

Lab Exercise 17 – Terraform

Multiple tf-vars Files

Name:-Vansh Bhatt

SapId:- 500125395

R.No:- R2142231689

Batch:- DevOps B1

To:- Hitesh Kumar Sharma Sir

Objective:

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

Prerequisites:

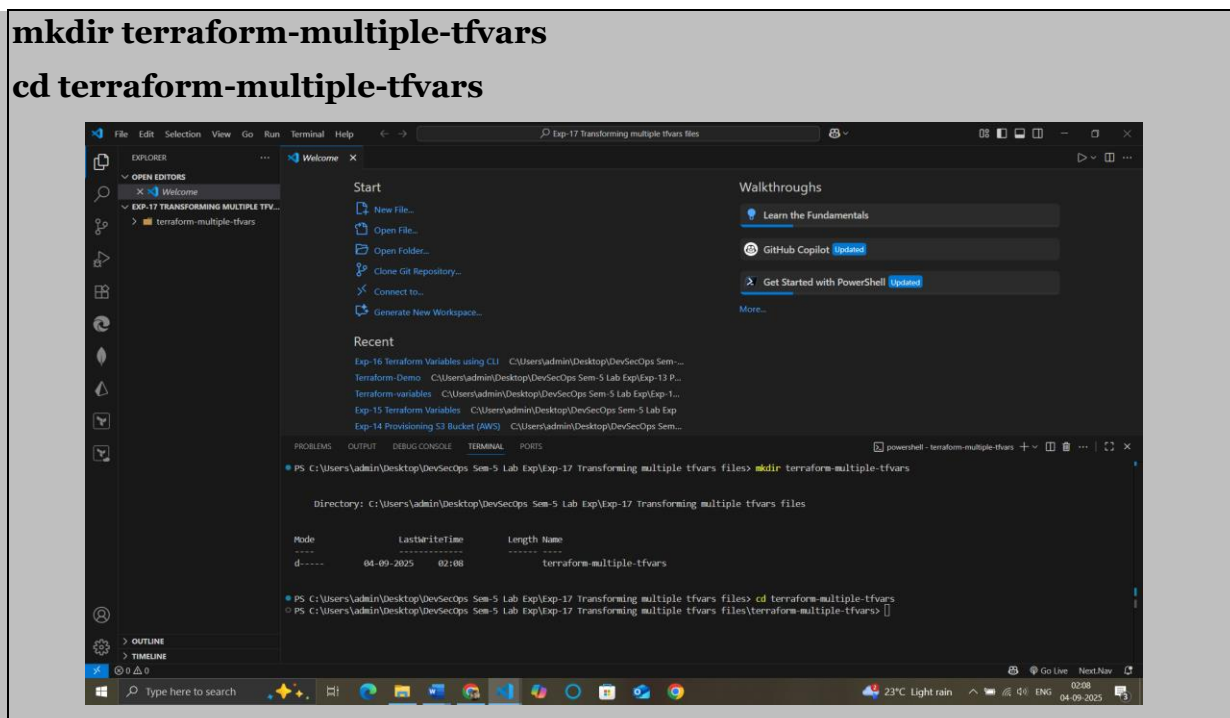
- Terraform installed on your machine.
- Basic knowledge of Terraform configuration and variables.

Steps:

1. Create a Terraform Directory:

mkdir terraform-multiple-tfvars

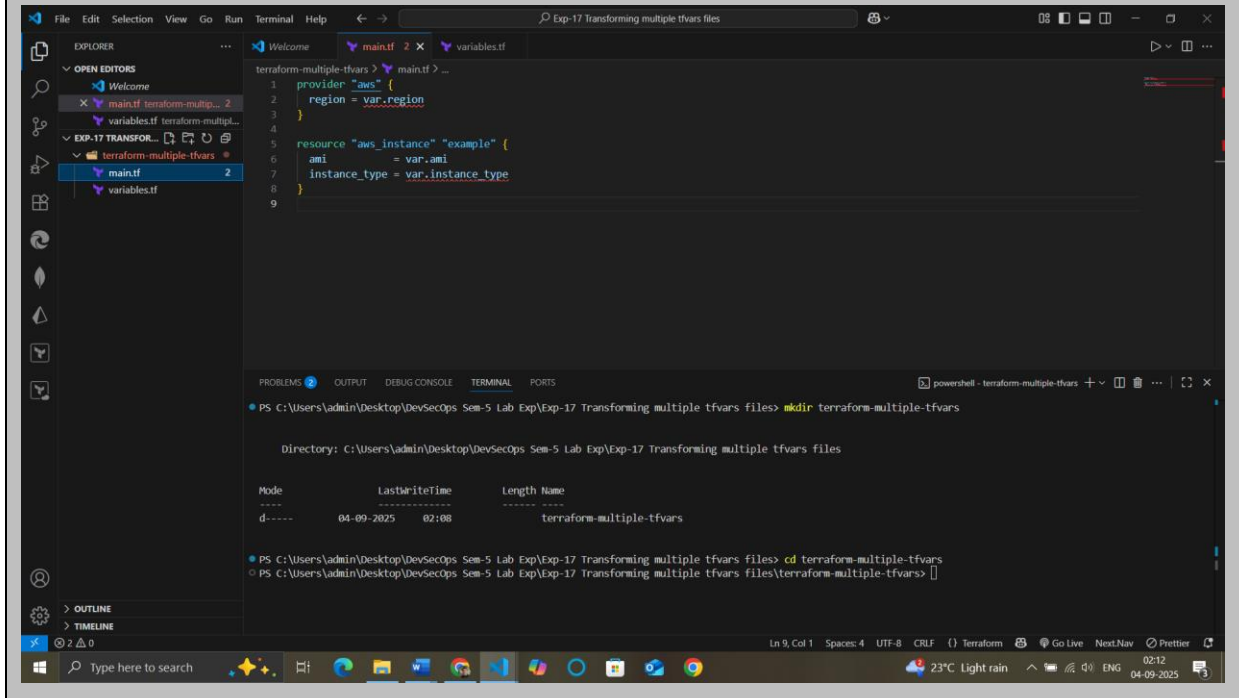
cd terraform-multiple-tfvars



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

main.tf

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

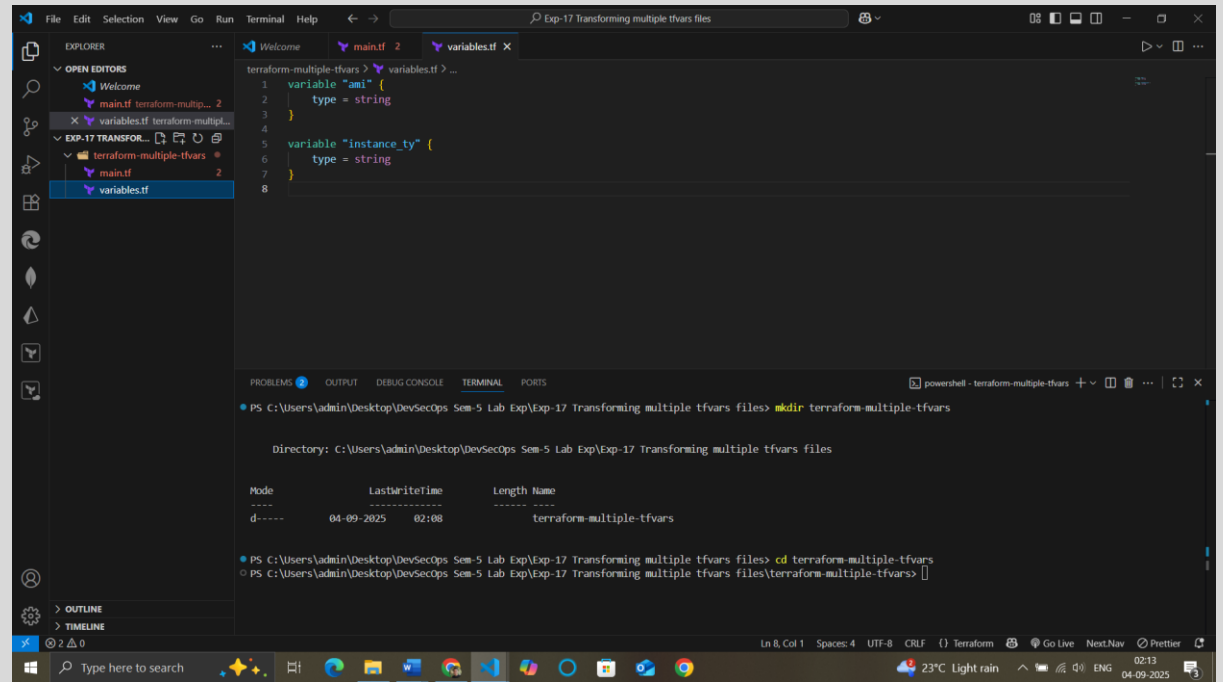


- Create a file named variables.tf:

variables.tf

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

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

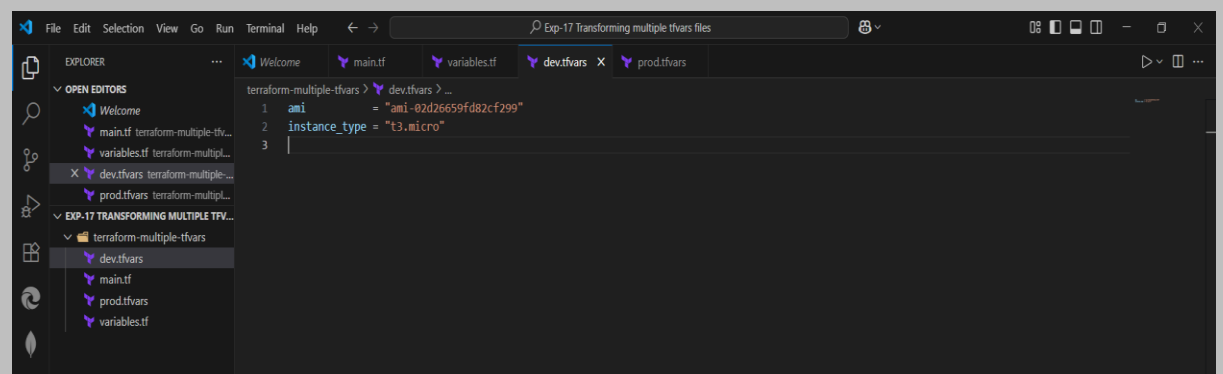


2. Create Multiple tfvars Files:

- Create a file named dev.tfvars:

dev.tfvars

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



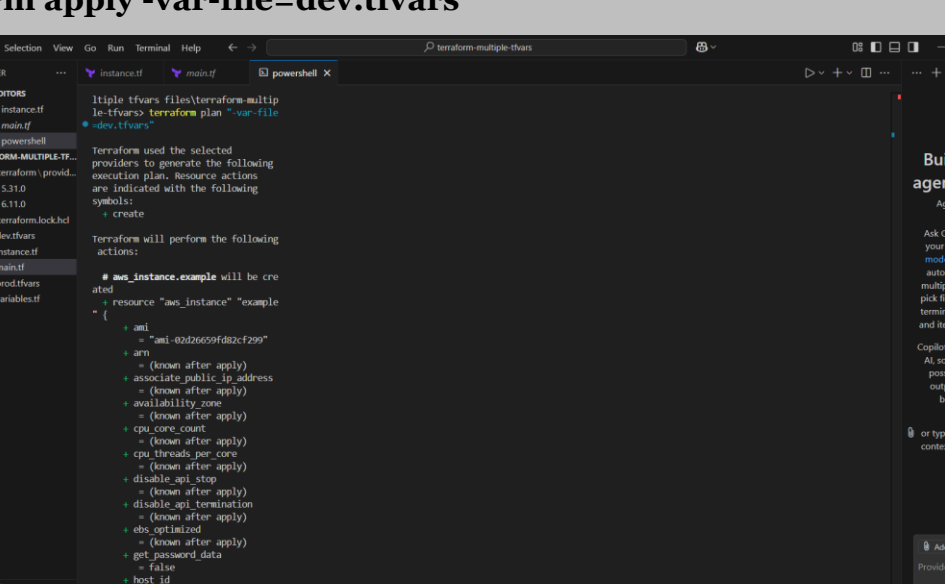
- ## # prod.tfvars

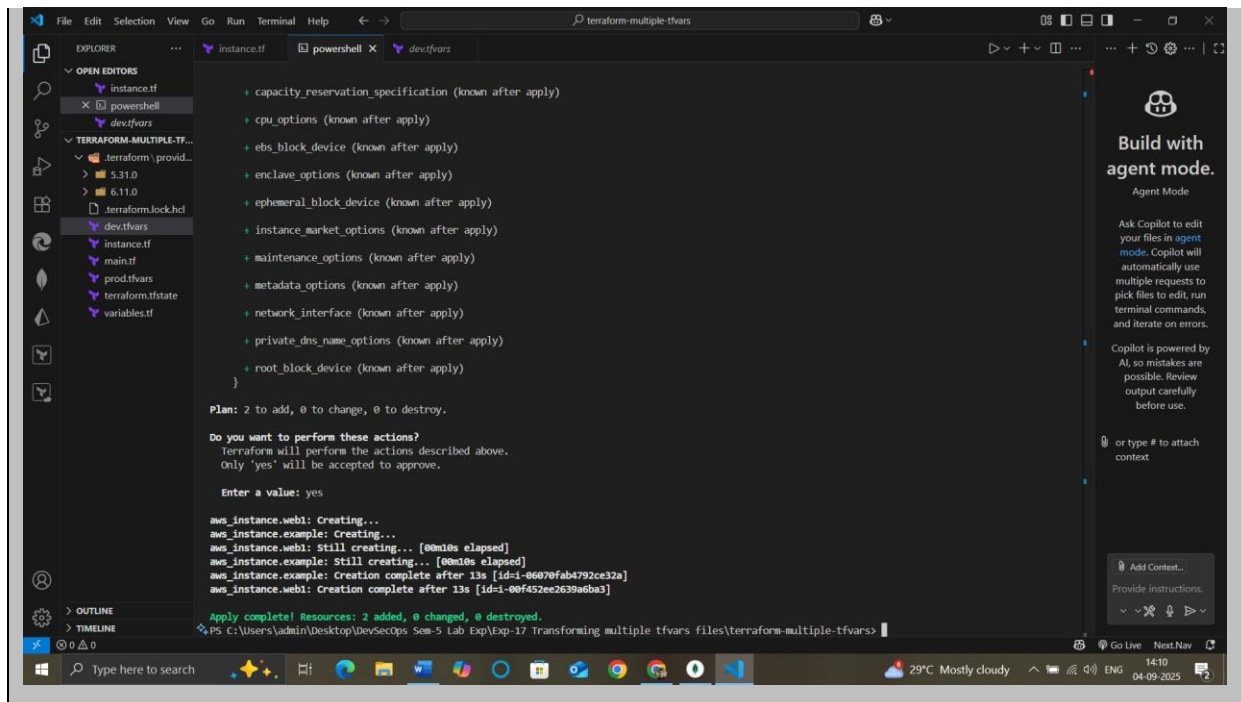
The screenshot shows the VS Code interface with the following details:

- Explorer View:** Displays the project structure. The 'prod.tfvars' file is selected under the 'terraform-multiple-tfvars' folder.
- Open Editors View:** Shows the 'prod.tfvars' file as the active editor.
- Editor View:** Displays the content of 'prod.tfvars', which includes a Terraform configuration snippet for an AWS instance:


```
1 ami = "ami-8861f4c788f5065c4"
2 instance_type = "t2.large"
3
```

- ### 3. Initialize and Apply for Dev Environment:

- # terraform init
- ## terraform apply -var-file=dev.tfvars
- 
- The screenshot shows a Windows IDE with the following components:
- Explorer:** Shows the project structure with files like `instance.tf`, `main.tf`, `dev.tfvars`, and `prod.tfvars`.
 - Editor:** Displays the content of `main.tf`, which includes the `aws_instance.example` resource definition.
 - Terminal:** Shows the output of the `terraform init` and `terraform apply -var-file=dev.tfvars` commands. The output indicates that Terraform has initialized the providers and generated the execution plan.
 - Outline:** Shows the resource `aws_instance.example` and its associated actions.
 - Right Panel:** Displays the "Build with agent mode" section, which includes a "Build with agent mode" button and a "Build with agent mode" button.
- ```
terraform init
terraform apply -var-file=dev.tfvars
```
- The terminal output shows the following steps:
- Terraform initialized the providers.
  - Terraform generated the execution plan.
  - Terraform applied the plan, creating the `aws_instance.example` resource.
- The resource definition in `main.tf` is as follows:
- ```
resource "aws_instance" "example" {
  ami           = "ami-02d26659fd02cf299"
  instance_type = "t2.micro"
  subnet_id     = "subnet-0a5a0503"
  vpc_id        = "vpc-0a5a0503"
  associate_public_ip_address = true
  availability_zone = "us-east-1a"
  cpu_core_count = 1
  cpu_threads_per_core = 2
  disable_api_stop = true
  disable_api_termination = true
  elb_optimized = true
  get_password_data = false
  host_id = "i-0a5a0503"
  host_resource_group_arn = "arn:aws:ec2:us-east-1:123456789012:resource-group/rg-0a5a0503"
}
```

