

Task 7

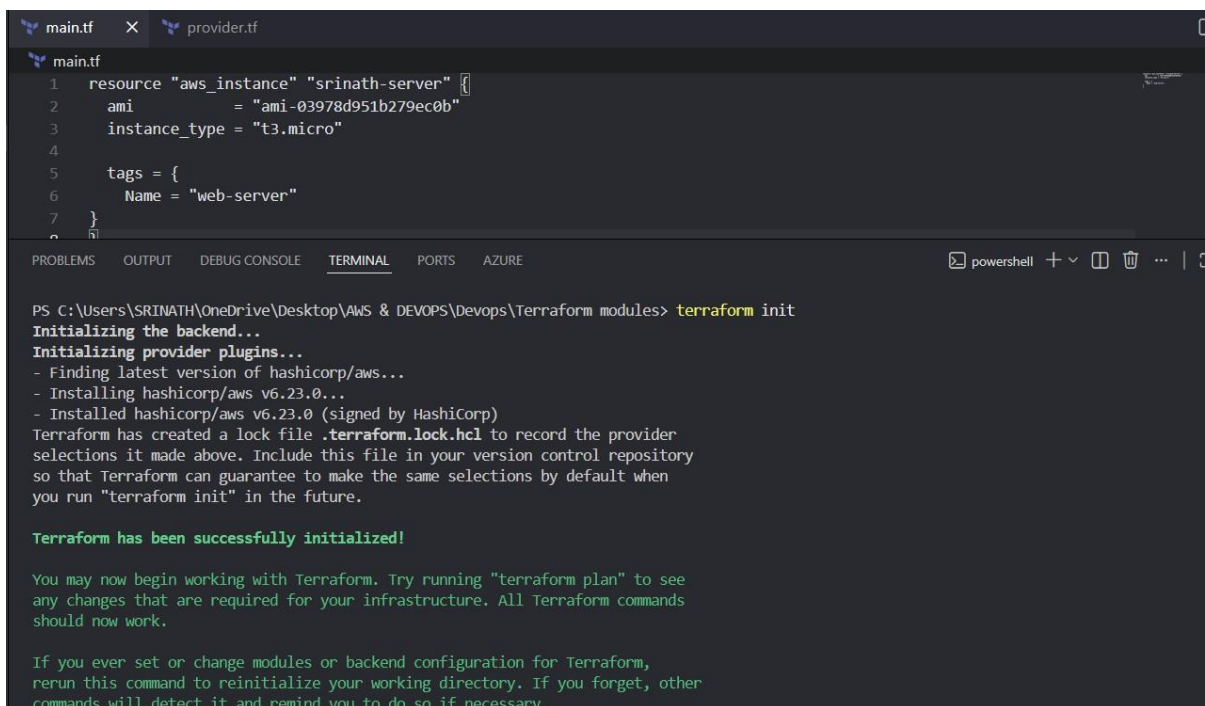
Terraform:

- Terraform is an open-source infrastructure as code (IaC) tool developed by HashiCorp that allows users to define and provision cloud and on-premises resources using a declarative configuration language.
- It enables users to manage infrastructure throughout its lifecycle by writing human-readable configuration files that can be versioned, reused, and shared.

Terraform Command:

Creation of EC2

1) terraform init:



The screenshot shows a VS Code editor with two tabs: 'main.tf' and 'provider.tf'. The 'main.tf' tab is active, displaying a Terraform configuration for an AWS EC2 instance. The configuration includes the resource name 'srinath-server', an AMI ID, instance type 't3.micro', and a tag 'web-server'. Below the editor, the 'TERMINAL' panel shows the output of the 'terraform init' command. The output indicates that the backend is initialized, provider plugins are installed, and a lock file is created. It concludes with a success message and instructions for running 'terraform plan'.

```
main.tf
1 resource "aws_instance" "srinath-server" {
2   ami           = "ami-03978d951b279ec0b"
3   instance_type = "t3.micro"
4
5   tags = {
6     Name = "web-server"
7   }
8 }
```

