Additional Task

**Name:Shaik Khaja Basha**

**Batch : Batch 11**

**Date : 24.07.2025**

**Task : Meta Arguments**

**Meta Arguments**

In Terraform, **Meta-Arguments** are special arguments that control **how resources behave**, rather than defining the resource itself. These are not specific to any provider and can be used with most resources.

**Dependency**

In Terraform, a dependency defines the order in which resources are created or destroyed. Terraform builds a dependency graph internally based on how resources reference each other. This ensures resources are created/destroyed in the correct order.

**Types of Dependencies**

**1. Implicit/direct Dependencies (Automatic)**

* Terraform automatically detects dependencies when you reference attributes from other resources.
* It is use as reference to other resources

**Example:**

I have resources like

resource "local\_file" "f3" {

filename = "123.txt"

content = "test"

}

resource "local\_file" "f2" {

filename = "12325.txt"

content = local\_file.f3.id # here we are using id of first resource

}

resource "local\_file" "f4" {

filename = local\_file.f2.id

content = local\_file.f3.id

}

resource "local\_file" "f1" {

filename = "15342541"

content = "sgfj"

}

* Here we are using direct resource form 1st resource to execute the 2nd resource by using the id of the first resource as shown above
* We will get id for the second resource when the first resource is executed and it’s a chain process which depend on each other

**Creating of resources in res.tf file**

resource "local\_file" "f3" {

filename = "123.txt"

content = "testtdtd"

}

resource "local\_file" "f2" {

filename = "12325.txt"

