

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" {

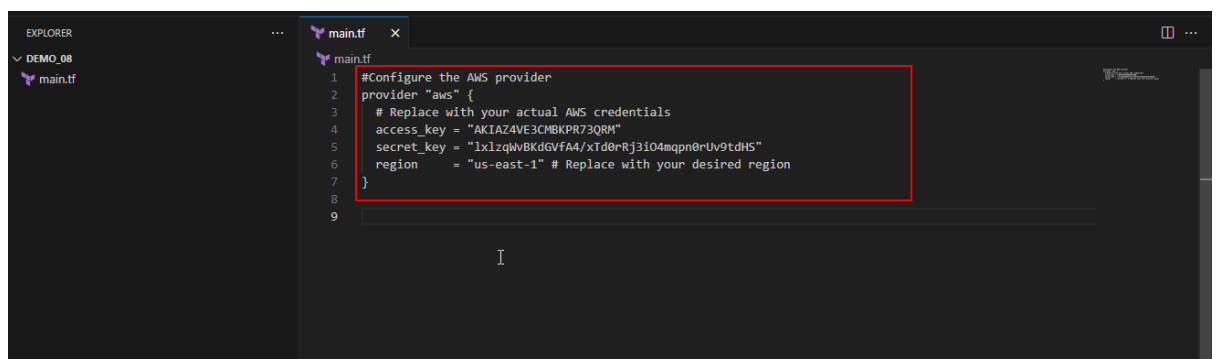
Replace with your actual AWS credentials

access_key = "YOUR_ACCESS_KEY"

secret_key = "YOUR_SECRET_KEY"

region = "us-east-1" # Replace with your desired region

}

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'DEMO_08' with a file 'main.tf'. The main editor window displays the content of 'main.tf'. The code is as follows:

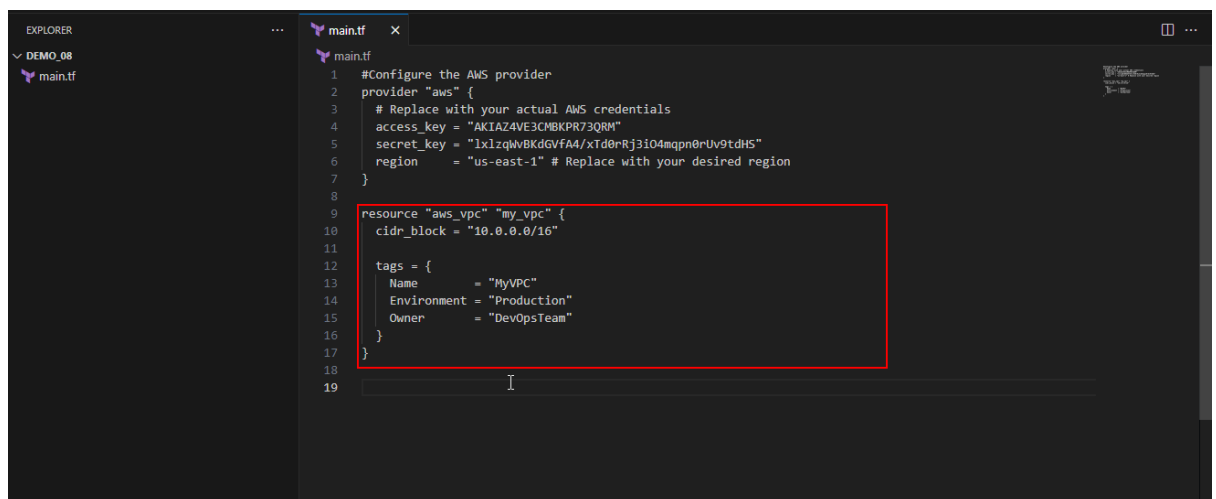
```
1 #Configure the AWS provider
2 provider "aws" {
3   # Replace with your actual AWS credentials
4   access_key = "AKIAZ4VE3CMBKPR73QRM"
5   secret_key = "1x1zqWvBKdGVfAA/xTd0rRj3iO4mqpn0rUV9tdHS"
6   region     = "us-east-1" # Replace with your desired region
7 }
8
9
```

The code block from line 2 to line 7 is highlighted with a red rectangular box. The cursor is positioned at the end of line 9.

