Lab Exercise 8– Terraform Multiple tfvars Files

Objective:

Aditya Tomar

500106015

R2142221060

Batch-2(DevOps)

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:

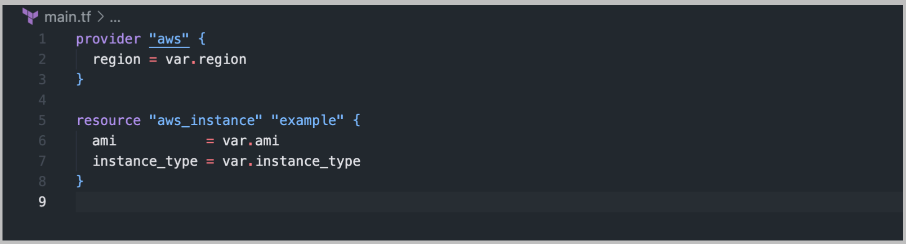
```
mkdir terraform-multiple-tfvars
cd terraform-multiple-tfvars
```

- 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
}
```

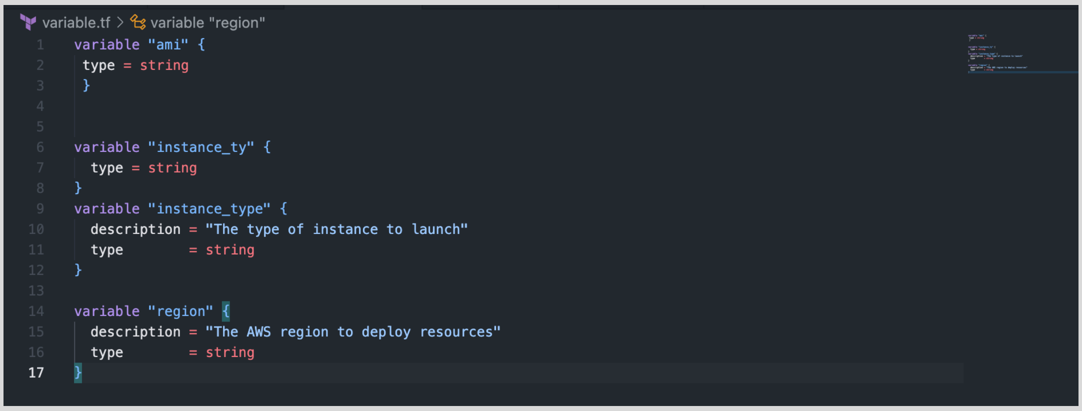


```
main.tf > ...
provider "aws" {
  region = var.region
}

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
}
```



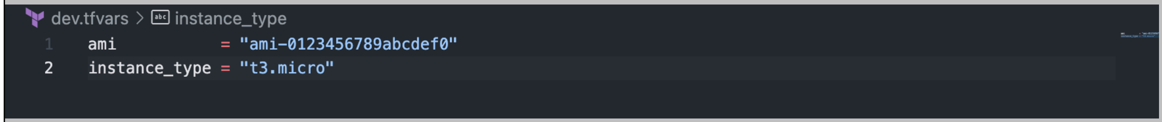
```
variable.tf > variable "region"
variable "ami" {
 type = string
 }


variable "instance_ty" {
  type = string
}
variable "instance_type" {
  description = "The type of instance to launch"
  type        = string
}

variable "region" {
  description = "The AWS region to deploy resources"
  type        = string
}
```

2. Create Multiple tfvars Files:

- Create a file named dev.tfvars:

# dev.tfvars

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

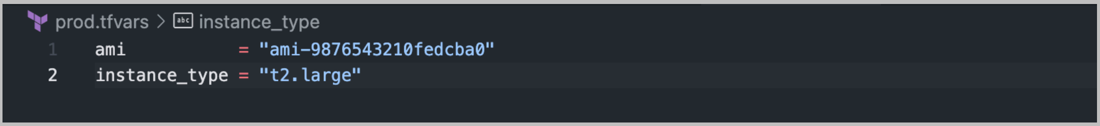