content = local\_file.f3.id

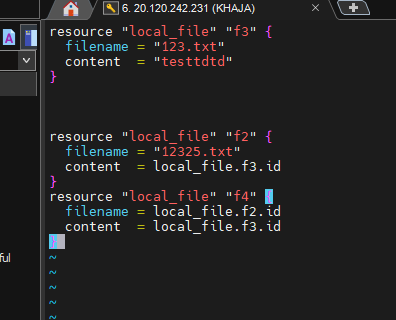
}

resource "local\_file" "f4" {

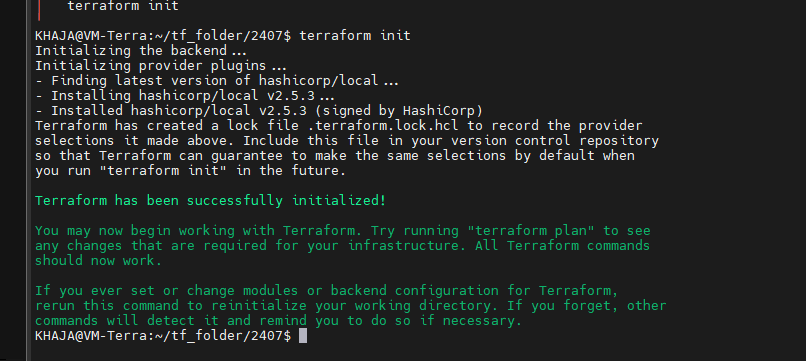
filename = local\_file.f2.id

content = local\_file.f3.id

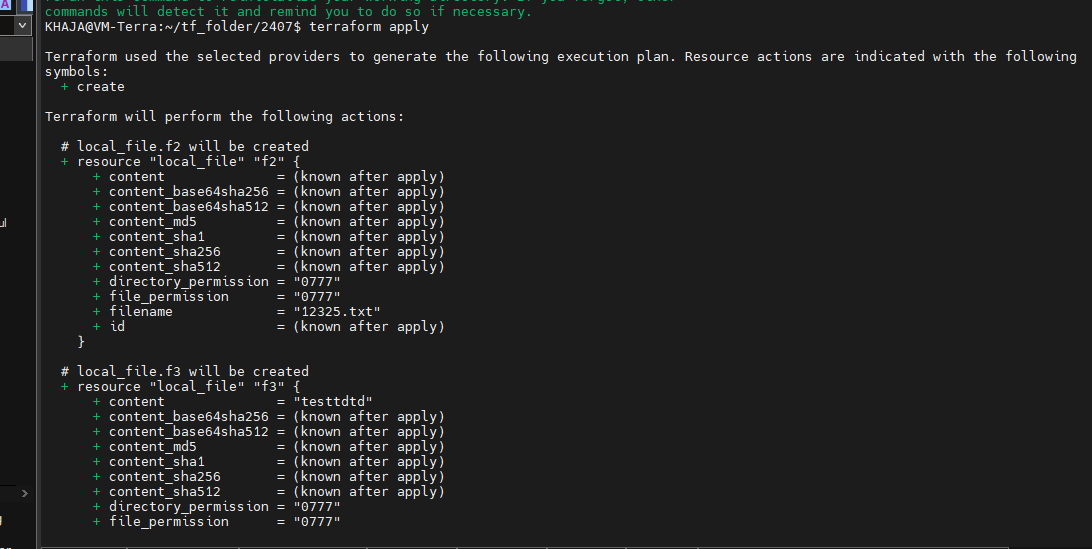
}

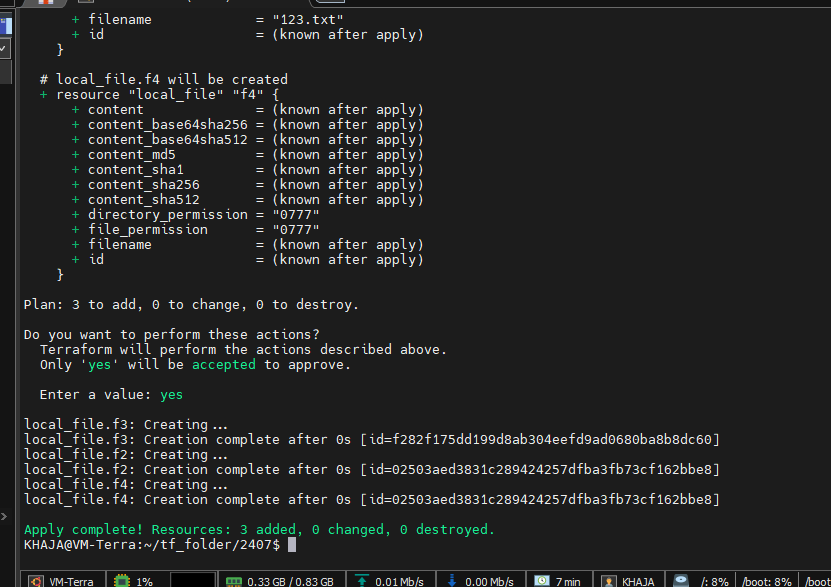


Execute the command terraform init



Execute the terraform apply to make changes

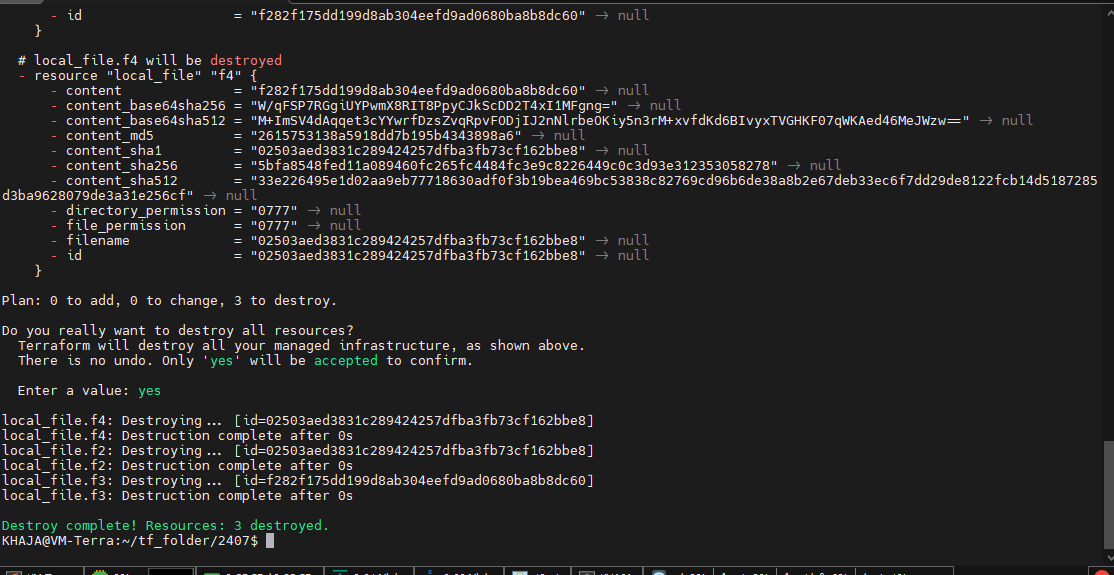




Form the output we can see after creating of file.f3 its executing file.f2 which depend on f3

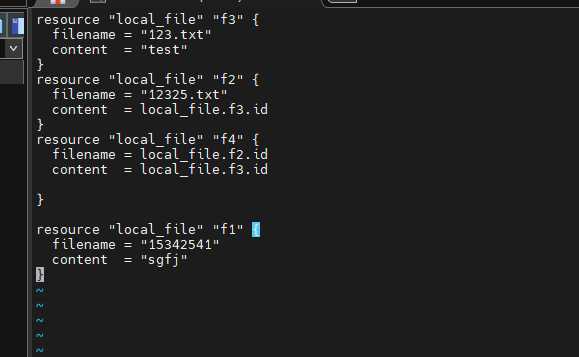
And like that for f4

Lets destroy it

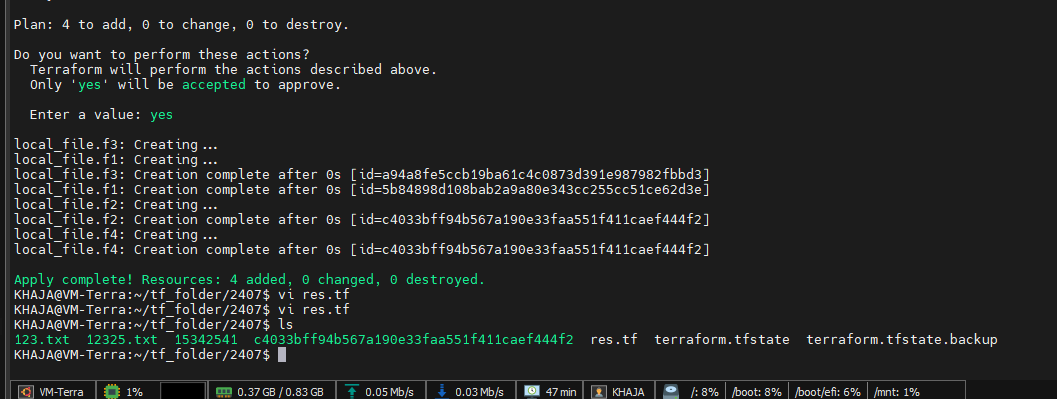


Which destroying also it do the same first its deleting the file f4 then the remaining

Dependency configuration



Here f1 and f3 are independent and other are dependency resource



**2. Explicit/indirect Dependency *(Using depends\_on)***

Use depends\_on when Terraform **can’t detect the dependency automatically**, such as when resources are connected only via provisioners or indirect outputs.

