Prepared by: Dr. Hitesh Kumar Sharma

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.

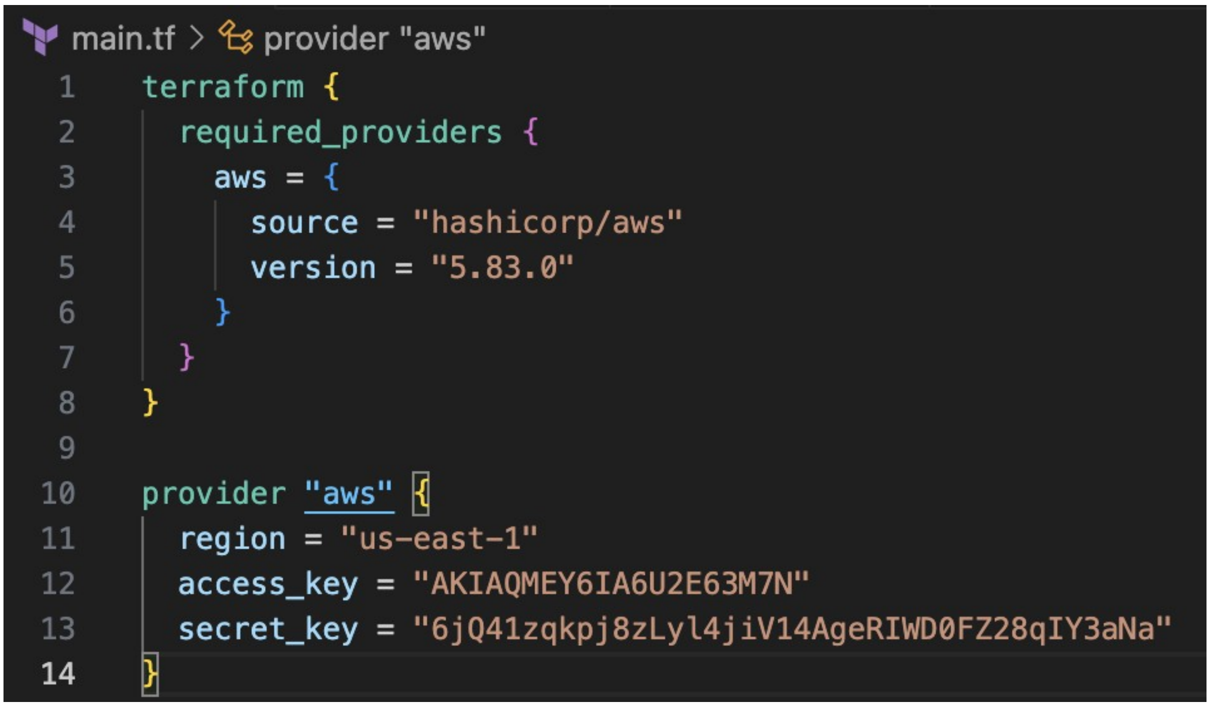
Steps:

1.Create a Terraform Directory:

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

Create Terraform Configuration Files:

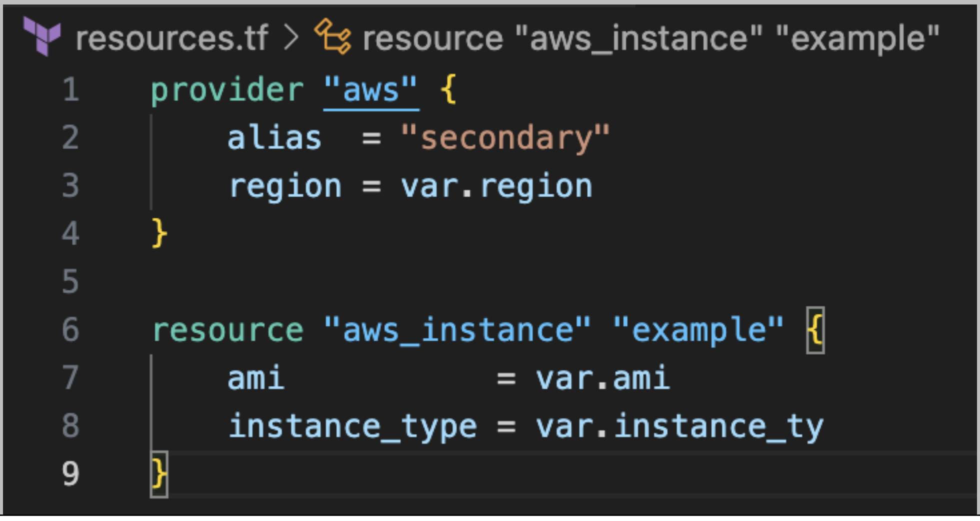
Create a file named main.tf:



```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.83.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  access_key = "AKIAQMEY6IA6U2E63M7N"
  secret_key = "6jQ41zqkpj8zLyl4jiV14AgeRIWD0FZ28qIY3aNa"
}
```

