Lab Exercise 8 – Terraform Multiple tfvars Files

Objective:

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

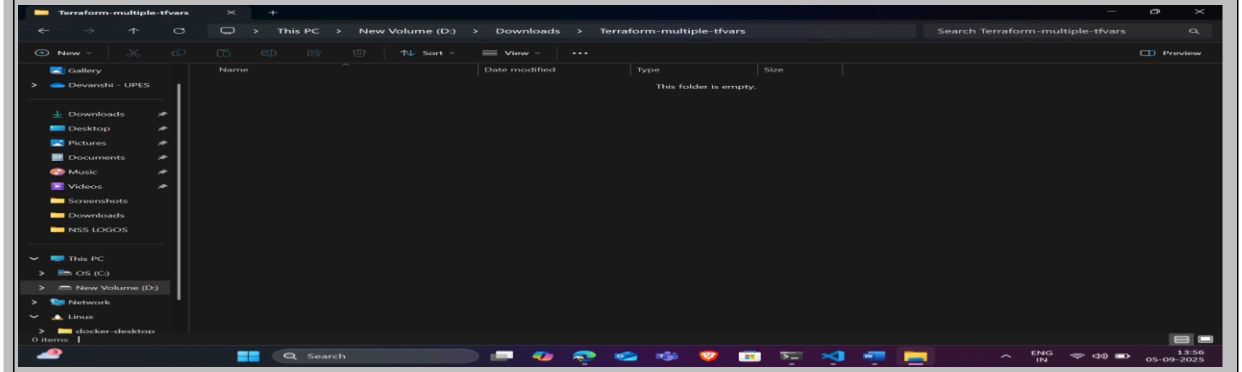
Prerequisites:

- 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
```



Name Date modified Type Size

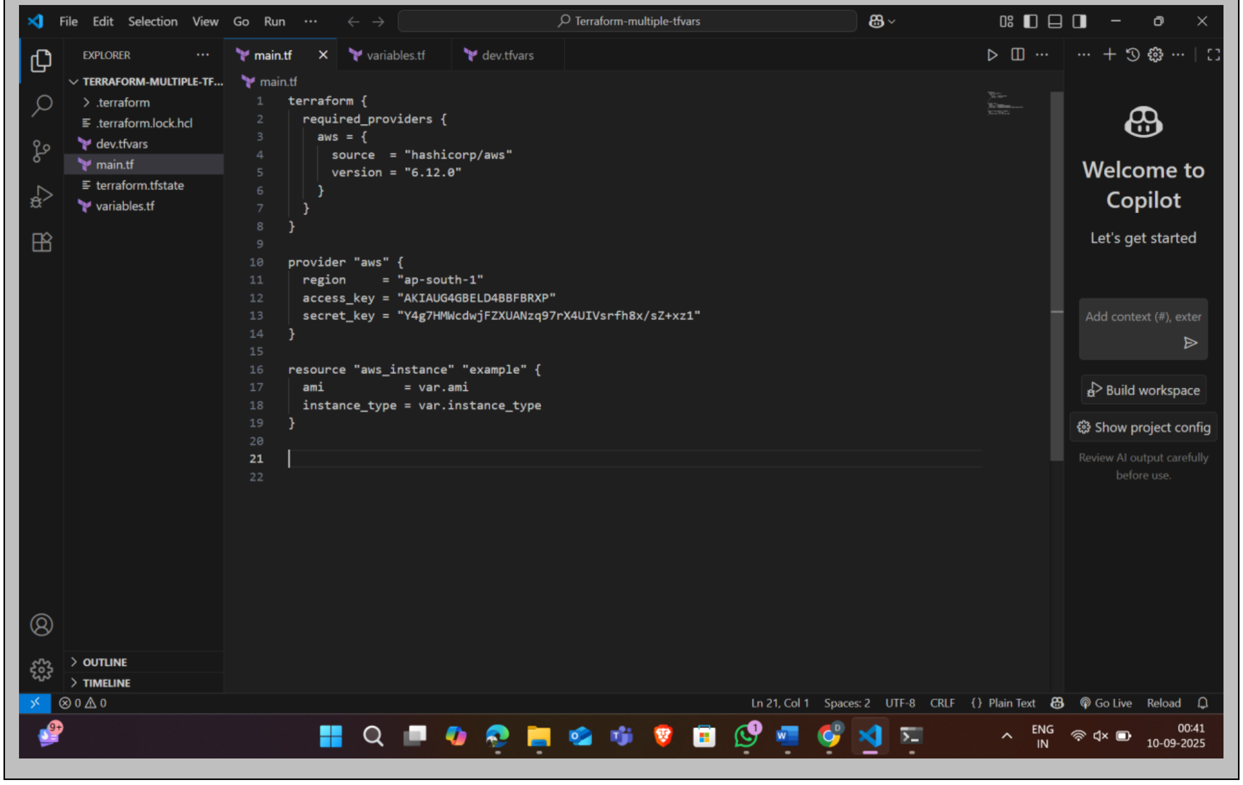
This folder is empty.

- 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
}
```

