

**Name – Rachit Shrivastava**

**Sap Id – 500119571**

**Batch 3 – DevOps**

## **Lab Exercise 17– 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:

**# 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"

```

- Create a file named prod.tfvars: #

**prod.tfvars**

```

ami      = "ami-9876543210fedcbao" 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

```
C:\terraform\terraform-multiple-tfvars>terraform init
Initializing the backend...
Initializing provider plugins...
```

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

```
C:\terraform\terraform-multiple-tfvars>terraform plan -var-file=dev.tfvars
```

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

```
C:\terraform\terraform-multiple-tfvars>terraform apply -var-file=dev.tfvars
```

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 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**

```
C:\terraform\terraform-multiple-tfvars>terraform apply -var-file=prod.tfvars
```

```
No changes. Your infrastructure matches the configuration.
```

```
Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.
```

## 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**

```
enable_resource_name_dns_record = false → null
enable_resource_name_dns_aaaa_record = false → null
hostname_type                    = "ip-name" → null
}

root_block_device {
  delete_on_termination = true → null
  device_name           = "/dev/sda1" → null
  encrypted             = false → null
  iops                  = 3000 → null
  tags                  = {} → null
  tags_all              = {} → null
  throughput            = 125 → null
  volume_id             = "vol-87b6bd3242c39ab71" → 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-809e9146ba7b37e9b]  
aws\_instance.example: Still destroying... [id=i-809e9146ba7b37e9b, 00m10s elapsed]  
aws\_instance.example: Still destroying... [id=i-809e9146ba7b37e9b, 00m20s elapsed]  
aws\_instance.example: Still destroying... [id=i-809e9146ba7b37e9b, 00m30s elapsed]  
aws\_instance.example: Still destroying... [id=i-809e9146ba7b37e9b, 00m40s elapsed]  
aws\_instance.example: Still destroying... [id=i-809e9146ba7b37e9b, 00m50s elapsed]  
aws\_instance.example: Destruction complete after 54s

Destroy complete! Resources: 1 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.