* we will be mentioning the resources as dependencies

example:

lets take resources like

resource "local\_file" "f3" {

filename = "123.txt"

content = "test"

}

resource "local\_file" "f1" {

filename = "15342541"

content = "sgfj"

}

resource "local\_file" "f5" {

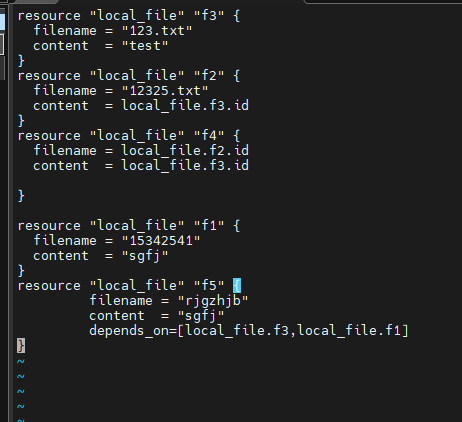
filename = "rjgzhjb"

content = "sgfj"

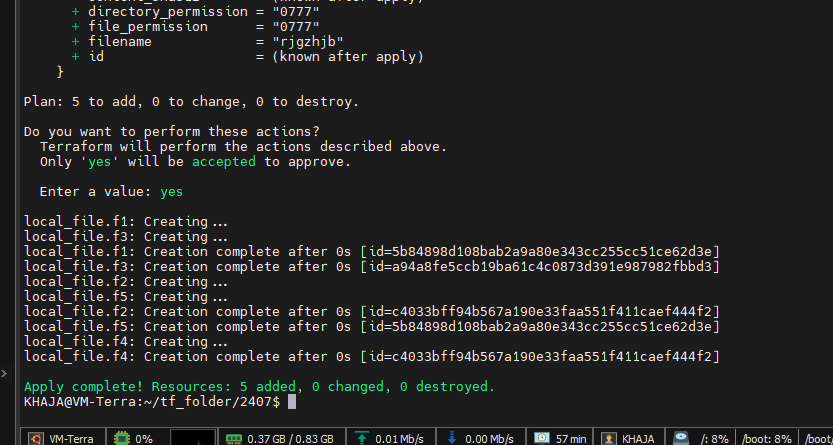
depends\_on=[local\_file.f3,local\_file.f1]

}

Here f5 will execute when f3 and f1 are executed until it will not executed



Terraform apply



Here we can see once f1 and f3 are created then f5 is created bcz its depend on both the files

**Lifecycle:**

* Used when changing one resource should force replacement of another, even if it’s not directly referenced.
* destroy and then create

**create before destroy:** Zero-downtime replacements:

**use case**

When enabled, Terraform first creates the new resource before destroying the old one — which helps avoid downtime for services like:

* VMs
* Load balancers
* App gateways

Useful when a field change would normally force resource replacement.

**Syntax:**

lifecycle {

create\_before\_destroy = true

}

**Prevent\_destroy:** A lifecycle rule that blocks resource deletion in Terraform, even during terraform destroy or resource replacement.

* creates a lock on the resource and doesn’t allow for modification

If a resource has prevent\_destroy = true:

* Running terraform destroy will **fail** if that resource is in the config
* Useful for critical resources like databases or production VMs