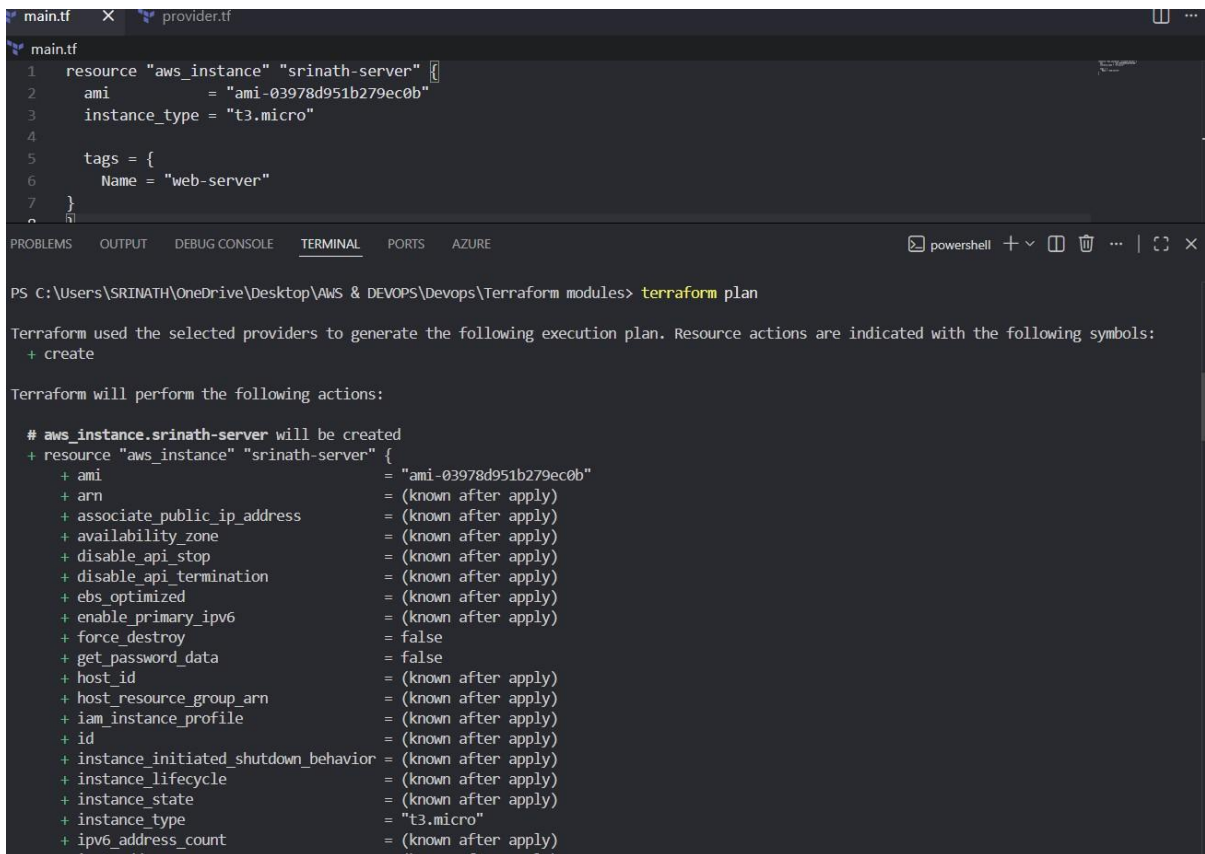
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
PS C:\Users\SRINATH\OneDrive\Desktop\AWS & DEVOPS\Devops\Terraform modules> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.23.0...
- Installed hashicorp/aws v6.23.0 (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.
```

2) terraform plan:



The screenshot shows a VS Code editor with two files: `main.tf` and `provider.tf`. The `main.tf` file contains a Terraform resource definition for an AWS instance. The terminal window shows the output of the `terraform plan` command, indicating that the resource will be created and listing its attributes.

