Lab Exercise 8– Terraform Multiple tfvars Files

Objective:

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

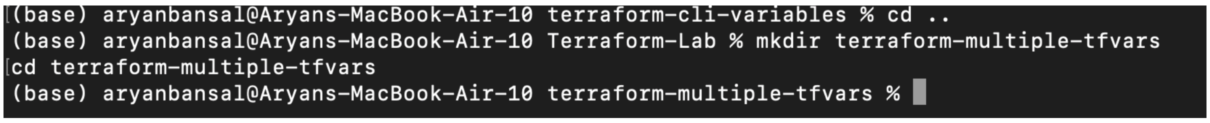
Prerequisites:

- 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
```

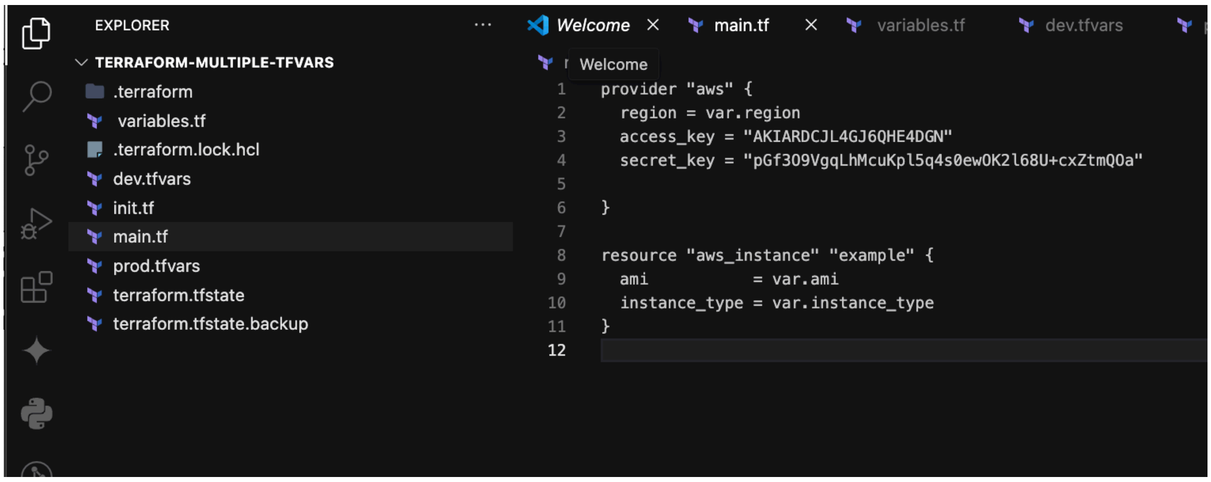