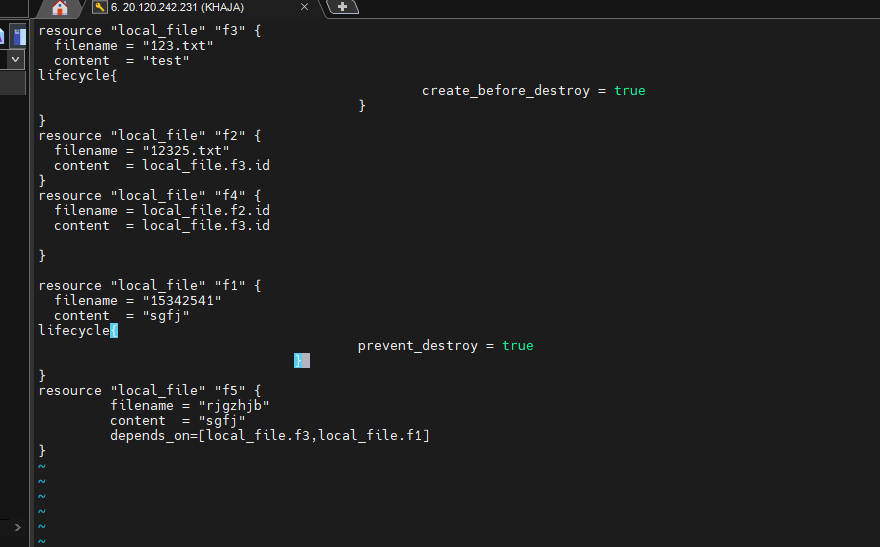
**Syntax:**

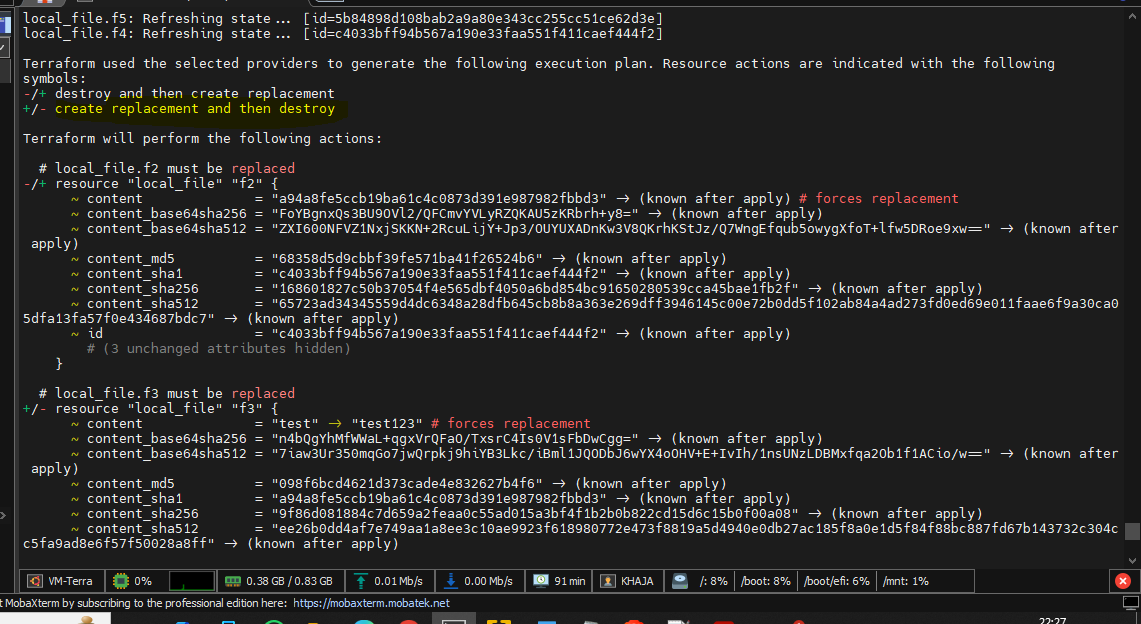
lifecycle {

prevent\_destroy = true

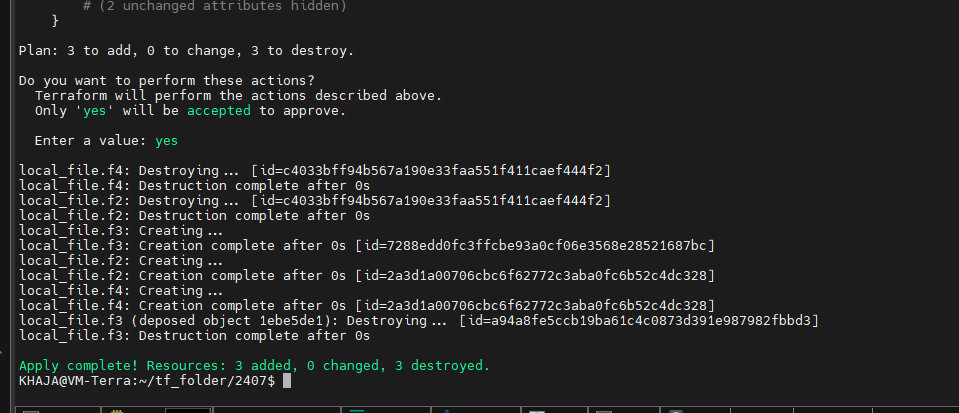
}

**Lets use in present resources**

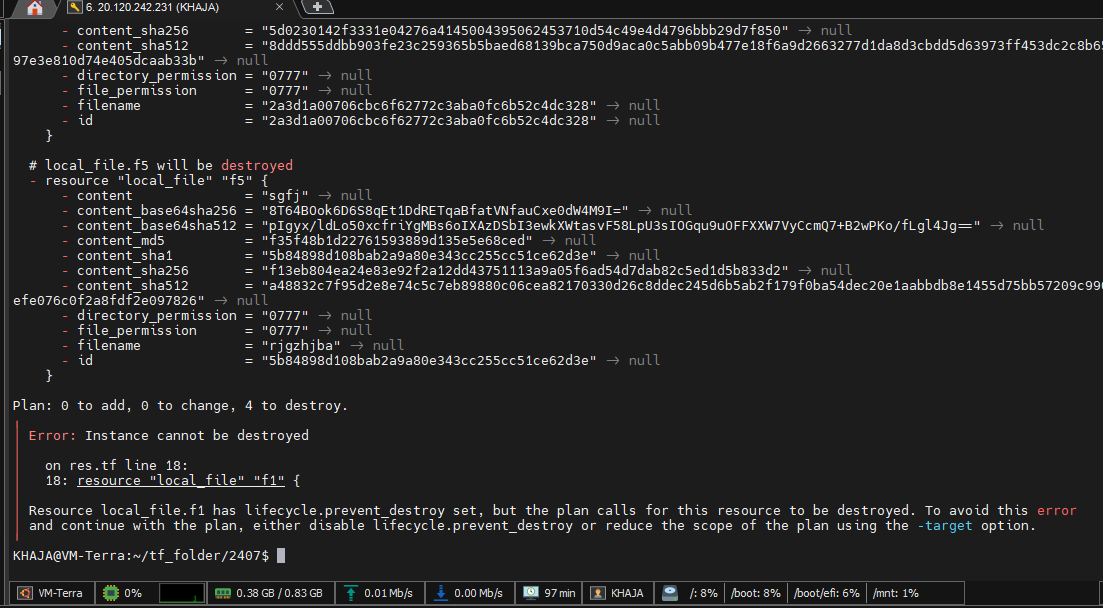