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

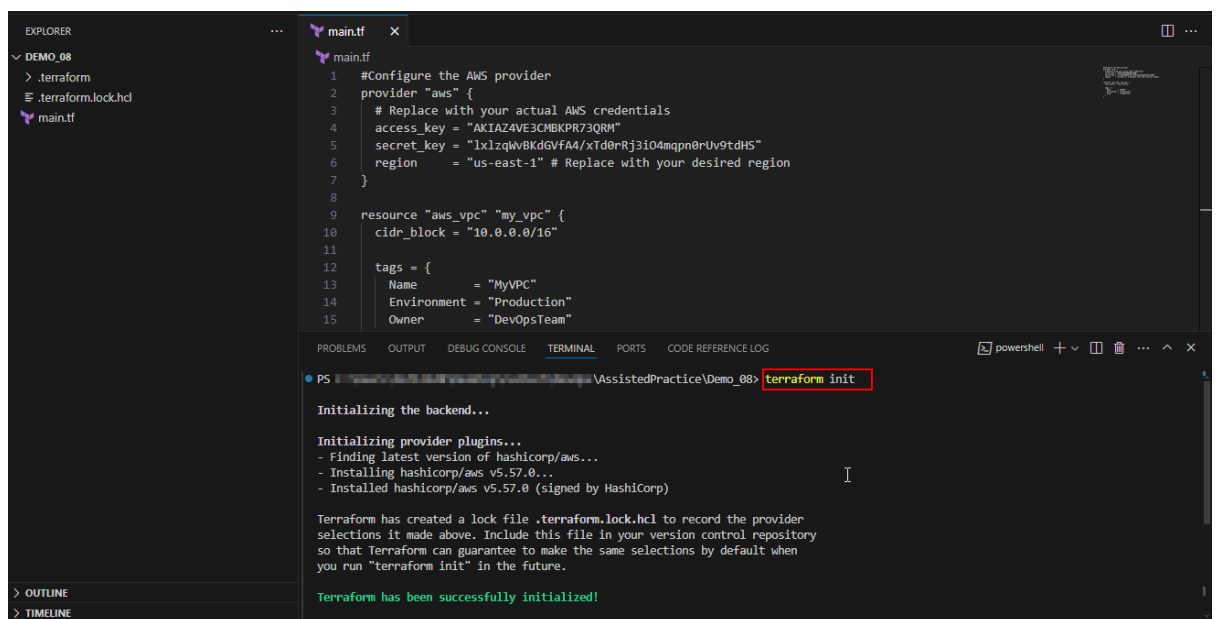
```
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"

  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



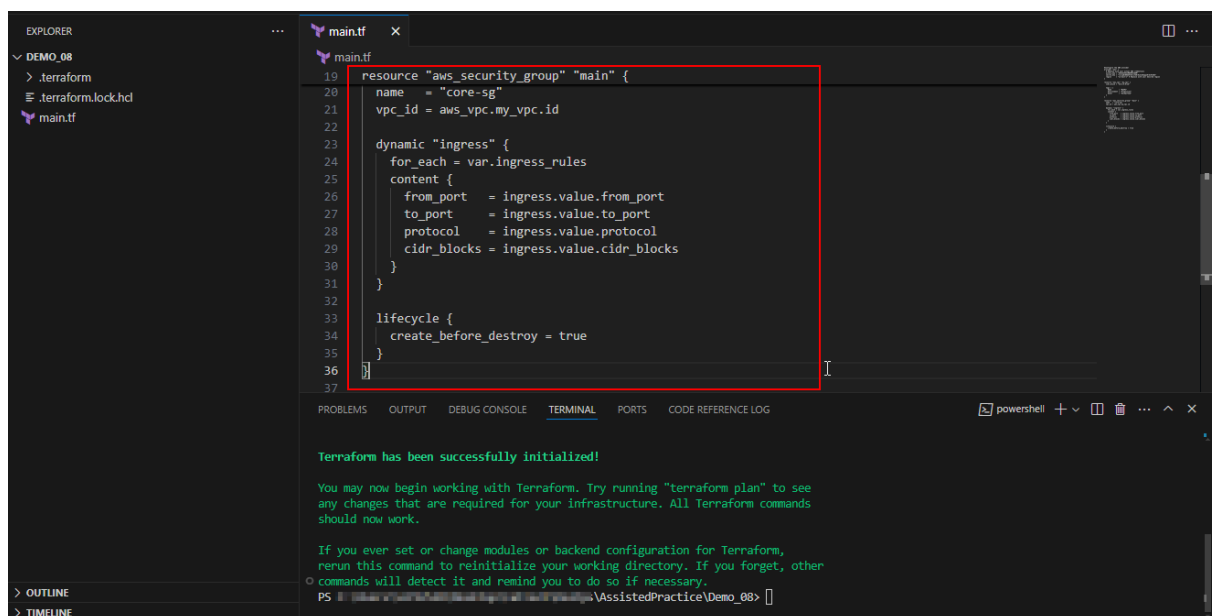
Step 2: Implement a dynamic security group with lifecycle modifications

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

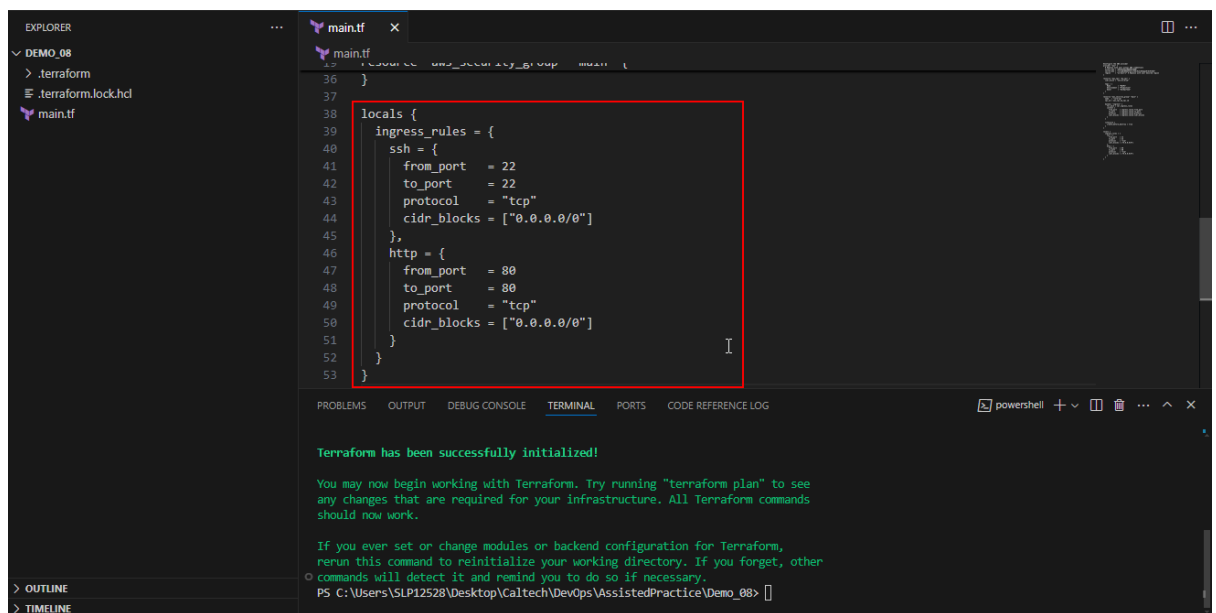
  dynamic "ingress" {
    for_each = var.ingress_rules
    content {
      from_port = ingress.value.from_port
      to_port   = ingress.value.to_port
      protocol  = ingress.value.protocol
      cidr_blocks = ingress.value.cidr_blocks
    }
  }

  lifecycle {
    create_before_destroy = true
  }
}
```