```
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-cli-variables % cd ..
(base) aryanbansal@Aryans-MacBook-Air-10 Terraform-Lab % mkdir terraform-multiple-tfvars
cd terraform-multiple-tfvars
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-multiple-tfvars %
```

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

# main.tf

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



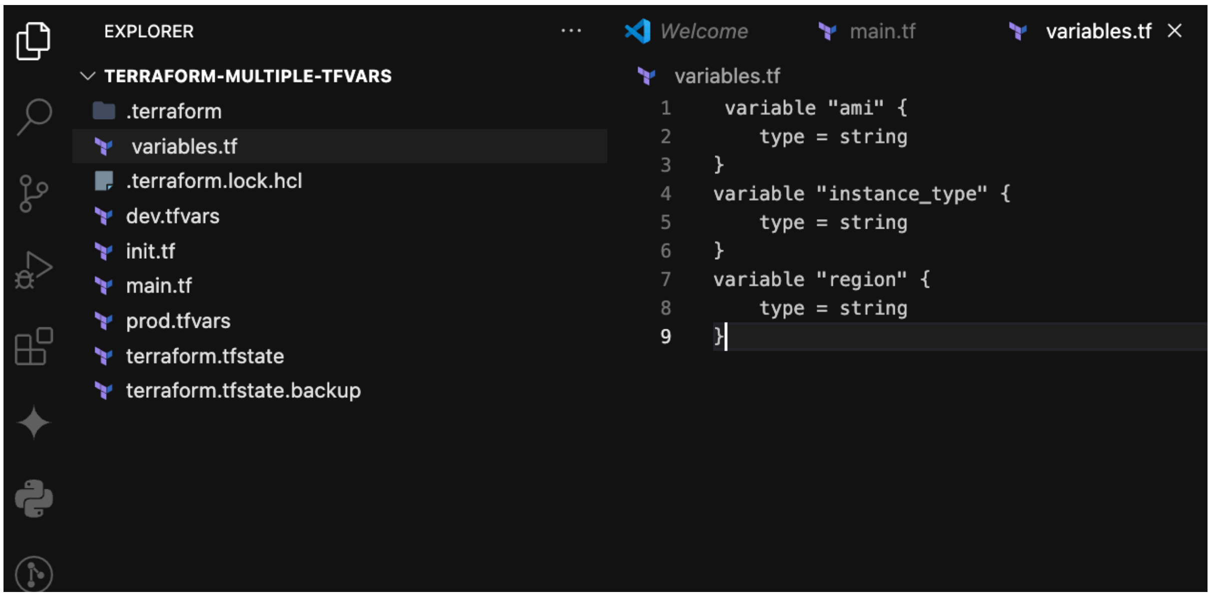
```
provider "aws" {
  region = var.region
  access_key = "AKIARDCJL4GJ6QHE4DGN"
  secret_key = "pGf309VgqLhMcuKpl5q4s0ewOK2l68U+cxZtmQ0a"

}

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

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

- Create a file named variables.tf:

# variables.tf

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


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



EXPLORER
TERRAFORM-MULTIPLE-TFVARS
.terraform
variables.tf
.terraform.lock.hcl
dev.tfvars
init.tf
main.tf
prod.tfvars
terraform.tfstate
terraform.tfstate.backup

Welcome | main.tf | variables.tf

variables.tf

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

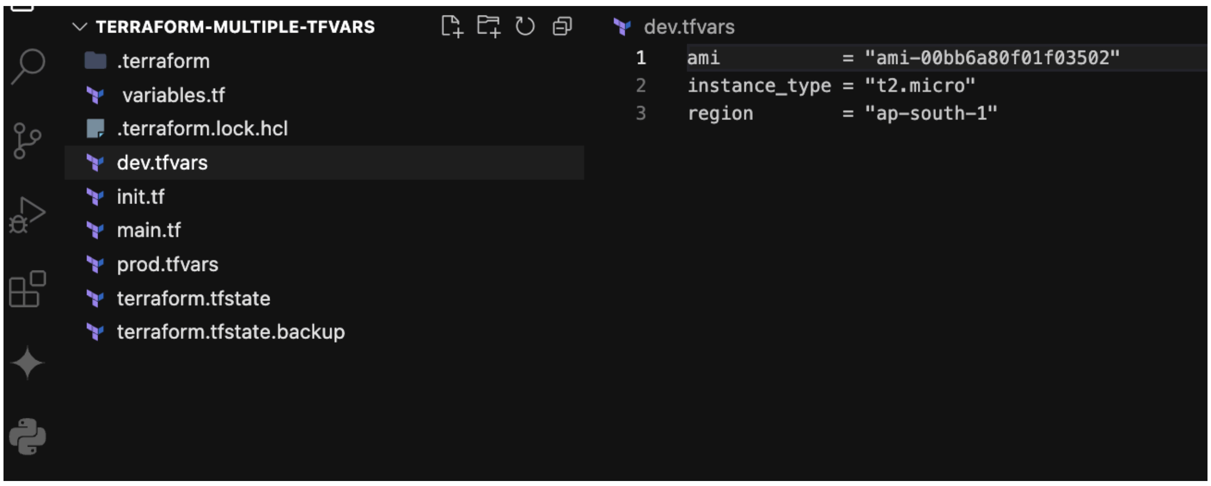
2. Create Multiple tfvars Files:

- Create a file named dev.tfvars:

# dev.tfvars

```
ami          = "ami-0123456789abcdef0"
```

```
instance_type = "t2.micro"
```



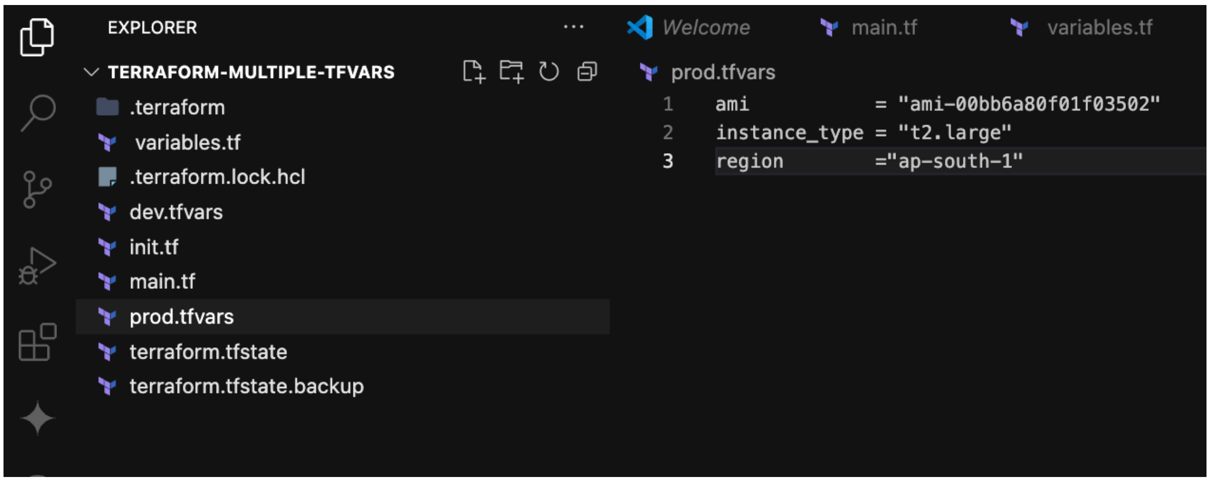
```
ami           = "ami-00bb6a80f01f03502"
instance_type = "t2.micro"
region        = "ap-south-1"
```

- Create a file named prod.tfvars:

# prod.tfvars

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

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

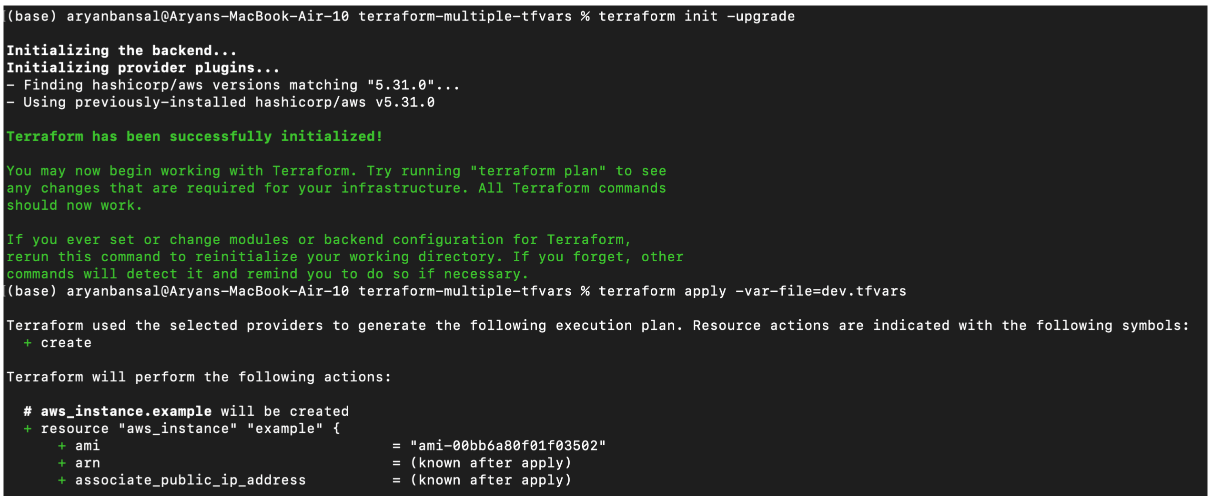


```
ami           = "ami-00bb6a80f01f03502"
instance_type = "t2.large"
region        ="ap-south-1"
```

