Lesson 11 Demo 08

Managing Terraform Resource Lifecycle

Objective: To manage the Terraform resource lifecycle effectively for infrastructure deployment

Tools required: Terraform**,** AWS, and Visual Studio Code

Prerequisites: Refer to **Demo 01** of **Lesson 11** for creating access and secret key

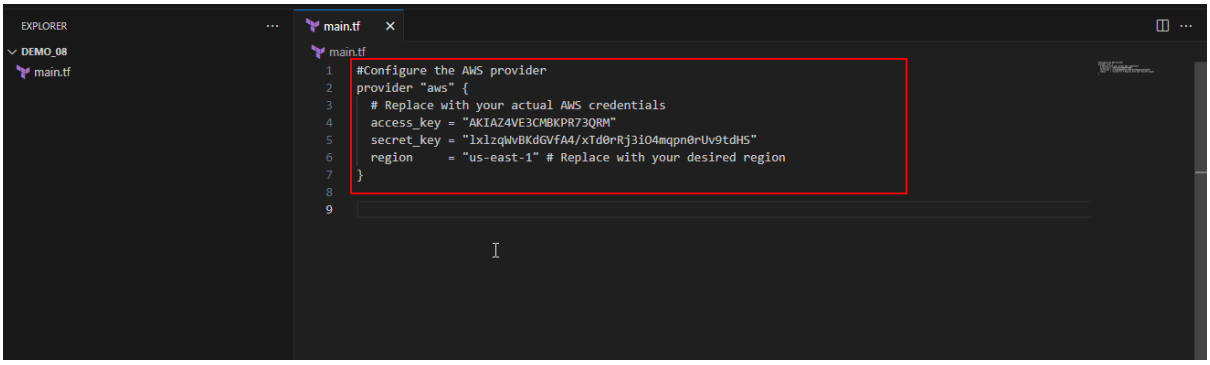
Steps to be followed:

1. Set up a basic AWS infrastructure
2. Implement a dynamic security group with lifecycle modifications
3. Apply configuration changes
4. Implement and test **prevent_destroy**

Step 1: Set up a basic AWS infrastructure

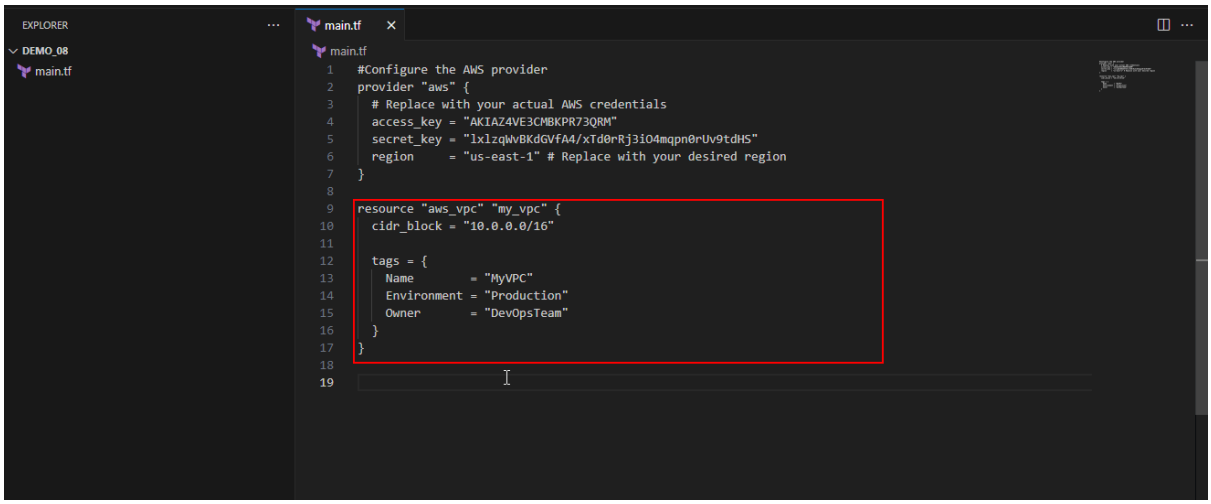
1.1 Open your Terraform configuration environment and create a file named **main.tf**. Add the following configuration block, as shown in the screenshot:

```
#Configure the AWS provider
provider "aws" {
 # Replace with your actual AWS credentials
 access_key = "YOUR_ACCESS_KEY"
 secret_key = "YOUR_SECRET_KEY"
 region     = "us-east-1" # Replace with your desired region
}
```