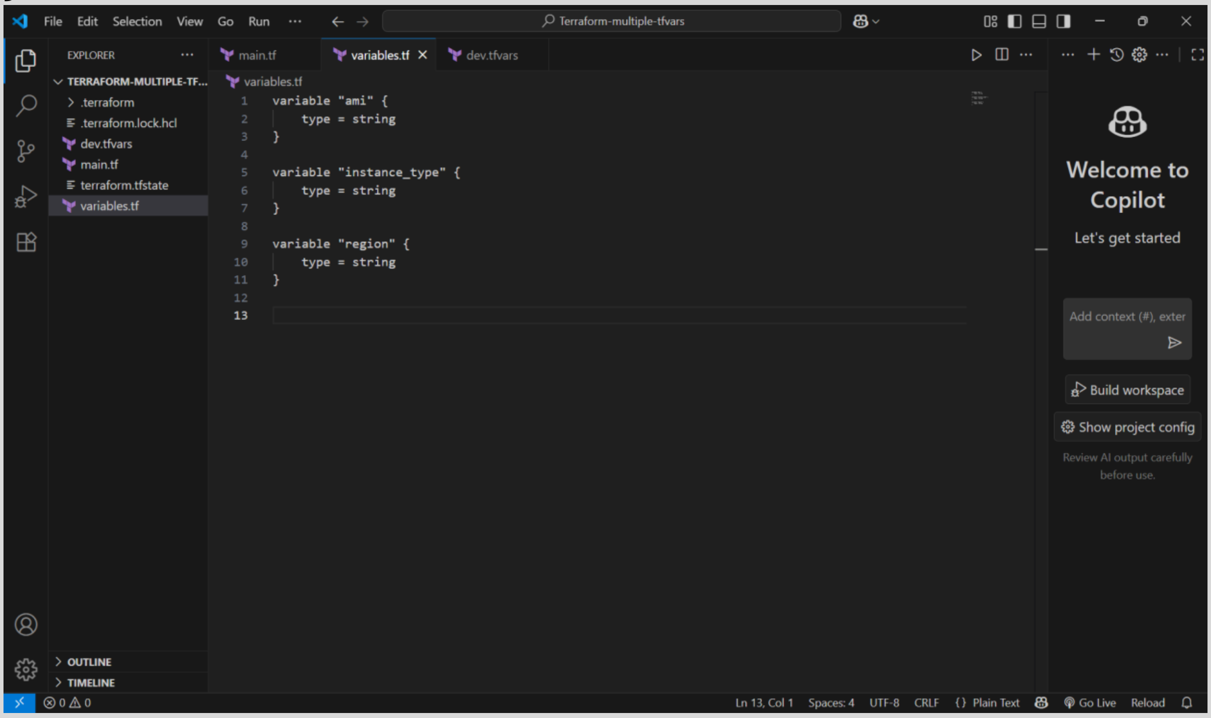


```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "6.12.0"
    }
  }
}

provider "aws" {
  region     = "ap-south-1"
  access_key = "AKIAUG4GBELD4BBFBRXP"
  secret_key = "Y4g7HMWcdwjFZXUANzq97rX4UIVsrfh8x/sZ+xz1"
}

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
}
```