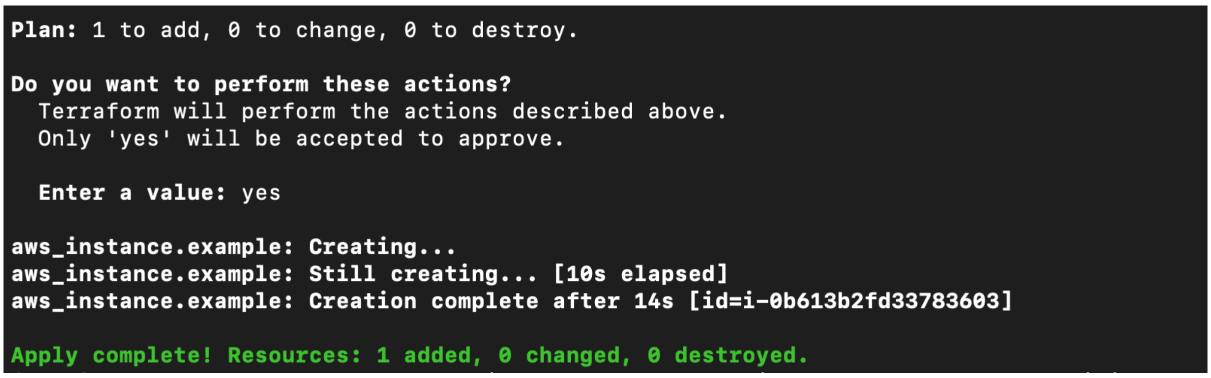
3. Initialize and Apply for Dev Environment:

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

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



```
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-multiple-tfvars % terraform init -upgrade

Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.31.0"...
- Using previously-installed hashicorp/aws v5.31.0

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.

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.
(base) aryanbansal@Aryans-MacBook-Air-10 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.example will be created
  + resource "aws_instance" "example" {
      + ami                                  = "ami-00bb6a80f01f03502"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
```

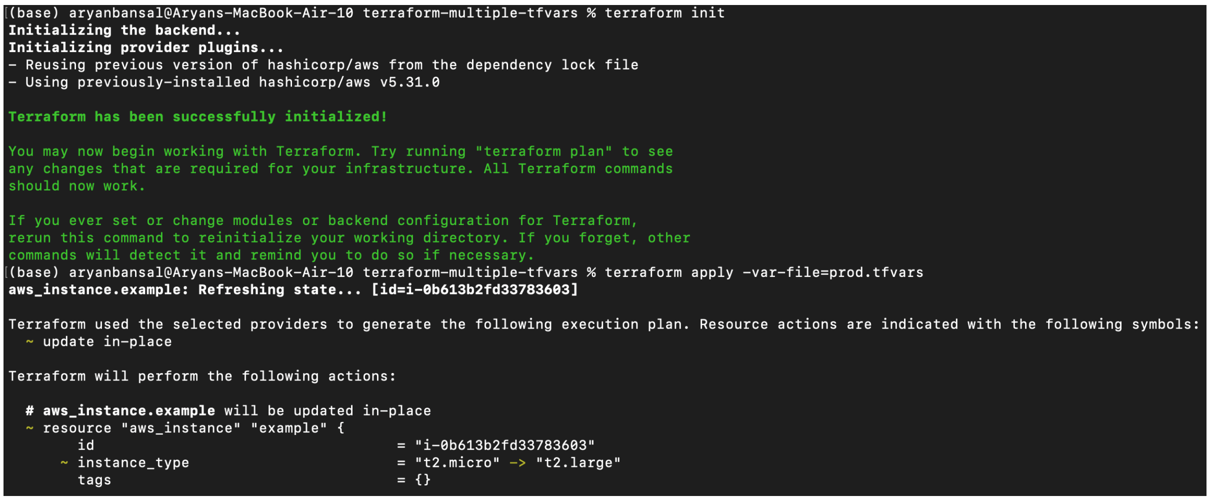


```
Plan: 1 to add, 0 to change, 0 to destroy.

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.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 14s [id=i-0b613b2fd33783603]

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

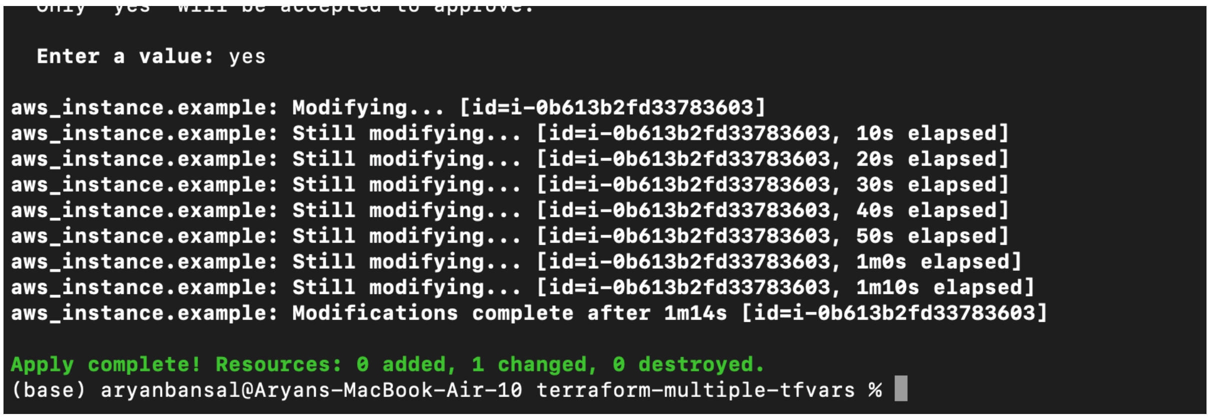
4. Initialize and Apply for Prod Environment:

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

```
terraform init
terraform apply -var-file=prod.tfvars
```



```
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-multiple-tfvars % terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.31.0

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.

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.
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-multiple-tfvars % terraform apply -var-file=prod.tfvars
aws_instance.example: Refreshing state... [id=i-0b613b2fd33783603]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_instance.example will be updated in-place
  ~ resource "aws_instance" "example" {
        id                                   = "i-0b613b2fd33783603"
      ~ instance_type                        = "t2.micro" -> "t2.large"
        tags                                 = {}
```



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

  Enter a value: yes

aws_instance.example: Modifying... [id=i-0b613b2fd33783603]
aws_instance.example: Still modifying... [id=i-0b613b2fd33783603, 10s elapsed]
aws_instance.example: Still modifying... [id=i-0b613b2fd33783603, 20s elapsed]
aws_instance.example: Still modifying... [id=i-0b613b2fd33783603, 30s elapsed]
aws_instance.example: Still modifying... [id=i-0b613b2fd33783603, 40s elapsed]
aws_instance.example: Still modifying... [id=i-0b613b2fd33783603, 50s elapsed]
aws_instance.example: Still modifying... [id=i-0b613b2fd33783603, 1m0s elapsed]
aws_instance.example: Still modifying... [id=i-0b613b2fd33783603, 1m10s elapsed]
aws_instance.example: Modifications complete after 1m14s [id=i-0b613b2fd33783603]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-multiple-tfvars %
```

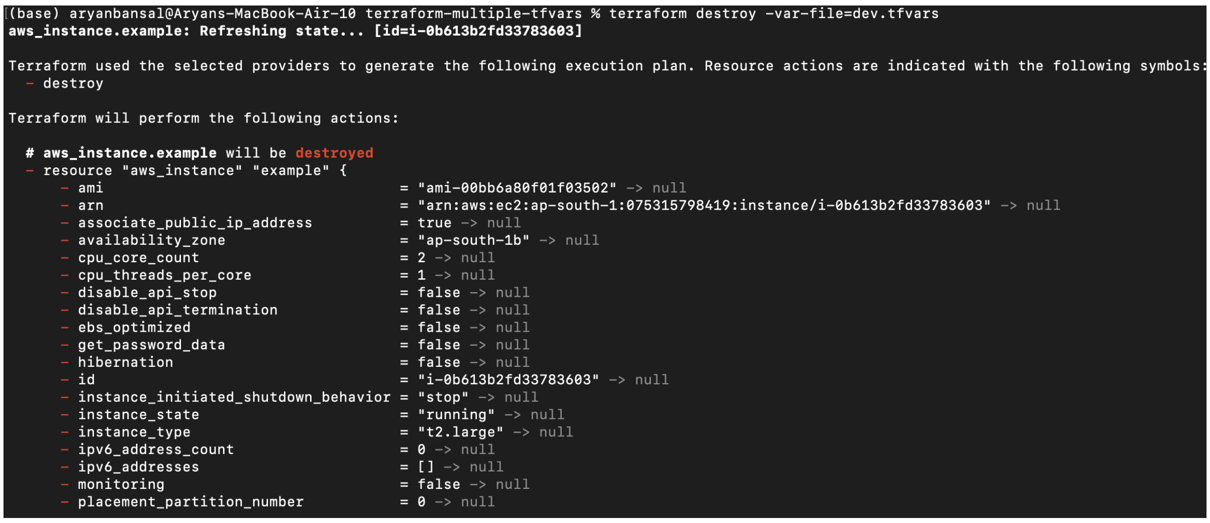
5. Test and Verify:

- 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.

6. Clean Up:

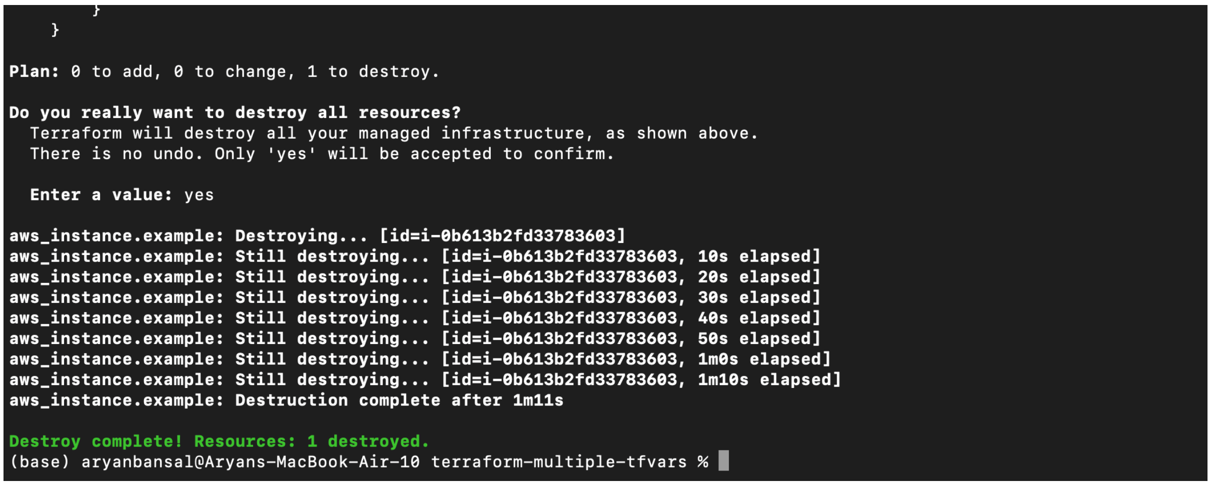
- After testing, you can clean up resources:

```
terraform destroy -var-file=dev.tfvars
terraform destroy -var-file=prod.tfvars
```

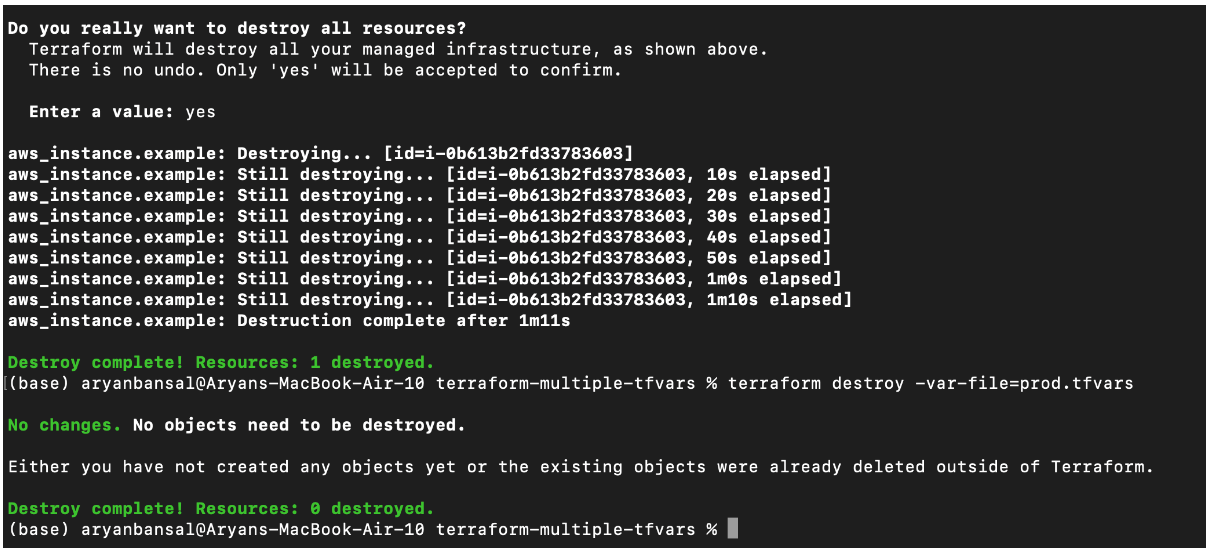


```
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-multiple-tfvars % terraform destroy -var-file=dev.tfvars
aws_instance.example: Refreshing state... [id=i-0b613b2fd33783603]

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.example will be destroyed
  - resource "aws_instance" "example" {
      - ami                                  = "ami-00bb6a80f01f03502" -> null
      - arn                                  = "arn:aws:ec2:ap-south-1:075315798419:instance/i-0b613b2fd33783603" -> null
      - associate_public_ip_address          = true -> null
      - availability_zone                    = "ap-south-1b" -> null
      - cpu_core_count                       = 2 -> null
      - cpu_threads_per_core                 = 1 -> null
      - disable_api_stop                     = false -> null
      - disable_api_termination              = false -> null
      - ebs_optimized                        = false -> null
      - get_password_data                    = false -> null
      - hibernation                          = false -> null
      - id                                   = "i-0b613b2fd33783603" -> null
      - instance_initiated_shutdown_behavior = "stop" -> null
      - instance_state                       = "running" -> null
      - instance_type                        = "t2.large" -> null
      - ipv6_address_count                   = 0 -> null
      - ipv6_addresses                       = [] -> null
      - monitoring                           = false -> null
      - placement_partition_number           = 0 -> null
```

- Confirm the destruction by typing yes.



```
        }
    }

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.example: Destroying... [id=i-0b613b2fd33783603]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 10s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 20s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 30s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 40s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 50s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 1m0s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 1m10s elapsed]
aws_instance.example: Destruction complete after 1m11s

Destroy complete! Resources: 1 destroyed.
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-multiple-tfvars %
```



```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.example: Destroying... [id=i-0b613b2fd33783603]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 10s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 20s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 30s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 40s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 50s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 1m0s elapsed]
aws_instance.example: Still destroying... [id=i-0b613b2fd33783603, 1m10s elapsed]
aws_instance.example: Destruction complete after 1m11s

Destroy complete! Resources: 1 destroyed.
(base) aryanbansal@Aryans-MacBook-Air-10 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.
(base) aryanbansal@Aryans-MacBook-Air-10 terraform-multiple-tfvars %
```

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.