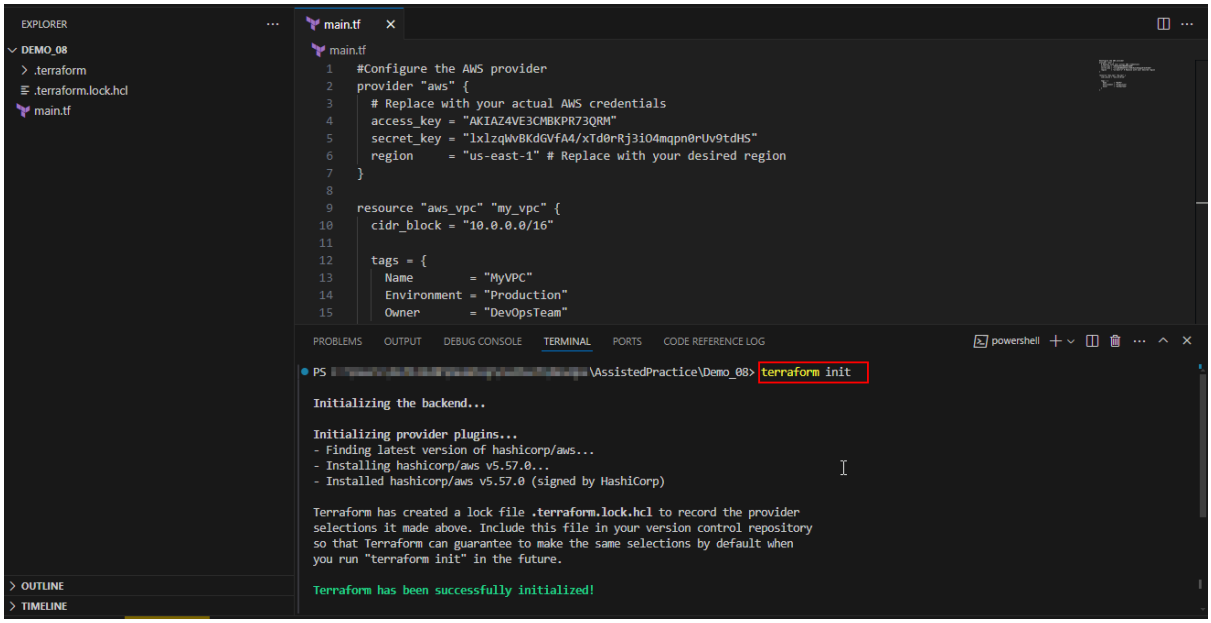
1.2 Define an AWS VPC to host the network components, as shown:

```
resource "aws_vpc" "my_vpc" {
 cidr_block = "10.0.0.0/16"

 tags = {
  Name        = "MyVPC"
  Environment = "Production"
  Owner       = "DevOpsTeam"
 }
}
```



