Terraform Task

**Name:Shaik Khaja Basha**

**Batch : Batch 11**

**Date : 19.07.2025**

**Task : File creations in .tf & make corrections**

1. **Create a file and Subsequent apply, destroy**

Ans: for subsequent apply we ill create a file name file\_kbs.tf

Enter the content in the file\_kbs.tf like

**resource "local\_file" "filecreation1" {**

**filename = "abc\_1.txt"**

**content = "This is the content of abc\_1."**

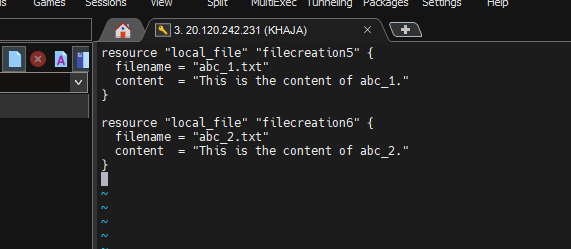
**}**

**resource "local\_file" "filecreation2" {**

**filename = "abc\_2.txt"**

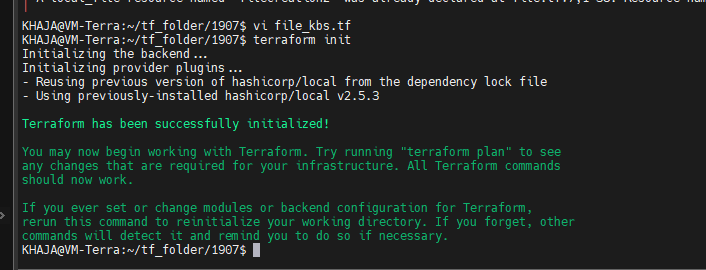
**content = "This is the content of abc\_2."**

**}**

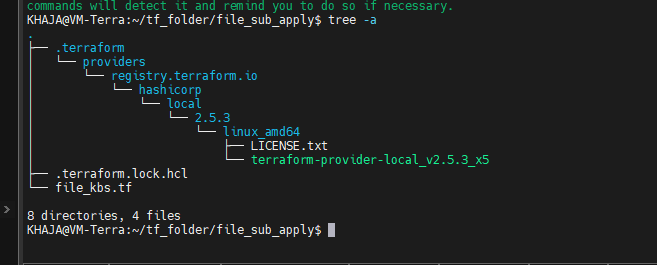


Initialize Terraform using command

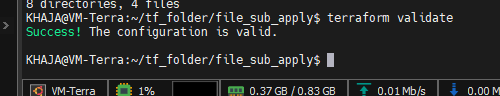
* terraform init



Check the contents using tree -a



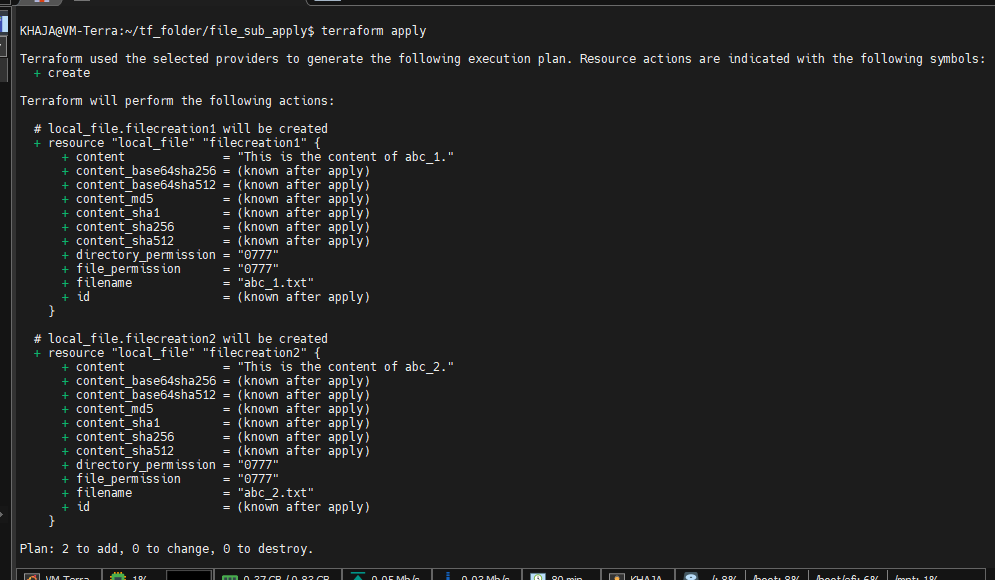
Run the command terraform validate

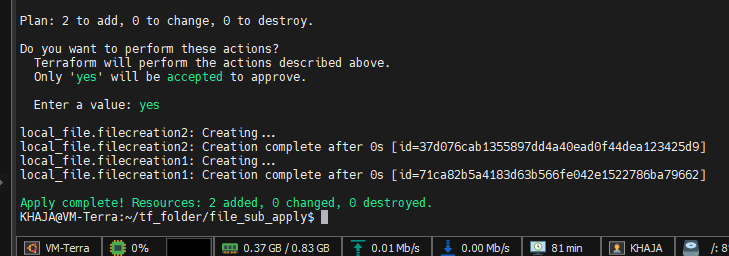


terraform plan

* it will show a preview or dry run.
* It shows like What Terraform wants to create, change, or delete.
* No changes are made to our real environment when running plan.