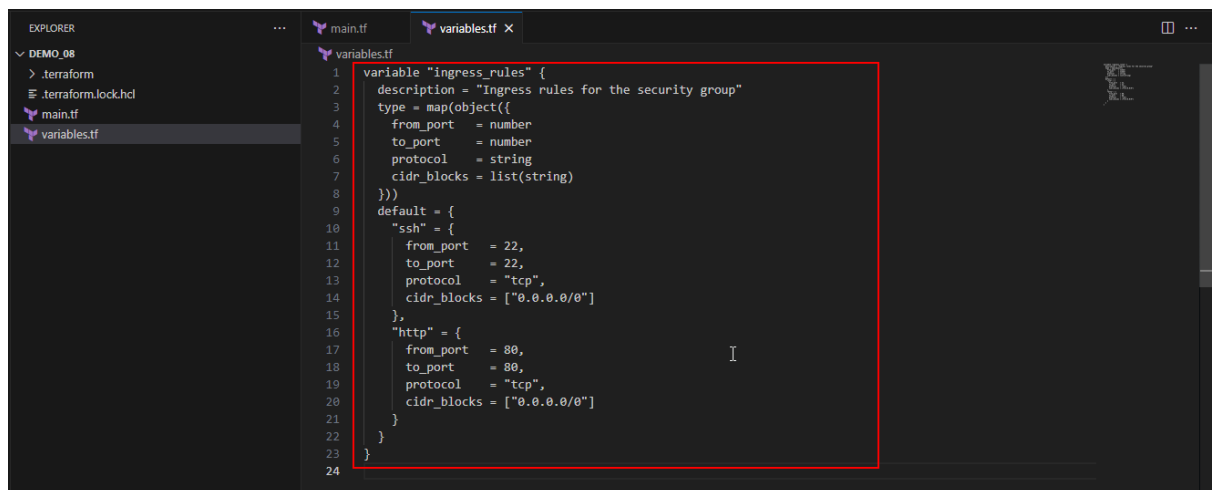
2.2 Define ingress rules using local variables or direct definition as shown in the screenshot:

```
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"]
    }
  }
}
```



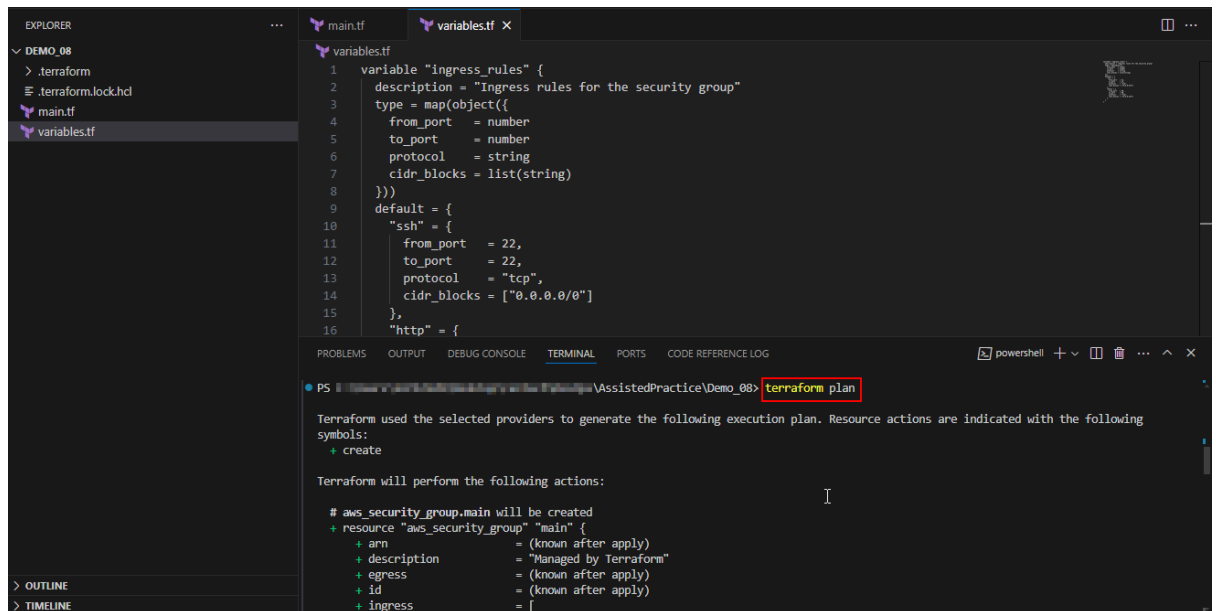
2.3 Create a file named **variables.tf** and add the following variable block, as shown in the screenshot:

```
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



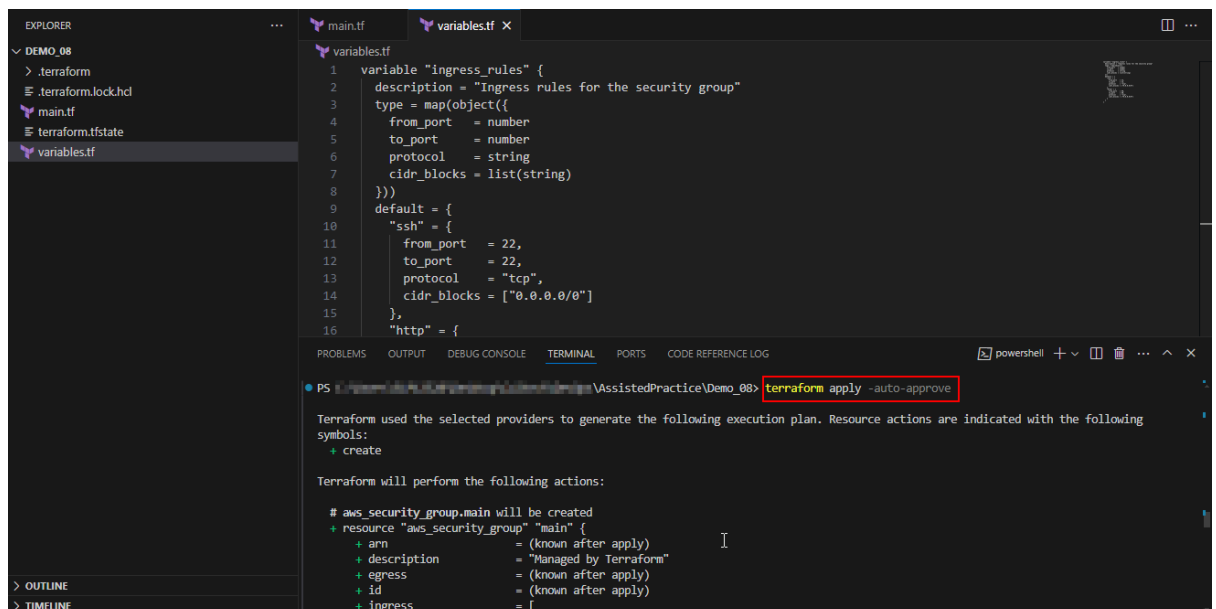
The screenshot shows the Visual Studio Code interface. The Explorer pane on the left shows a project named 'DEMO_08' with files: '.terraform', '.terraform.lock.hcl', 'main.tf', and 'variables.tf'. The 'variables.tf' file is selected and its content is displayed in the editor. The content of 'variables.tf' is as follows:

```
1 variable "ingress_rules" {
2   description = "Ingress rules for the security group"
3   type = map(object({
4     from_port = number
5     to_port   = number
6     protocol  = string
7     cidr_blocks = list(string)
8   }))
9   default = {
10     "ssh" = {
11       from_port = 22,
12       to_port   = 22,
13       protocol  = "tcp",
14       cidr_blocks = ["0.0.0.0/0"]
15     },
16     "http" = {
```

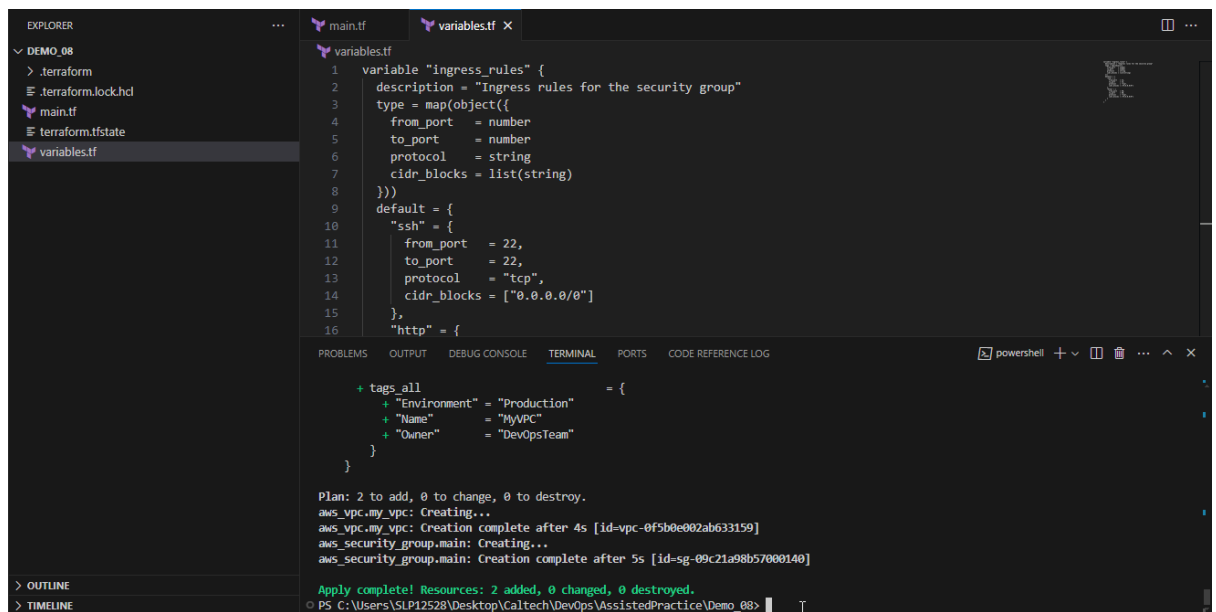
The Terminal pane at the bottom shows the command 'terraform plan' being executed. The output indicates that Terraform will create an 'aws_security_group' resource named 'main'. The actions to be performed are listed as follows:

```
# aws_security_group.main will be created
+ resource "aws_security_group" "main" {
+   arn              = (known after apply)
+   description      = "Managed by Terraform"
+   egress            = (known after apply)
+   id                = (known after apply)
+   ingress           = [
```

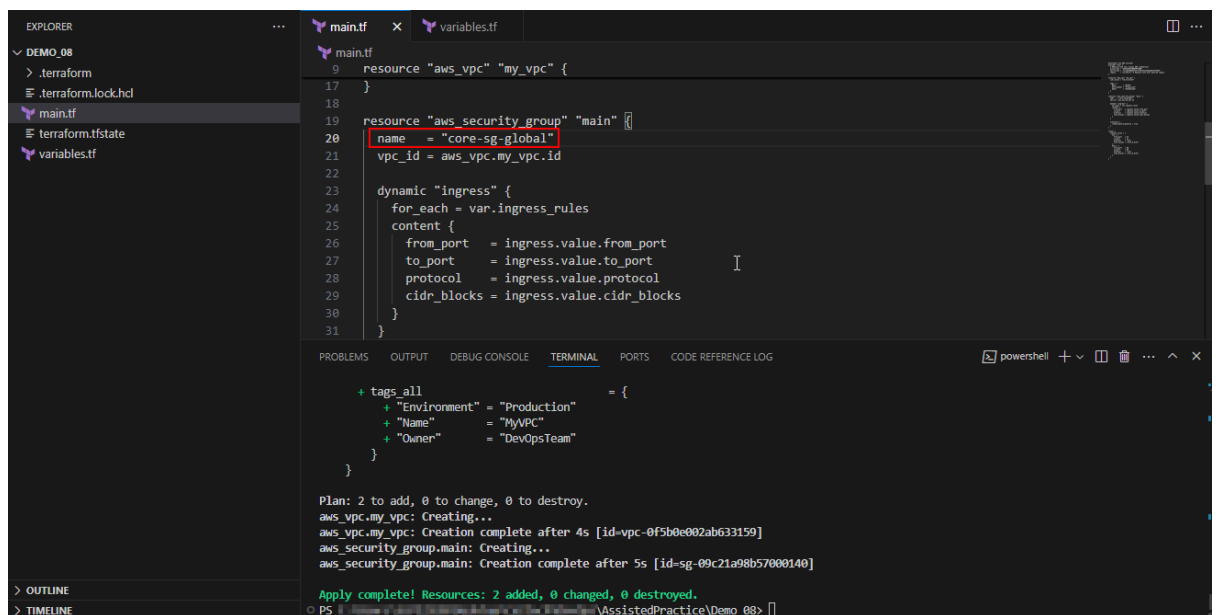
- 3.2 Apply the configuration using the following command to deploy the changes, as shown in the screenshot:
terraform apply -auto-approve



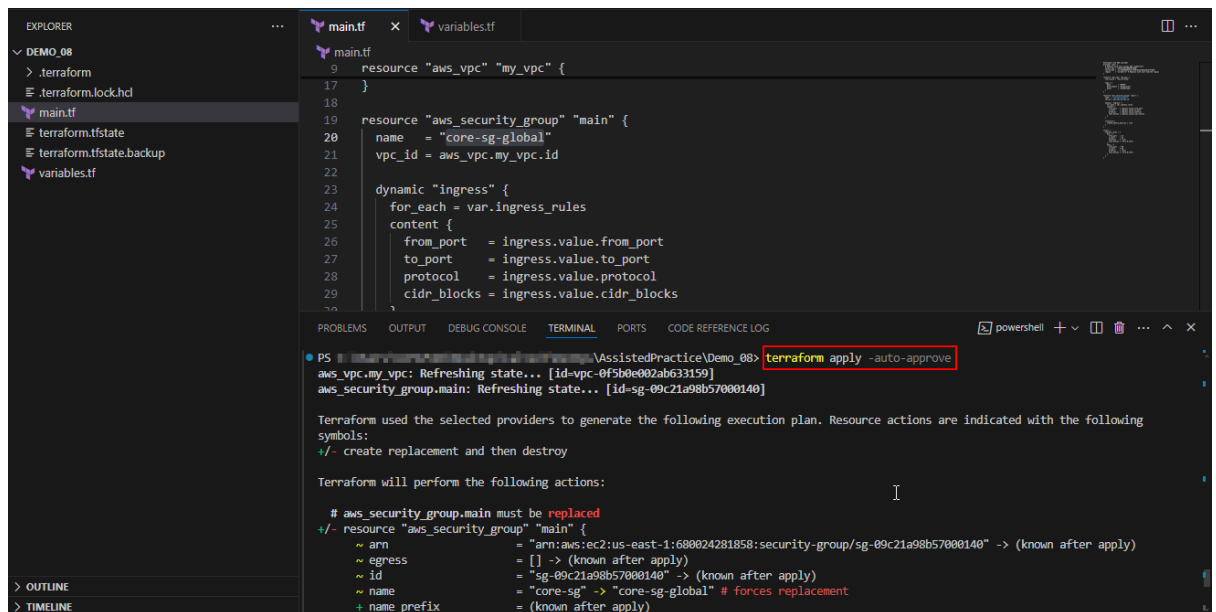
The screenshot shows the Visual Studio Code interface. The Explorer pane on the left shows the same project 'DEMO_08' with an additional file 'terraform.tfstate' now present. The 'variables.tf' file is still selected. The Terminal pane at the bottom shows the command 'terraform apply -auto-approve' being executed. The output is identical to the previous screenshot, showing the plan for creating the 'aws_security_group' resource.



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**



The screenshot shows the VS Code interface with the Explorer pane on the left displaying the project structure for 'DEMO_08'. The main editor shows the 'main.tf' file with Terraform configuration for an AWS VPC and a Security Group. The terminal pane at the bottom shows the command 'terraform apply -auto-approve' being executed. The output indicates that the security group 'main' must be replaced and shows the planned actions for the 'aws_security_group.main' resource.

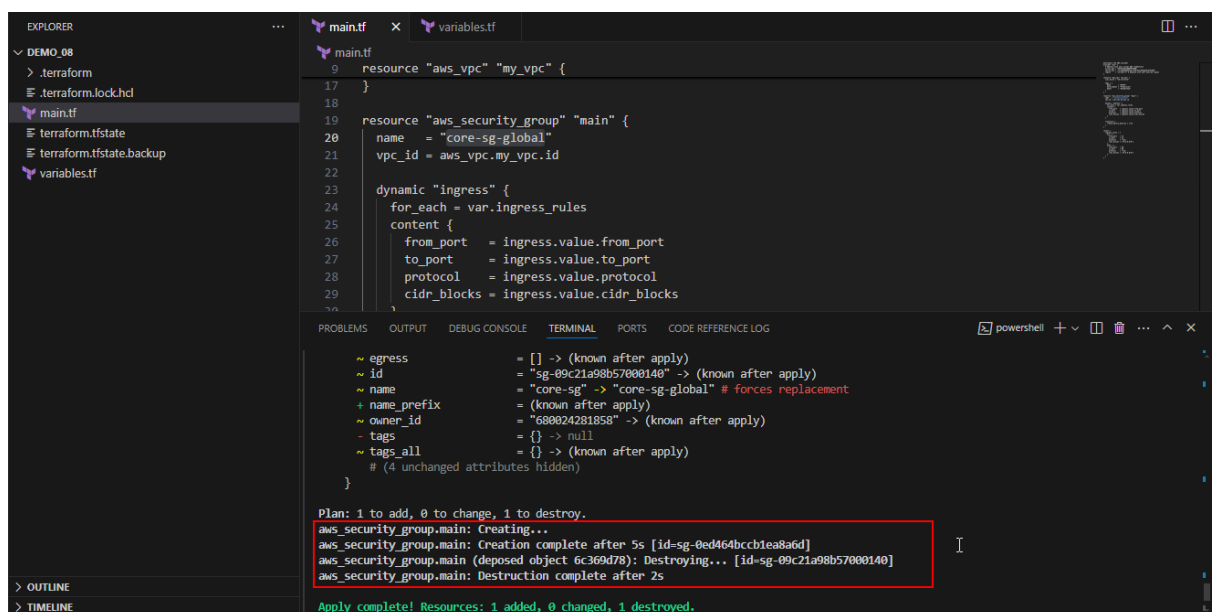
```
main.tf
9 resource "aws_vpc" "my_vpc" {
17 }
18
19 resource "aws_security_group" "main" {
20   name = "core-sg-global"
21   vpc_id = aws_vpc.my_vpc.id
22
23   dynamic "ingress" {
24     for_each = var.ingress_rules
25     content {
26       from_port = ingress.value.from_port
27       to_port = ingress.value.to_port
28       protocol = ingress.value.protocol
29       cidr_blocks = ingress.value.cidr_blocks
30     }
31   }
32 }
```