1.3 Initialize the Terraform project using the following command to set up the necessary plugins, as shown:

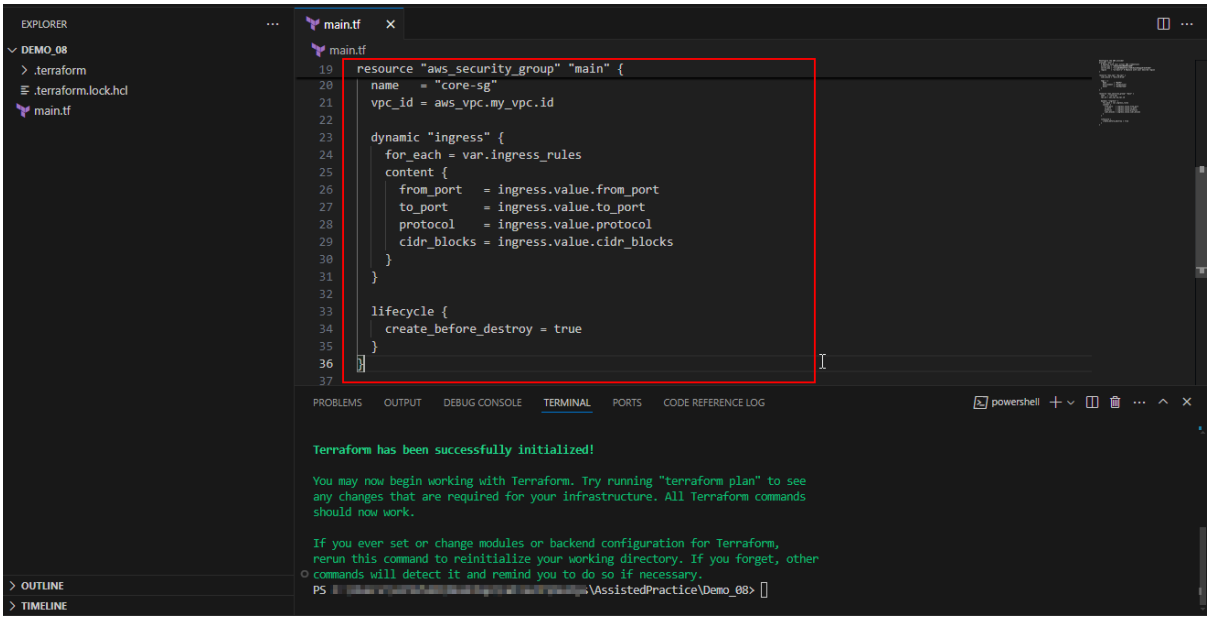
terraform init



Step 2: Implement a dynamic security group with lifecycle modifications

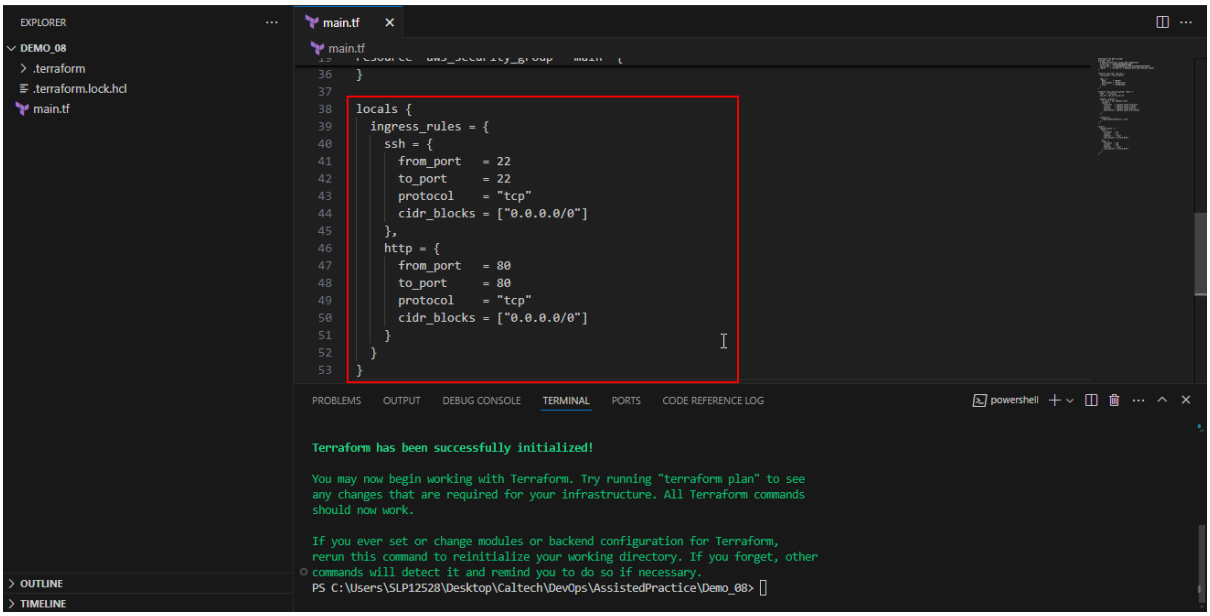
2.1 Create a security group in the specified VPC with dynamic ingress rules, as shown in the screenshot:

```
resource "aws_security_group" "main" {
 name   = "core-sg"
 vpc_id = aws_vpc.my_vpc.id

 dynamic "ingress" {
  for_each = var.ingress_rules
  content {
   from_port   = ingress.value.from_port
   to_port     = ingress.value.to_port
   protocol    = ingress.value.protocol
   cidr_blocks = ingress.value.cidr_blocks
  }
 }

 lifecycle {
  create_before_destroy = true
 }
}
```



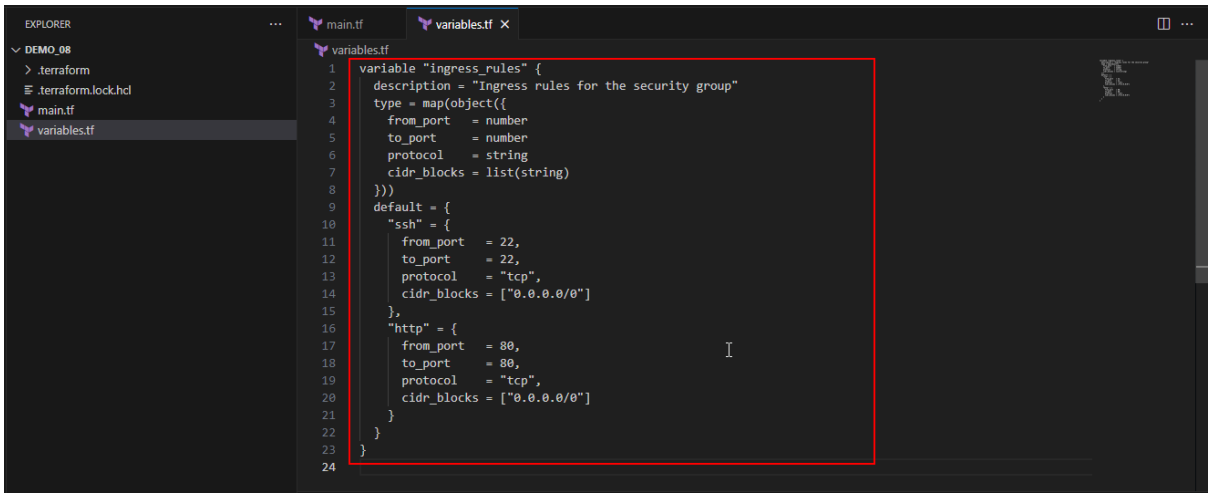
2.2 Define ingress rules using local variables or direct definition as shown in the screenshot:

```
locals {
 ingress_rules = {
  ssh = {
   from_port   = 22
   to_port     = 22
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
  },
  http = {
   from_port   = 80
   to_port     = 80
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
  }
 }
}
```