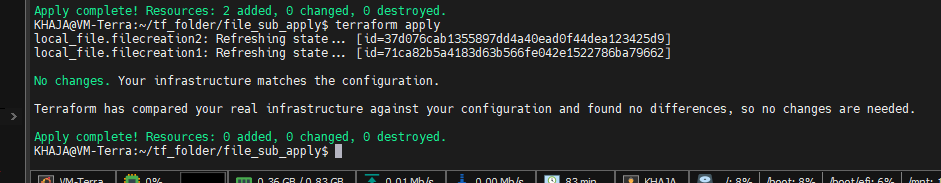
Run the command terraform plan





Subsequent apply

* It will show that No changes are made to our real environment when running plan.



Terraform tracks resource definitions using **resource blocks**, not by file content or filename. So:

* **Removing a block** = Terraform **destroys** that resource
* **Adding a new block** = Terraform **creates** that resource
* Even if the **content is same**, if the **resource name changes**, Terraform treats it as new.

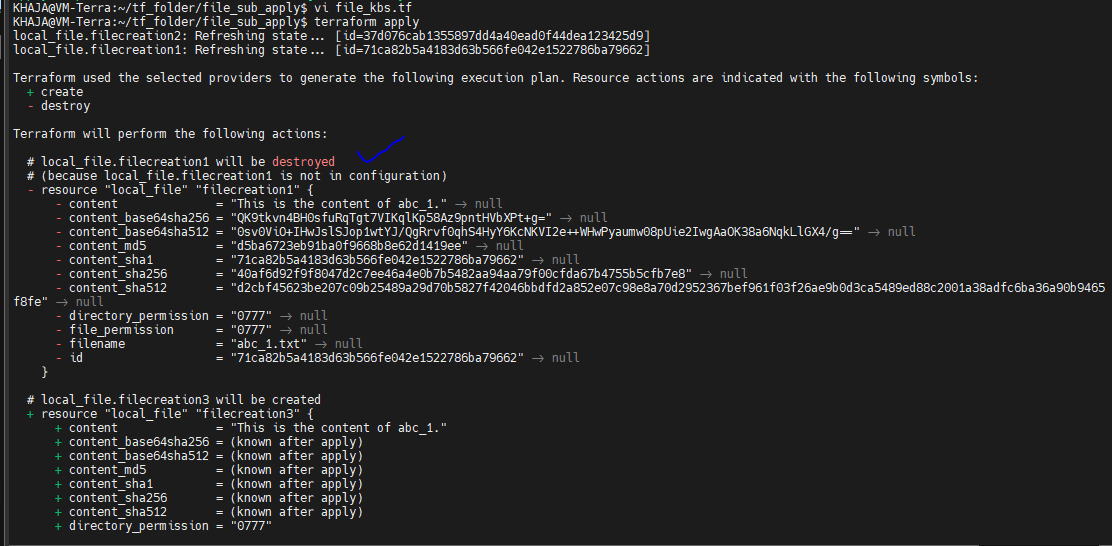
Now, in the updated file\_kbs.tf, you removed filecreation1 and added:

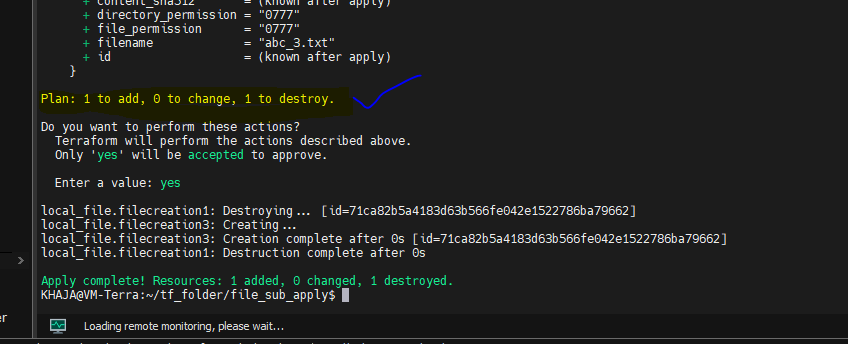
resource "local\_file" "filecreation3" {

filename = "abc\_3.txt"

content = "This is the content of abc\_1."

