**Terraform statefile notes and lab as of 03-01-25**

**Day 3: Terraform.tfstate file**

The terraform.tfstate file is a critical component of Terraform that stores the state of your infrastructure. In simple words, the statefile track the resources from desired state to current state, contains current state info(remote).

**What is terraform.tfstate?**

The terraform.tfstate file is a JSON file that keeps track of the current state of your infrastructure as managed by Terraform. It acts as a **mapping** between your configuration (.tf files) and the actual resources in the cloud or other providers.

Notes to remember:

 The terraform.tfstate file is essential for tracking and managing your infrastructure state.

 Handle it with care to avoid issues like state corruption, conflicts, or security breaches.

 Use remote backends and follow best practices to manage the state effectively.

A diagram of a state

Description automatically generated

A diagram of a business process

Description automatically generated

Let’s get started with lab …

First create a folder name -> Day-3-statefile

Second add provider.tf and main.tf files to the Day-3 folder and initiate running **terraform init** command.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

When you run – **terraform apply -auto-approve**

One instance gets created in AWS ec2 end, also you get to see the terraform.tfstate file also getting created in the working directory.

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

When we re-run the **terraform apply -auto-approve**

Output of the result is NO CHANGE.

A screen shot of a computer

Description automatically generated

Let’s see some use cases and studies with terraform.tfstate file

Case 1: Remote change

We are going to manually edit and create some changes in the remote, to see if we get any result out of it.

A screenshot of a computer

Description automatically generated

I’m going to change the Name from TerrformTest to Dev

A screenshot of a computer

Description automatically generated

Im going to re-run the command - **terraform apply -auto-approve**

**It takes the desired state only, and doesn’t take the manually change done by us in the remote(ec2 instance)**

**It says 1 to change**

A computer screen with white text

Description automatically generated

It goes back to original state main.tf tag name = TerrformTest

A screenshot of a computer

Description automatically generated

Case 2: Removing or adding a change in main.tf

Now, im going to delete tags and re-run from desired state to see the current state in the ec2 instance

From

resource "aws\_instance" "test" {

ami = "ami-019374baf467d6601"

instance\_type = "t2.micro"

key\_name = "LondonKP"

tags = {

Name = "TerraformTest"

}

}

To

resource "aws\_instance" "test" {

ami = "ami-019374baf467d6601"

instance\_type = "t2.micro"

key\_name = "LondonKP"

}

Now re-run apply command …

Plan: 0 to add, 1 to change, 0 to destroy.

aws\_instance.test: Modifying... [id=i-03a36ef57213687f2]

aws\_instance.test: Modifications complete after 2s [id=i-03a36ef57213687f2]

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

The result in the ec2 instance will be with no name like below …

A screenshot of a computer

Description automatically generated

So, conclusion is whatever change we do in the ec2 end manually, once we apply the terraform apply -auto-approve, the recent changes done via ec2 end will be gone or it will override with the desired state to current state.

Case 3: Changing inside statefile

Im going to delete the ami inside the terraform.tfstate file, and going to run terraform apply -auto-approve

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

It automatically gets updated back, after running apply command

Lumlas-MacBook-Pro:Day-3-statefile Lumla$ terraform apply -auto-approve

aws\_instance.test: Refreshing state... [id=i-03a36ef57213687f2]

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.

A screen shot of a computer code

Description automatically generated

**Please Note:** It is not advisable to change the **terraform.tfstate** file at any point to avoid inconsistencies

Case 4: Applying destroy command

When we apply -> **terraform destroy -auto-approve** command, all the previous information inside the statefile will be gone. So, if an instance is running, it provides all information about the resources inside the statefile, but once we destroy an instance, all information will be gone inside the terraform.tfstate file.

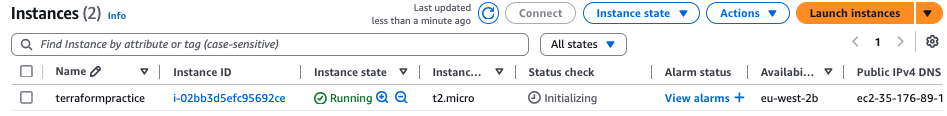
It comes to null stage … like below

A screen shot of a computer program

Description automatically generated

Case 5: with 2 developer’s collaboration problems

Connect the ec2 instance(Amazon Linux) and install **terraform** and **git**.