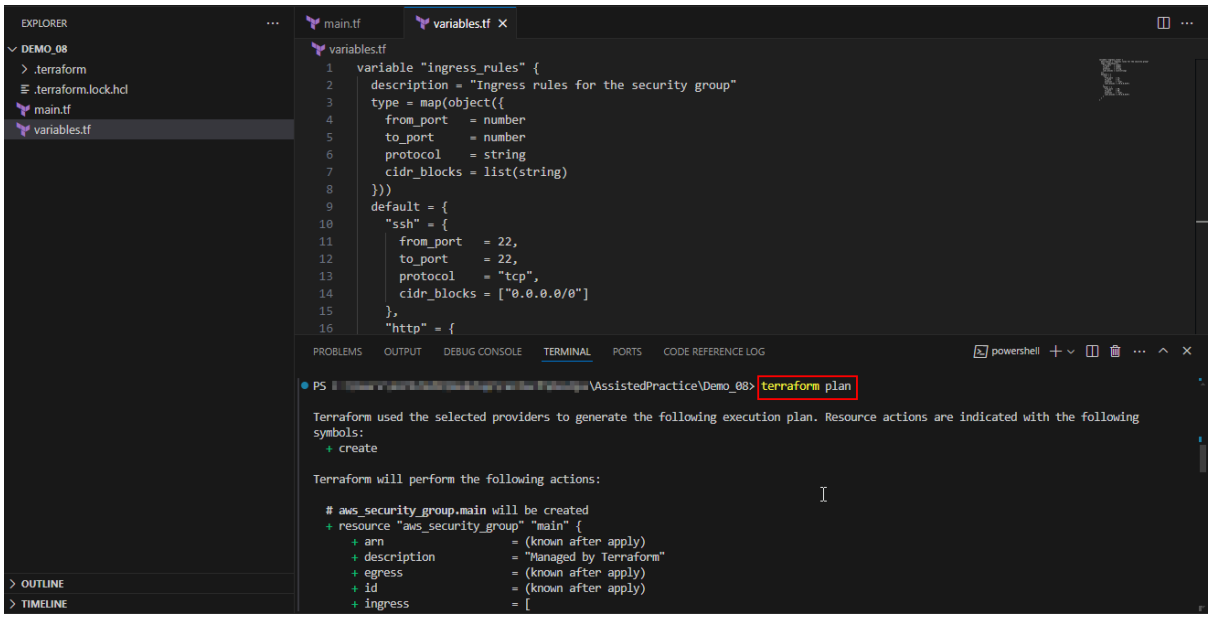
2.3 Create a file named **variables.tf** and add the following variable block, as shown in the screenshot:

```
variable "ingress_rules" {
 description = "Ingress rules for the security group"
 type = map(object({
  from_port   = number
  to_port     = number
  protocol    = string
  cidr_blocks = list(string)
 }))
 default = {
  "ssh" = {
   from_port   = 22,
   to_port     = 22,
   protocol    = "tcp",
   cidr_blocks = ["0.0.0.0/0"]
  },
  "http" = {
   from_port   = 80,
   to_port     = 80,
   protocol    = "tcp",
   cidr_blocks = ["0.0.0.0/0"]
  }
 }
}
```