```
dev.tfvars > instance_type
ami           = "ami-0123456789abcdef0"
instance_type = "t3.micro"
```

- Create a file named prod.tfvars:

# prod.tfvars

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



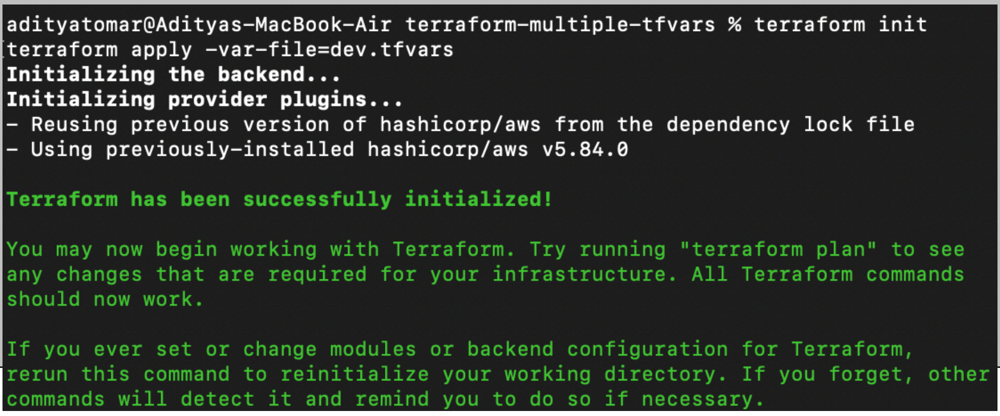
```
prod.tfvars > instance_type
ami           = "ami-9876543210fedcba0"
instance_type = "t2.large"
```

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

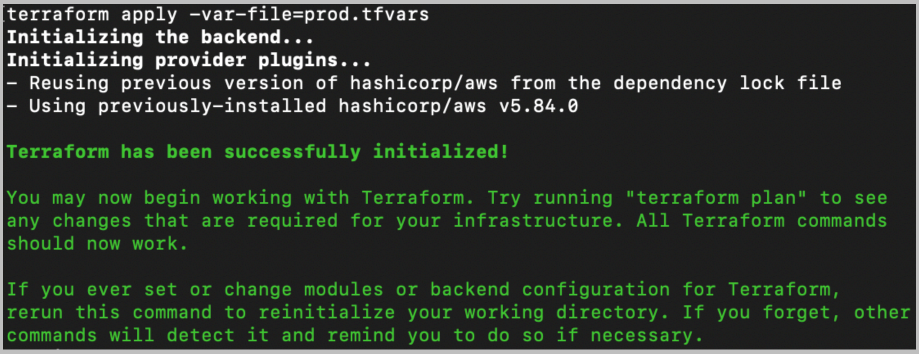


```
adityatomar@Adityas-MacBook-Air terraform-multiple-tfvars % terraform init
terraform apply -var-file=dev.tfvars
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.84.0

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.
```

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
```



```
terraform apply -var-file=prod.tfvars
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.84.0

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.
```

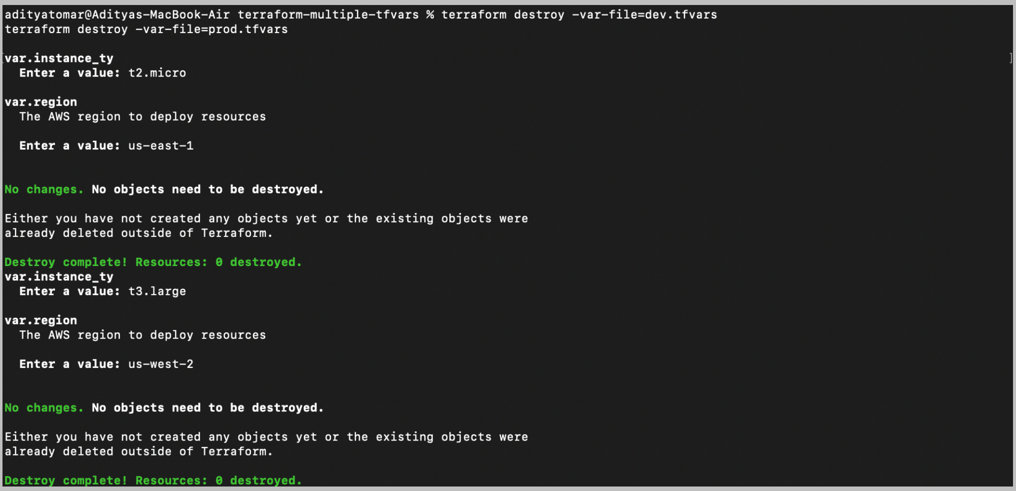
5. Test and Verify:

- 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
```



```
adityatomar@Adityas-MacBook-Air terraform-multiple-tfvars % terraform destroy -var-file=dev.tfvars
terraform destroy -var-file=prod.tfvars

var.instance_ty
  Enter a value: t2.micro

var.region
  The AWS region to deploy resources

  Enter a value: us-east-1


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.
var.instance_ty
  Enter a value: t3.large

var.region
  The AWS region to deploy resources

  Enter a value: us-west-2


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.