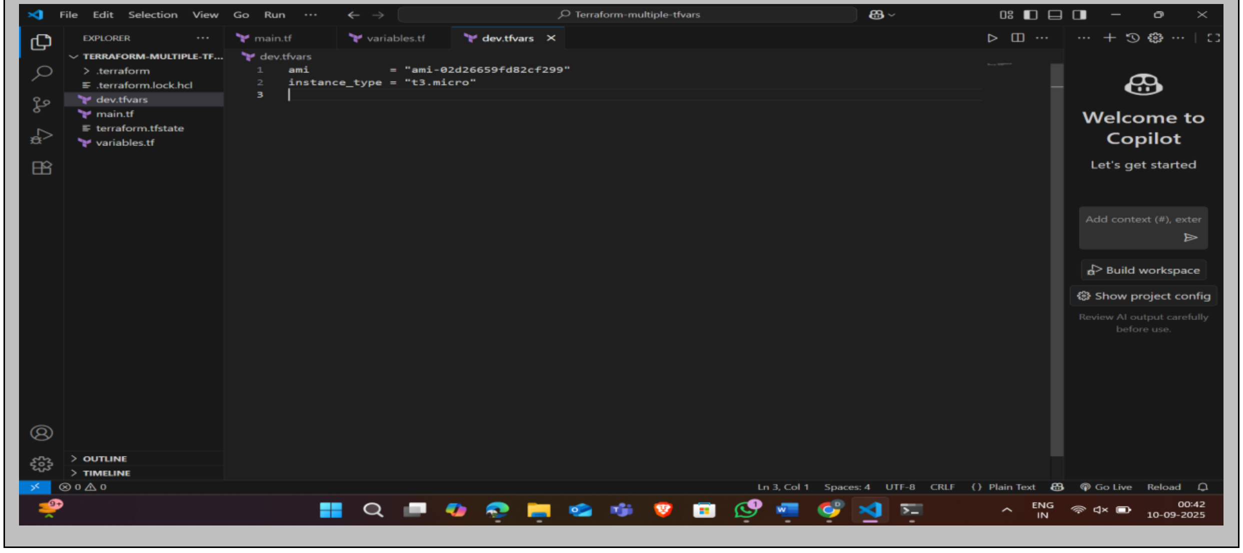
```
variable "ami" {
    type = string
}

variable "instance_type" {
    type = string
}

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

2. Create Multiple tfvars Files:

- Create a file named dev.tfvars:

# dev.tfvars

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

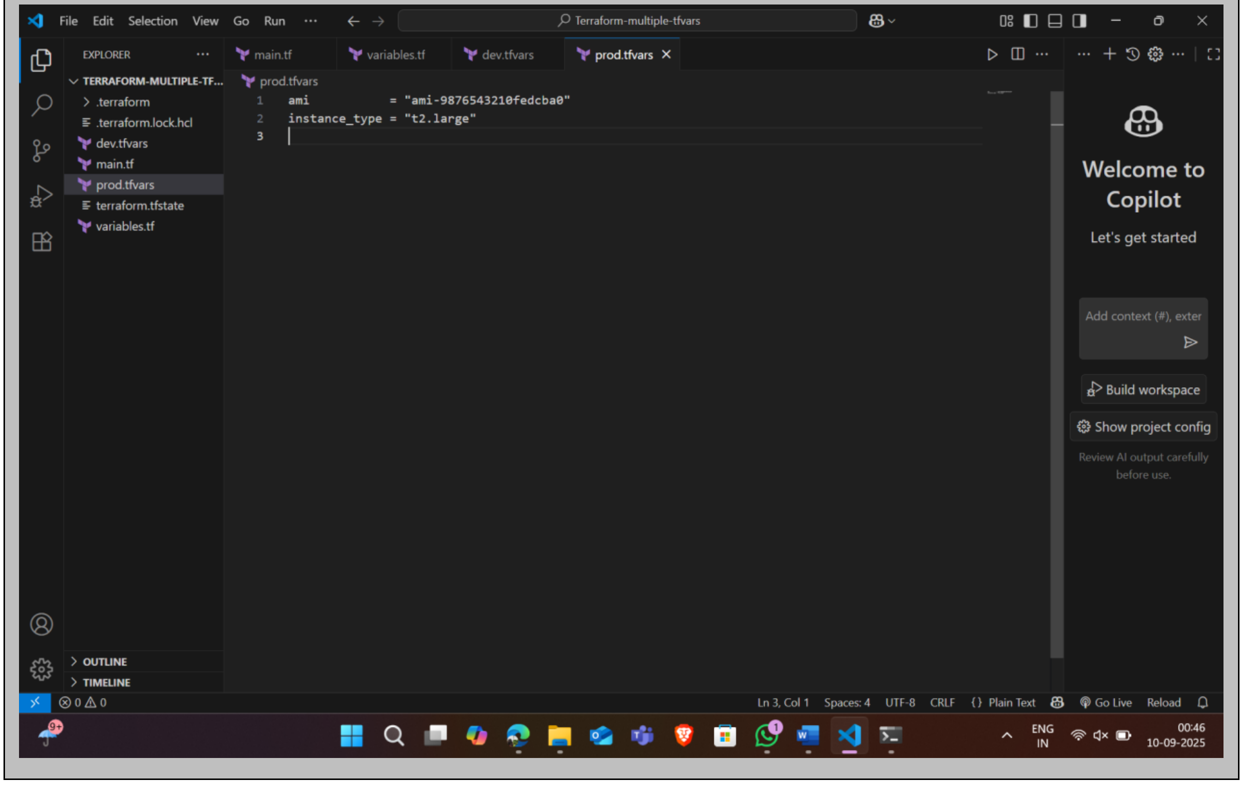