```
PS C:\AssistedPractice\Demo_08> terraform apply -auto-approve
aws_vpc.my_vpc: Refreshing state... [id=vpc-0f5b0e002ab633159]
aws_security_group.main: Refreshing state... [id=sg-09c21a98b57000140]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
+/- create replacement and then destroy

Terraform will perform the following actions:

# aws_security_group.main must be replaced
+/- resource "aws_security_group" "main" {
  ~ arn = "arn:aws:ec2:us-east-1:680024281858:security-group/sg-09c21a98b57000140" -> (known after apply)
  ~ egress = [] -> (known after apply)
  ~ id = "sg-09c21a98b57000140" -> (known after apply)
  ~ name = "core-sg" -> "core-sg-global" # forces replacement
  ~ name_prefix = (known after apply)
}
```



This screenshot shows the continuation of the Terraform apply process. The terminal output displays the detailed plan for the 'aws_security_group.main' resource, including attributes like 'arn', 'egress', 'id', 'name', 'name_prefix', 'owner_id', 'tags', and 'tags_all'. It then shows the execution progress: 'Creation complete after 5s', 'Destroying...' (for the old resource), and 'Destruction complete after 2s'. The final summary states 'Apply complete! Resources: 1 added, 0 changed, 1 destroyed.'

```
~ egress = [] -> (known after apply)
~ id = "sg-09c21a98b57000140" -> (known after apply)
~ name = "core-sg" -> "core-sg-global" # forces replacement
+ name_prefix = (known after apply)
~ owner_id = "680024281858" -> (known after apply)
~ tags = {} -> null
~ tags_all = {} -> (known after apply)
# (4 unchanged attributes hidden)
}

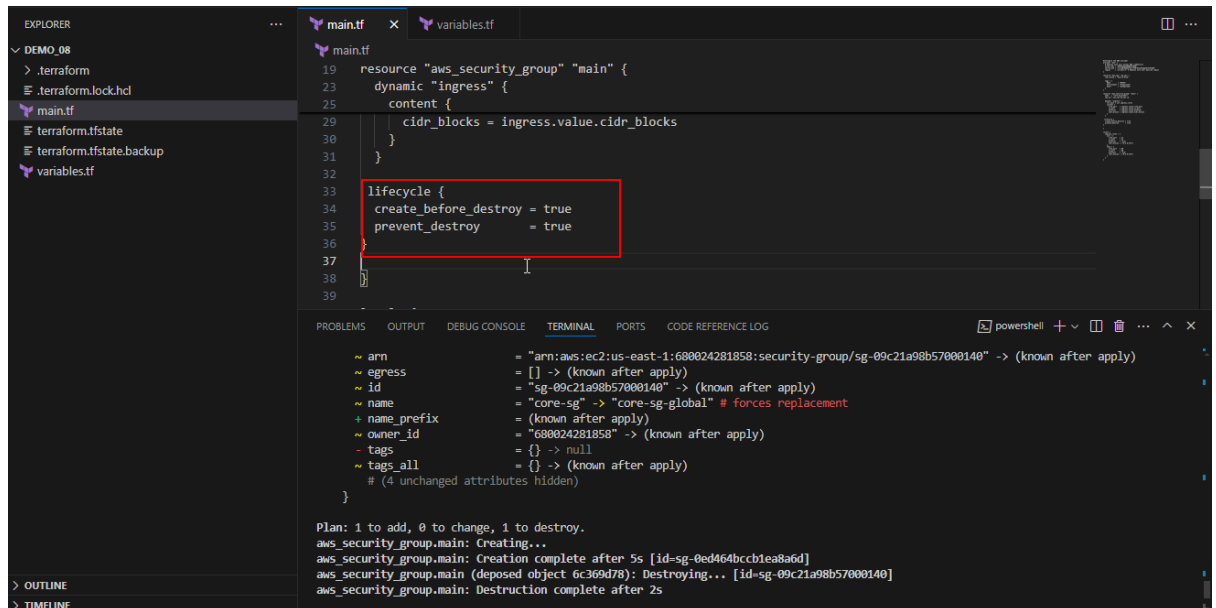
Plan: 1 to add, 0 to change, 1 to destroy.
aws_security_group.main: Creating...
aws_security_group.main: Creation complete after 5s [id=sg-0ed464bccb1ea8a6d]
aws_security_group.main (deposed object 6c369d78): Destroying... [id=sg-09c21a98b57000140]
aws_security_group.main: Destruction complete after 2s

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

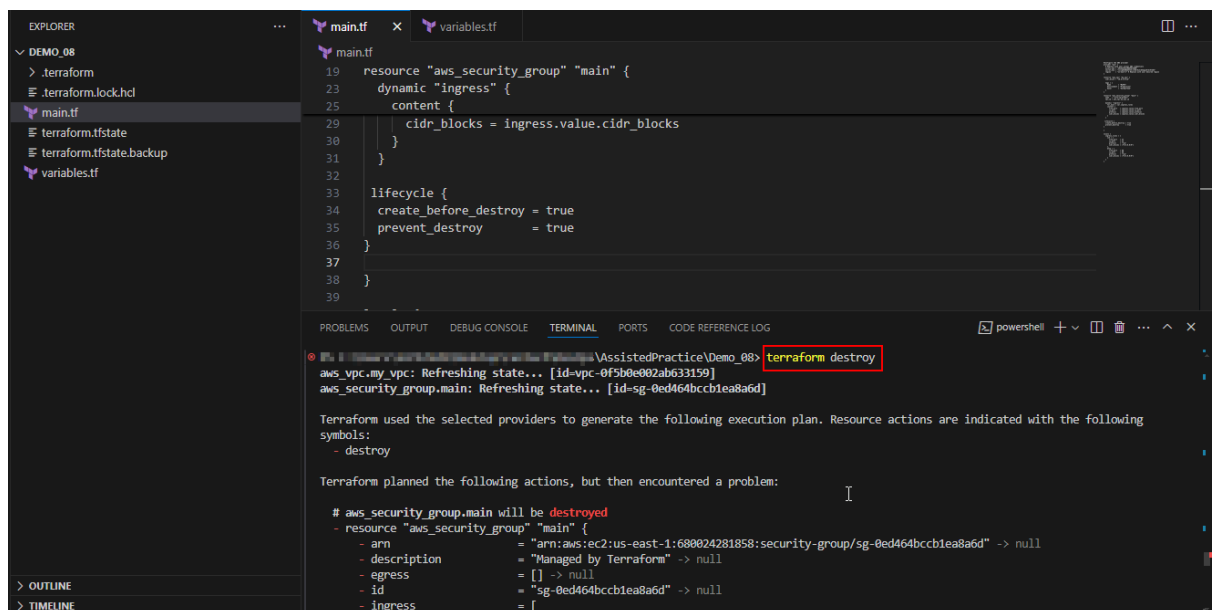

Step 4: Implement and test prevent_destroy

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

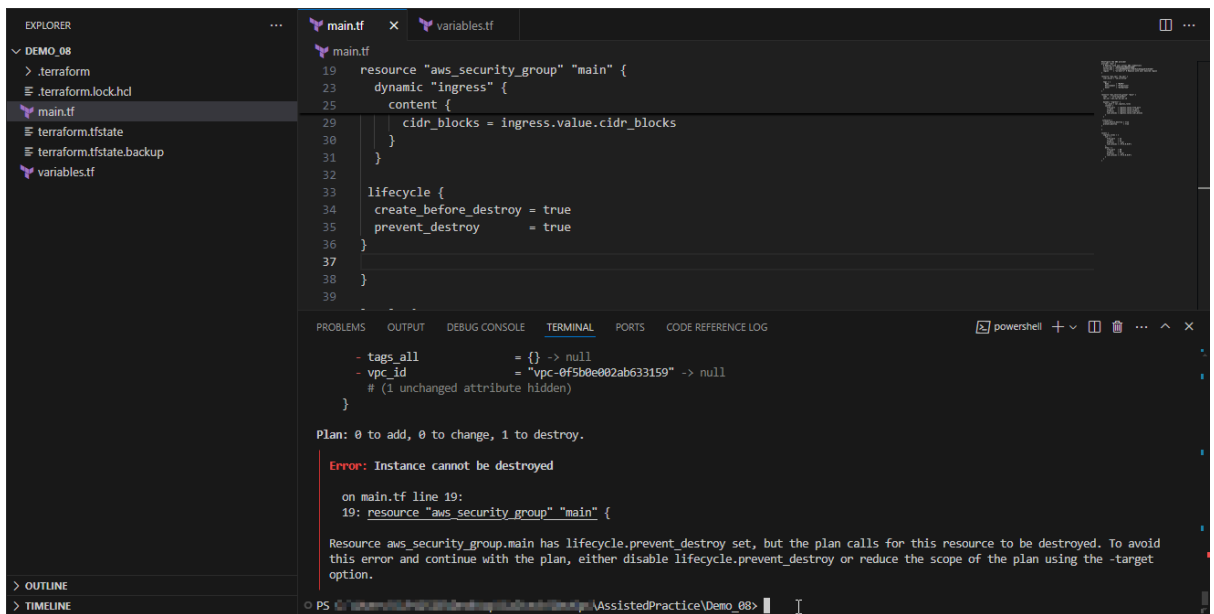
```
lifecycle {
  create_before_destroy = true
  prevent_destroy      = true
}
```



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



You will see the following error:



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

```
19 resource "aws_security_group" "main" {
23   dynamic "ingress" {
25     content {
29       | cidr_blocks = ingress.value.cidr_blocks
30     }
31   }
32 }
33 lifecycle {
34   create_before_destroy = true
35   prevent_destroy       = true
36 }
37 }
38 }
39 }
```

The terminal window shows the output of a Terraform plan and an error message:

```
- tags_all      = {} -> null
- vpc_id        = "vpc-0f5b0e002ab633159" -> null
# (1 unchanged attribute hidden)

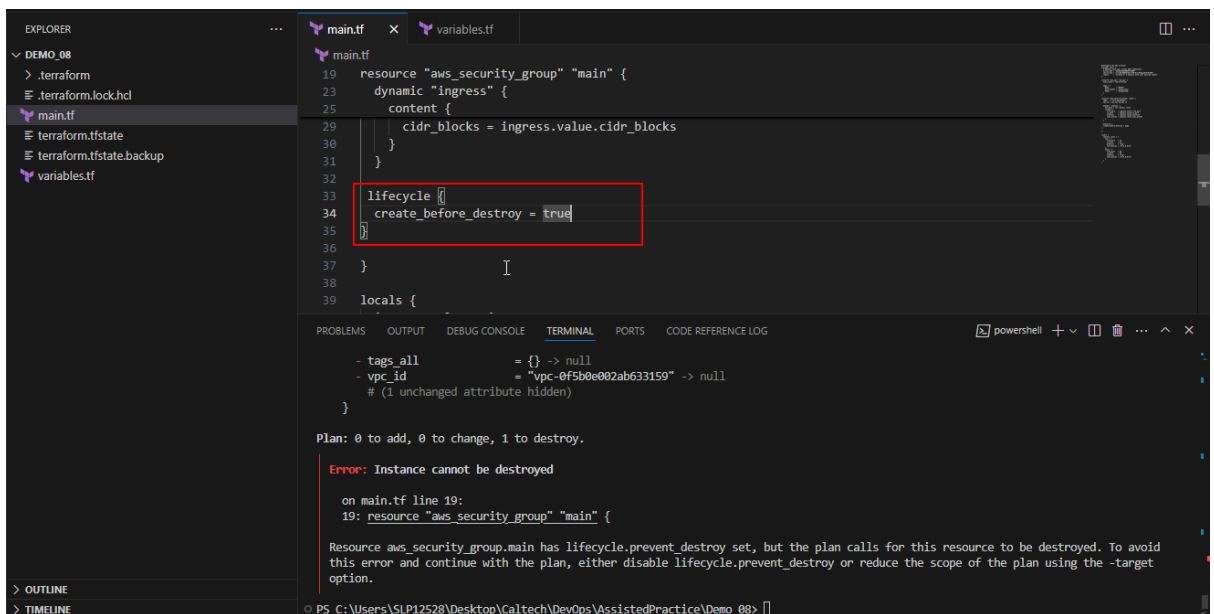
Plan: 0 to add, 0 to change, 1 to destroy.

Error: Instance cannot be destroyed

on main.tf line 19:
19: resource "aws_security_group" "main" {

Resource aws_security_group.main has lifecycle.prevent_destroy set, but the plan calls for this resource to be destroyed. To avoid this error and continue with the plan, either disable lifecycle.prevent_destroy or reduce the scope of the plan using the -target option.
```

4.3 Remove the `prevent_destroy` attribute, as shown:

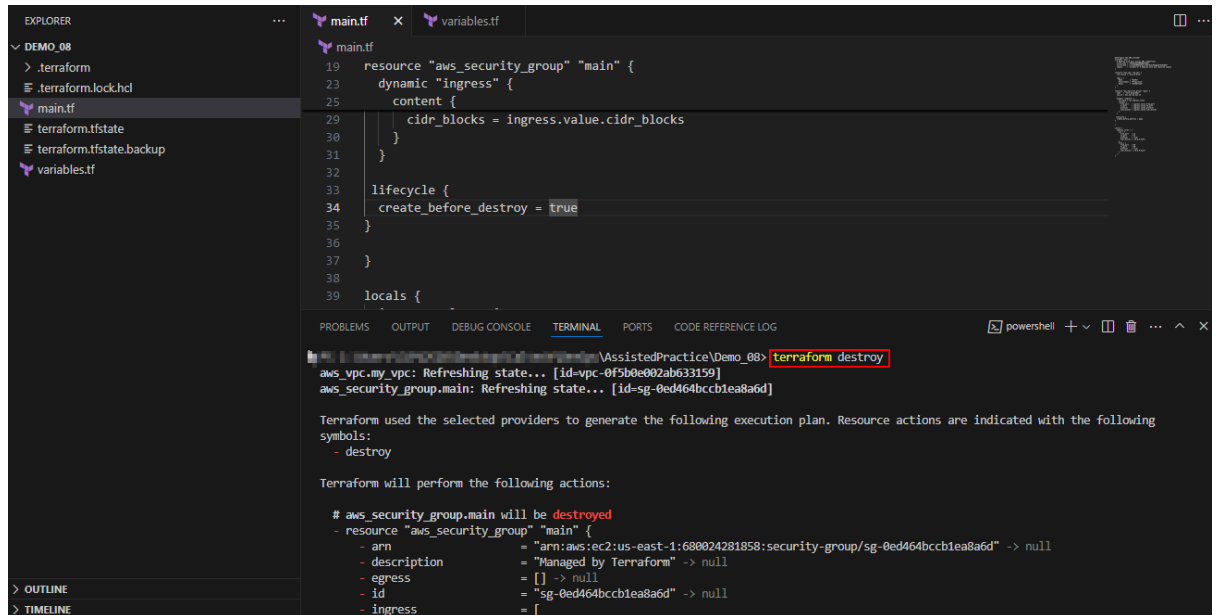


The screenshot shows the VS Code interface with the `main.tf` file. The `lifecycle` block has been modified to remove the `prevent_destroy` attribute. The code is as follows:

```
19 resource "aws_security_group" "main" {
23   dynamic "ingress" {
25     content {
29       | cidr_blocks = ingress.value.cidr_blocks
30     }
31   }
32 }
33 lifecycle {
34   create_before_destroy = true
35 }
36 }
37 }
38 }
39 }
```

The terminal window shows the same error message as before, indicating that the resource cannot be destroyed because `prevent_destroy` is still set to `true`.

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



The screenshot shows a Visual Studio Code editor with two files open: `main.tf` and `variables.tf`. The `main.tf` file contains Terraform configuration for an AWS security group. The terminal window at the bottom shows the execution of the `terraform destroy` command, which refreshes the state and displays the plan to destroy the `aws_security_group.main` resource.

```
main.tf
19 resource "aws_security_group" "main" {
23   dynamic "ingress" {
25     content {
29       | cidr_blocks = ingress.value.cidr_blocks
30     }
31   }
32 }
33 lifecycle {
34   create_before_destroy = true
35 }
36 }
37 }
38 }
39 locals {
```

```
aws_vpc.my_vpc: Refreshing state... [id=vpc-0f5b0e002ab633159]
aws_security_group.main: Refreshing state... [id=sg-0ed464bccb1ea8a6d]

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:

# aws_security_group.main will be destroyed
- resource "aws_security_group" "main" {
  - arn              = "arn:aws:ec2:us-east-1:680024281858:security-group/sg-0ed464bccb1ea8a6d" -> null
  - description      = "Managed by Terraform" -> null
  - egress            = [] -> null
  - id                = "sg-0ed464bccb1ea8a6d" -> null
  - ingress           = []
```

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