Get the code

<https://github.com/CloudTechDevOps/Terraform-main/blob/main/project-terraform-devops-tools-install/install.sh>

or install from official site too - https://developer.hashicorp.com/terraform/install

Use the below commands to install two platforms …

sudo su –

sudo yum install -y yum-utils

sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo

sudo yum -y install terraform

sudo yum install git -y



First developer pulled our repo from git clone, which has only two days …

Getting inside day-1, initialising terraform and running apply command

If role is not configured, you will get this error ..

Applying role to EC2 instance

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Now, run – terraform apply -auto-approve

A screen shot of a computer

Description automatically generated

One instance got created in Day-1 by developer 1

A screenshot of a computer

Description automatically generated

But the second developer working in local and created day-3 too, what if he also run apply command in their end in Day-1 directory?

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Now, two instances got created by two developers from day-1… since resource are same and, but state file is different. If state file is different, we cannot work collaboratively. So, to store sensitive information, we need s3 bucket and to work collaboratively.

Please Note: if working inside ec2, **role** is needed, outside ec2 connection of ec2 needs **key**

Case 6: Working with multiple developer’s solution

The statefile of the 2 developers working in a same project is different unless they have the common storage location(s3 bucket) to store the terraform.tfstate file.

It is also not recommended to work individually in the same project, if the state file doesn’t have the same storage location, as it creates complexities/conflicts in the working project.

So, to avoid confusion, adding the statefile in the s3 bucket is preferred.

Create a file called -> backend.tf and inside add the following …

terraform {

backend "s3" {

bucket = "lumlabucket"

key = "terraform.tfstate"

region = "eu-west-2"

}

}

Once you run the terraform apply -auto-approve

One instance gets created in the aws end, also you can see the **terraform.tfstate** also gets created in the s3 location

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Please note,** before adding statefile in the s3 bucket, make sure, you have a bucket present to store the file, in the same region as ec2 creation, also give the same in the provider.tf.

Case 7: Conflict resolved with collaboration

Now git push the added day-3 files in the remote GitHub

A screenshot of a computer screen

Description automatically generated

Now, let the other developer working in remote end, into the Day-3 and try running terraform apply -auto-approve command

[root@ip-172-31-38-191 TerraformClass]# ls

Day-1-Terraform-Basics Day-2-Terraform-tfvars

[root@ip-172-31-38-191 TerraformClass]# git pull

remote: Enumerating objects: 7, done.

remote: Counting objects: 100% (7/7), done.

remote: Compressing objects: 100% (4/4), done.

remote: Total 6 (delta 1), reused 6 (delta 1), pack-reused 0 (from 0)

Unpacking objects: 100% (6/6), 633 bytes | 316.00 KiB/s, done.

From https://github.com/lumlathomas/TerraformClass

1bda28f..4d3c3ed main -> origin/main

Updating 1bda28f..4d3c3ed

Fast-forward

Day-3-statefile/backend.tf | 7 +++++++

Day-3-statefile/main.tf | 6 ++++++

Day-3-statefile/provider.tf | 3 +++

3 files changed, 16 insertions(+)

create mode 100644 Day-3-statefile/backend.tf

create mode 100644 Day-3-statefile/main.tf

create mode 100644 Day-3-statefile/provider.tf

[root@ip-172-31-38-191 TerraformClass]# ls

Day-1-Terraform-Basics Day-2-Terraform-tfvars Day-3-statefile

[root@ip-172-31-38-191 TerraformClass]# cd Day-3-statefile/

[root@ip-172-31-38-191 Day-3-statefile]# terraform init

Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically

use this backend unless the backend configuration changes.

Initializing provider plugins...

- Finding latest version of hashicorp/aws...

- Installing hashicorp/aws v5.82.2...

- Installed hashicorp/aws v5.82.2 (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.

[root@ip-172-31-38-191 Day-3-statefile]# terraform apply -auto-approve

aws\_instance.test: Refreshing state... [id=i-0ac535e6b73375a9d]

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.

[root@ip-172-31-38-191 Day-3-statefile]#

No, extra resource gets created, because state file is common to both developers as it is stored in s3 bucket, hence no extra ec2 gets created or no duplication of instance created.