```
ami           = "ami-02d26659fd82cf299"
instance_type = "t3.micro"
```

- Create a file named prod.tfvars:

# prod.tfvars

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



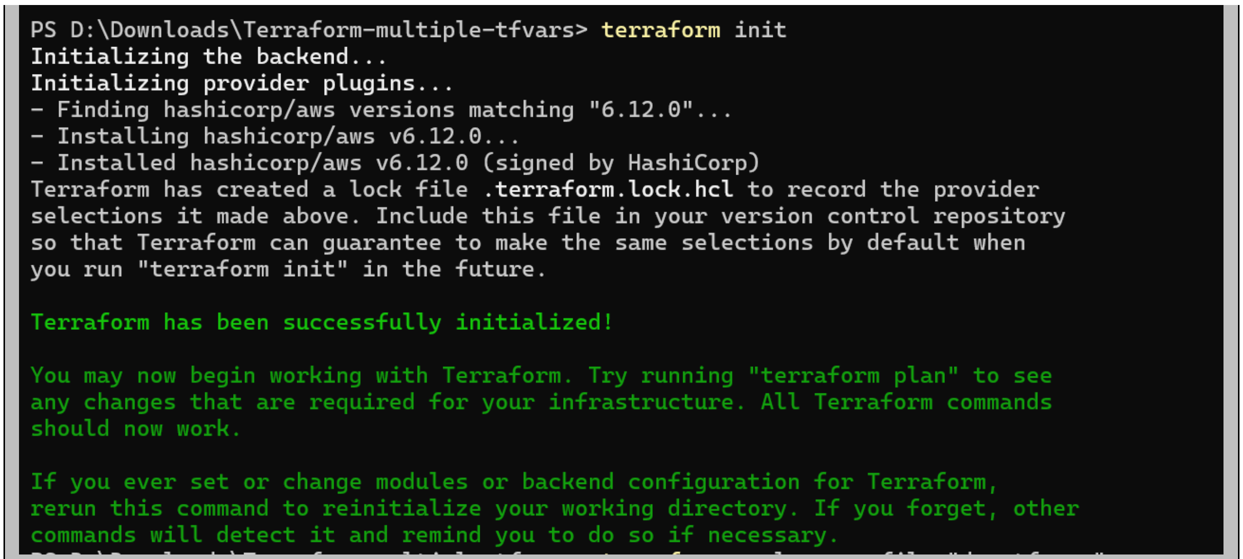
```
ami           = "ami-9876543210fedcba0"