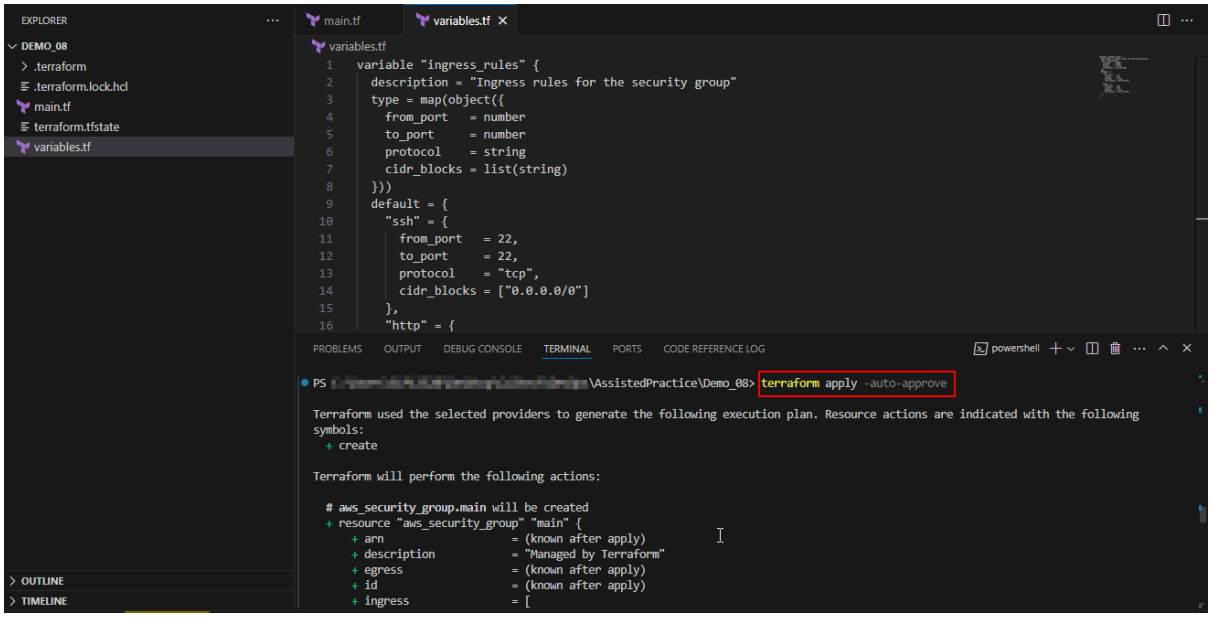


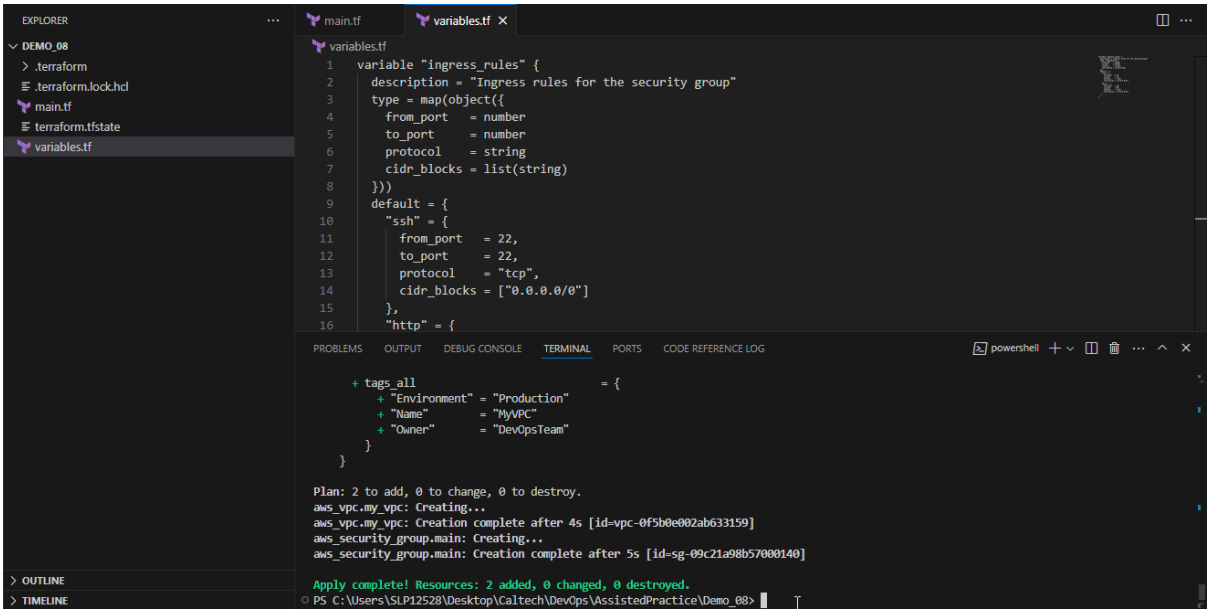
Step 3: Apply configuration changes

3.1 Execute the following command to preview the proposed changes to the infrastructure:
terraform plan

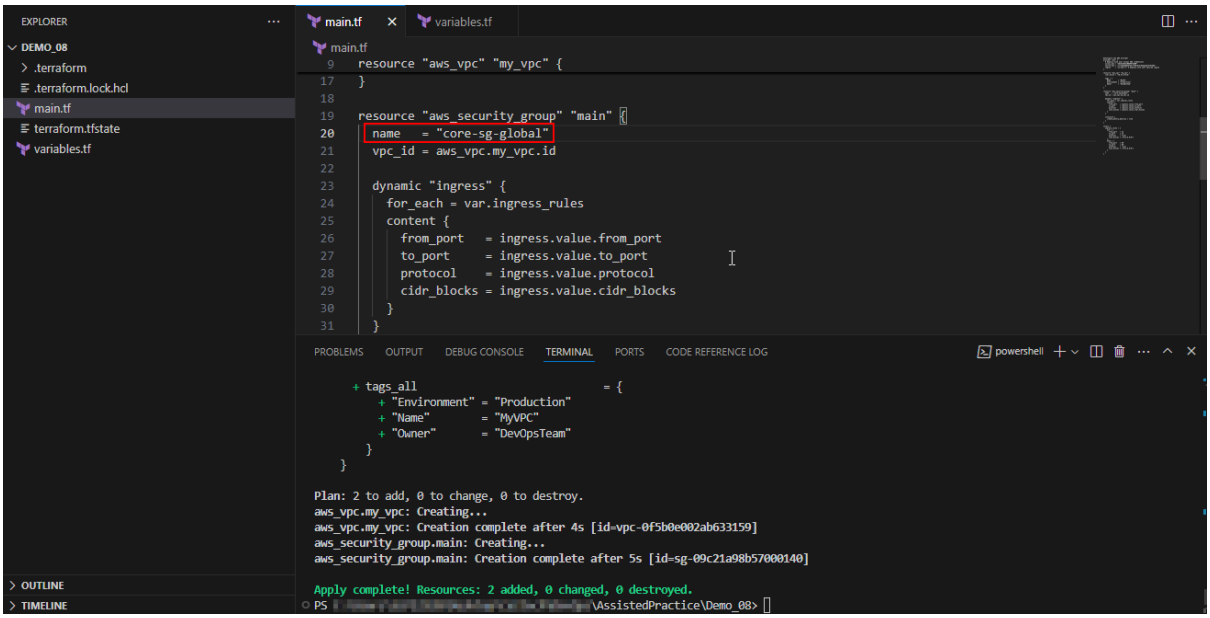


3.2 Apply the configuration using the following command to deploy the changes, as shown in the screenshot:
terraform apply -auto-approve