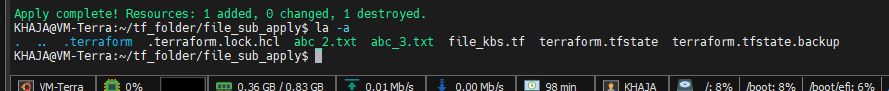
}



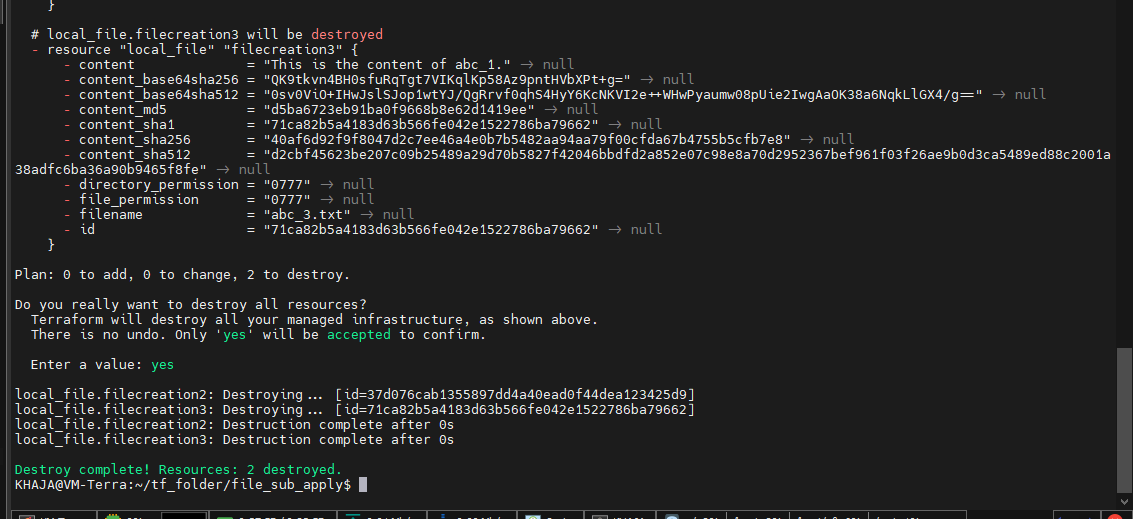
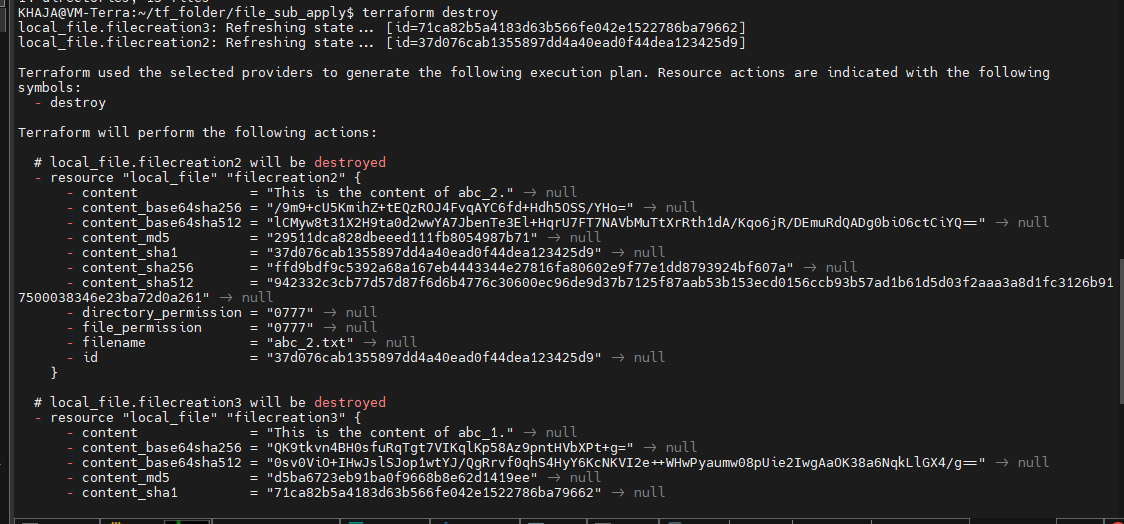


**Output**

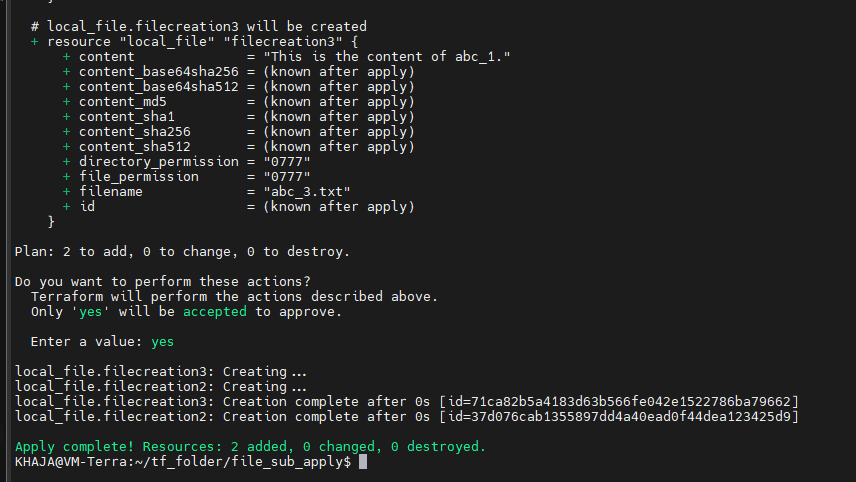
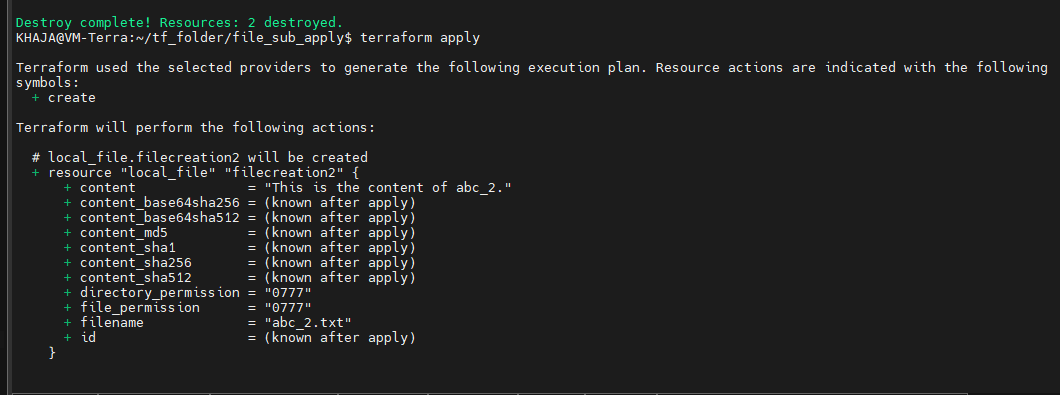
As a result when we change and apply , it will remove the change file and create a new resource



Use terraform detroy to destroy the resources



If we run **terraform apply** again with the same configuration the file will be recreated



1. **explore these providers -> local, random, null**

**Ans: Definition:**

Terraform providers are essentially plugins that allow Terraform to interact with various infrastructure platforms (like cloud providers, SaaS providers, or APIs).