instance_type = "t2.large"
```

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

3. Initialize and Apply for Dev Environment:

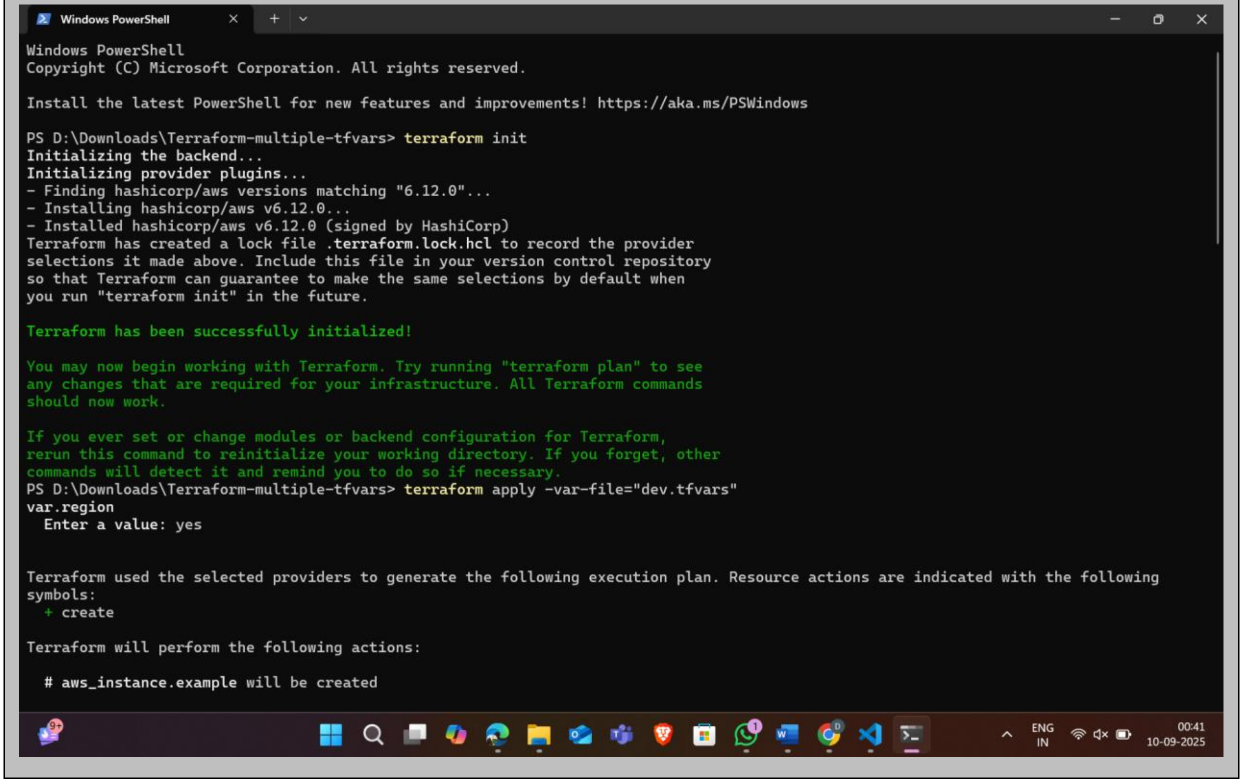
- Run the following Terraform commands to initialize and apply the configuration for the dev environment:

```
terraform init
```



```
PS D:\Downloads\Terraform-multiple-tfvars> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "6.12.0"...
- Installing hashicorp/aws v6.12.0...
- Installed hashicorp/aws v6.12.0 (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=dev.tfvars



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Downloads\Terraform-multiple-tfvars> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "6.12.0"...
- Installing hashicorp/aws v6.12.0...
- Installed hashicorp/aws v6.12.0 (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.
PS D:\Downloads\Terraform-multiple-tfvars> terraform apply -var-file="dev.tfvars"
var.region
  Enter a value: yes


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
```

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

--this command didn't work because t2.large instance is not available in free tier account
```

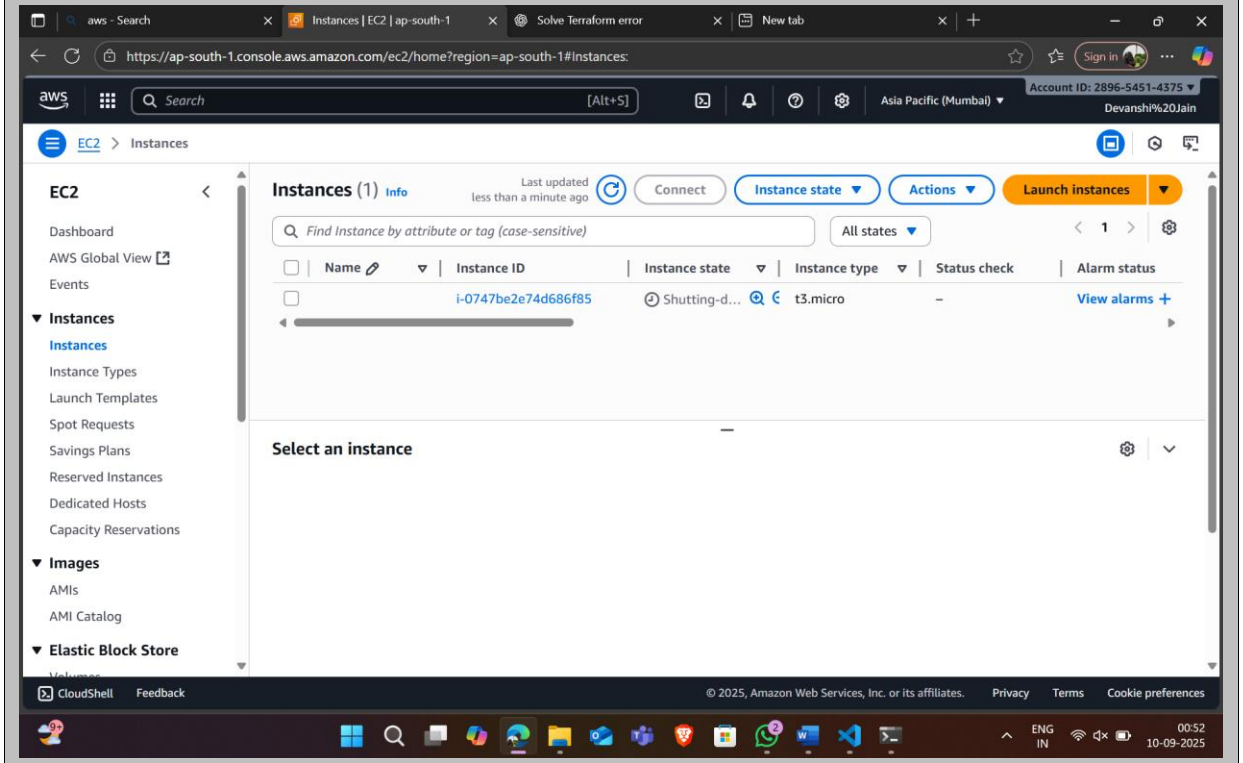
5. Test and Verify:

- 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:

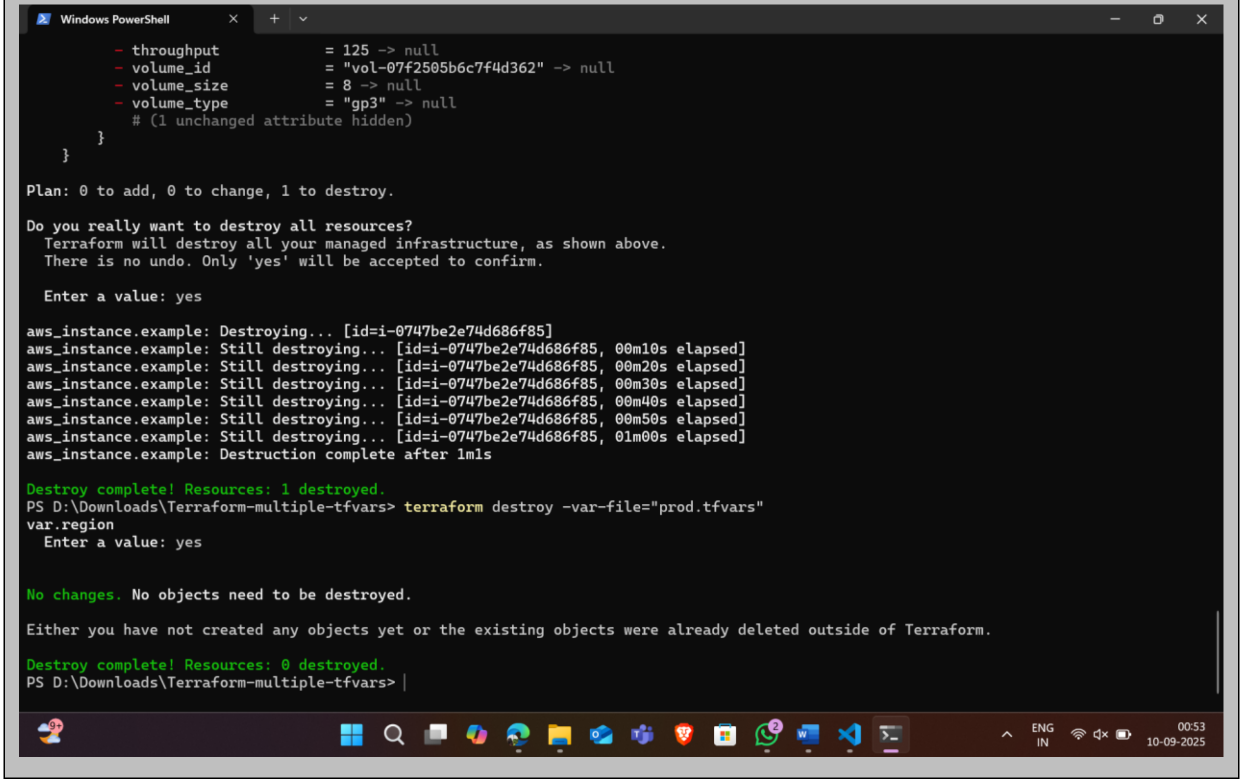
- After testing, you can clean up resources:

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



Name	Instance ID	Instance state	Instance type	Status check	Alarm status
	i-0747be2e74d686f85	Shutting-d...	t3.micro	–	View alarms +

terraform destroy -var-file=prod.tfvars



```
          - throughput            = 125 -> null
          - volume_id             = "vol-07f2505b6c7f4d362" -> null
          - volume_size           = 8 -> null
          - volume_type           = "gp3" -> null
            # (1 unchanged attribute hidden)
        }
    }

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-0747be2e74d686f85]
aws_instance.example: Still destroying... [id=i-0747be2e74d686f85, 00m10s elapsed]
aws_instance.example: Still destroying... [id=i-0747be2e74d686f85, 00m20s elapsed]
aws_instance.example: Still destroying... [id=i-0747be2e74d686f85, 00m30s elapsed]
aws_instance.example: Still destroying... [id=i-0747be2e74d686f85, 00m40s elapsed]
aws_instance.example: Still destroying... [id=i-0747be2e74d686f85, 00m50s elapsed]
aws_instance.example: Still destroying... [id=i-0747be2e74d686f85, 01m00s elapsed]
aws_instance.example: Destruction complete after 1m1s

Destroy complete! Resources: 1 destroyed.
PS D:\Downloads\Terraform-multiple-tfvars> terraform destroy -var-file="prod.tfvars"
var.region
  Enter a value: yes


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.
PS D:\Downloads\Terraform-multiple-tfvars> |
```

- 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.