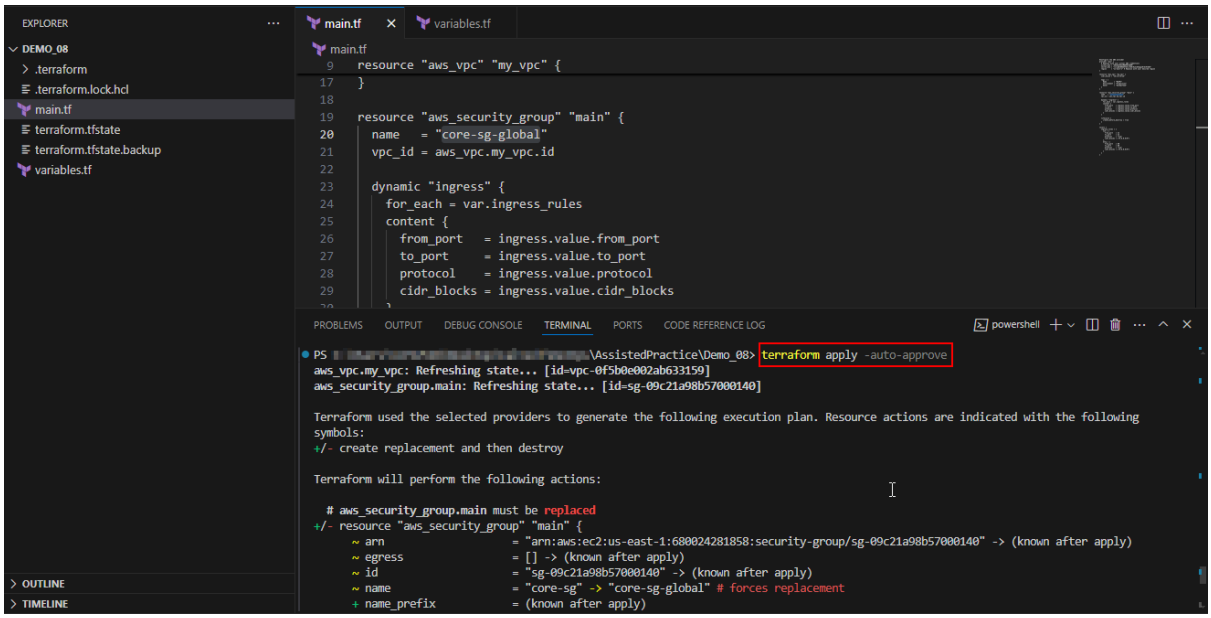


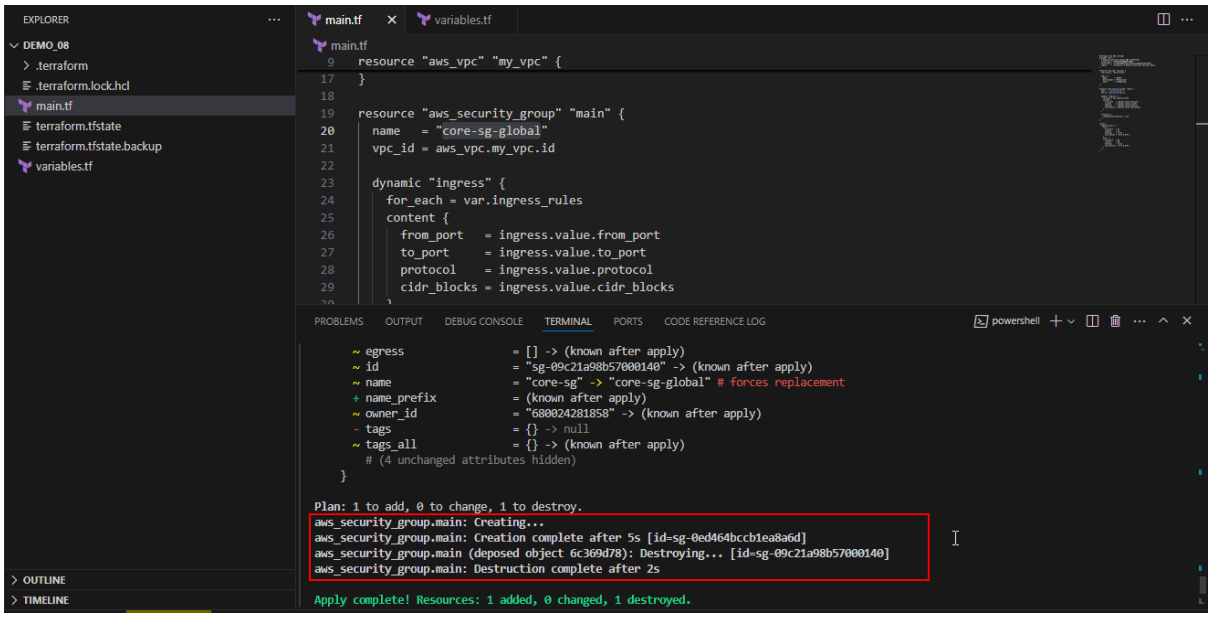


3.3 Modify the name of the security group to **core-sg-global**, as shown in the screenshot:



3.4 Apply the configuration using the following command to ensure the new security group is created before the old one is destroyed:
terraform apply -auto-approve