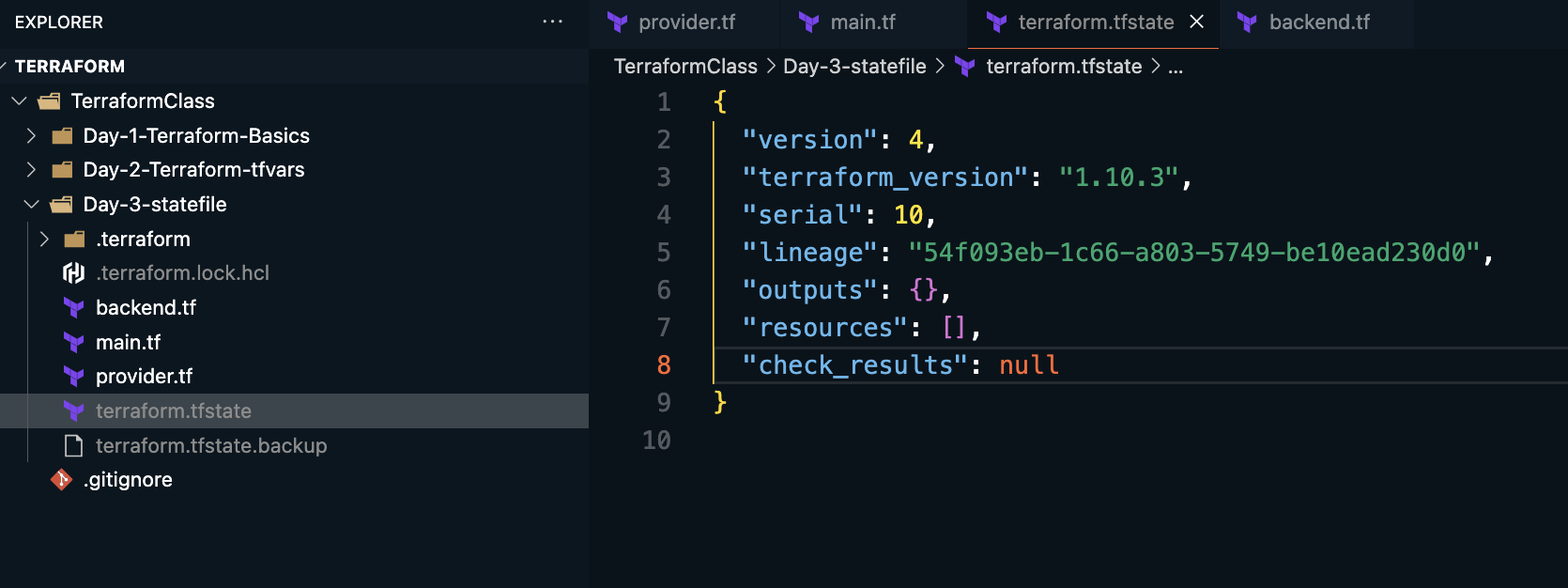
A screenshot of a phone

Description automatically generated

So, if we apply or delete a resource from any end remote or local, the result will be same.

I’ve destroyed the newly created ec2 instance from remote end, in local also it gets deleted.

Now, we can delete the terraform.tfstate file in the local, as we have kept the state file in the s3 bucket.



A screenshot of a computer

Description automatically generated

Case 8: Changing az from 1a to 1b

A screenshot of a computer program

Description automatically generated

The running instance will get destroy first and create again with the given AZ.

A screenshot of a computer

Description automatically generated

Run terraform apply -auto-approve

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Please remember to update, every time you made a change, hence there won’t be any collaboration issue.

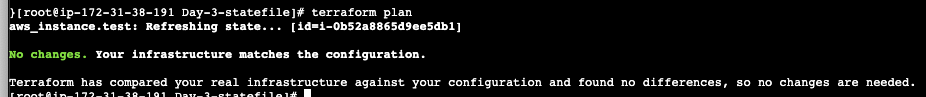
Also, before applying any changes, always check terraform plan for double confirmation, if it says,

A black background with white text

Description automatically generated

It means, there is a difference between local and remote developers, so remember to git pull always and do the apply changes.

We must see (No Changes) to proceed further.



Day 4: Terraform.tfstate file – Lock system

A screenshot of a computer

Description automatically generated

If 2 developers try to run terraform apply at the same time with different changes, if locking system(dynamoDB) is enable means we wouldn’t get any conflicts ( *like 2 instances/buckets will not be created, where one gets the information in the state file and the other ones wouldn’t have one, and it may be hard to destroy from local end, we may have to manually delete the changes made by one of the developer’s work, see below for clarification* )

What is DynamoDB?