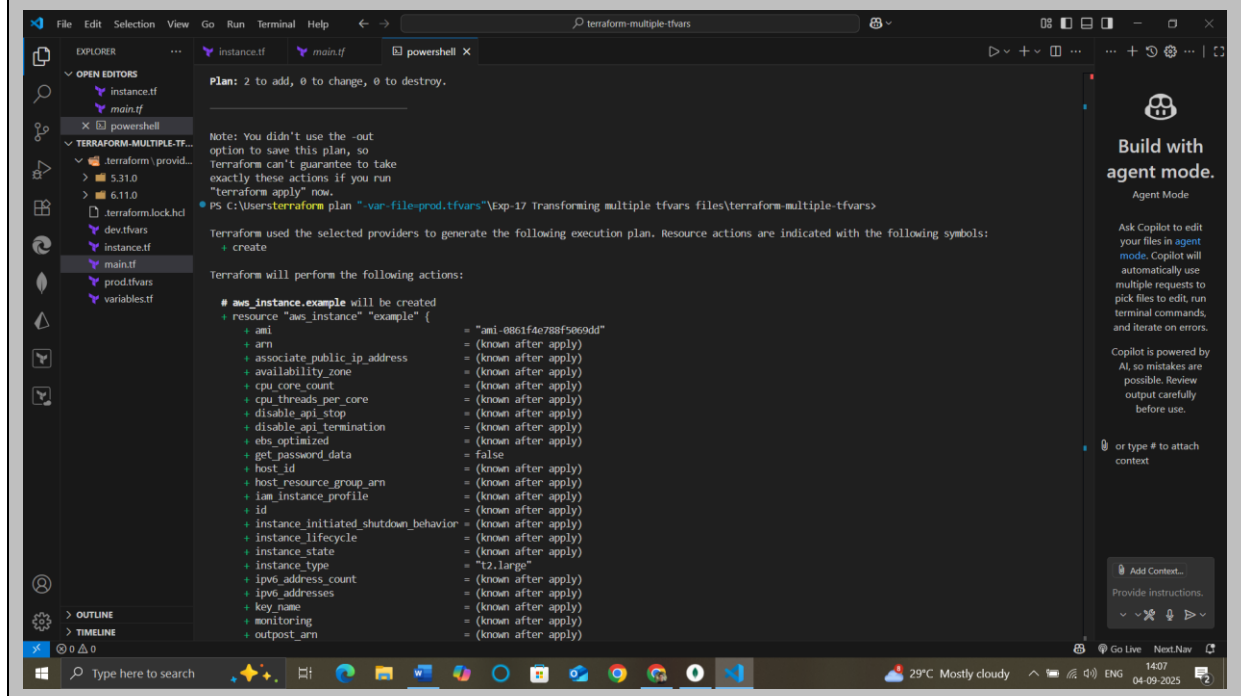


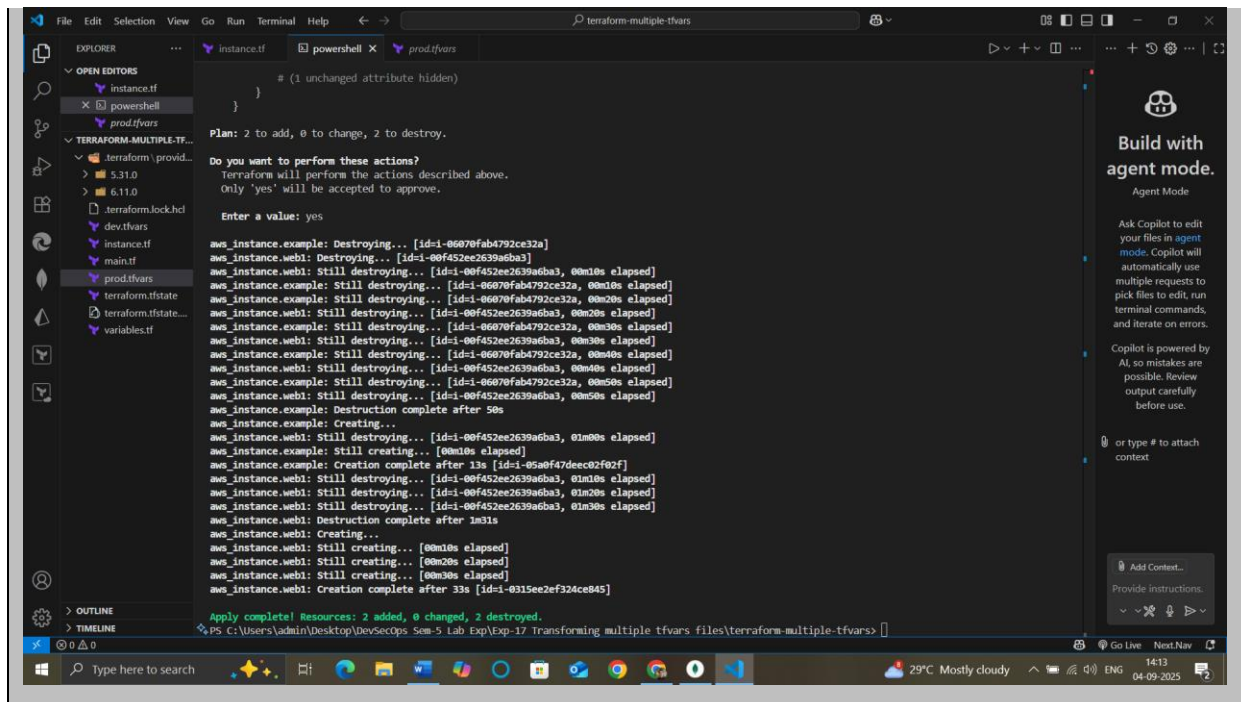
4. Initialize and Apply for Prod Environment:

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

terraform init

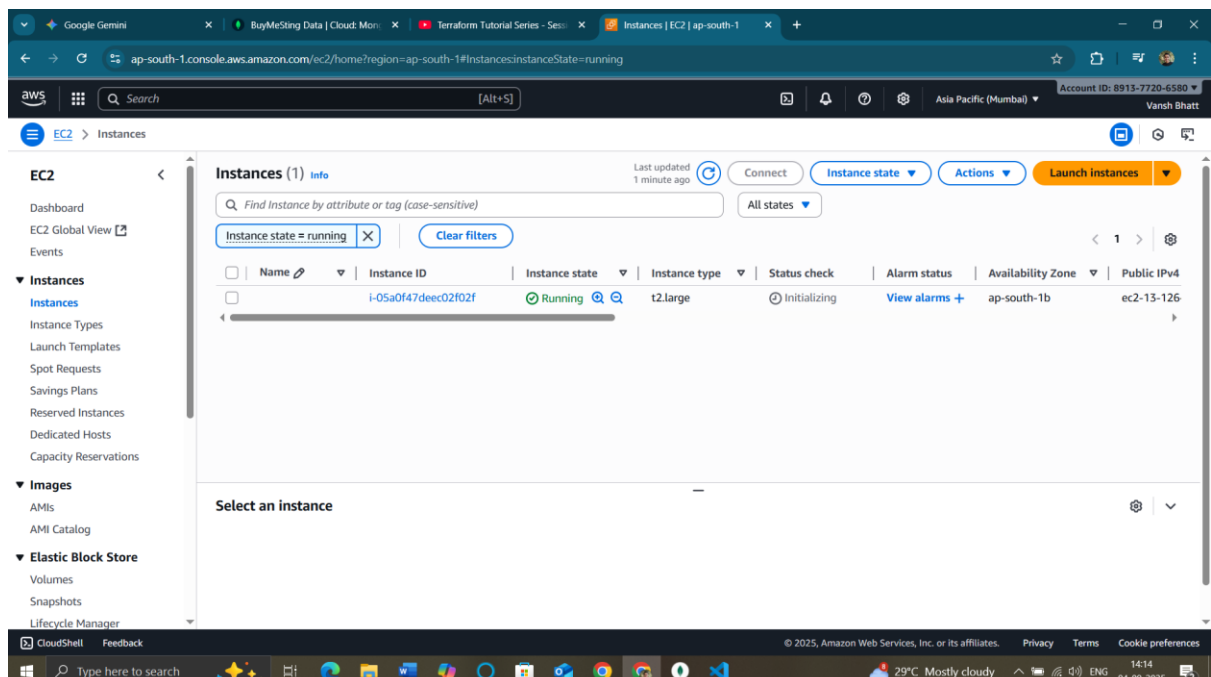
terraform apply -var-file=prod.tfvars





5. Test and Verify:

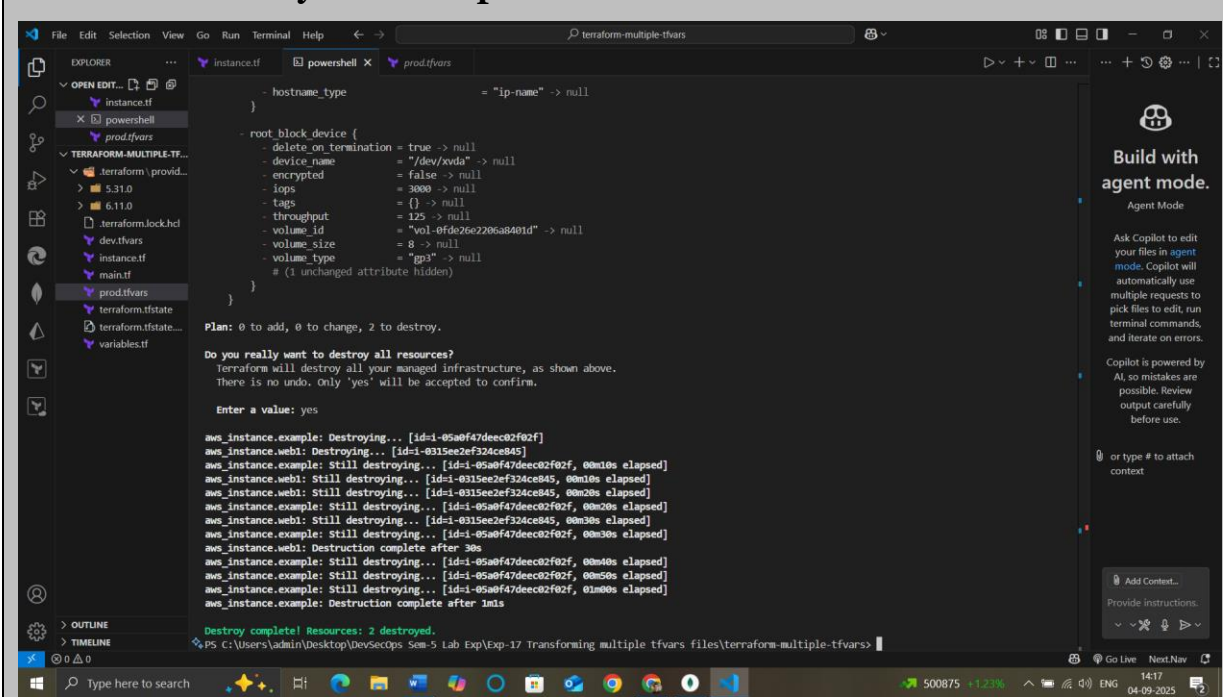
- Observe how different tfvars files are used to set variable values for different environments during the apply process.
- Access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified regions and instance types.



6. Clean Up:

- After testing, you can clean up resources:

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



```
Plan: 0 to add, 0 to change, 2 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-05a0f47deec02f02f]
aws_instance.web1: Destroying... [id=i-0315ee2ef324ce845]
aws_instance.example: Still destroying... [id=i-05a0f47deec02f02f, 00m10s elapsed]
aws_instance.web1: Still destroying... [id=i-0315ee2ef324ce845, 00m10s elapsed]
aws_instance.example: Still destroying... [id=i-05a0f47deec02f02f, 00m20s elapsed]
aws_instance.web1: Still destroying... [id=i-0315ee2ef324ce845, 00m20s elapsed]
aws_instance.example: Still destroying... [id=i-05a0f47deec02f02f, 00m30s elapsed]
aws_instance.web1: Still destroying... [id=i-0315ee2ef324ce845, 00m30s elapsed]
aws_instance.example: Destruction complete after 30s
aws_instance.web1: Destruction complete after 30s
aws_instance.example: Still destroying... [id=i-05a0f47deec02f02f, 00m40s elapsed]
aws_instance.web1: Still destroying... [id=i-05a0f47deec02f02f, 00m50s elapsed]
aws_instance.example: Still destroying... [id=i-05a0f47deec02f02f, 01m00s elapsed]
aws_instance.web1: Still destroying... [id=i-05a0f47deec02f02f, 01m00s elapsed]
aws_instance.example: Destruction complete after 1m1s
aws_instance.web1: Destruction complete after 1m1s

Destroy complete! Resources: 2 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.