# main.tf

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

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



resources.tf > resource "aws_instance" "example"

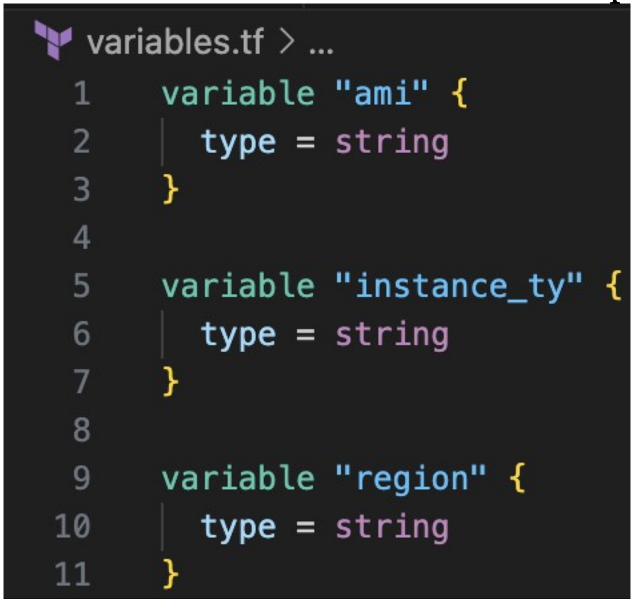
```
provider "aws" {
    alias  = "secondary"
    region = var.region
}

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

```
}
```

Create a file named variables.tf:

# variables.tf

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

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



variables.tf > ...

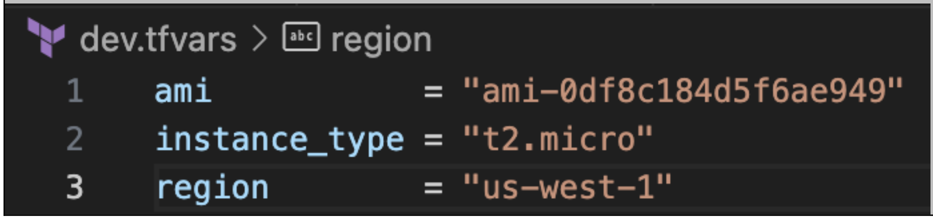
```
variable "ami" {
  type = string
}

variable "instance_ty" {
  type = string
}

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

1.Create Multiple tfvars Files:

Create a file named dev.tfvars:

# dev.tfvars

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



dev.tfvars > region

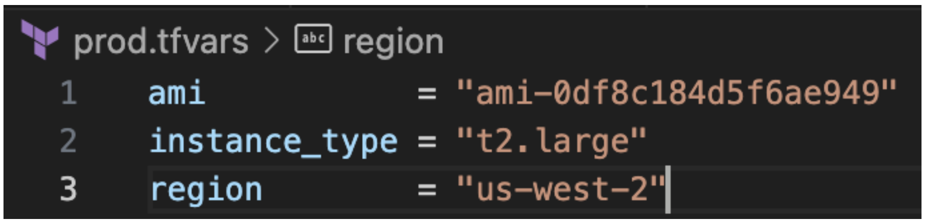
```
ami           = "ami-0df8c184d5f6ae949"
instance_type = "t2.micro"
region        = "us-west-1"
```

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.