```
main.tf
1 resource "aws_instance" "srinath-server" {
2   ami           = "ami-03978d951b279ec0b"
3   instance_type = "t3.micro"
4
5   tags = {
6     Name = "web-server"
7   }
8 }
```

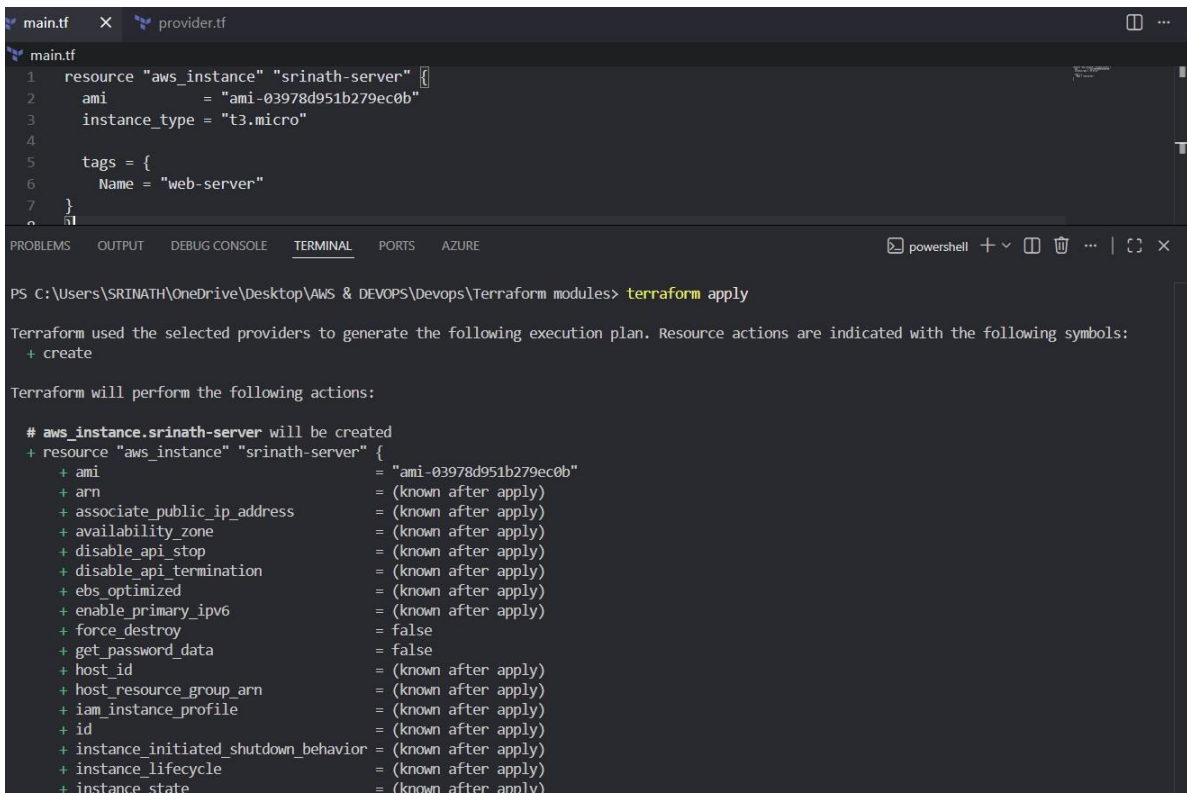
```
PS C:\Users\SRINATH\OneDrive\Desktop\AWS & DEVOPS\Devops\Terraform modules> terraform plan

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

Terraform will perform the following actions:

# aws_instance.srinath-server will be created
+ resource "aws_instance" "srinath-server" {
  + ami                  = "ami-03978d951b279ec0b"
  + arn                  = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone     = (known after apply)
  + disable_api_stop     = (known after apply)
  + disable_api_termination = (known after apply)
  + ebs_optimized         = (known after apply)
  + enable_primary_ipv6   = (known after apply)
  + force_destroy         = false
  + get_password_data     = false
  + host_id               = (known after apply)
  + host_resource_group_arn = (known after apply)
  + iam_instance_profile  = (known after apply)
  + id                   = (known after apply)
  + instance_initiated_shutdown_behavior = (known after apply)
  + instance_lifecycle    = (known after apply)
  + instance_state        = (known after apply)
  + instance_type         = "t3.micro"
  + ipv6_address_count    = (known after apply)
  + ipv6_addresses        = (known after apply)
  + key_name               = (known after apply)
  + monitoring              = (known after apply)
  + outpost_id             = (known after apply)
  + placement_group        = (known after apply)
  + primary_interface      = (known after apply)
  + private_ip              = (known after apply)
  + private_ip_prefix     = (known after apply)
  + provisioner             = (known after apply)
  + subnet_id              = (known after apply)
  + tags                   = {Name = "web-server"}
  + tenancy                = (known after apply)
  + user_data               = (known after apply)
  + vpc_security_group_ids = (known after apply)
}
```

3) terraform apply



The screenshot shows a VS Code editor with two files: `main.tf` and `provider.tf`. The `main.tf` file contains a Terraform resource definition for an AWS instance. The terminal window shows the output of the `terraform apply` command, indicating that the resource will be created and listing its attributes.

```
main.tf
1 resource "aws_instance" "srinath-server" {
2   ami           = "ami-03978d951b279ec0b"
3   instance_type = "t3.micro"
4
5   tags = {
6     Name = "web-server"
7   }
8 }
```

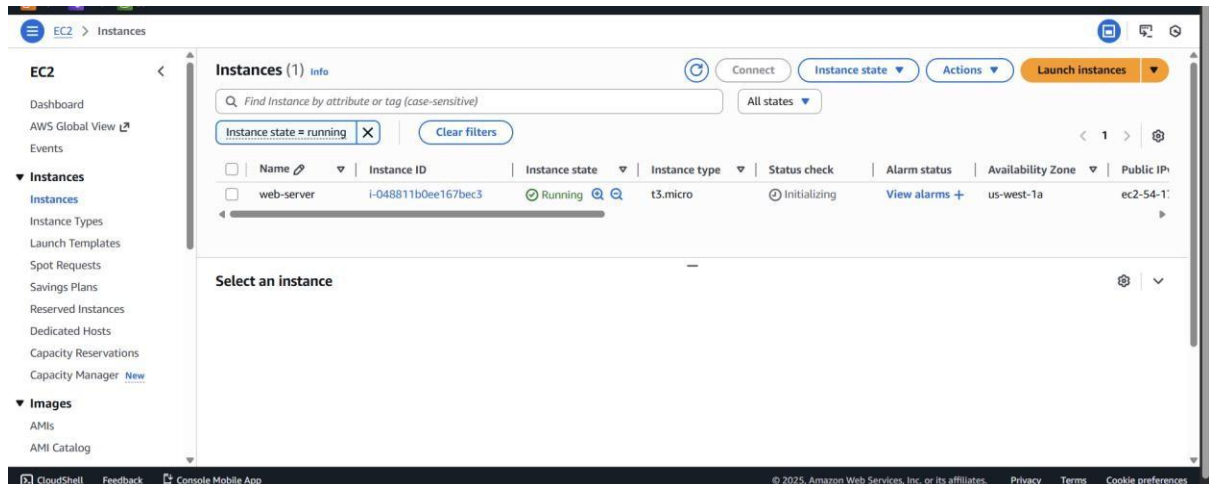
```
PS C:\Users\SRINATH\OneDrive\Desktop\AWS & DEVOPS\Devops\Terraform modules> terraform apply

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

Terraform will perform the following actions:

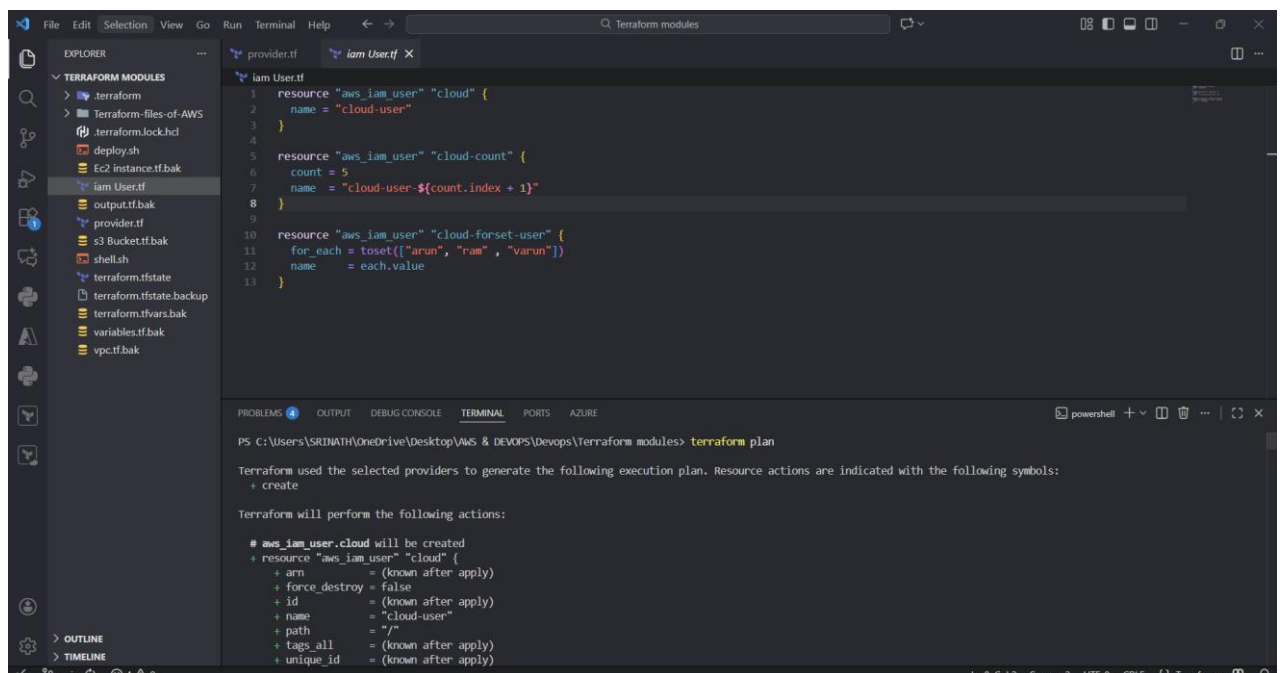
# aws_instance.srinath-server will be created
+ resource "aws_instance" "srinath-server" {
  + ami                  = "ami-03978d951b279ec0b"
  + arn                  = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone     = (known after apply)
  + disable_api_stop     = (known after apply)
  + disable_api_termination = (known after apply)
  + ebs_optimized         = (known after apply)
  + enable_primary_ipv6   = (known after apply)
  + force_destroy         = false
  + get_password_data     = false
  + host_id               = (known after apply)
  + host_resource_group_arn = (known after apply)
  + iam_instance_profile  = (known after apply)
  + id                   = (known after apply)