* **Local Providers:**

**Definition:**

Local providers handle tasks within the same environment as Terraform, often for testing or local resource management.

* **Usage:**

They can also be used for other local operations, such as generating random passwords, IDs, or other unique values.

* Generating configuration files (e.g., for apps, scripts, or cloud-init).
* Saving outputs (e.g., certificates, keys) to disk.

**Key Resource: local\_file**

resource "local\_file" "config" {

filename = "app.conf"

content = "key=value" *# Dynamic content via variables/templates*

file\_permission = "0644" *# Linux file permissions (optional)*

}

**Lifecycle Control:**

lifecycle {

ignore\_changes = [content] *# Prevents overwrites if file is modified externally*

}

**Example: Generate a Dynamic Script**

resource "local\_file" "startup\_script" {

filename = "start.sh"

content = <<-EOT

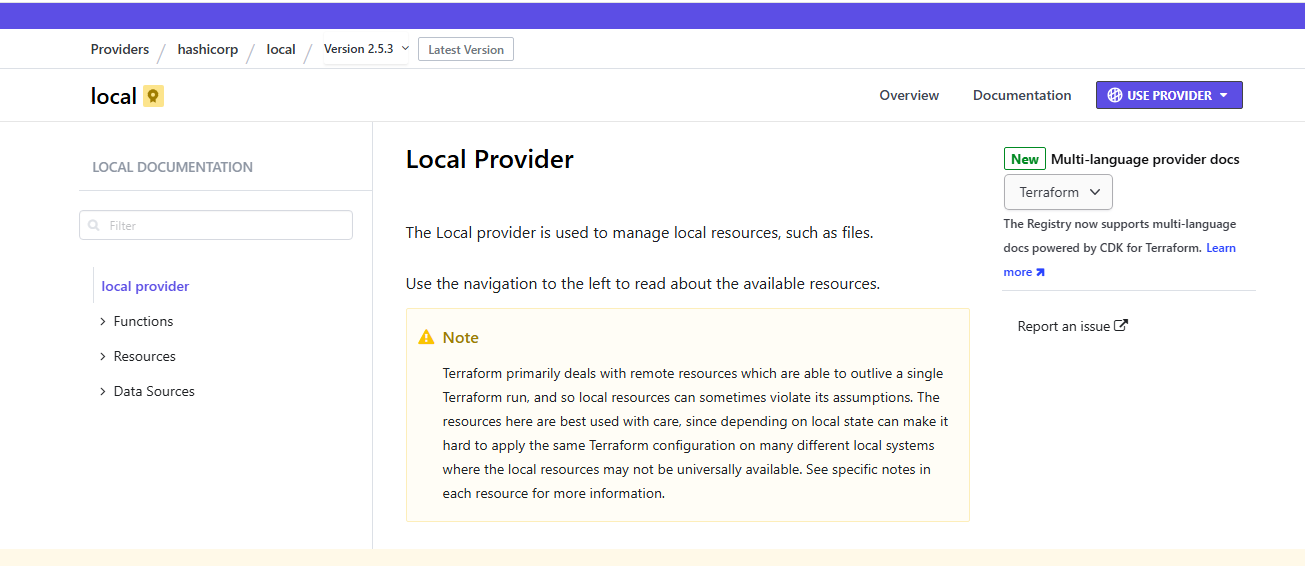
#!/bin/bash

echo "Hello, ${var.username}!"