**Here its showing create replacement before destroying**

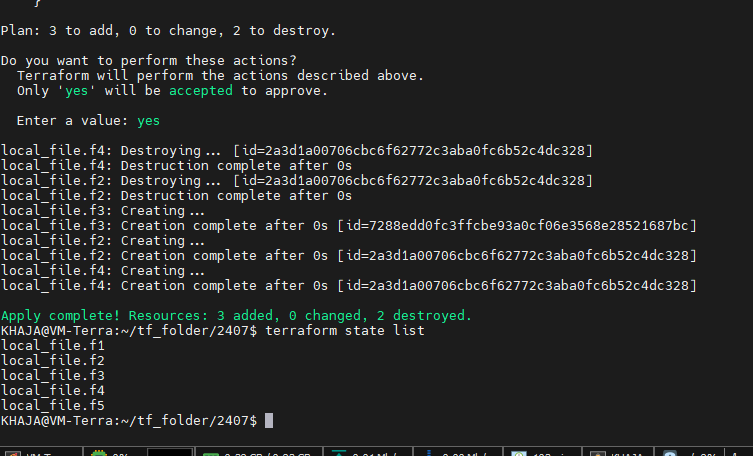


**Execute the command terraform destroy**



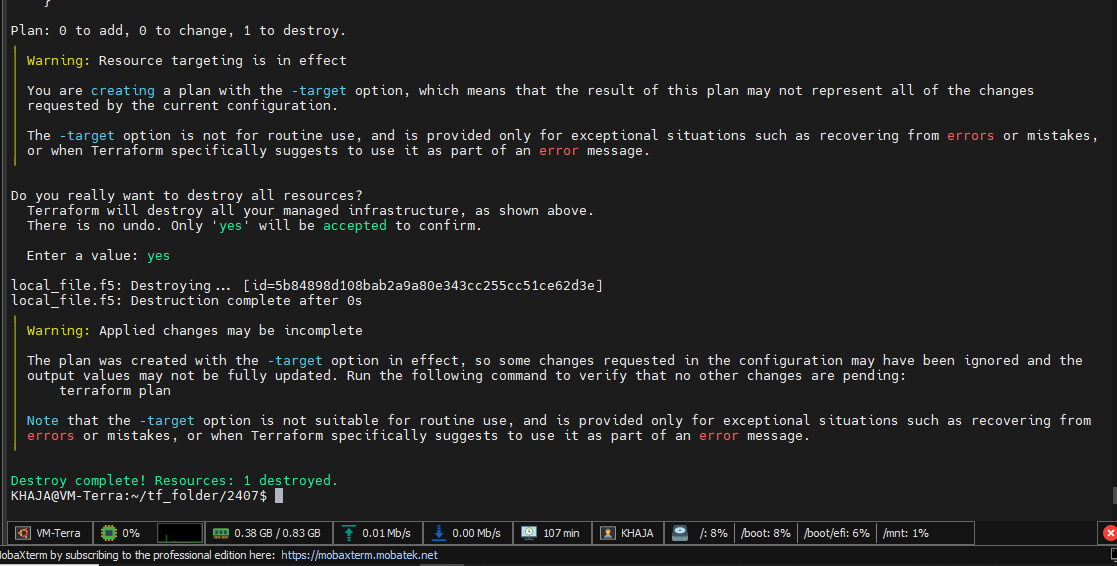
Its through an error because its have a Lifecyle prevent\_before\_destroy so it will be not deleted , until we delete manually