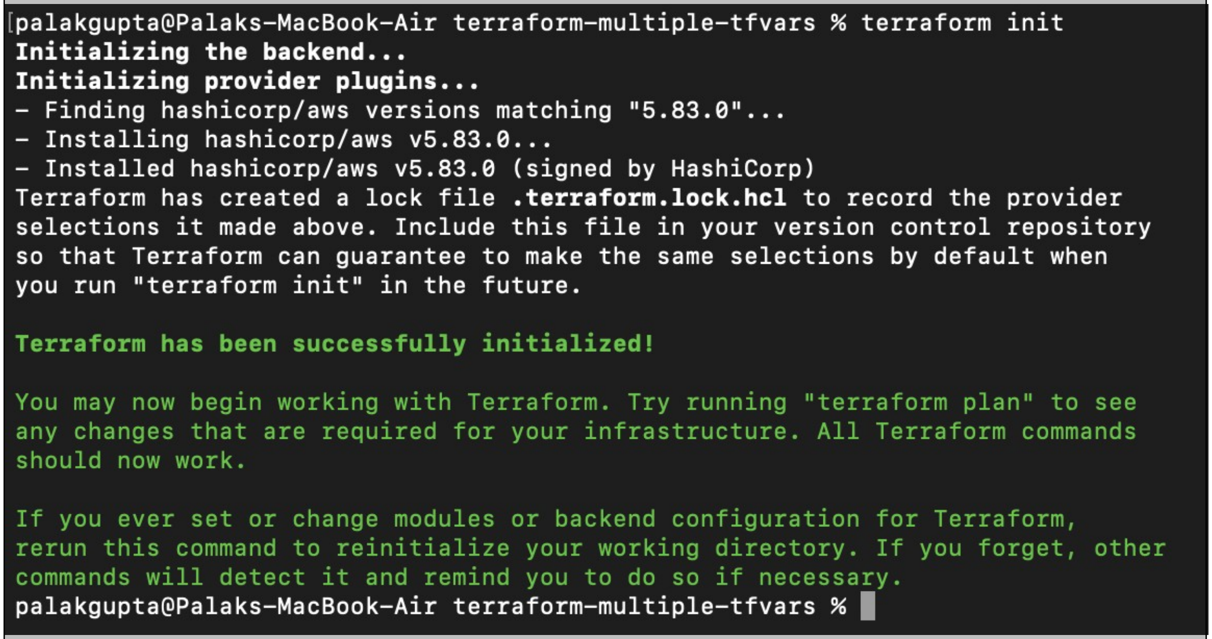


```
ami           = "ami-0df8c184d5f6ae949"
instance_type = "t2.large"
region        = "us-west-2"
```

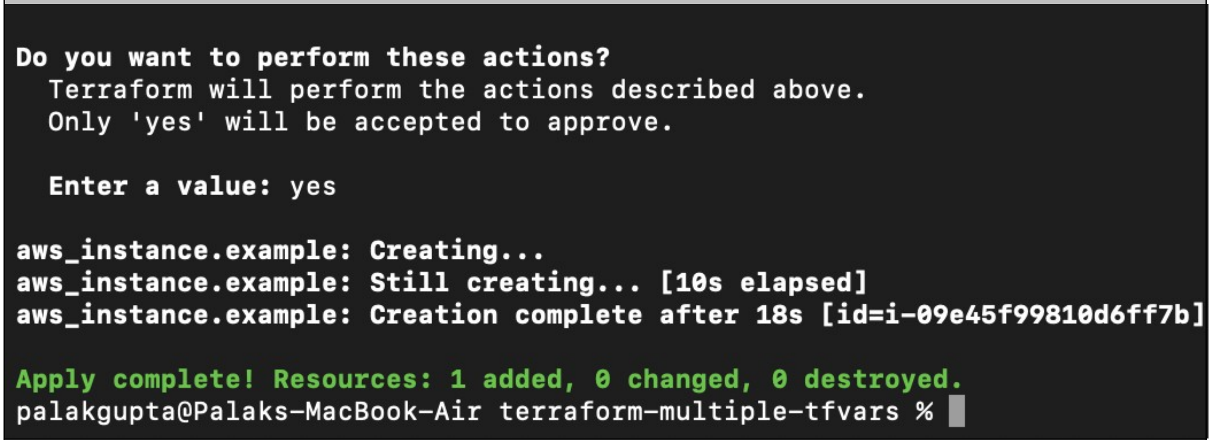
1.Initialize and Apply for Dev Environment:

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

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



```
[palakgupta@Palaks-MacBook-Air terraform-multiple-tfvars % 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.
palakgupta@Palaks-MacBook-Air terraform-multiple-tfvars %
```



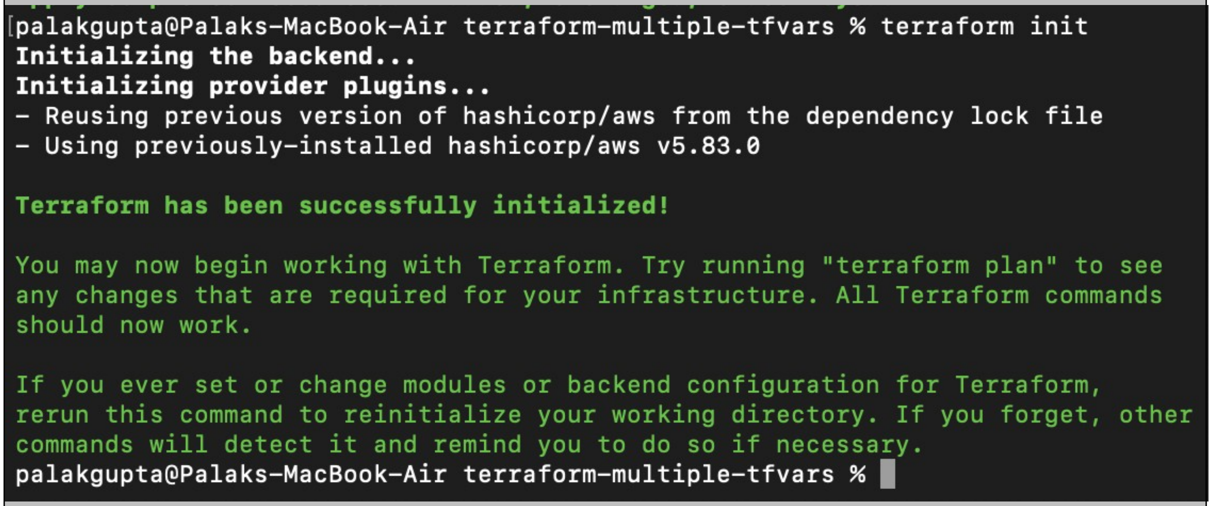
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 18s [id=i-09e45f99810d6ff7b]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
palakgupta@Palaks-MacBook-Air terraform-multiple-tfvars %
```

1.Initialize and Apply for Prod Environment:

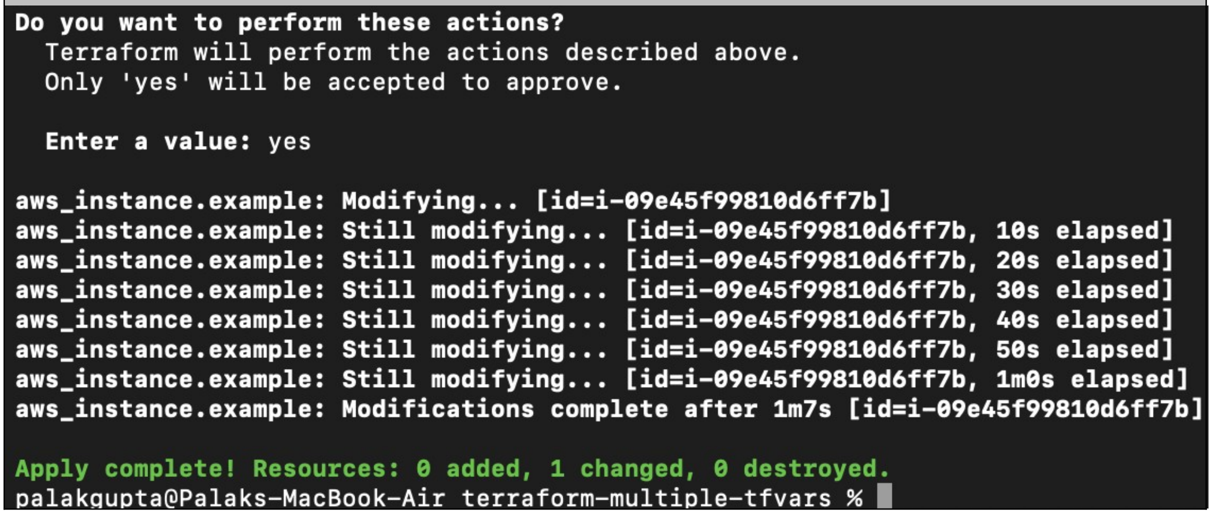
Run the following Terraform commands to initialize and apply the configuration for the prod environment:

```
terraform init
```

terraform apply -var-file=prod.tfvars



```
[palakgupta@Palaks-MacBook-Air 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 v5.83.0

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.
palakgupta@Palaks-MacBook-Air terraform-multiple-tfvars %
```



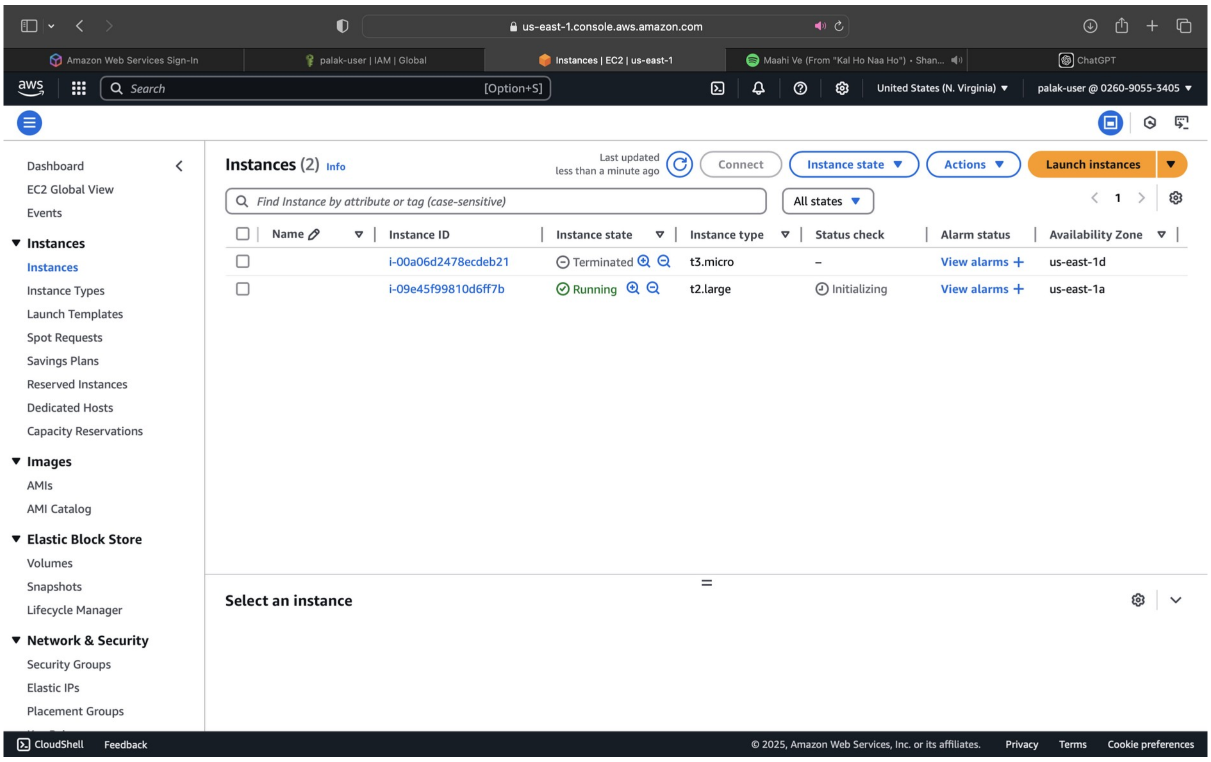
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.example: Modifying... [id=i-09e45f99810d6ff7b]
aws_instance.example: Still modifying... [id=i-09e45f99810d6ff7b, 10s elapsed]
aws_instance.example: Still modifying... [id=i-09e45f99810d6ff7b, 20s elapsed]
aws_instance.example: Still modifying... [id=i-09e45f99810d6ff7b, 30s elapsed]
aws_instance.example: Still modifying... [id=i-09e45f99810d6ff7b, 40s elapsed]
aws_instance.example: Still modifying... [id=i-09e45f99810d6ff7b, 50s elapsed]
aws_instance.example: Still modifying... [id=i-09e45f99810d6ff7b, 1m0s elapsed]
aws_instance.example: Modifications complete after 1m7s [id=i-09e45f99810d6ff7b]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
palakgupta@Palaks-MacBook-Air terraform-multiple-tfvars %
```