  + instance_initiated_shutdown_behavior = (known after apply)
  + instance_lifecycle    = (known after apply)
  + instance_state        = (known after apply)
  + instance_type         = "t3.micro"
  + ipv6_address_count    = (known after apply)
  + ipv6_addresses        = (known after apply)
  + key_name               = (known after apply)
  + monitoring              = (known after apply)
  + outpost_id             = (known after apply)
  + placement_group        = (known after apply)
  + primary_interface      = (known after apply)
  + private_ip              = (known after apply)
  + private_ip_prefix     = (known after apply)
  + provisioner             = (known after apply)
  + subnet_id              = (known after apply)
  + tags                   = {Name = "web-server"}
  + tenancy                = (known after apply)
  + user_data               = (known after apply)
  + vpc_security_group_ids = (known after apply)
}
```

Output

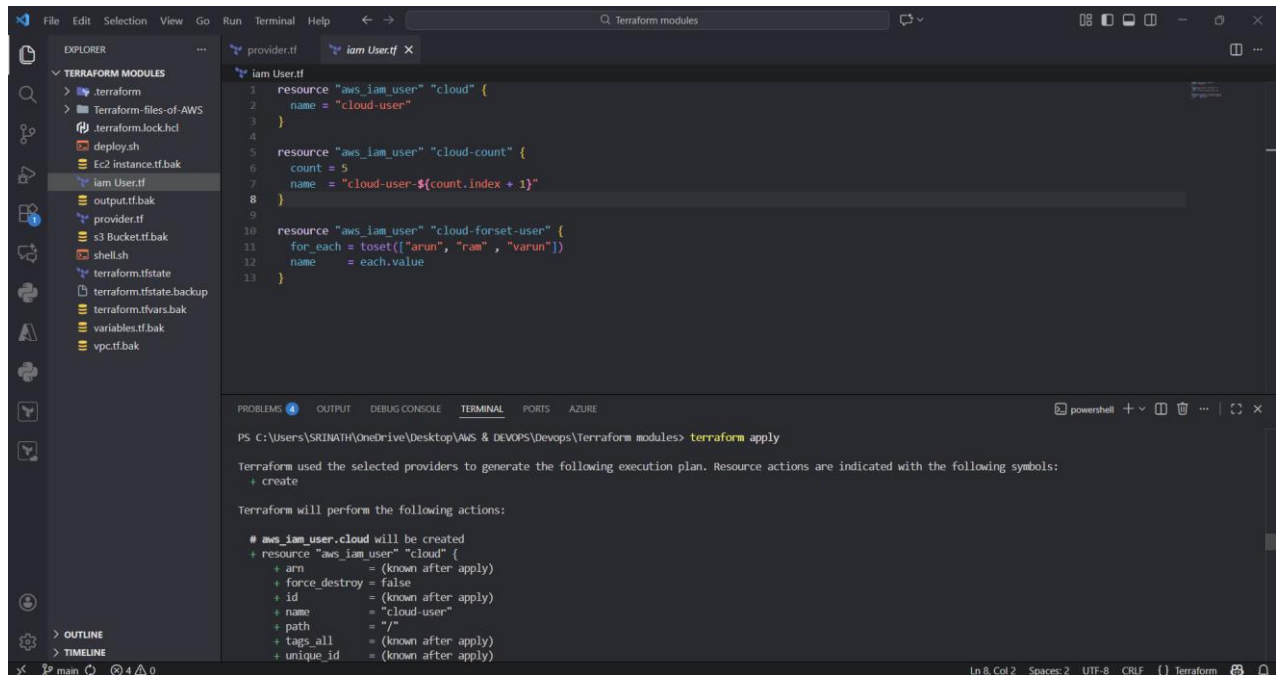


Creation of IAM:

1) terraform plan:



2) terraform apply:



The screenshot shows the Visual Studio Code interface with the Terraform modules explorer on the left and the terminal on the right. The terminal displays the output of the `terraform apply` command, showing the execution plan and the actions Terraform will perform.

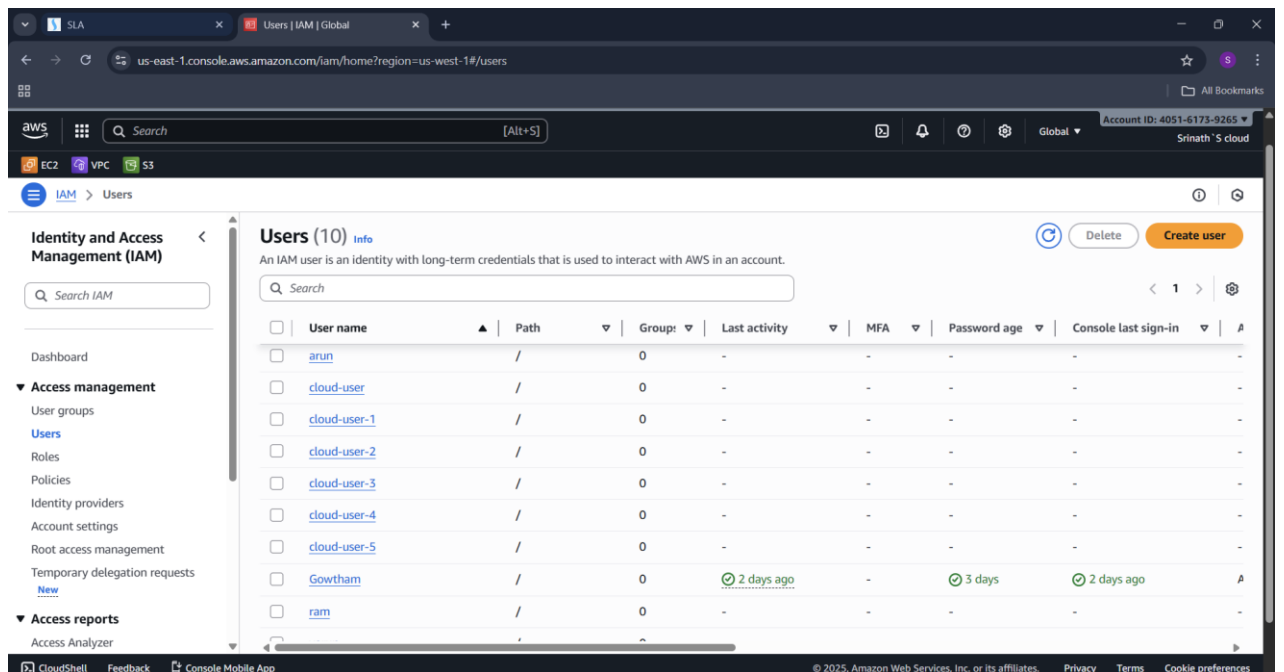
```
PS C:\Users\SRINATH\Desktop\AWS & DEVOPS\Devops\Terraform modules> terraform apply