* Even it wont allow us to modify the content

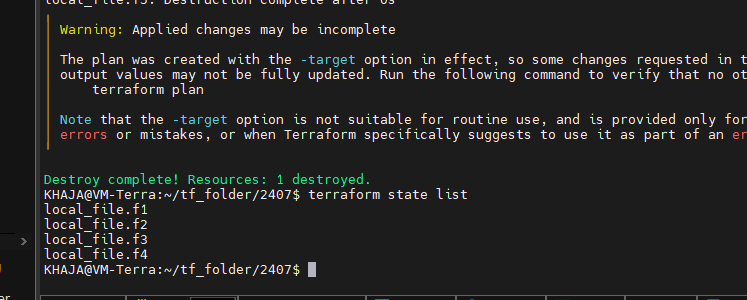


Here I had multiple resource files in state file

Suppose I want to delete specific file I can deleted using **target**

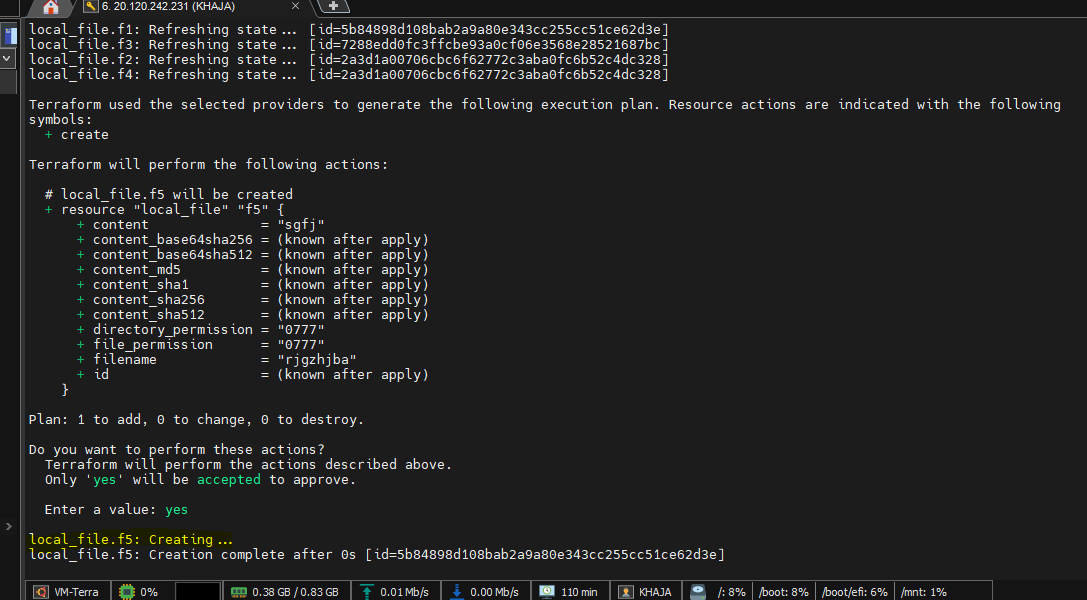


Here we can see explicitly we destroy the define targe file



When we do terraform apply

It will get effected the target delete file



**Looping:**

1. **count**- Basic Index-Based Looping

Creates multiple resource instances using a counter.