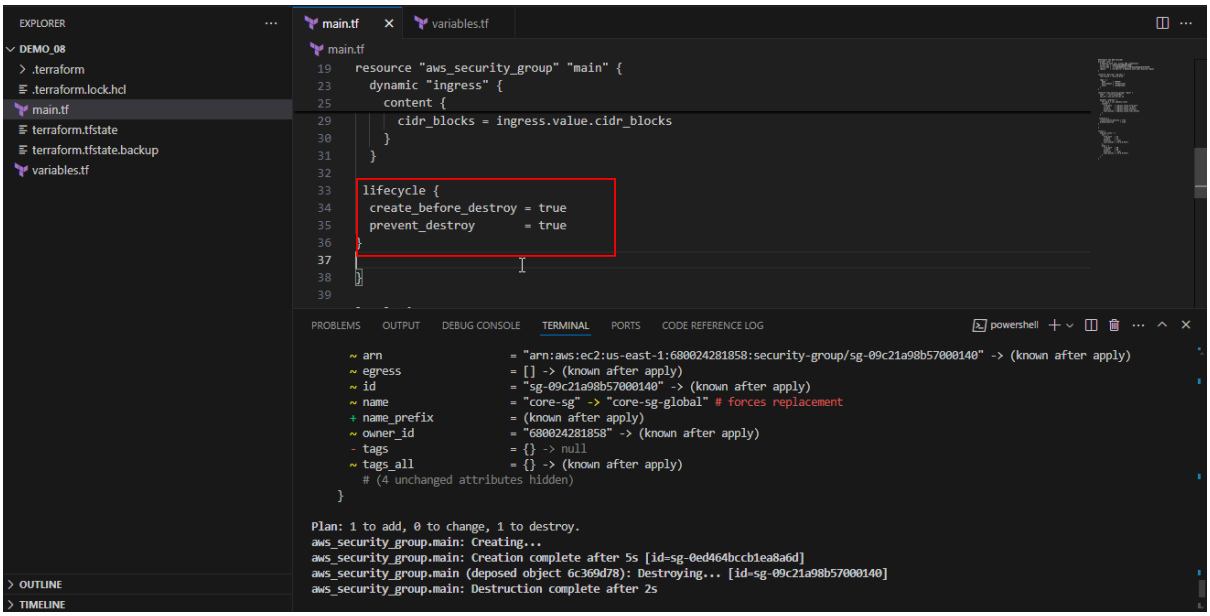




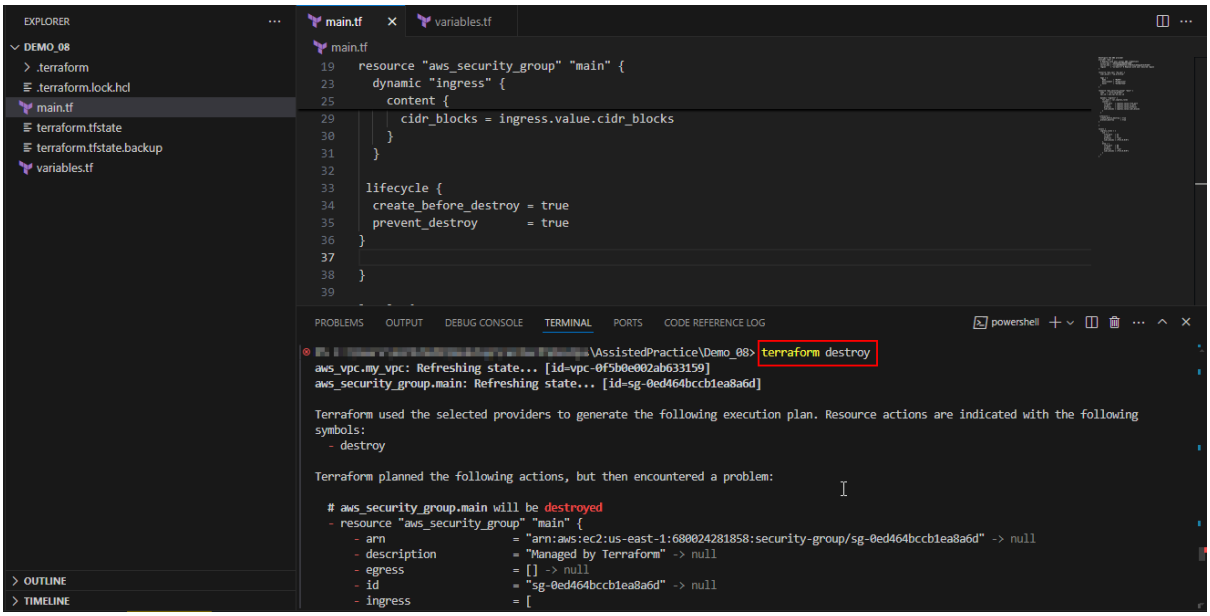
Step 4: Implement and test prevent_destroy

4.1 Add the **prevent_destroy** directive to the lifecycle block of the security group to prevent accidental deletion**:**

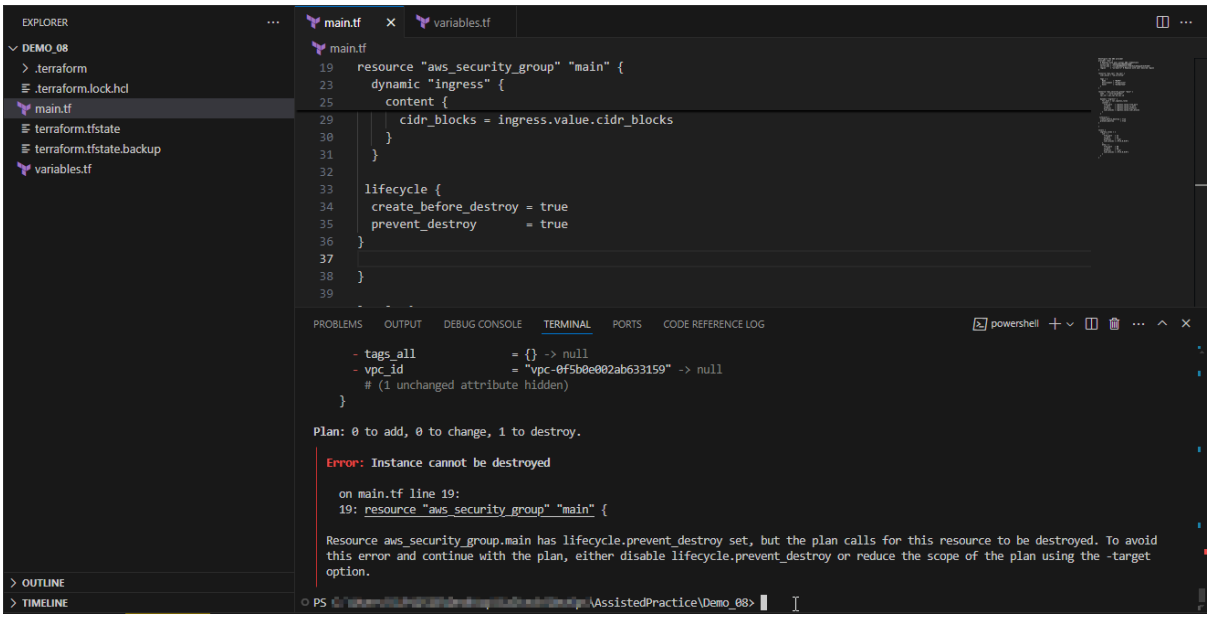
```
lifecycle {
  create_before_destroy = true
  prevent_destroy       = true
}
```



