ANSHIKA SRIVASTAVA

ROLL NUMBER – R2142220907

SAP ID – 500107049

LAB EXERCISE 8

Lab Exercise 8– Terraform Multiple tfvars Files

Objective:

Learn how to use multiple tfvars files in Terraform for different environments.

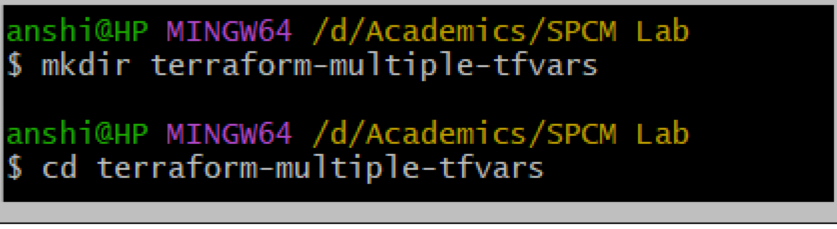
Prerequisites:

- Terraform installed on your machine.
- Basic knowledge of Terraform configuration and variables.

Steps:

1. Create a Terraform Directory:

```
mkdir terraform-multiple-tfvars
cd terraform-multiple-tfvars
```

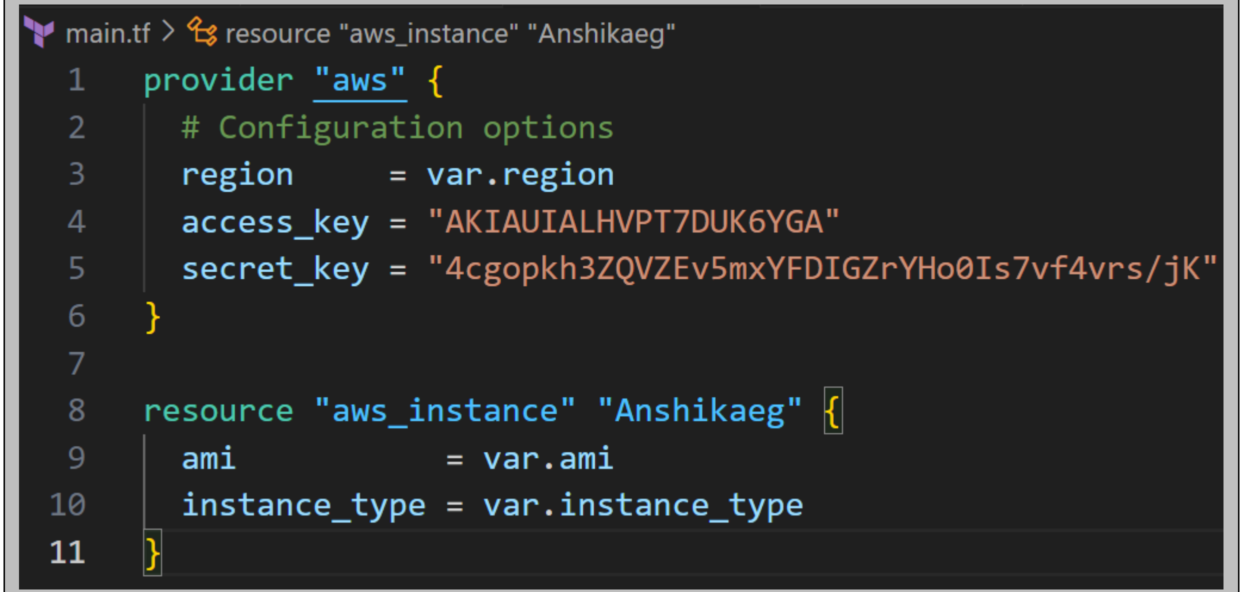


```
anshi@HP MINGW64 /d/Academics/SPCM Lab
$ mkdir terraform-multiple-tfvars

anshi@HP MINGW64 /d/Academics/SPCM Lab
$ cd terraform-multiple-tfvars
```

- Create Terraform Configuration Files:
- Create a file named main.tf:

# main.tf

```
provider "aws" {
  region = var.region
}

resource "aws_instance" "example" {
  ami           = var.ami
  instance_type = var.instance_type
}
```

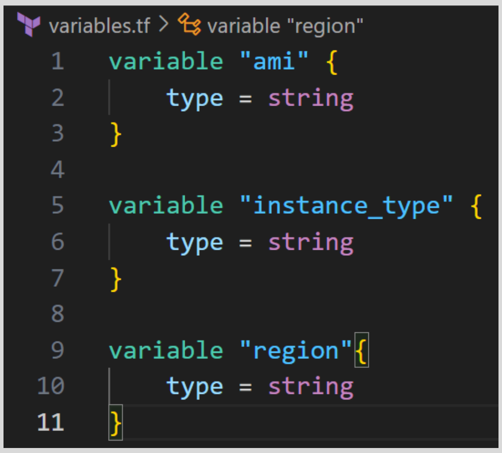


```
provider "aws" {
  # Configuration options
  region     = var.region
  access_key = "AKIAUIALHVPT7DUK6YGA"
  secret_key = "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"
}

resource "aws_instance" "Anshikaeg" {
  ami           = var.ami
  instance_type = var.instance_type
}
```

- Create a file named variables.tf:

# variables.tf

```
variable "ami" {
  type = string
}

variable "instance_type" {
  type = string
}
```



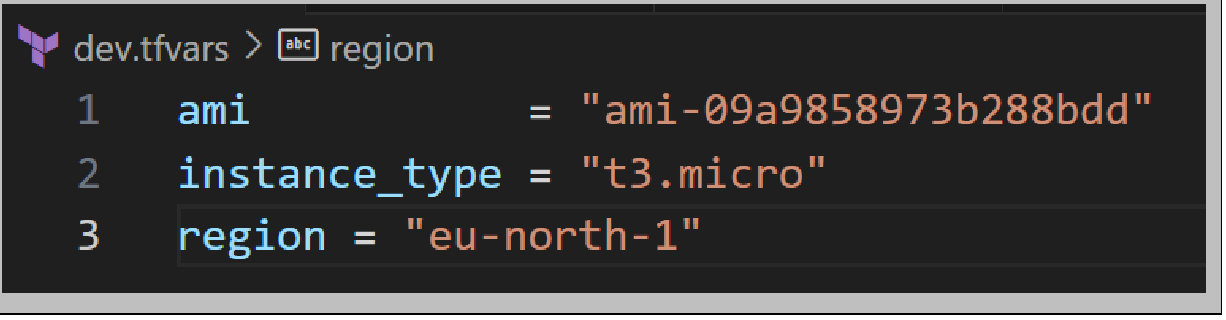
```
variable "ami" {
    type = string
}

variable "instance_type" {
    type = string
}

variable "region"{
    type = string
}
```

2. Create Multiple tfvars Files:

- Create a file named dev.tfvars:

# dev.tfvars

```
ami          = "ami-0123456789abcdef0"
instance_type = "t2.micro"
```



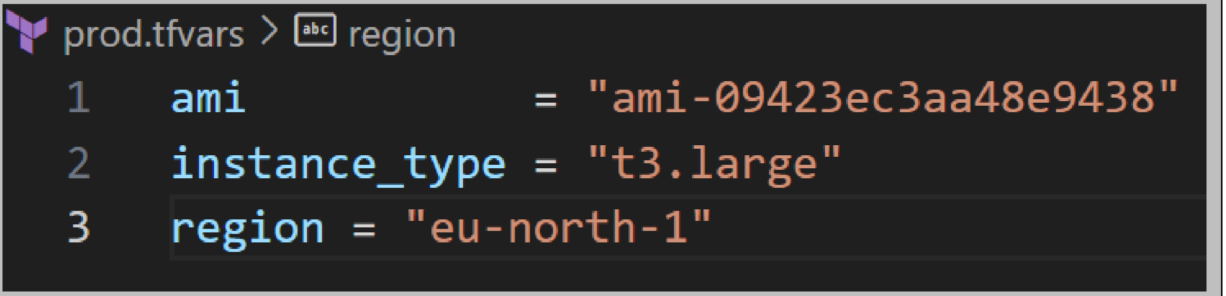
dev.tfvars > region

```
ami           = "ami-09a9858973b288bdd"
instance_type = "t3.micro"
region = "eu-north-1"
```

- Create a file named prod.tfvars:

# prod.tfvars

```
ami          = "ami-9876543210fedcba0"
instance_type = "t2.large"
```



prod.tfvars > region

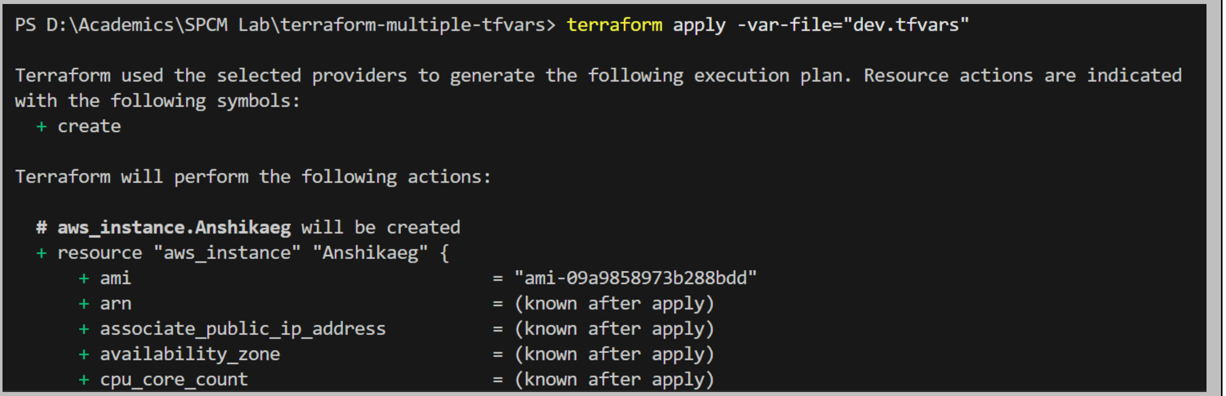
```
ami           = "ami-09423ec3aa48e9438"
instance_type = "t3.large"
region = "eu-north-1"
```

- In these files, provide values for the variables based on the environments.

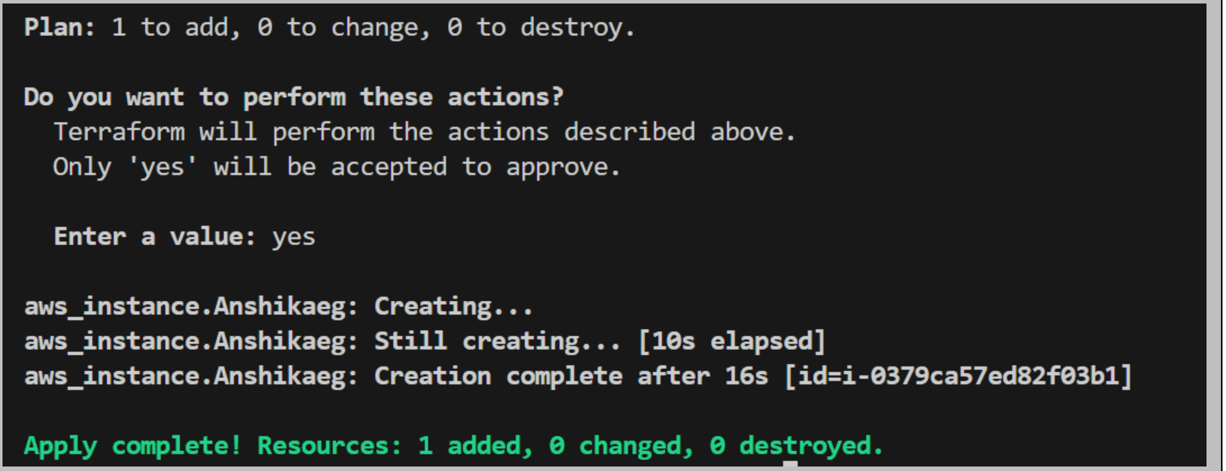
3. Initialize and Apply for Dev Environment:

- Run the following Terraform commands to initialize and apply the configuration for the dev environment:

```
terraform init

terraform apply -var-file="dev.tfvars"
```



```
PS D:\Academics\SPCM Lab\terraform-multiple-tfvars> terraform apply -var-file="dev.tfvars"

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.Anshikaeg will be created
  + resource "aws_instance" "Anshikaeg" {
      + ami                                  = "ami-09a9858973b288bdd"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```

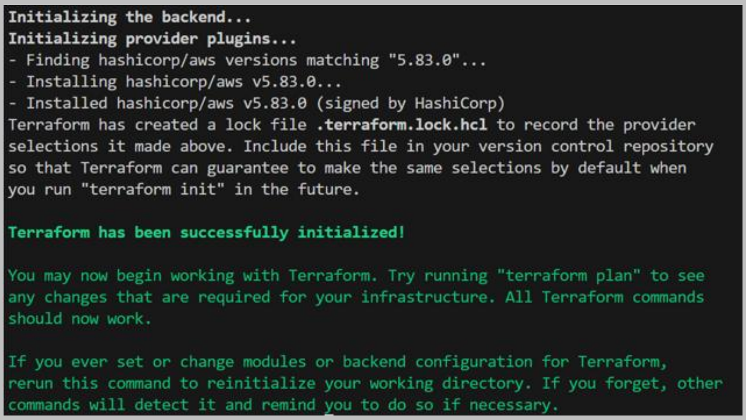


```
Plan: 1 to add, 0 to change, 0 to destroy.

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

aws_instance.Anshikaeg: Creating...
aws_instance.Anshikaeg: Still creating... [10s elapsed]
aws_instance.Anshikaeg: Creation complete after 16s [id=i-0379ca57ed82f03b1]

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

4. Initialize and Apply for Prod Environment:

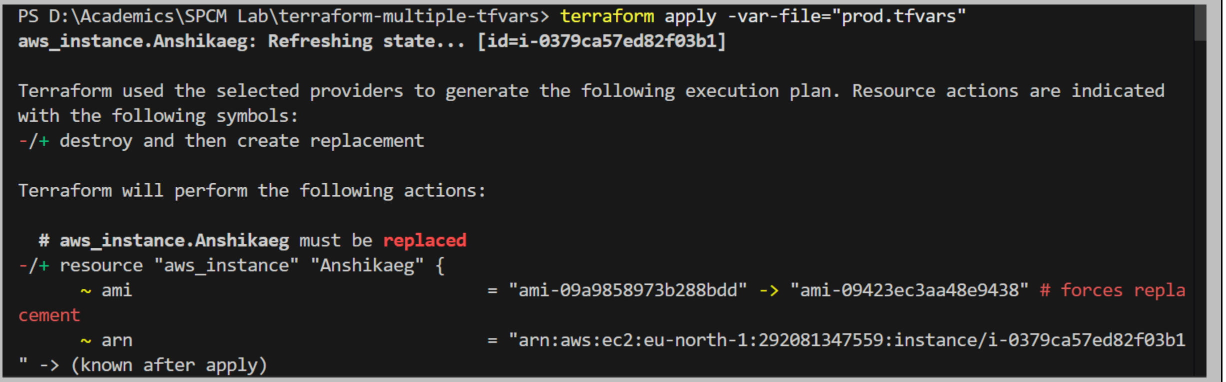
- Run the following Terraform commands to initialize and apply the configuration for the prod environment:

terraform init

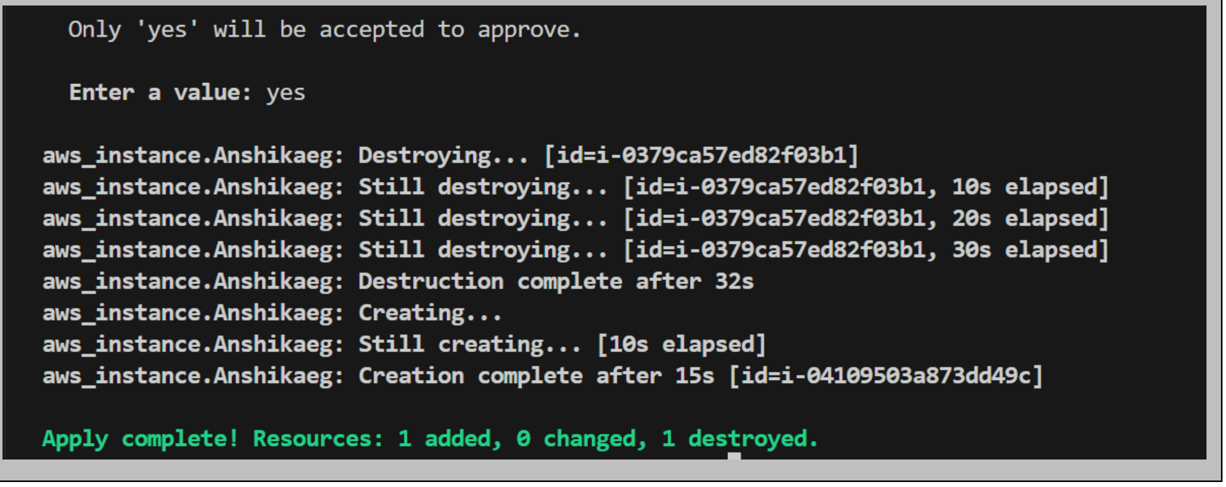


```
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.83.0"...
- Installing hashicorp/aws v5.83.0...
- Installed hashicorp/aws v5.83.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections 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.
```

terraform apply -var-file=”prod.tfvars”



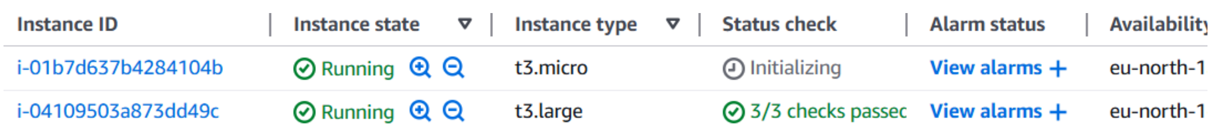
```
PS D:\Academics\SPCM Lab\terraform-multiple-tfvars> terraform apply -var-file="prod.tfvars"
aws_instance.Anshikaeg: Refreshing state... [id=i-0379ca57ed82f03b1]

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

Terraform will perform the following actions:

  # aws_instance.Anshikaeg must be replaced
-/+ resource "aws_instance" "Anshikaeg" {
      ~ ami                                  = "ami-09a9858973b288bdd" -> "ami-09423ec3aa48e9438" # forces repla
cement
      ~ arn                                  = "arn:aws:ec2:eu-north-1:292081347559:instance/i-0379ca57ed82f03b1
" -> (known after apply)
```



```
    Only 'yes' will be accepted to approve.

    Enter a value: yes

  aws_instance.Anshikaeg: Destroying... [id=i-0379ca57ed82f03b1]
  aws_instance.Anshikaeg: Still destroying... [id=i-0379ca57ed82f03b1, 10s elapsed]
  aws_instance.Anshikaeg: Still destroying... [id=i-0379ca57ed82f03b1, 20s elapsed]
  aws_instance.Anshikaeg: Still destroying... [id=i-0379ca57ed82f03b1, 30s elapsed]
  aws_instance.Anshikaeg: Destruction complete after 32s
  aws_instance.Anshikaeg: Creating...
  aws_instance.Anshikaeg: Still creating... [10s elapsed]
  aws_instance.Anshikaeg: Creation complete after 15s [id=i-04109503a873dd49c]

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

5. Test and Verify:

- Observe how different tfvars files are used to set variable values for different environments during the apply process.
- Access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified regions and instance types.

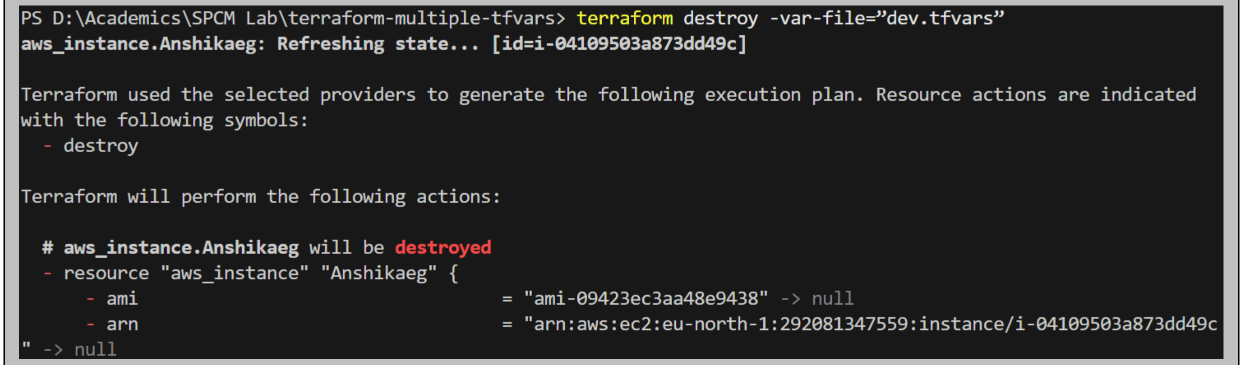


Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
i-01b7d637b4284104b	Running	t3.micro	Initializing	View alarms +	eu-north-1
i-04109503a873dd49c	Running	t3.large	3/3 checks passec	View alarms +	eu-north-1

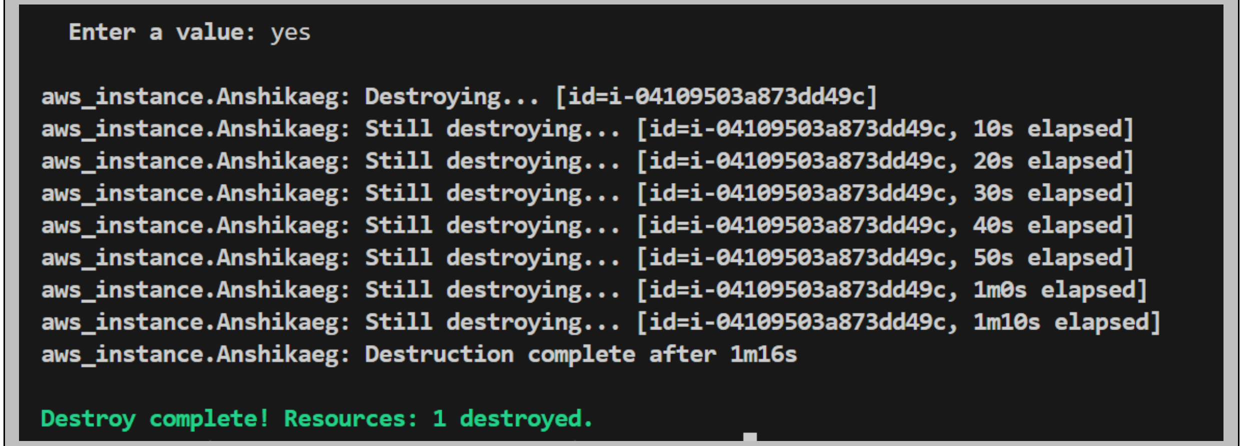
6. Clean Up:

- After testing, you can clean up resources:

terraform destroy -var-file=”dev.tfvars”

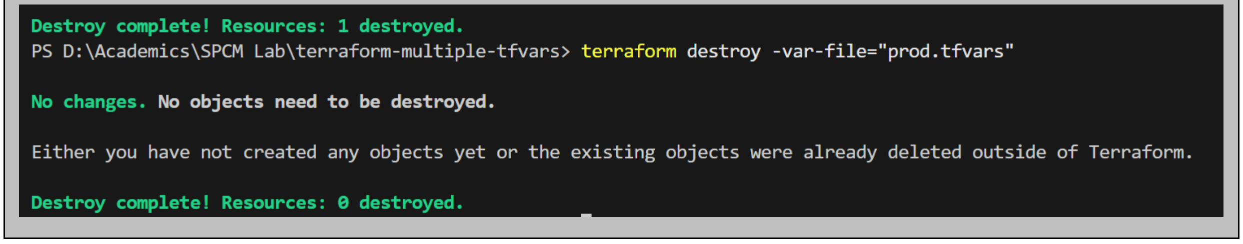


```
PS D:\Academics\SPCM Lab\terraform-multiple-tfvars> terraform destroy -var-file=”dev.tfvars”
aws_instance.Anshikaeg: Refreshing state... [id=i-04109503a873dd49c]

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_instance.Anshikaeg will be destroyed
  - resource "aws_instance" "Anshikaeg" {
      - ami                                  = "ami-09423ec3aa48e9438" -> null
      - arn                                  = "arn:aws:ec2:eu-north-1:292081347559:instance/i-04109503a873dd49c
" -> null
```



```
  Enter a value: yes

aws_instance.Anshikaeg: Destroying... [id=i-04109503a873dd49c]
aws_instance.Anshikaeg: Still destroying... [id=i-04109503a873dd49c, 10s elapsed]
aws_instance.Anshikaeg: Still destroying... [id=i-04109503a873dd49c, 20s elapsed]
aws_instance.Anshikaeg: Still destroying... [id=i-04109503a873dd49c, 30s elapsed]
aws_instance.Anshikaeg: Still destroying... [id=i-04109503a873dd49c, 40s elapsed]
aws_instance.Anshikaeg: Still destroying... [id=i-04109503a873dd49c, 50s elapsed]
aws_instance.Anshikaeg: Still destroying... [id=i-04109503a873dd49c, 1m0s elapsed]
aws_instance.Anshikaeg: Still destroying... [id=i-04109503a873dd49c, 1m10s elapsed]
aws_instance.Anshikaeg: Destruction complete after 1m16s

Destroy complete! Resources: 1 destroyed.
```

terraform destroy -var-file=”prod.tfvars”



```
Destroy complete! Resources: 1 destroyed.
PS D:\Academics\SPCM Lab\terraform-multiple-tfvars> terraform destroy -var-file="prod.tfvars"

No changes. No objects need to be destroyed.

Either you have not created any objects yet or the existing objects were already deleted outside of Terraform.

Destroy complete! Resources: 0 destroyed.
```

- Confirm the destruction by typing yes.

7. Conclusion:

This lab exercise demonstrates how to use multiple tfvars files in Terraform to manage variable values for different environments. It allows you to maintain separate configuration files for different environments, making it easier to manage and maintain your infrastructure code. Experiment with different values in the dev.tfvars and prod.tfvars files to observe how they impact the infrastructure provisioning process for each environment.