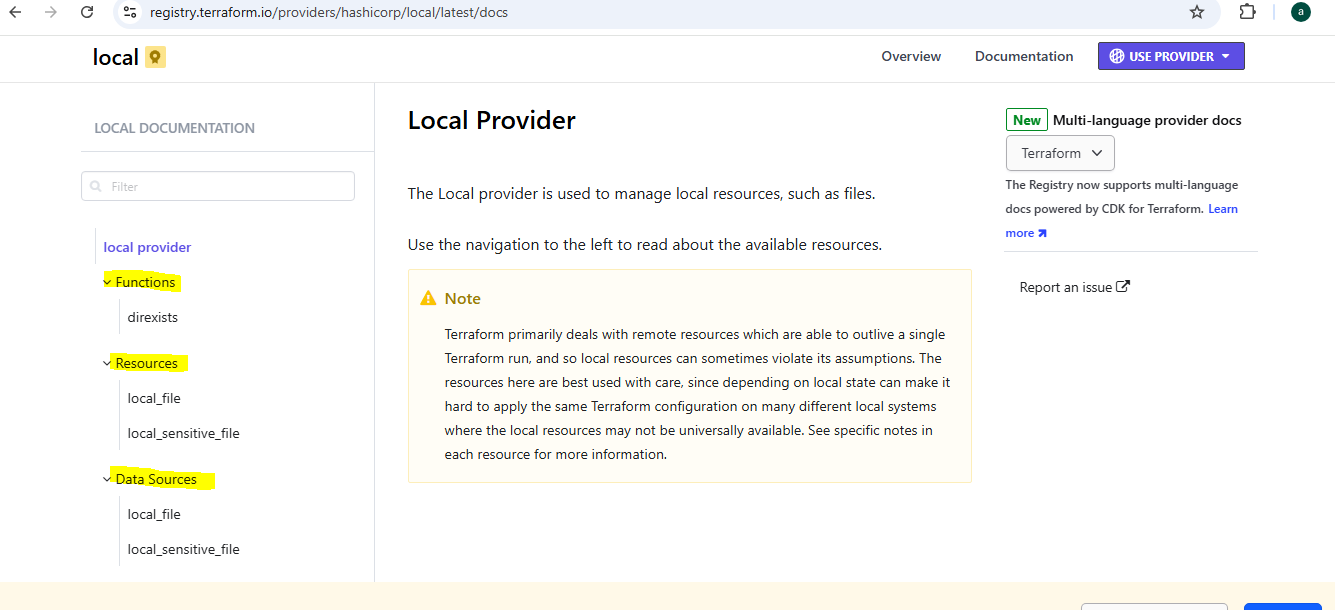
EOT

}

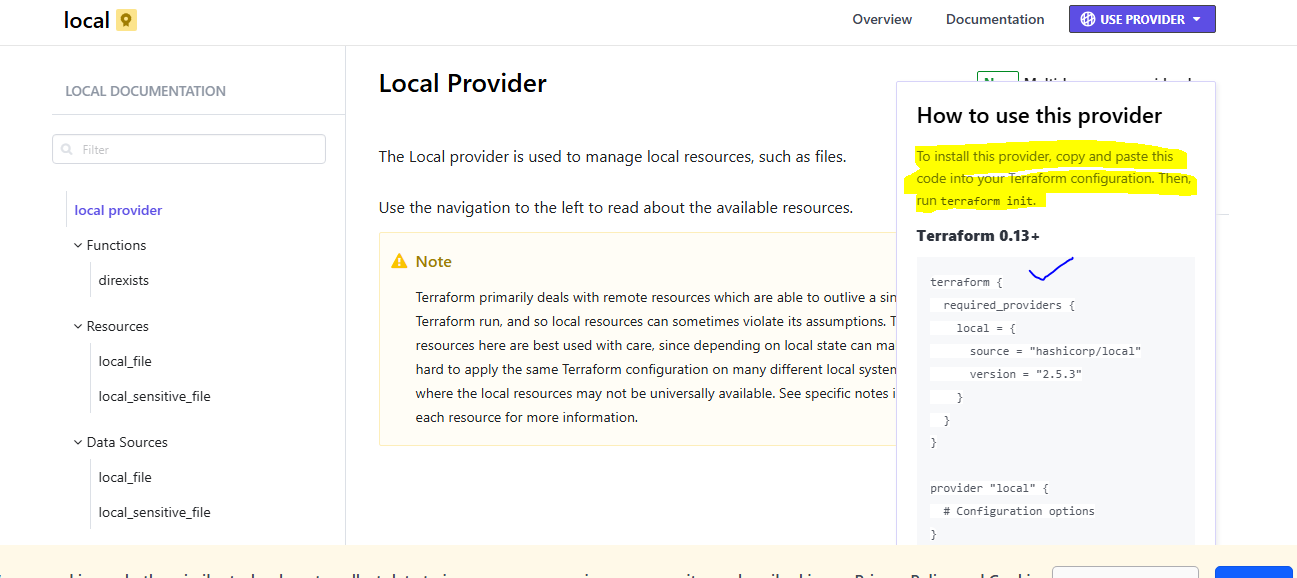
They are the certified trusted providers



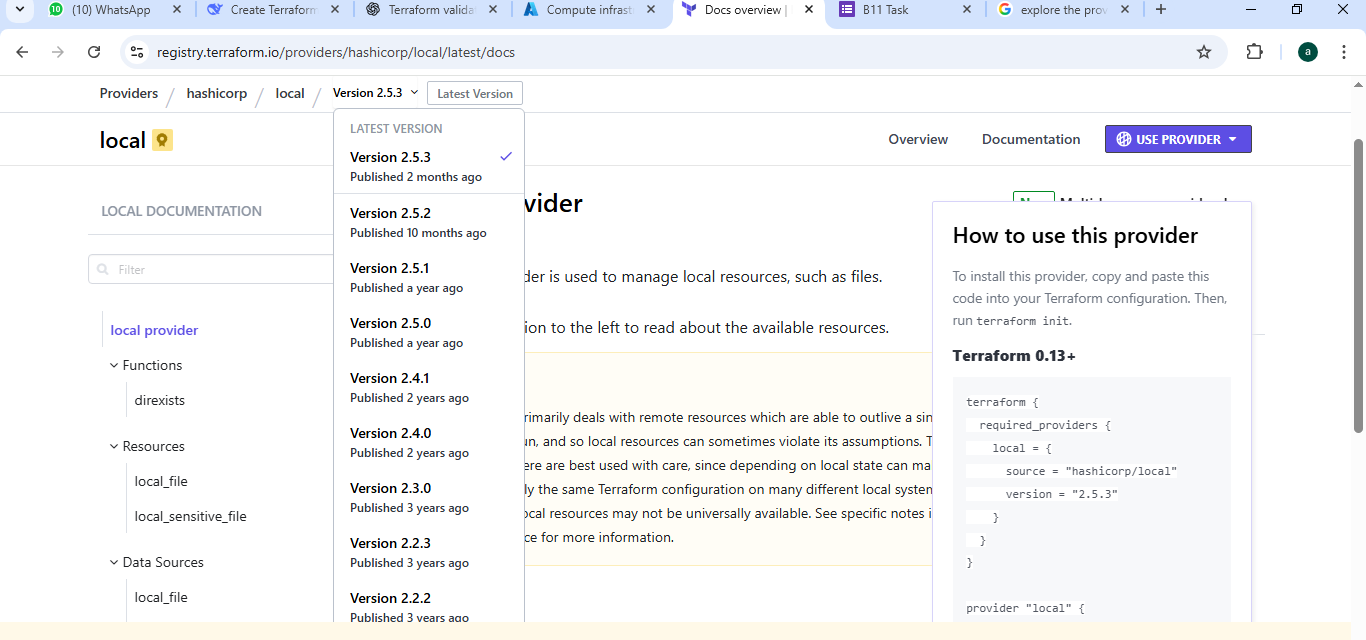
We can check are the services Local Provider let you