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

Terraform will perform the following actions:

# aws_iam_user.cloud will be created
+ resource "aws_iam_user" "cloud" {
  + arn                = (known after apply)
  + force_destroy      = false
  + id                 = (known after apply)
  + name               = "cloud-user"
  + path               = "/"
  + tags_all           = (known after apply)
  + unique_id          = (known after apply)
}
```

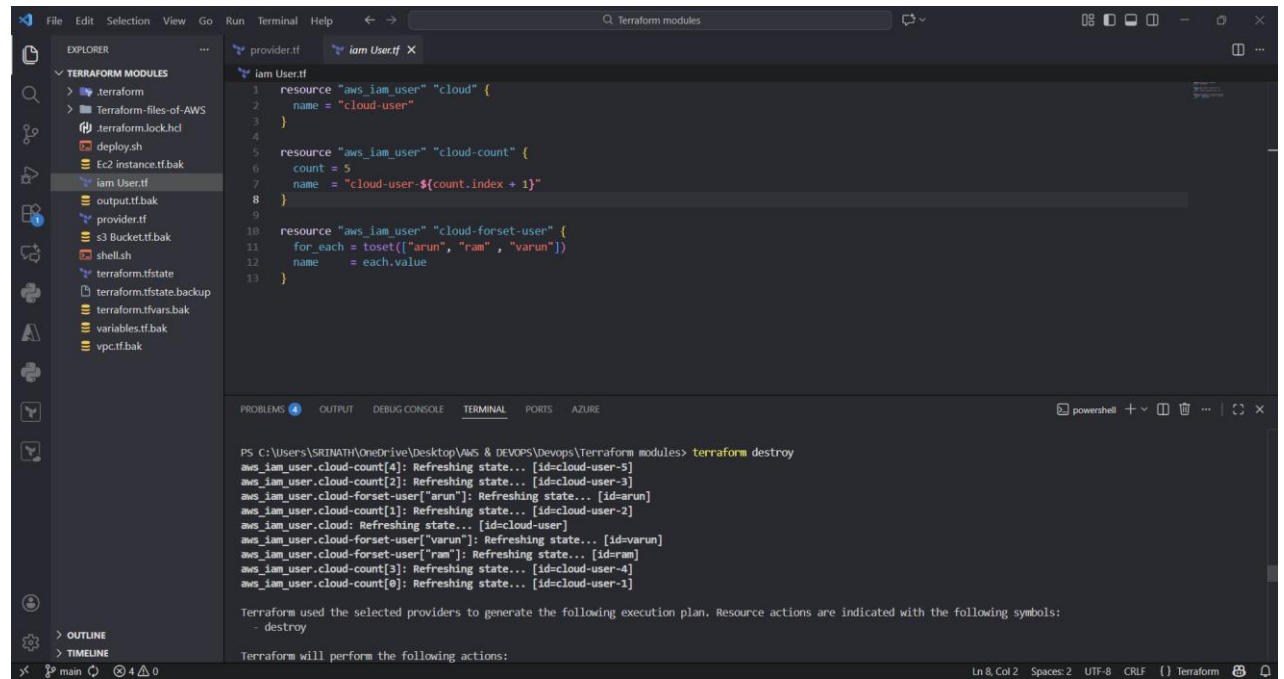
Output:



The screenshot shows the AWS IAM console with the 'Users' page selected. The table lists the following users:

User name	Path	Group	Last activity	MFA	Password age	Console last sign-in
arun	/	0	-	-	-	-
cloud-user	/	0	-	-	-	-
cloud-user-1	/	0	-	-	-	-
cloud-user-2	/	0	-	-	-	-
cloud-user-3	/	0	-	-	-	-
cloud-user-4	/	0	-	-	-	-
cloud-user-5	/	0	-	-	-	-
Gowtham	/	0	2 days ago	-	3 days	2 days ago
ram	/	0	-	-	-	-

3) terraform destroy:



```
1 resource "aws_iam_user" "cloud" {
2   name = "cloud-user"
3 }
4
5 resource "aws_iam_user" "cloud-count" {
6   count = 5
7   name = "cloud-user-${count.index + 1}"
8 }
9
10 resource "aws_iam_user" "cloud-forset-user" {
11   for_each = toset(["arun", "ram", "varun"])
12   name     = each.value
13 }
```

PS C:\Users\SRINATH\OneDrive\Desktop\AWS & DEVOPS\devops\Terraform modules> terraform destroy

aws_iam_user.cloud-count[4]: Refreshing state... [id=cloud-user-5]
aws_iam_user.cloud-count[2]: Refreshing state... [id=cloud-user-3]
aws_iam_user.cloud-forset-user["arun"]: Refreshing state... [id=arun]
aws_iam_user.cloud-count[1]: Refreshing state... [id=cloud-user-2]
aws_iam_user.cloud: Refreshing state... [id=cloud-user]
aws_iam_user.cloud-forset-user["varun"]: Refreshing state... [id=varun]
aws_iam_user.cloud-forset-user["ram"]: Refreshing state... [id=ram]
aws_iam_user.cloud-count[3]: Refreshing state... [id=cloud-user-4]
aws_iam_user.cloud-count[0]: Refreshing state... [id=cloud-user-1]

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: