "tcp"
28     cidr_blocks = ["0.0.0.0/0"]
29   }
30
31   ingress {
32     from_port = 80
33     to_port   = 80
34     protocol  = "tcp"
35     cidr_blocks = ["0.0.0.0/0"]
36   }
37 }
```

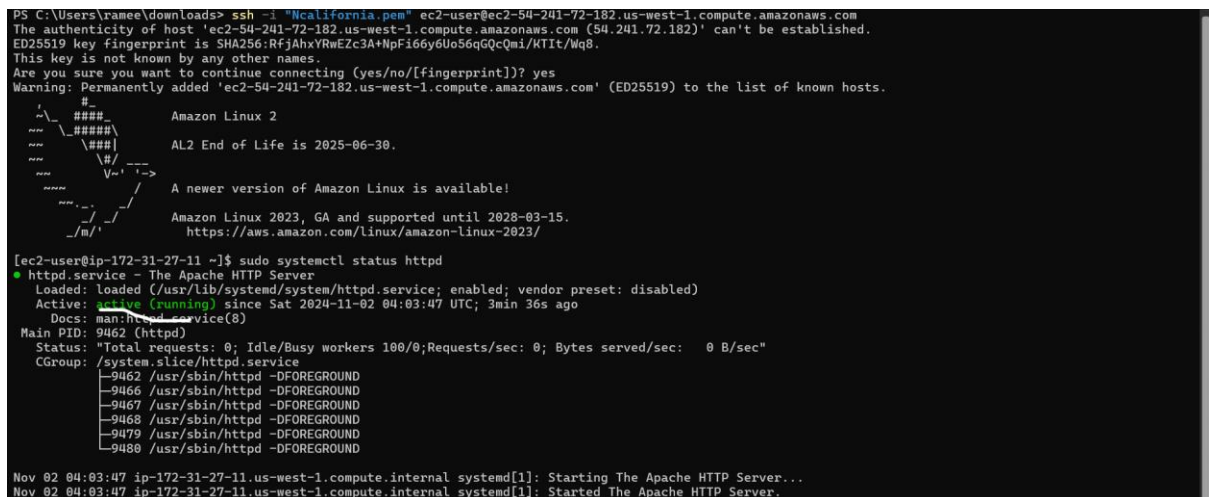
```
httpd.bash
1 #!/bin/bash
2 sudo yum update -y
3 sudo yum install -y httpd
4 sudo systemctl start httpd
5 sudo systemctl enable httpd
6
```

Here our instance is created with port 22 and 80.



Name	Security group rule ID	Port range	Protocol	Source	Security group
-	sgr-05c70e1dc6924be31	80	TCP	0.0.0.0/0	allow_ssh
-	sgr-07ccb1e540f53da49	22	TCP	0.0.0.0/0	allow_ssh

Our httpd server status is Running.



```
PS C:\Users\ramee\downloads> ssh -i "Ncalifornia.pem" ec2-user@ec2-54-241-72-182.us-west-1.compute.amazonaws.com
The authenticity of host 'ec2-54-241-72-182.us-west-1.compute.amazonaws.com (54.241.72.182)' can't be established.
ED25519 key fingerprint is SHA256:RfjAhxYRwEZc3A+NpF166y6Uo56qGQcQmi/KTIt/Wq8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-241-72-182.us-west-1.compute.amazonaws.com' (ED25519) to the list of known hosts.

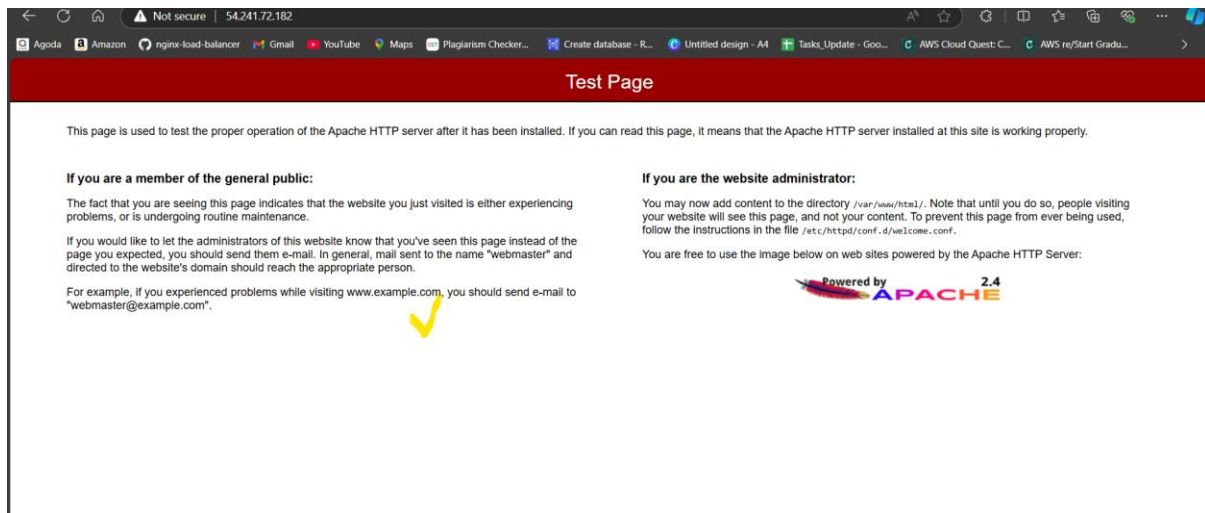
Amazon Linux 2
AL2 End of Life is 2025-06-30.