1.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.

1.Clean Up:



Instances (2) Info

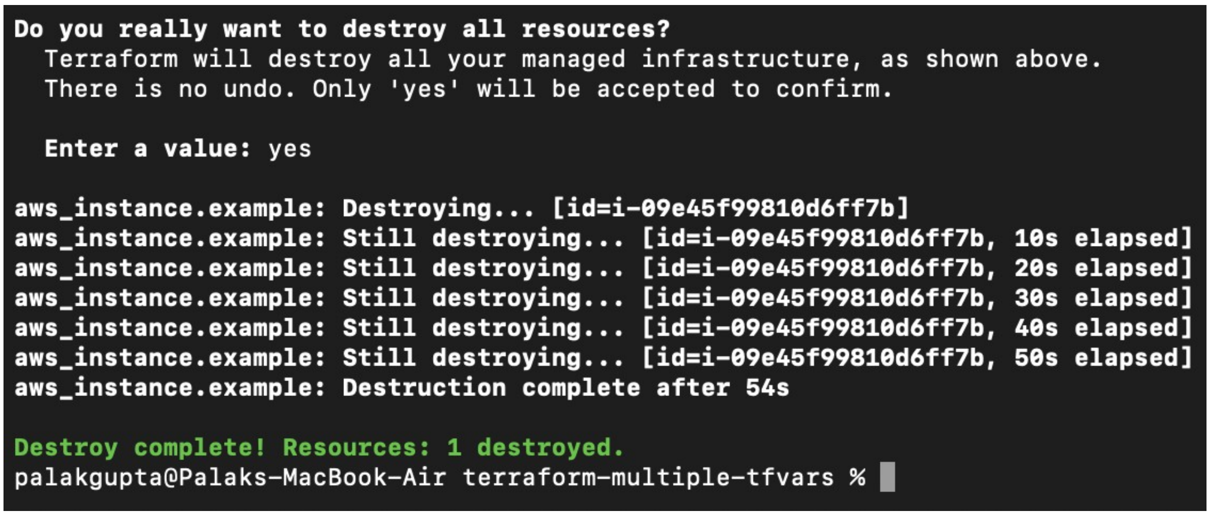
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
☐		i-00a06d2478ecdeb21	Terminated	t3.micro	–	View alarms +	us-east-1d
☐		i-09e45f99810d6ff7b	Running	t2.large	Initializing	View alarms +	us-east-1a

Select an instance

After testing, you can clean up resources:

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

Confirm the destruction by typing yes.



```
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-09e45f99810d6ff7b]
aws_instance.example: Still destroying... [id=i-09e45f99810d6ff7b, 10s elapsed]
aws_instance.example: Still destroying... [id=i-09e45f99810d6ff7b, 20s elapsed]
aws_instance.example: Still destroying... [id=i-09e45f99810d6ff7b, 30s elapsed]
aws_instance.example: Still destroying... [id=i-09e45f99810d6ff7b, 40s elapsed]
aws_instance.example: Still destroying... [id=i-09e45f99810d6ff7b, 50s elapsed]
aws_instance.example: Destruction complete after 54s

Destroy complete! Resources: 1 destroyed.
palakgupta@Palaks-MacBook-Air terraform-multiple-tfvars %
```

1.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.