Lab 2 – Terraform Folder Structure, Files & Basic Workflow

Creator: Sandeep Kumar Sharma

Learning Objectives

- Understand the essential files used in every Terraform project.
- Learn how Terraform organizes configuration, state, and variables.
- Understand the purpose of `main.tf`, `variables.tf`, `outputs.tf`, and `providers.tf`.
- Learn how to write clean, production-ready Terraform folder structures.
- Deploy a simple AWS resource using a properly structured Terraform project.

Learning Outcome

By the end of this lab, the learner will be able to: - Describe Terraform's recommended folder layout. - Split Terraform configuration into multiple `.tf` files. - Use variables and outputs in a real project. - Run Terraform using best practices. - Deploy infrastructure on AWS following clean standards.

Concept Explanation (Natural Style)

In the previous lab, we wrote everything inside a single `main.tf` file. This is okay for small experiments, but not good for real projects.

In real-world environments, your Terraform code must be clean, readable, and maintainable. Imagine writing a 2,000-line configuration in one file — impossible to understand and difficult to manage.

So Terraform gives you flexibility: - You can split your code into different `.tf` files. - Terraform automatically loads all `.tf` files in the folder. - This helps your team collaborate and maintain the infrastructure easily.

Think of this like arranging items in your home: - Clothes in a cupboard - Utensils in the kitchen - Tools in the workshop

Everything has its own place.

Terraform also expects your project to be organized so the next team member can understand it quickly.

Terraform Project File Structure (Standard Layout)

A simple, clean Terraform project looks like this:

```
terraform-lab2/
│
├── main.tf
├── variables.tf
├── outputs.tf
└── providers.tf
```

Each file has a purpose: - **providers.tf** → tells Terraform which cloud provider to use - **variables.tf** → stores dynamic values to avoid hardcoding - **main.tf** → stores actual resources that you create - **outputs.tf** → displays information after Terraform finishes

---

Step-by-Step Hands-On Lab

Step 1: Create the Lab Folder

```
mkdir terraform-lab2
cd terraform-lab2
```

Step 2: Create the Required Terraform Files

```
touch providers.tf
touch variables.tf
touch main.tf
touch outputs.tf
```

Step 3: Configure the Provider (providers.tf)

Open **providers.tf** and add:

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

This tells Terraform to use AWS and take the region from a variable.

Step 4: Declare Variables (variables.tf)

Open **variables.tf**:

```
variable "aws_region" {
  description = "AWS region where resources will be created"
  type        = string
  default     = "ap-south-1"
}

variable "bucket_name" {
  description = "Name of the S3 bucket"
  type        = string
}
```

Here we created: - A region variable (with default value) - A bucket name (no default — user must input it)

Step 5: Create AWS Resource (main.tf)

Open **main.tf**:

```
resource "aws_s3_bucket" "lab2_bucket" {
  bucket = var.bucket_name
  acl    = "private"
}
```

This uses variables instead of hardcoded values.

Step 6: Defining Outputs (outputs.tf)

Open **outputs.tf**:

```
output "bucket_name_output" {
  description = "Name of the bucket created"
  value       = aws_s3_bucket.lab2_bucket.bucket
}
```

After applying, Terraform will show the bucket name as output.

Step 7: Initialize Terraform

```
terraform init
```

This sets up the project and installs AWS provider.

Step 8: Provide Input Variables

Since `bucket_name` has no default, use:

```
echo "bucket_name=\"sandeep-lab2-terraform-bucket-001\"" > terraform.tfvars
```

Or you can pass manually during plan/apply.

Step 9: Run Terraform Plan

```
terraform plan
```

Terraform will show: - One resource to add - Variables used

Step 10: Apply the Configuration

```
terraform apply
```

Type **yes** when prompted.

Your S3 bucket is now created using a professional Terraform project structure.

Step 11: Validate in AWS Console

Go to AWS → S3 → search for your bucket.

Step 12: Destroy the Resources (Optional)

```
terraform destroy
```

This removes everything.

Summary

In this lab, you learned: - Why Terraform code should be modular and clean. - How to split configurations into `main.tf`, `variables.tf`, `outputs.tf`, and `providers.tf`. - How to use variables and outputs properly. - How to deploy AWS infrastructure using a production-ready file structure.

You now have a strong foundation to move towards more complex labs.

End of Lab 2