Amazon DynamoDB is a fully managed, serverless, NoSQL database service from Amazon Web Services (AWS):

**Features**

DynamoDB is a key-value database that allows users to create tables to store and retrieve data. It automatically scales to handle the amount of data and request traffic, and it's designed to provide fast and predictable performance.

**DynamoDB Locking in Terraform**

DynamoDB locking is a mechanism used by Terraform to ensure **state file consistency** and prevent simultaneous operations that could corrupt the terraform.tfstate file. This is especially useful when multiple users or CI/CD pipelines are interacting with the same Terraform state.

**How DynamoDB Locking Works**

1. **State Lock Table**:
   * A DynamoDB table is created and acts as a locking mechanism for the Terraform state.
   * When a terraform plan or terraform apply operation is initiated, Terraform writes a "lock record" to the DynamoDB table.
2. **Locking**:
   * Terraform checks for an existing lock before proceeding with any operation.
   * If a lock exists, Terraform waits or exits with an error, depending on the retry settings.
3. **Releasing the Lock**:
   * After the operation is complete, Terraform removes the lock record from the DynamoDB table, allowing other operations to proceed.

DynamoDB locking ensures safe and consistent access to the terraform.tfstate file when using the S3 backend. It prevents simultaneous operations, protecting the state from corruption. Setting up and using DynamoDB locking is straightforward and is a best practice for multi-user or automated workflows.

Let’s dive into lab to see conflicts and resolution…

Let’s see some conflicts if 2-developers try to implement two different changes at the same time …

One developer changing instance\_type = “t2.micro” and the other with “t2.nano”

Remote developer with t2.micro

A computer screen with text and symbols

Description automatically generated

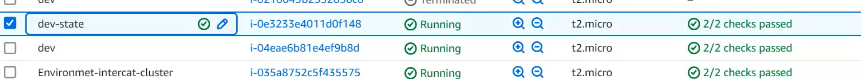
Local developer changed t2.nano

A screen shot of a computer program

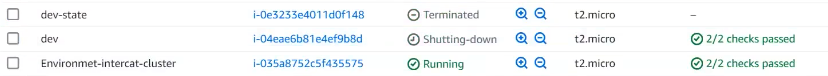
Description automatically generated

Now, try and run, terraform apply -auto-approve at the same time at both the end.

Two servers got created inspite of having. tfstate file, that’s called the conflict, as state file reading or running our request as two, and not ONE. We got only one input from .tfstate (dev-state) and not from (dev).



We have to manually delete one, as it will be impossible to delete from the other end, as we could find the history in the .tfstate file.



That will be the behaviour if we run at the same time, but we run in different time, two servers wouldn’t have created like above. Also,, if .tfstate file is busy it would not take the request at the same time.

So, what is the solution if multiple developers are working at the same time, and that’s when Locking via DynamoDB come to place.

In the backend block, we need to add this code for lock system to work.

**Setting up our S3 Backend**

Create a new file in your working directory labelled Backend.tf

Copy and paste this configuration in your source code editor in your backend.tf file.

terraform {

backend "s3" {

encrypt = true bucket = "hella-buckets"

dynamodb\_table = "terraform-state-lock-dynamo"

key = "terraform.tfstate"

region = "us-east-1"

}

}

**Creating our DynamoDB Table**

Create a new file in your working directory labelled dynamo.tf

Copy and paste this configuration in your source code editor in your main.tf file.

resource "aws\_dynamodb\_table" "dynamodb-terraform-state-lock" {

name = "terraform-state-lock-dynamo"

hash\_key = "LockID"

read\_capacity = 20

write\_capacity = 20

attribute {

name = "LockID"

type = "S"

}

}

Take this code and modify accordingly …

# Run first terraform-10-30am/day-4-resources\_for\_backend\_s3\_dynamodb to create reources

# This backend configuration instructs Terraform to store its state in an S3 bucket.

terraform {

backend "s3" {

bucket = "nareshmulticloud" # Name of the S3 bucket where the state will be stored.

region = "us-east-1"

key = "/day-2/terraform.tfstate" # Path within the bucket where the state will be read/written.

dynamodb\_table = "terraform-state-lock-dynamo" # DynamoDB table used for state locking, note: first run day-4-bckend resources then day-5-backend config

encrypt = true # Ensures the state is encrypted at rest in S3.

