Lab 4 – Terraform State File & Remote Backend (S3) Explained

Creator: Sandeep Kumar Sharma

Learning Objectives

- Understand what the Terraform state file is and why it is critically important.
- Learn where and how Terraform stores infrastructure information.
- Understand the lifecycle of the `terraform.tfstate` file.
- Learn problems with local state and why remote backends are used.
- Configure an AWS S3 backend for Terraform state.
- Learn about state locking using DynamoDB.

Learning Outcome

By the end of this lab, the learner will be able to: - Explain Terraform state in simple, real-world terms. - Identify the risks of using local state files. - Configure Terraform to use S3 as a remote backend. - Enable DynamoDB table for state locking. - Run Terraform with a production-ready backend setup.

Concept Explanation (Natural Style)

Let's understand the Terraform state file in the simplest way possible.

Imagine Terraform is a builder constructing your AWS infrastructure. To know: - what is already created, - what needs to be updated, - what needs to be deleted, Terraform needs a memory.

This memory is the **Terraform State File**.

🧠What is the Terraform State File?

Terraform creates a file called `terraform.tfstate` in your working directory. This file contains: - IDs of created resources - Current configuration details - Dependencies - Metadata

Think of it like a *map* of all infrastructure Terraform manages.

👋🏼Why Do We Need a State File?

Without the state file, Terraform cannot: - know what already exists - calculate a proper `plan` - understand changes - destroy the right resources

👉🏿Problems With Local State

If the state file is stored locally: - It can get deleted accidentally. - Multiple team members can override each other. - CI/CD pipelines cannot access it. - Sensitive data may be exposed. - No state locking → leads to corruption.

That's why local state is **not recommended** for production.

👍Remote Backend (S3 + DynamoDB)

To solve these problems, Terraform supports **remote backends**, such as: - S3 - Azure Blob - GCP Storage - Terraform Cloud

In AWS, the most common production setup is: - **S3 bucket** → stores the state file - **DynamoDB table** → provides state locking

This prevents multiple users from making changes at the same time.

Step-by-Step Hands-On Lab

Step 1: Create a New Lab Folder

```
mkdir terraform-lab4-state
cd terraform-lab4-state
```

Step 2: Create Required Files

```
touch main.tf
```

Step 3: Create S3 Bucket for State Storage

Before configuring the backend, we must first create an S3 bucket manually or with Terraform.

Create S3 bucket using Terraform:

```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_s3_bucket" "tf_state_bucket" {
  bucket = "sandeep-terraform-state-bucket-lab4" # must be globally unique
  acl    = "private"

  versioning {
    enabled = true
  }
}
```

Apply this once:

```
terraform init
terraform apply
```

After this step, delete this code from `main.tf` because backend configuration cannot use resources from the same config.

Copy the bucket name for next steps.

Step 4: Create DynamoDB Table for Locking

Use Terraform to create a lock table:

```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_dynamodb_table" "terraform_locks" {
  name         = "terraform-lock-table-lab4"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}
```

Apply this once:

```
terraform init
terraform apply
```

Now the DynamoDB table is ready.

Step 5: Configure Remote Backend

Now remove all previous resource code.

Update `main.tf` to:

```
terraform {
  backend "s3" {
    bucket         = "sandeep-terraform-state-bucket-lab4"
    key            = "global/terraform.tfstate"
    region         = "ap-south-1"
    dynamodb_table = "terraform-lock-table-lab4"
    encrypt        = true
  }
}

provider "aws" {
  region = "ap-south-1"
}

resource "aws_s3_bucket" "sample_bucket" {
  bucket = "sandeep-lab4-sample-bucket-12345"
  acl    = "private"
}
```

What this does:

- Tells Terraform to store `terraform.tfstate` in S3
- Enables locking using DynamoDB
- Encrypts state at rest

Step 6: Initialize Backend

```
terraform init
```

Terraform will detect backend configuration and ask to migrate state.

Type **yes**.

State file will move from local → S3.

Step 7: Apply the Configuration

```
terraform plan
terraform apply
```

This creates a sample bucket and stores state remotely.

Step 8: Verify State in AWS

In S3:

- Open your state bucket
- Navigate to `global/terraform.tfstate`
- Confirm the file exists

In DynamoDB:

- Open your table → terraform-lock-table-lab4
- Run any Terraform command to see lock entries

Step 9: Destroy Resources (Optional)

Only destroy the sample bucket, not the backend itself:

```
terraform destroy
```

Your backend remains intact.

Summary

In this lab, you learned: - What the Terraform state file is and how Terraform uses it. - Why local state is dangerous for teams and production. - How to configure S3 as a remote backend. - How DynamoDB provides state locking to prevent corruption. - How to migrate local state to remote state. - How to store Terraform state securely and professionally.

You are now ready for more advanced Terraform concepts like: variables, modules, EC2 with security groups, VPC creation, and reusable infrastructure.

End of Lab 4