*resource "local\_file" "f3" {*

*count = length(var.filename)*

*filename = var.filename[count.index]*

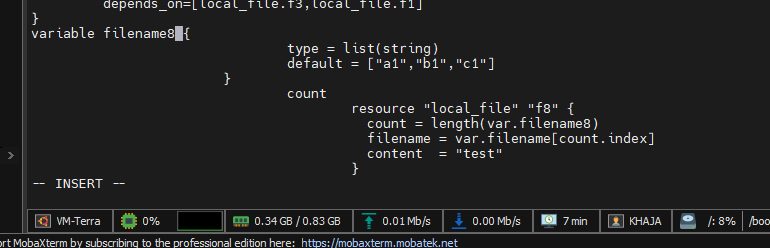
*content = "test"*

*}*

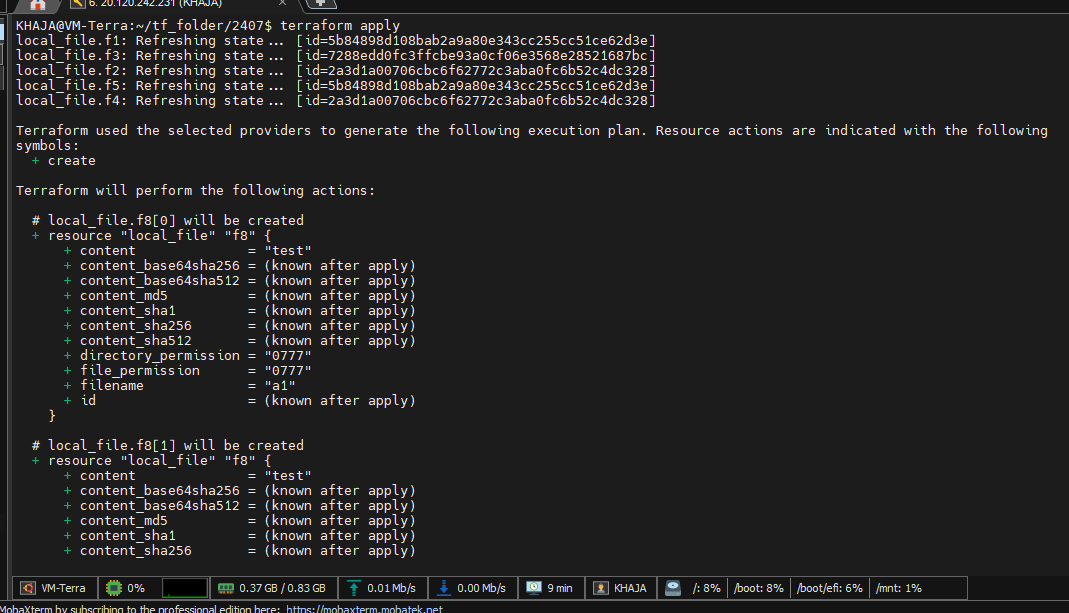
**When to Use:**

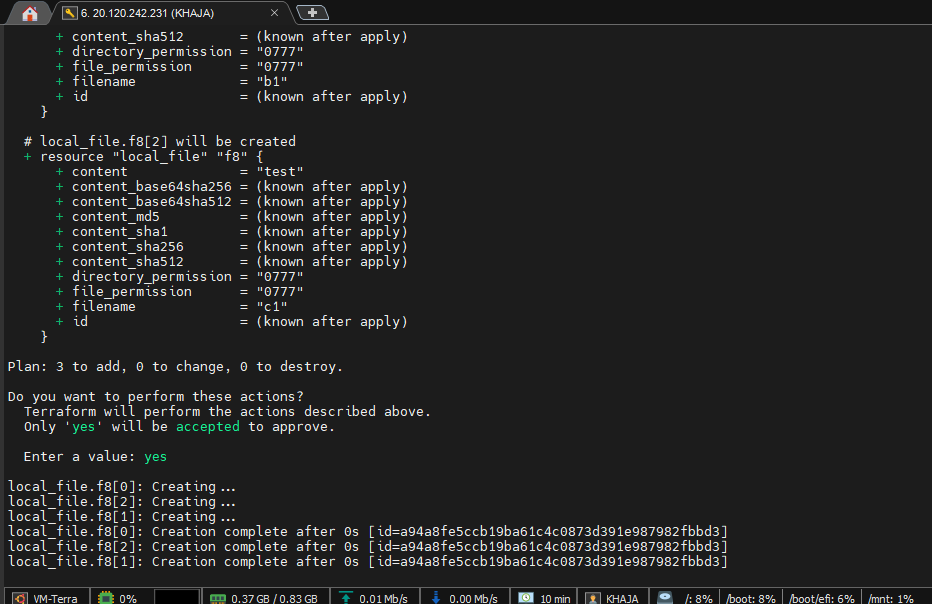
* Simple numeric iteration
* Identical resources with numeric suffixes

Let create a resource and variable practically

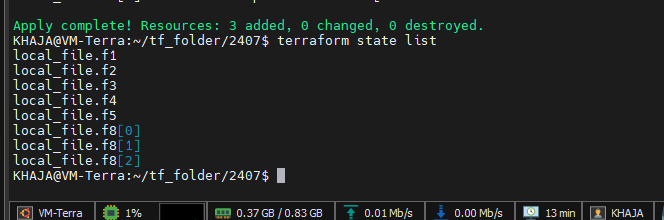


Execute the command terraform apply





It will be creating along with index



2. **for\_each -** Map/Set-Based Looping

Creates resources from a map or set, preserving unique identifiers.