}

}

To make things work, we need to create a resource block before and call DynamoDB table like below in the main.tf ..

provider "aws" {

}

resource "aws\_s3\_bucket" "terraformbucket" {

bucket = "lumlabucketterraform"

}

resource "aws\_dynamodb\_table" "dynamodb-terraform-statefile-lock" {

name = "terraform-state-file-dynamo"

hash\_key = "LockID"

read\_capacity = 20

write\_capacity = 20

attribute {

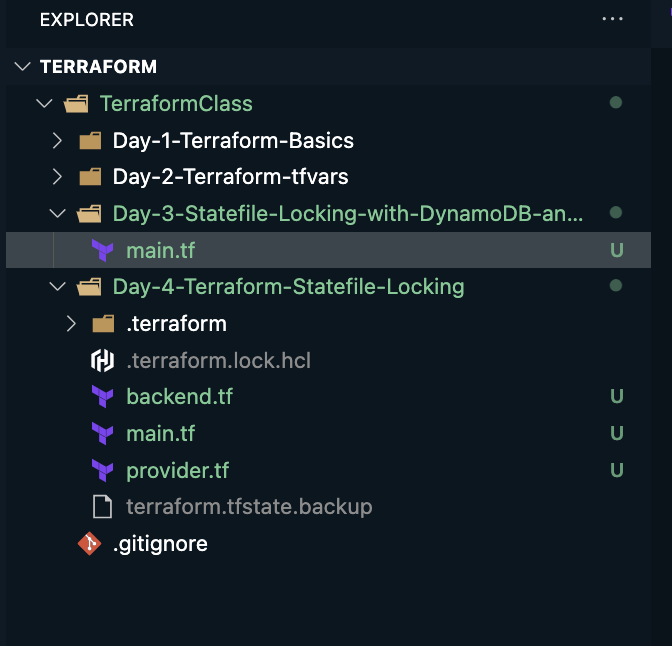
name = "LockID"

type = "S"

}

}

We have to run this file first to create dynamodb to make the locking work.



Now, Day-3 has only one file (main.tf) with the below information …

A screenshot of a computer program

Description automatically generated

First, we need to run the day-3, initialize and run apply command …

One bucket gets created in s3

A screenshot of a web page

Description automatically generated

Also, one dynamodb also get created …

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

Now, Cd into Day-4, initialize it.

terraform init -reconfigure

then git add .

git commit -m “Day-4 added with dynamodb and s3”

git push

A screenshot of a computer

Description automatically generated

Use the below commands to install two platforms (terraform and git), also make sure role is configured too…

sudo su –

sudo yum install -y yum-utils

sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo

sudo yum -y install terraform

sudo yum install git -y

git clone https://github.com/lumlathomas/TerraformClass.git

Now cd into day-4 in local and rempote

In remote- apply terraform apply -auto-approve

A screenshot of a computer program

Description automatically generated

One Terraformtest instance is running with local apply run command

A screenshot of a chat

Description automatically generated

Now go to remote and run

terraform init

check terraform plan

then run terraform apply -auto-approve

(you will see no change)

A screenshot of a computer program

Description automatically generated

And when we try to create simultaneously after making a change( it wont work, it will say lock system is enabled), we won’t create an extra resource, because we have given lock system

Let’s make a change in main.tf, change name to **dev** and other to **test**

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Let’s apply both the end at the same time … Which ever comes comes first, it takes the first one – Like First In First Out (FIFO)

We wil get this message “**Error**: Error acquiring the state lock”

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

You can create dynamoDb manual also …

Just give table name and Partition key and create table

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Now, let’s learn how work with statefile right without overriding the content …

Now create a folder – Day-5-satefile-override

Create two files – main.tf and backend.tf

A screenshot of a computer

Description automatically generated

Main.tf

A screen shot of a computer

Description automatically generated

Backend.tf

A screenshot of a computer program

Description automatically generated

Initiate it first

terraform init ( if needed you have to give terraform init -reconfigure)

then check terraform plan

since we have ec2 already created, it has to destroy ec2 to create s3 bucket

**A screen shot of a computer

Description automatically generated**

**A computer screen shot of white text

Description automatically generated**

**It will say, one to add and one to destroy, so best practice is to give a folder (main.tf -> key – folder1/terraform.tfstate)for not overriding**

A screenshot of a computer program

Description automatically generated

Now, initiate with (terraform init -reconfigure) once more and then check (terraform plan)

Now, it shows right, 1 to add …

A screen shot of a computer

Description automatically generated

Now git add .

Git commit -m “modified”

Git push

A screenshot of a computer

Description automatically generated

Done!