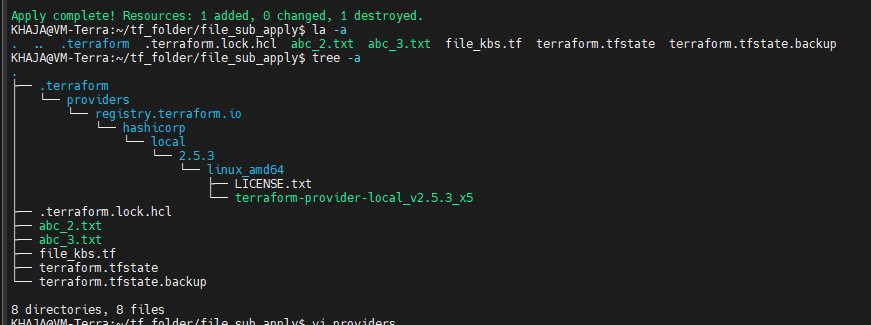
It will also let us know the configuration file with current version

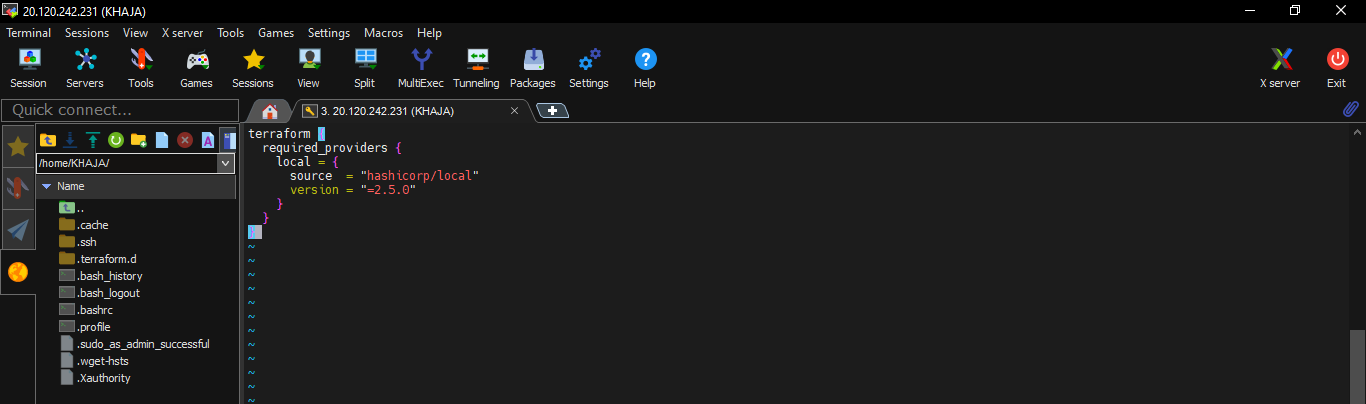


Here we can use the versions which are provided if suppose I want the previous version I can have that version like we have certain



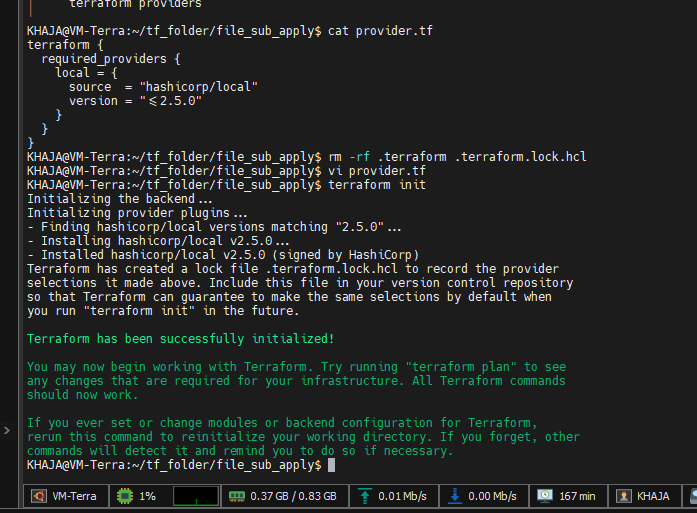
I have the version like version 2.5.3 know I want to change to version 2.5.0





Run the command to Initialize Terraform

* terraform init



Here the version has change form latest version 2.5.3 to previous version 2.5.0 as per the requirement

**2.**random**Provider**

**Purpose**: Generates random values (strings, numbers, pets) for unique resource naming or secrets.  
**Common Use Cases**:

* Creating unique resource names (e.g., S3 buckets, VM hostnames).
* Generating temporary passwords/keys.

**Key Resources**

**a)**random\_string

resource "random\_string" "bucket\_suffix" {

length = 8

upper = false

special = false *# No special chars (e.g., `-`, `\_`)*

}

**Output**: bucket-name-3a7b9c2d

**b)**random\_pet (Human-readable random names)

resource "random\_pet" "server\_name" {

length = 2 *# e.g., "happy-lemur"*

}

**Output**: happy-lemur

**c)**random\_password**(Sensitive)**

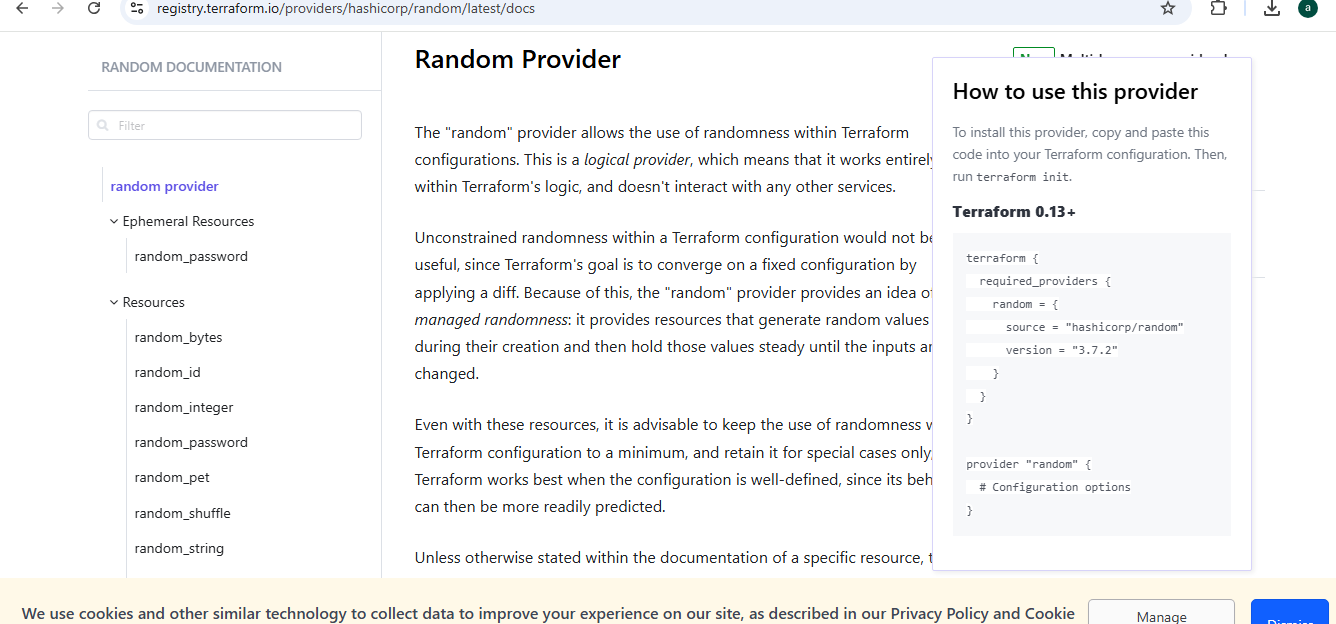
resource "random\_password" "db\_password" {

length = 16

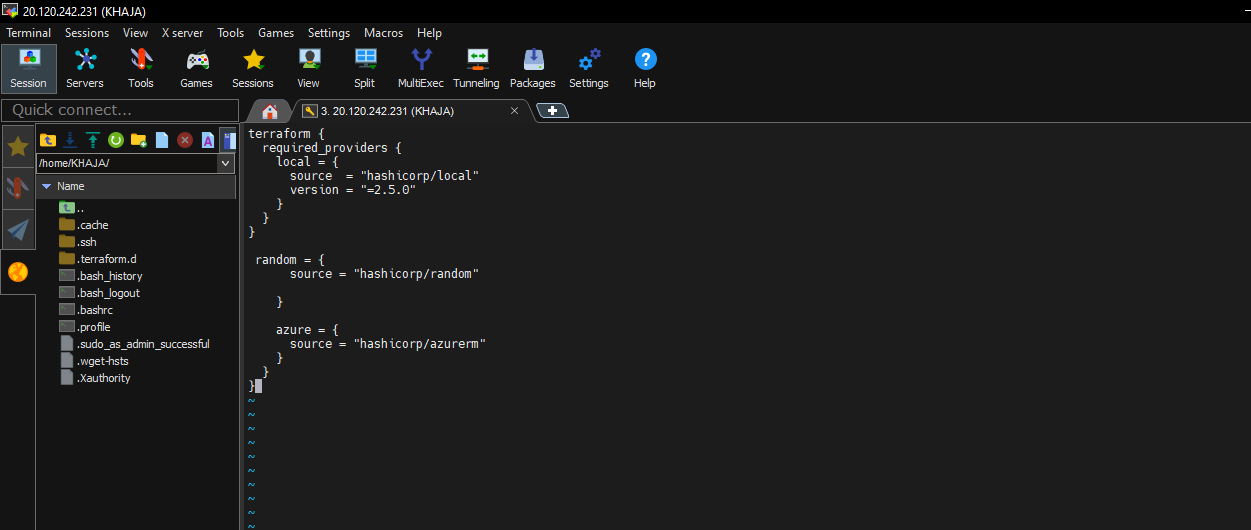
special = true

}

**Note**: Mark outputs as sensitive to hide them in logs.

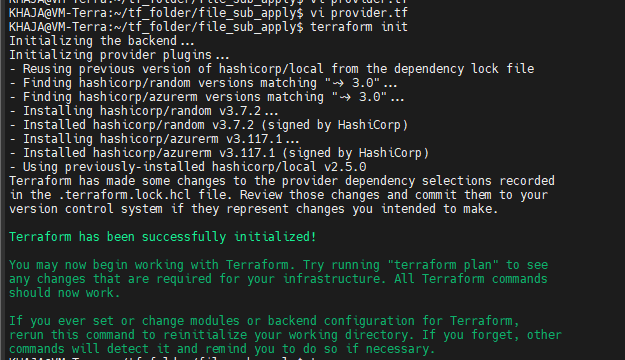


When we use the provider like

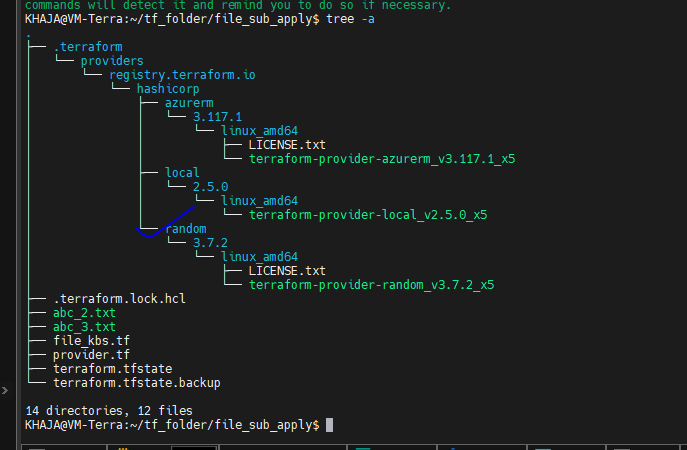


Execute the command init

Terraform init



Check the tree for version



1. Null Provider

Provides constructs that intentionally do nothing – useful in various situations to help orchestrate tricky behavior or work around limitations.

The null provider is a rather-unusual provider that has constructs that intentionally do nothing. This may sound strange, and indeed these constructs do not need to be used in most cases, but they can be useful in various situations to help orchestrate tricky behavior or work around limitations.

The documentation of each feature of this provider, accessible via the navigation, gives examples of situations where these constructs may prove useful.

Usage of the null provider can make a Terraform configuration harder to understand. While it can be useful in certain cases, it should be applied with care and other solutions preferred when available.

**Make a configuration**

terraform {

required\_providers {

null = {

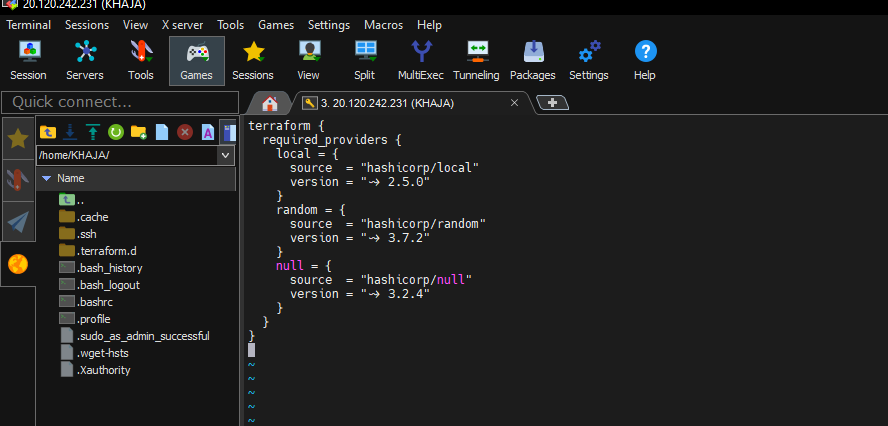
source = "hashicorp/null"

version = "3.2.4"

}

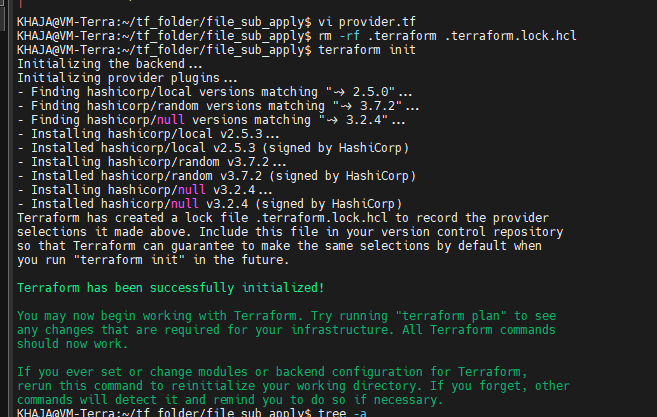
}

}



Execute the command init

Terraform init



Check the contents of directory using tree -a

