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:**

 # main.tf

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

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

main.tf lab3 | main.tf lab5 | main.tf lab6

lab8 > 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
}
```

```
main.tf lab5    main.tf lab6
lab8 > variables.tf > ...
1  variable "ami" {
2    type = string
3  }
4
5  variable "instance_ty" {
6    type = string
7  }
8
```

2. Create Multiple tfvars Files:

- **Create a file named dev.tfvars:**

 # dev.tfvars

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

```
main.tf lab6    instance.tf lab6    variables.tf
lab8 > dev.tfvars > ...
1  ami           = "ami-0123456789abcdef0"
2  instance_type = "t2.micro"
3
```

- **Create a file named prod.tfvars:**

 # prod.tfvars

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

instance.tf lab6 | variables.tf lab6 | main.tf lab7

lab8 > prod.tfvars > ...

```
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 C:\Github Repositores\Terraform-Demo\lab8> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.83.1...
- Installed hashicorp/aws v5.83.1 (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

```
aws_instance.example: Destroying... [id=i-039ae5d11e807fdb5]
aws_instance.example: Still destroying... [id=i-039ae5d11e807fdb5, 10s elapsed]
aws_instance.example: Still destroying... [id=i-039ae5d11e807fdb5, 20s elapsed]
aws_instance.example: Still destroying... [id=i-039ae5d11e807fdb5, 30s elapsed]
aws_instance.example: Still destroying... [id=i-039ae5d11e807fdb5, 40s elapsed]
aws_instance.example: Still destroying... [id=i-039ae5d11e807fdb5, 50s elapsed]
aws_instance.example: Still destroying... [id=i-039ae5d11e807fdb5, 1m0s elapsed]
aws_instance.example: Destruction complete after 1m0s
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 12s [id=i-04c24518cca9ea461]

Apply complete! Resources: 1 added, 0 changed, 1 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

```
PS C:\Github Repositores\Terraform-Demo\lab8> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.83.1...
- Installed hashicorp/aws v5.83.1 (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=prod.tfvars

```
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 12s [id=i-039ae5d11e807fdb5]

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

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.**
-

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
	i-04c24518cca9ea461	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1b	ec2-13-1

6. Clean Up:

- **After testing, you can clean up resources:**

terraform destroy -var-file=dev.tfvars

```
PS C:\Github Repositores\Terraform-Demo\lab8> terraform destroy -var-file="dev.tfvars"
aws_instance.example: Refreshing state... [id=i-04c24518cca9ea461]

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-07b69f62c1d38b012" -> null
      - arn                                  = "arn:aws:ec2:ap-south-1:979211864468:instance/i-04c24518cca9ea461" -> null
      - associate_public_ip_address          = true -> null
      - availability_zone                    = "ap-south-1b" -> null
```

terraform destroy -var-file=prod.tfvars

```
PS C:\Github Repositores\Terraform-Demo\lab8> 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.
```

- **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.