* Its an element base

***for\_each***

*resource "local\_file" "f9" {*

*filename = each.value*

*for\_each = toset(var.filename1)*

*content = "test"*

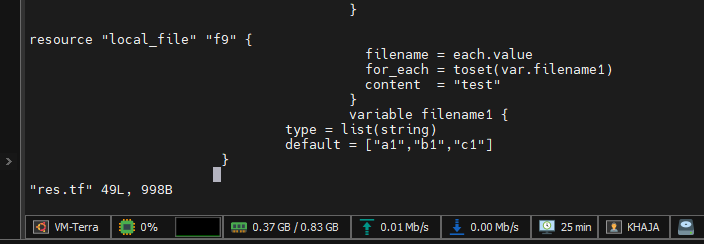
*}*

*variable filename1 {*

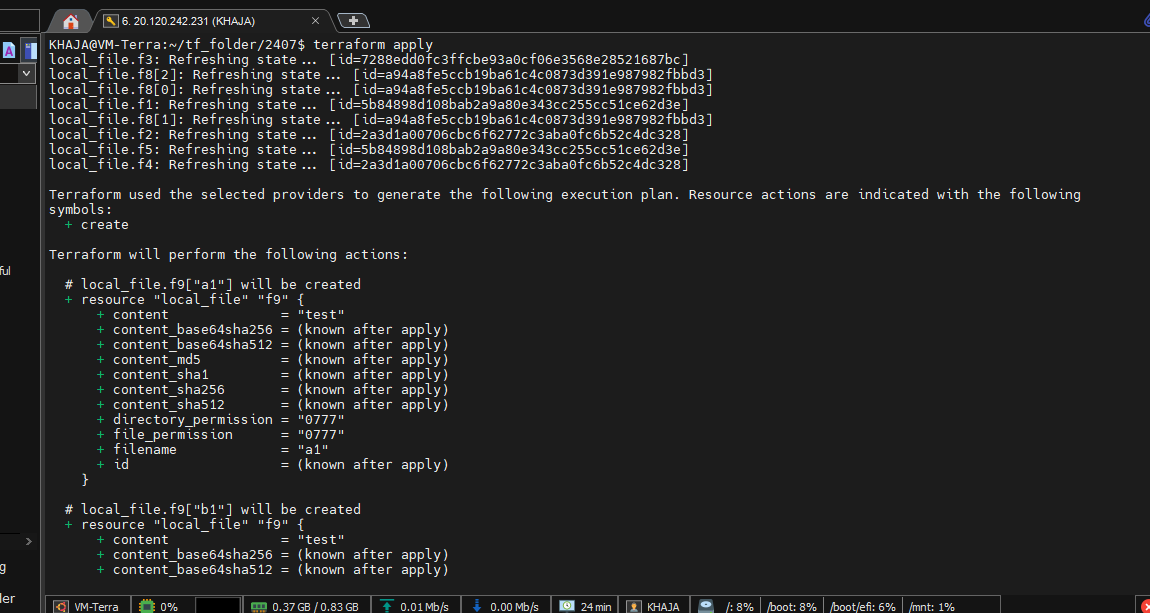
*type = list(string)*

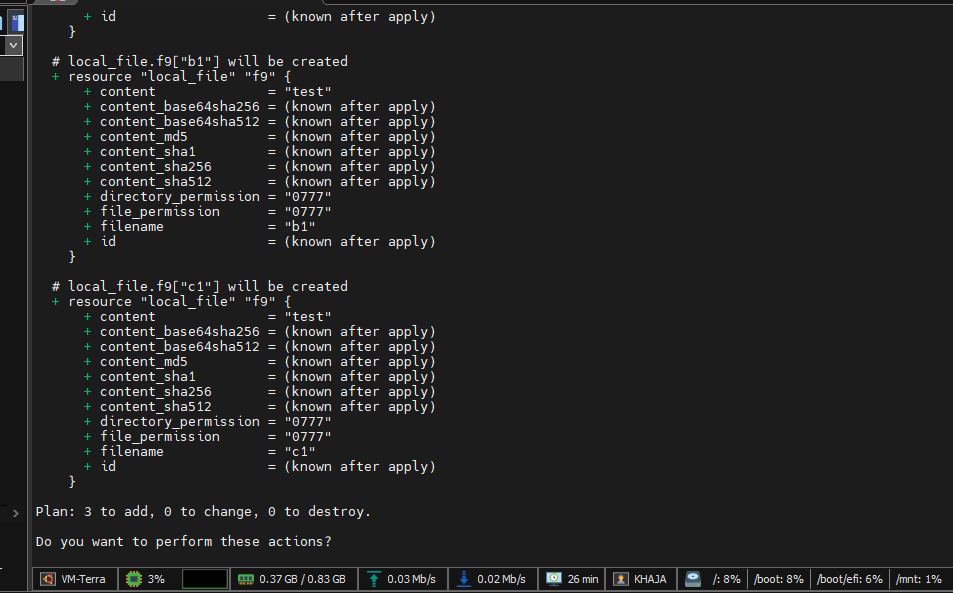
*default = ["a1","b1","c1"]*

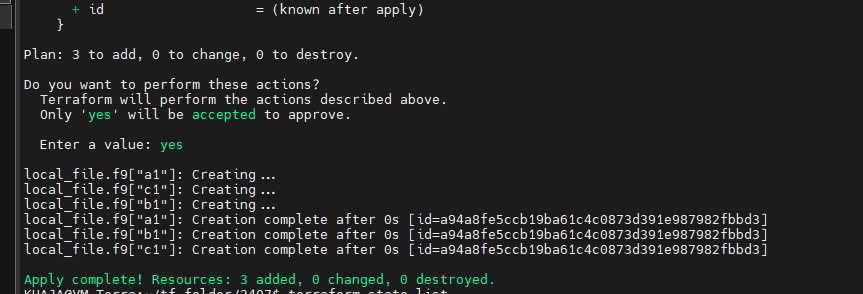
*}*



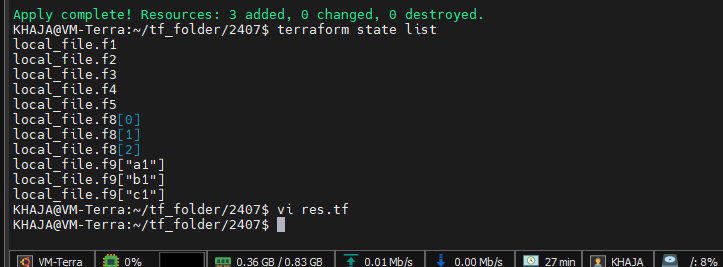
Execute the command terraform apply





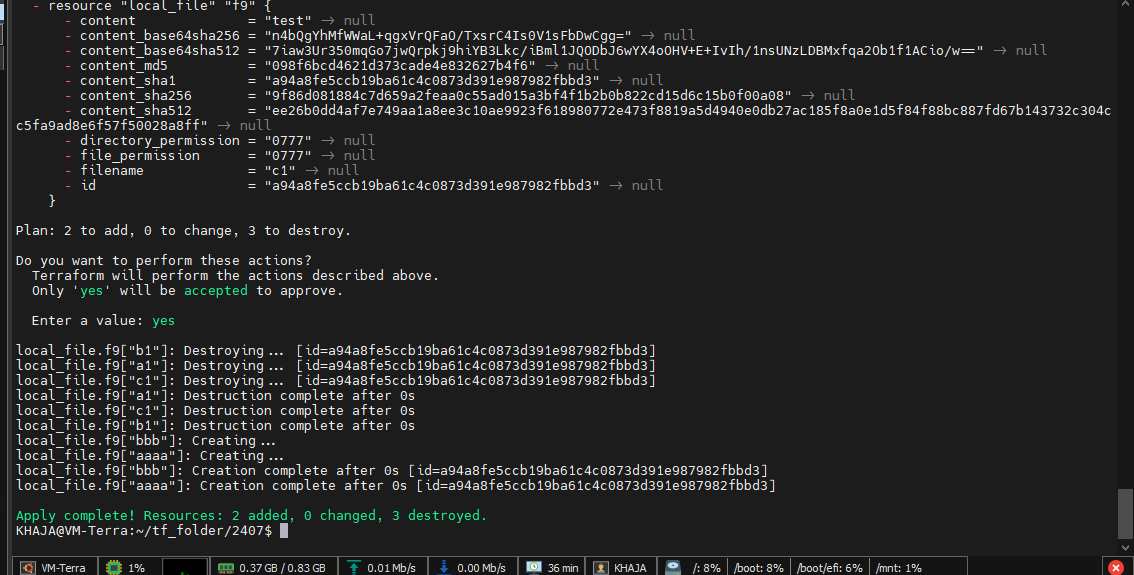


The output will be based on element not the index



When we inject the value from outsite like

Terraform apply -var='filename=["aaaa","bbb"]'



**3.for expressions (inside locals or variables)**

variable "cities" {

default = ["delhi", "mumbai", "hyd"]

}

output "upper\_cities" {

value = [for city in var.cities : upper(city)]

}

Result: ["DELHI", "MUMBAI", "HYD"]