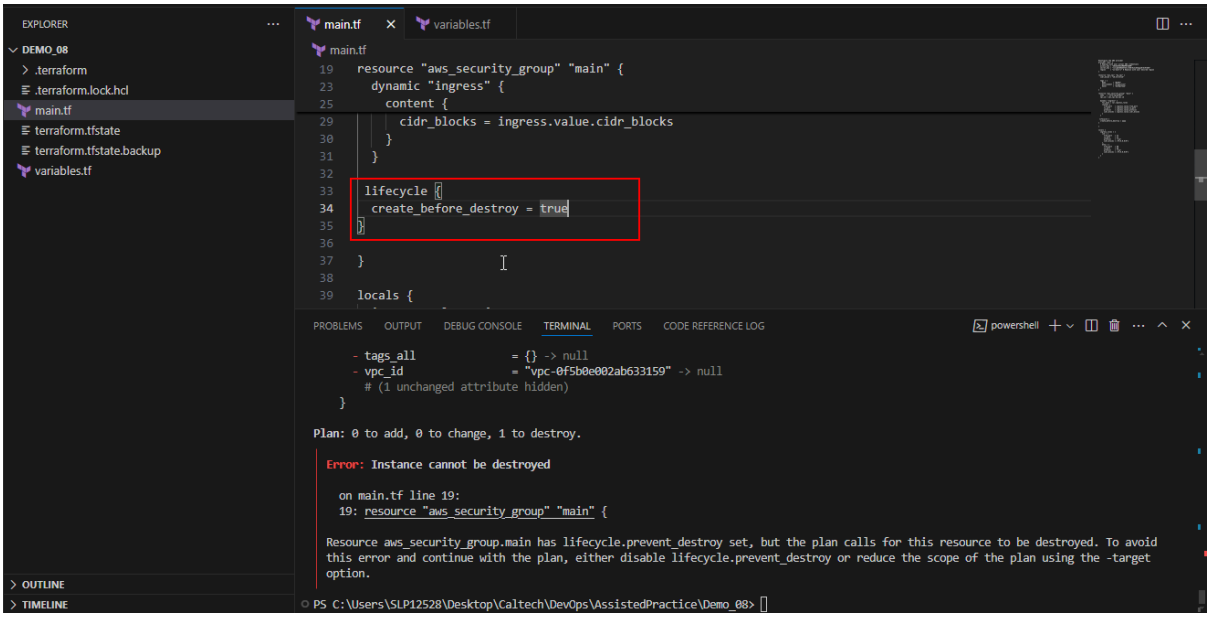
4.2 Execute the following command to destroy the infrastructure:
terraform destroy



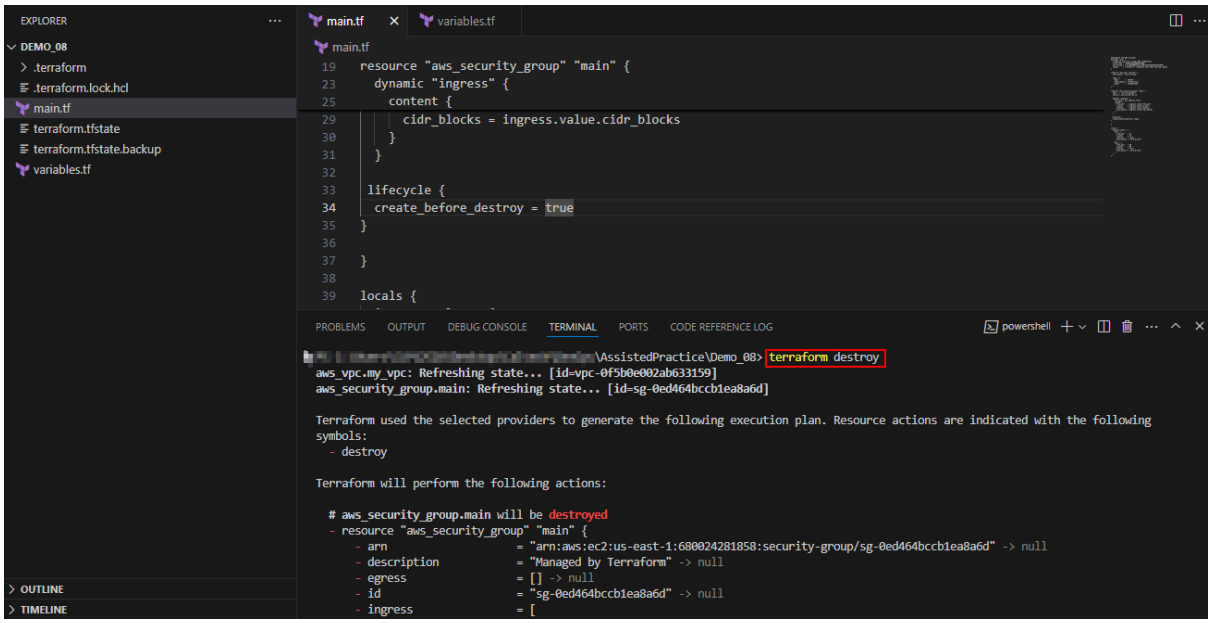
You will see the following error:



4.3 Remove the **prevent_destroy** attribute, as shown:



4.4 Execute the following command to destroy the infrastructure:
terraform destroy



By following these steps, you have successfully managed the Terraform resource lifecycle for infrastructure deployment.