A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/Linux/amazon-linux-2023/

[ec2-user@ip-172-31-27-11 ~]$ sudo systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Sat 2024-11-02 04:03:47 UTC; 3min 36s ago
     Docs: man:httpd.service(8)
   Main PID: 9462 (httpd)
    Status: "Total requests: 0; Idle/Busy workers 100/0; Requests/sec: 0; Bytes served/sec: 0 B/sec"
   CGroup: /system.slice/httpd.service
           └─9462 /usr/sbin/httpd -DFOREGROUND
             └─9466 /usr/sbin/httpd -DFOREGROUND
               └─9467 /usr/sbin/httpd -DFOREGROUND
                 └─9468 /usr/sbin/httpd -DFOREGROUND
                   └─9479 /usr/sbin/httpd -DFOREGROUND
                     └─9480 /usr/sbin/httpd -DFOREGROUND

Nov 02 04:03:47 ip-172-31-27-11.us-west-1.compute.internal systemd[1]: Starting The Apache HTTP Server...
Nov 02 04:03:47 ip-172-31-27-11.us-west-1.compute.internal systemd[1]: Started The Apache HTTP Server.
```


Our httpd page also accessible .



Done.

Terraform state file

The Terraform state file is essentially a blueprint that tracks all the infrastructure Terraform manages. It stores information about each resource, including its unique ID, configuration metadata, and the actual state of the infrastructure. This allows Terraform to efficiently check for any differences between what's currently deployed and the desired configuration in the code.

Terraform Remote State and State Locking

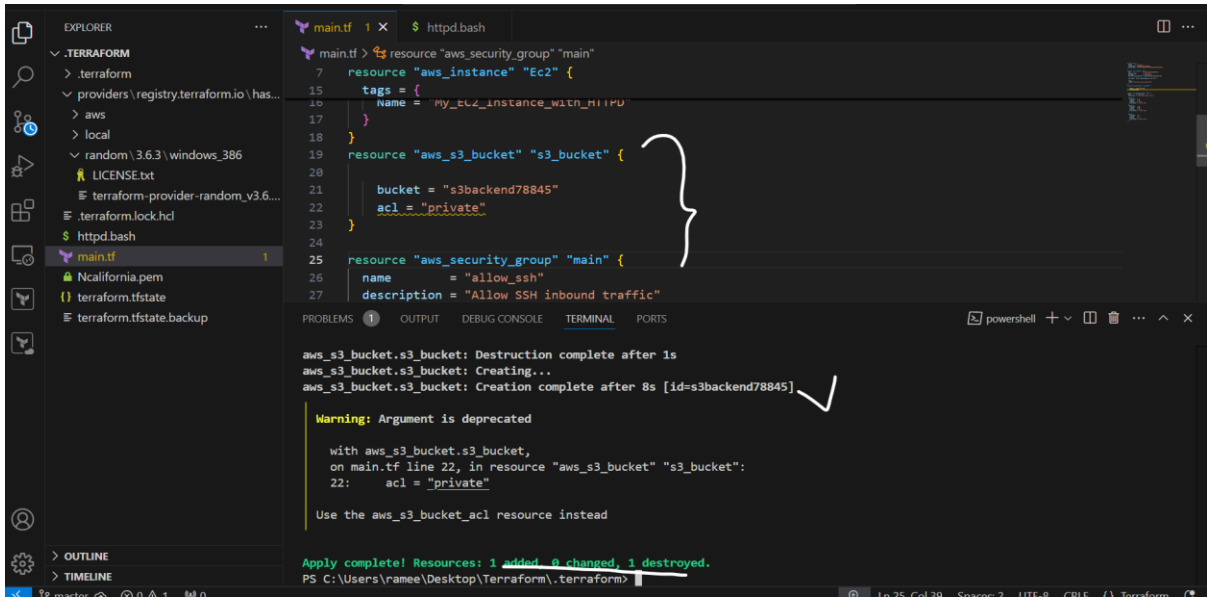
Storing Terraform Files

While you can store Terraform configuration and state files in GitHub, it's not the best practice. Instead, use services like **Amazon S3** or **Terraform Cloud** to keep your files secure.

Benefits of Remote State:

- **Central Access:** Team members can work from the same state file.
- **Better Security:** Services like S3 provide encryption and access controls.
- **Versioning:** Easily track changes and revert to previous states if needed.

1. Setup s3 as backend to the task 3.
Here using this script I am created one s3 bucket.



The screenshot shows a VS Code editor with a Terraform script in `main.tf` and its execution output in the terminal. The script defines an AWS S3 bucket named `s3_bucket` with the following configuration:

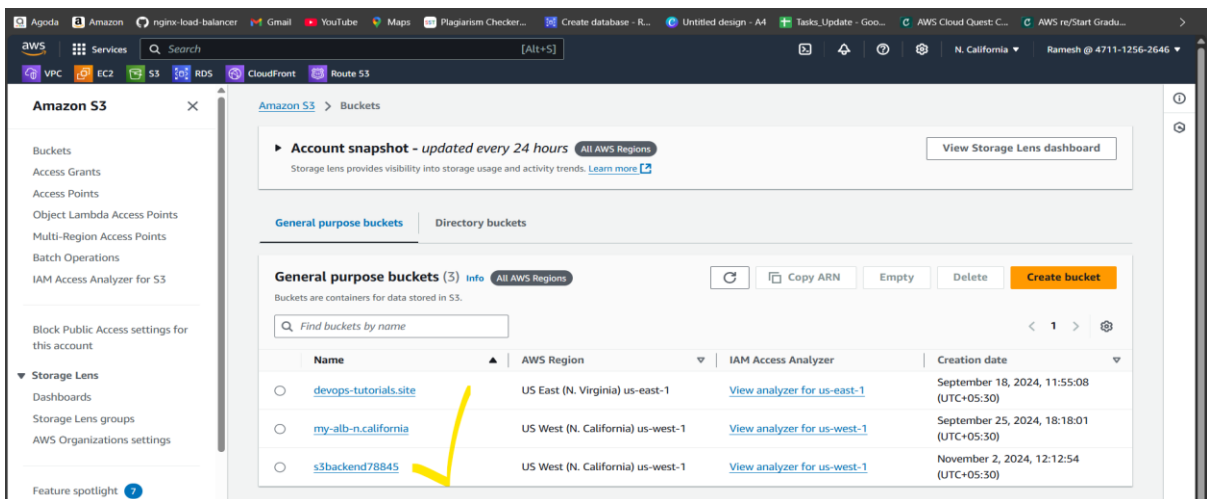
```
resource "aws_s3_bucket" "s3_bucket" {
  bucket = "s3backend78845"
  acl    = "private"
}
```

The terminal output shows the successful creation of the bucket:

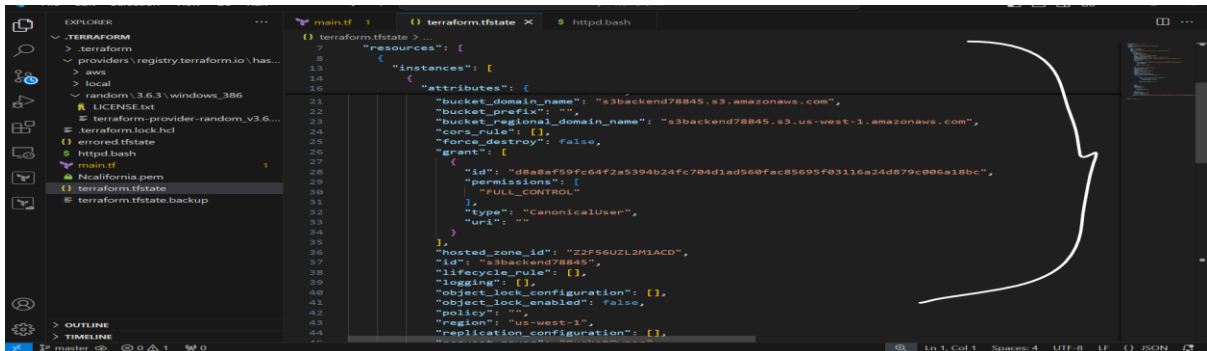
```
aws_s3_bucket.s3_bucket: Destruction complete after 1s
aws_s3_bucket.s3_bucket: Creating...
aws_s3_bucket.s3_bucket: Creation complete after 8s [id=s3backend78845]
Warning: Argument is deprecated
with aws_s3_bucket.s3_bucket,
on main.tf line 22, in resource "aws_s3_bucket" "s3_bucket":
22:   acl = "private"
Use the aws_s3_bucket_acl resource instead
Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

Now we have to check in aws console.

Here our s3 bucket is created.

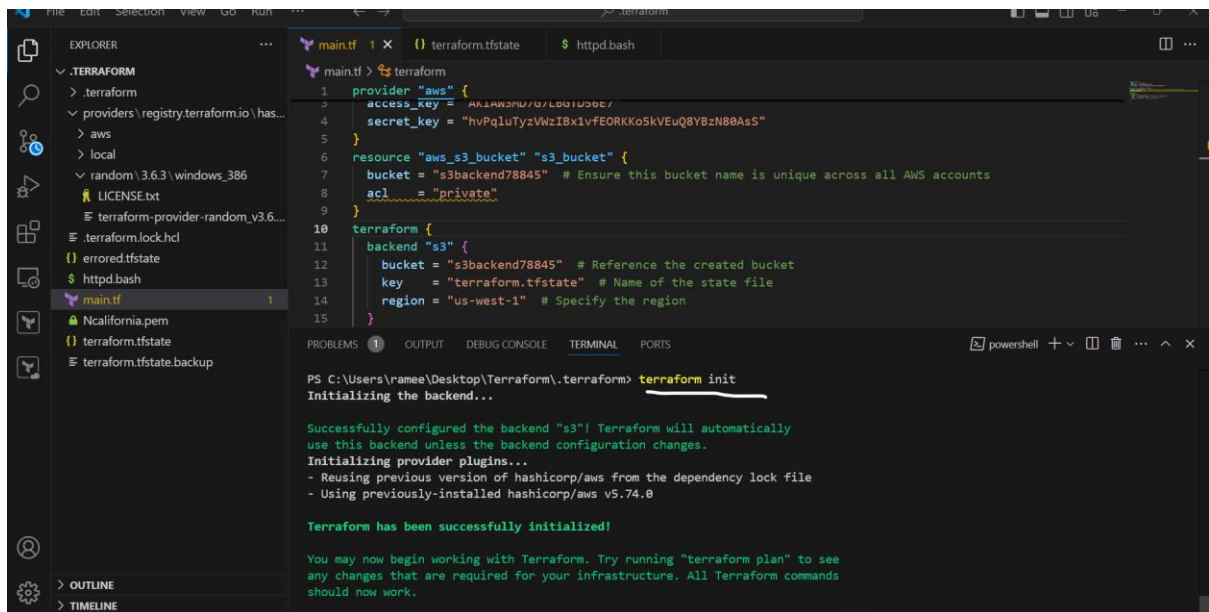


Now check here terraform.tfstate file we see content.



Now we have to setup backend.

First you need to initialize the all dependency's.



The screenshot shows a VS Code editor with a Terraform configuration file `main.tf` and a terminal window. The `main.tf` file contains the following configuration:

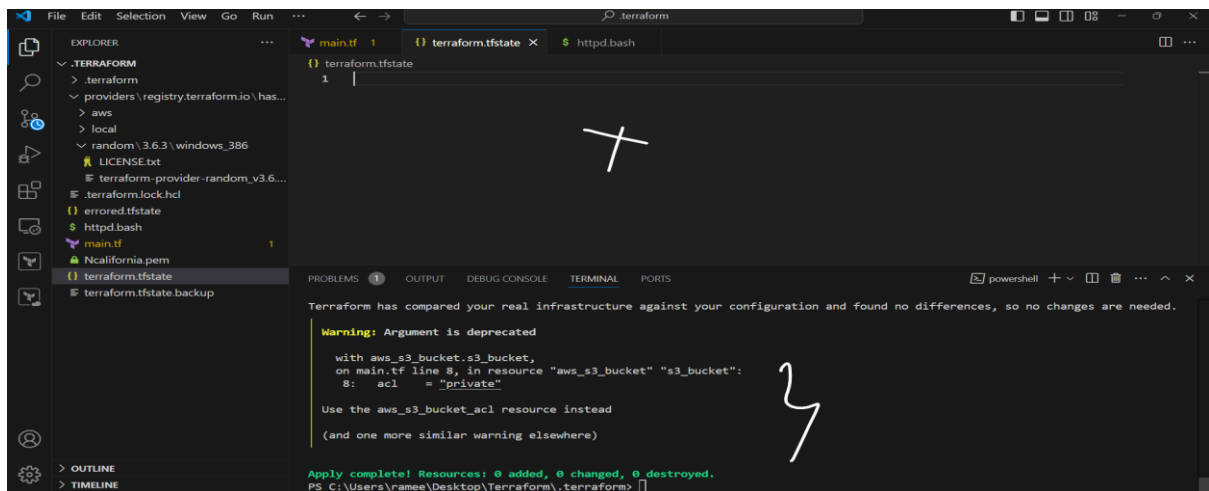
```
1 provider "aws" {  
2   access_key = "AKIAW3PU/0/LB01U0DE/  
3   secret_key = "hVPq1uTyZVWzIBx1vFEORKK0sKVEuQ8YBzN80AsS"  
4 }  
5  
6 resource "aws_s3_bucket" "s3_bucket" {  
7   bucket = "s3backend78845" # Ensure this bucket name is unique across all AWS accounts  
8   acl     = "private"  
9 }  
10  
11 terraform {  
12   backend "s3" {  
13     bucket = "s3backend78845" # Reference the created bucket  
14     key    = "terraform.tfstate" # Name of the state file  
15     region = "us-west-1" # Specify the region  
16   }  
17 }
```

The terminal window shows the output of the `terraform init` command:

```
PS C:\Users\ramee\Desktop\Terraform\Terraform> terraform init  
Initializing the backend...  
  
Successfully configured the backend "s3"! Terraform will automatically  
use this backend unless the backend configuration changes.  
Initializing provider plugins...  
- Reusing previous version of hashicorp/aws from the dependency lock file  
- Reusing previously-installed hashicorp/aws v5.74.0  
  
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.
```

Now try apply the changes using the command `terraform init`.

After running `terraform apply`, the `terraform.tfstate` file is stored in the specified S3 bucket, which allows for remote state management. This means that instead of being visible in the local directory, the state file is securely maintained in S3, enabling collaboration and consistency across different environments and team members.



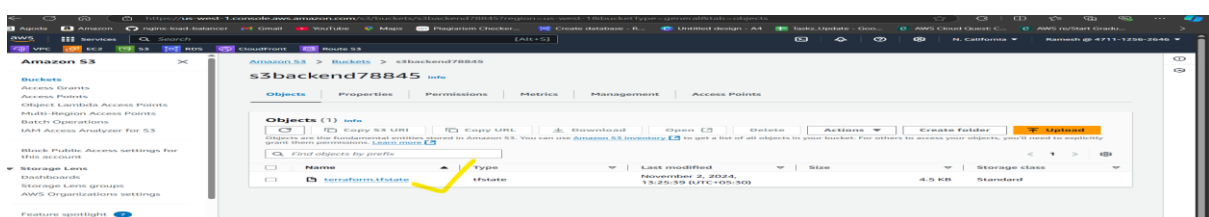
The screenshot shows a VS Code editor with a Terraform configuration file `main.tf` and a terminal window. The `main.tf` file contains the following configuration:

```
1 terraform {  
2   backend "s3" {  
3     bucket = "s3backend78845"  
4     key    = "terraform.tfstate"  
5     region = "us-west-1"  
6   }  
7 }  
8 resource "aws_s3_bucket" "s3_bucket" {  
9   bucket = "s3backend78845"  
10  acl     = "private"  
11 }
```

The terminal window shows the output of the `terraform apply` command:

```
Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.  
  
Warning: Argument is deprecated  
with aws_s3_bucket.s3_bucket,  
on main.tf line 8, in resource "aws_s3_bucket" "s3_bucket":  
8:   acl     = "private"  
  
Use the aws_s3_bucket_acl resource instead  
(and one more similar warning elsewhere)  
  
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.  
PS C:\Users\ramee\Desktop\Terraform\Terraform>
```

Now here our `terraform.tfstate` file is there.



3) Setup dynamo db locking for task3.

What is State Locking?

State locking prevents multiple users from making changes to the infrastructure at the same time. This is important because:

- **Avoid Conflicts:** Only one person can make changes at a time, preventing errors.
- **Maintain Consistency:** The state file always reflects the actual infrastructure.

Using S3 and DynamoDB

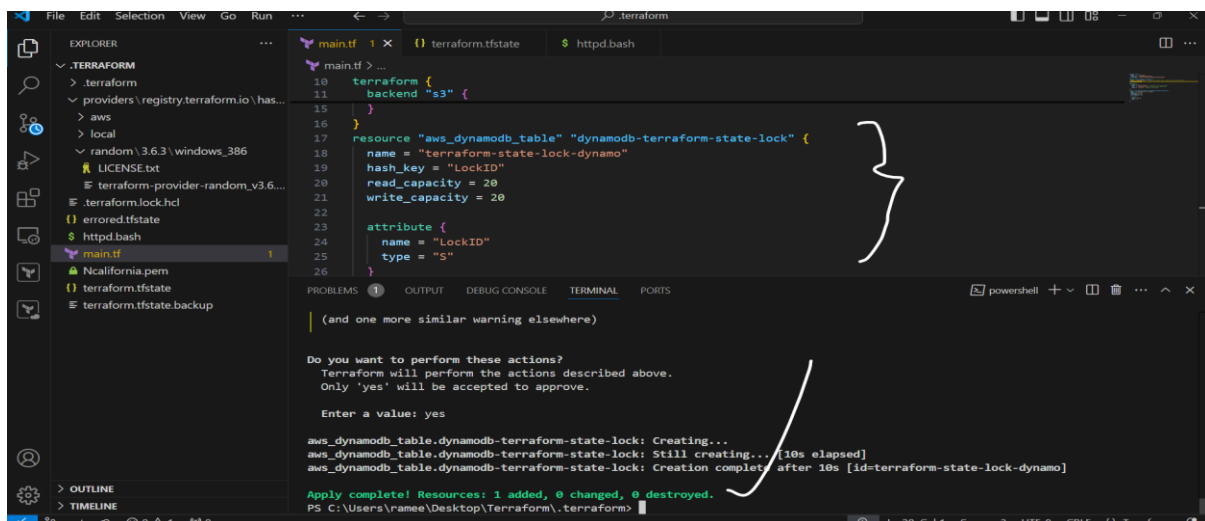
To effectively manage your state, use **Amazon S3** for storing the state file and **DynamoDB** for locking it.

2. **S3:** Stores the state file securely.
3. **Dynamo DB:** Locks the state so no one else can make changes while one user is working.

Create dynamo db using terraform:

```
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"  
  }  
}
```

After save the file and initialize the all dependency's.
Then enter terraform apply.

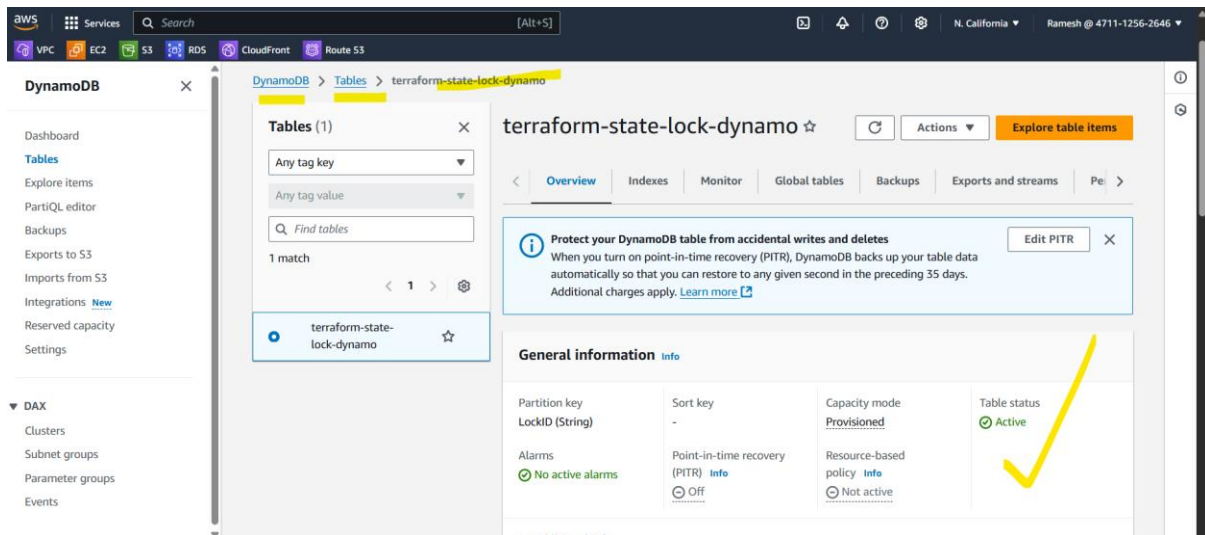
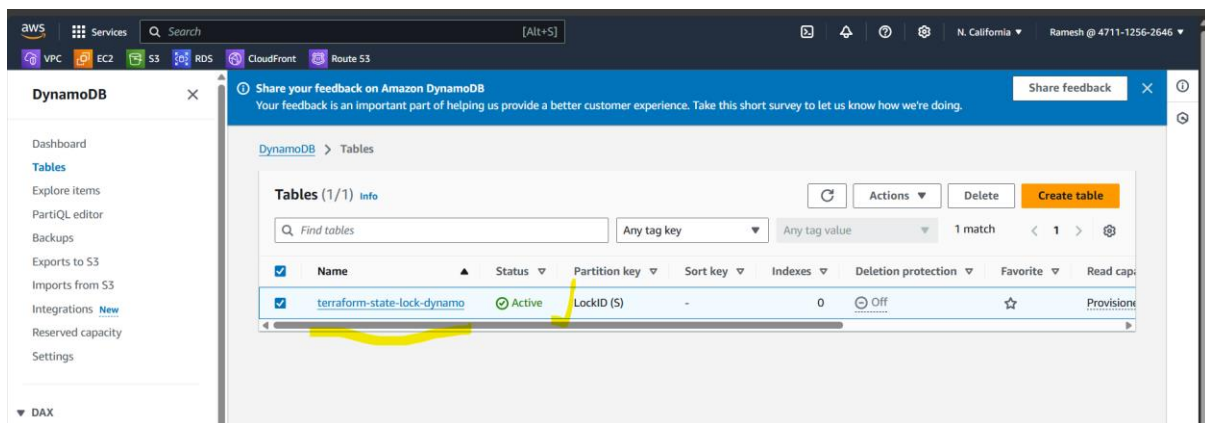


The screenshot shows a VS Code editor with a file explorer on the left displaying a project structure for Terraform. The main editor window shows a Terraform configuration file named `main.tf` with the following content:

```
10 terraform {  
11   backend "s3" {  
12   }  
13 }  
14  
15 resource "aws_dynamodb_table" "dynamodb-terraform-state-lock" {  
16   name = "terraform-state-lock-dynamo"  
17   hash_key = "LockID"  
18   read_capacity = 20  
19   write_capacity = 20  
20  
21   attribute {  
22     name = "LockID"  
23     type = "S"  
24   }  
25 }  
26
```

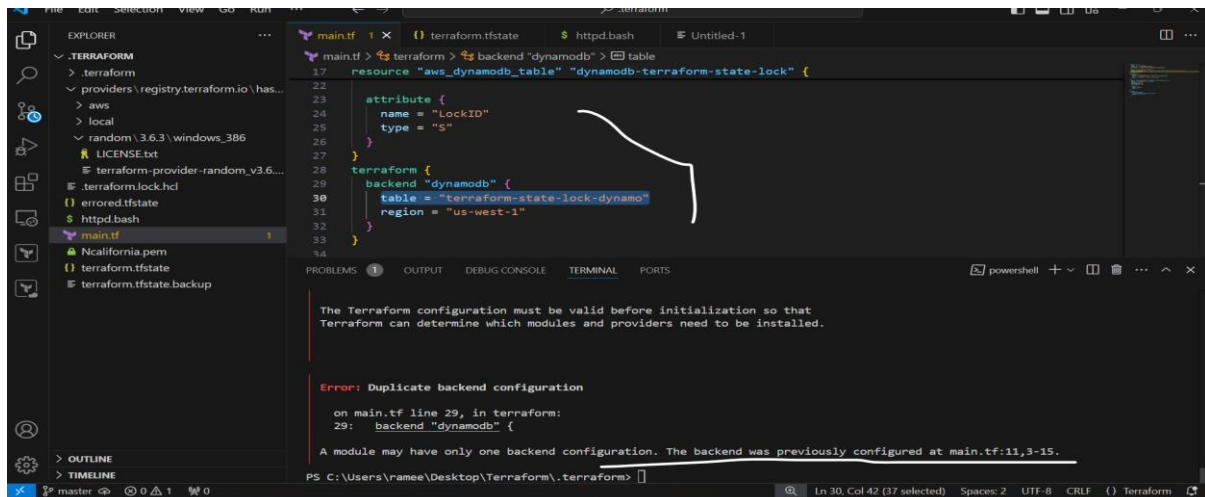
The terminal window at the bottom shows the output of the `terraform apply` command. It displays a warning about the `terraform.tfstate` file, followed by a confirmation prompt: "Do you want to perform these actions? Terraform will perform the actions described above. Only 'yes' will be accepted to approve. Enter a value: yes". The output then shows the creation of the `aws_dynamodb_table.dynamodb-terraform-state-lock` resource, with a status of "Creation complete" after 10s. The final output is "Apply complete! Resources: 1 added, 0 changed, 0 destroyed."

Now we have to check the dynamodb table is created or not.



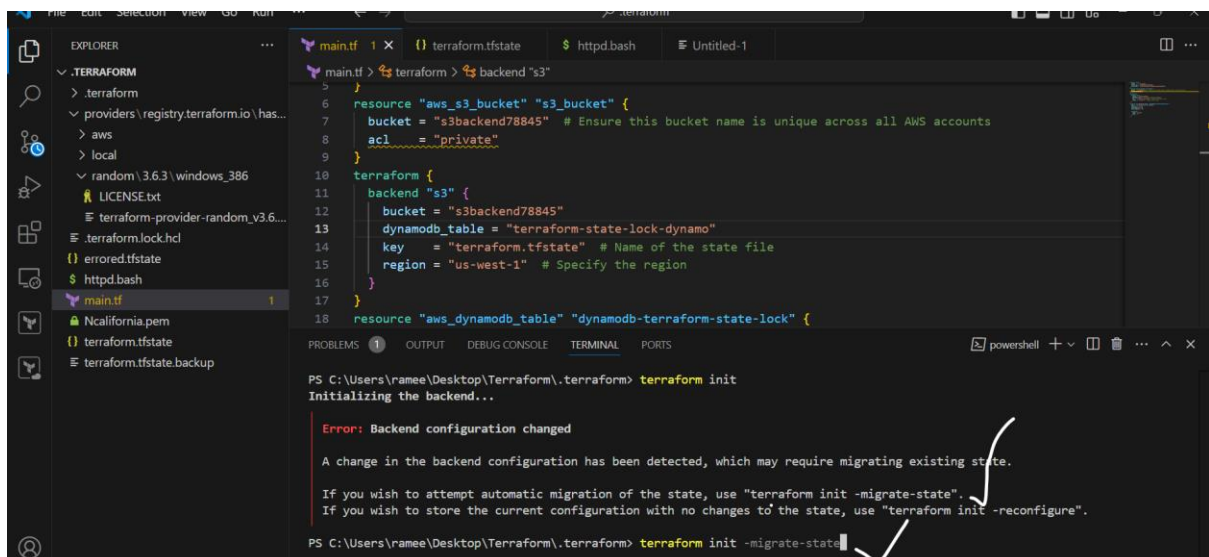
Now I want to Setup dynamo db locking.

Here I am facing with issue is am already created on backend for s3 bucket. So it's not allowing second dynamo db backend locking.



So came s3 backend there I am attached.

Here I am try to initialize it 's asking – use terraform init –migrate-state



```
main.tf 1 x {} terraform.tfstate $ httpd.bash Untitled-1
main.tf > terraform > backend "s3"
6 resource "aws_s3_bucket" "s3_bucket" {
7   bucket = "s3backend78845" # Ensure this bucket name is unique across all AWS accounts
8   acl    = "private"
9 }
10 terraform {
11   backend "s3" {
12     bucket = "s3backend78845"
13     dynamodb_table = "terraform-state-lock-dynamo"
14     key          = "terraform.tfstate" # Name of the state file
15     region       = "us-west-1" # Specify the region
16   }
17 }
18 resource "aws_dynamodb_table" "dynamodb-terraform-state-lock" {

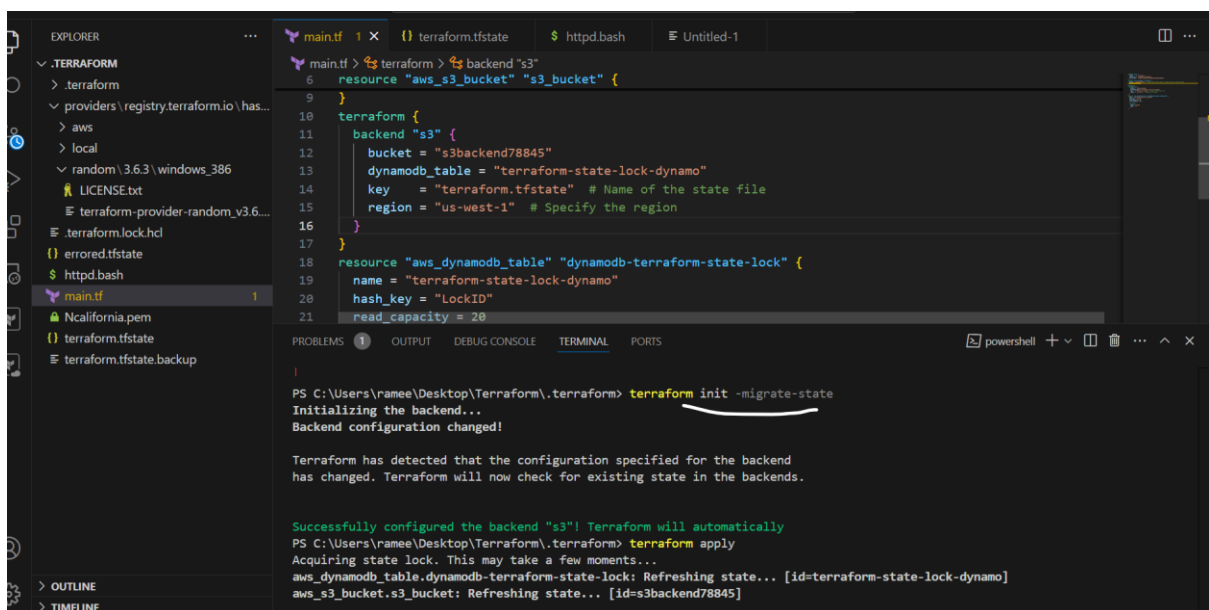
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ramee\Desktop\Terraform\..terraform> terraform init
Initializing the backend...

Error: Backend configuration changed

A change in the backend configuration has been detected, which may require migrating existing state.

If you wish to attempt automatic migration of the state, use "terraform init -migrate-state".
If you wish to store the current configuration with no changes to the state, use "terraform init -reconfigure".

PS C:\Users\ramee\Desktop\Terraform\..terraform> terraform init -migrate-state
```



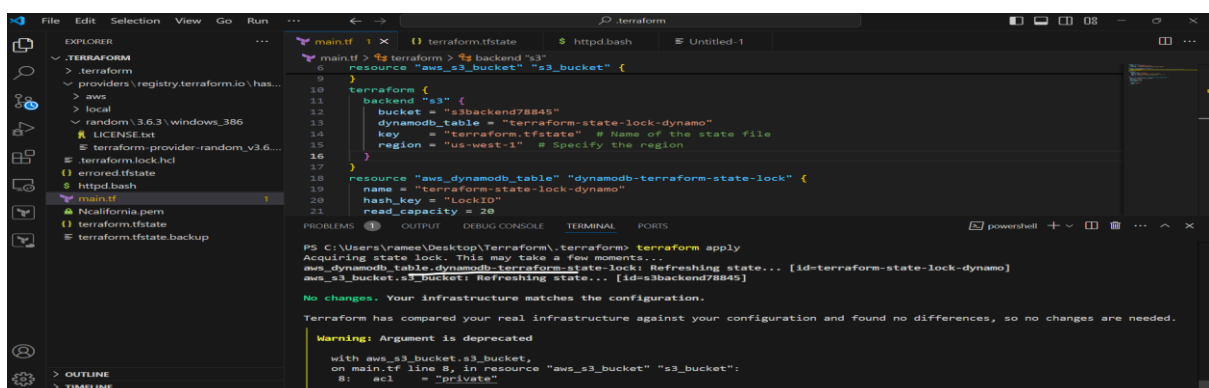
```
main.tf 1 x {} terraform.tfstate $ httpd.bash Untitled-1
main.tf > terraform > backend "s3"
6 resource "aws_s3_bucket" "s3_bucket" {
9 }
10 terraform {
11   backend "s3" {
12     bucket = "s3backend78845"
13     dynamodb_table = "terraform-state-lock-dynamo"
14     key          = "terraform.tfstate" # Name of the state file
15     region       = "us-west-1" # Specify the region
16   }
17 }
18 resource "aws_dynamodb_table" "dynamodb-terraform-state-lock" {
19   name = "terraform-state-lock-dynamo"
20   hash_key = "LockID"
21   read_capacity = 20
22 }

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ramee\Desktop\Terraform\..terraform> terraform init -migrate-state
Initializing the backend...
Backend configuration changed!

Terraform has detected that the configuration specified for the backend
has changed. Terraform will now check for existing state in the backends.

Successfully configured the backend "s3"! Terraform will automatically
PS C:\Users\ramee\Desktop\Terraform\..terraform> terraform apply
Acquiring state lock. This may take a few moments...
aws_dynamodb_table.dynamodb-terraform-state-lock: Refreshing state... [id=terraform-state-lock-dynamo]
aws_s3_bucket.s3_bucket: Refreshing state... [id=s3backend78845]
```

It will success.



```
main.tf 1 x {} terraform.tfstate $ httpd.bash Untitled-1
main.tf > terraform > backend "s3"
6 resource "aws_s3_bucket" "s3_bucket" {
11   backend "s3" {
12     bucket = "s3backend78845"
13     dynamodb_table = "terraform-state-lock-dynamo"
14     key          = "terraform.tfstate" # Name of the state file
15     region       = "us-west-1" # Specify the region
16   }
17 }
18 resource "aws_dynamodb_table" "dynamodb-terraform-state-lock" {
19   name = "terraform-state-lock-dynamo"
20   hash_key = "LockID"
21   read_capacity = 20
22 }

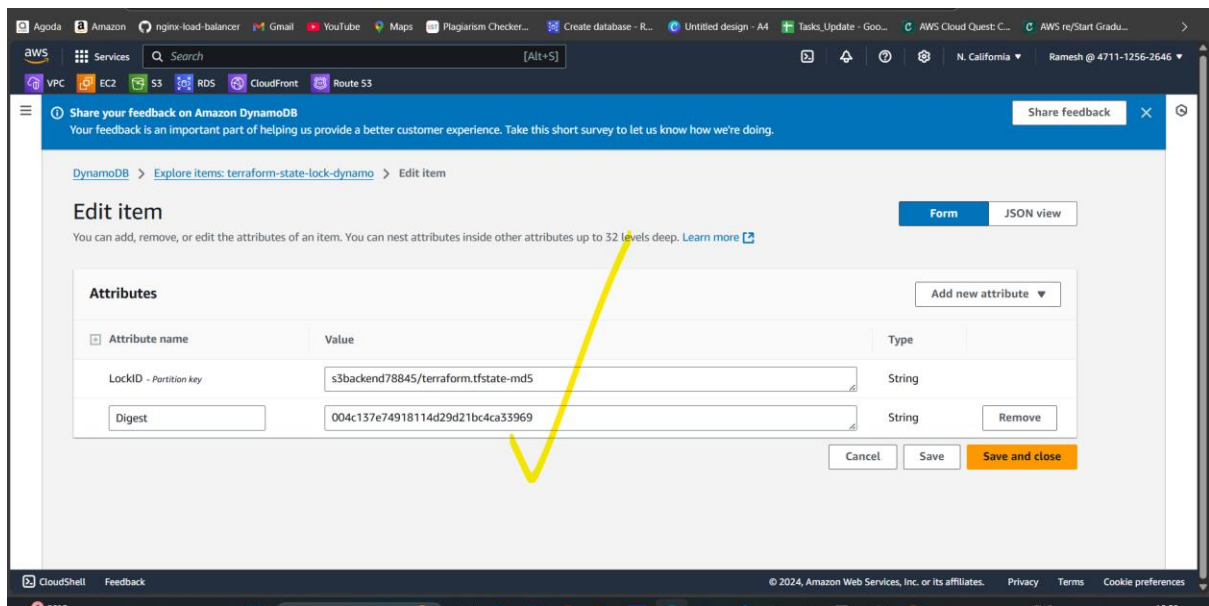
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ramee\Desktop\Terraform\..terraform> terraform apply
Acquiring state lock. This may take a few moments...
aws_dynamodb_table.dynamodb-terraform-state-lock: Refreshing state... [id=terraform-state-lock-dynamo]
aws_s3_bucket.s3_bucket: Refreshing state... [id=s3backend78845]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Warning: Argument is deprecated
with aws_s3_bucket.s3_bucket,
on main.tf line 8, in resource "aws_s3_bucket" "s3_bucket":
8:   acl    = "private"
```

Now we want to check it Setup dynamo db locking or not.



Its created.

Terraform provides a range of log levels to help troubleshoot issues, each offering a different depth of information:

1. **INFO** gives basic updates, showing what Terraform is doing, like creating or updating resources.
2. **WARNING** flags potential issues that might not block execution but could lead to unexpected results, helping to catch misconfigurations early.
3. **ERROR** logs critical problems that stop Terraform from proceeding, such as syntax or config errors.
4. **DEBUG** dives deeper, showing detailed internal processing. It's great for understanding Terraform's internal actions.
5. **TRACE** is the most detailed, logging every action Terraform takes. It's ideal when I need to trace down to the smallest operation.

To enable any of these, I'd just set the `TF_LOG` environment variable to the required level, like `export TF_LOG=DEBUG`. And if I want to keep the logs, I can set a path with `TF_LOG_PATH` so the logs save to a file for later analysis.