Terraform Task

**Name:Shaik Khaja Basha**

**Batch : Batch 11**

**Date : 22.07.2025**

**Task : datatypes**

1. **Defines what kind of data a variable can hold?**

**Ans: Terraform Data Types**

Data types define what kind of data a variable can hold in Terraform.

**Primitive Types**

**1. String**

* Alpha-numeric values that can include symbols
* Enclosed in double quotes " "
* \n for multi-line strings

variable "example" {

type = string

default = "test123"

*# Multi-line example*

default = "test123\nshgf"

}

**2. Number**

* Numeric values (both integer and decimal)
* No separate float/integer distinction

variable "example" {

type = number

default = 125 *# Integer*

default = 10.5 *# Decimal*

}

**3. Boolean**

* Conditional data type
* Only true or false values

variable "example" {

type = bool

default = true

}

**4. Any (Default)**

* Accepts any data type
* Type is inferred from the default value

variable "example" {

*# type not specified defaults to 'any'*

default = 10 *# becomes number*

default = "test" *# becomes string*

default = false *# becomes bool*

}

**Complex/Composite Types**

**1. List**

* Ordered collection of similar values
* Can be nested (lists of lists)
* Accessed by index (0-based)

*# Unconstrained list*

variable "untyped\_list" {

type = list

default = ["test", 123, true, "test", 123]

}

*# Typed list*

variable "number\_list" {

type = list(number)

default = [1,2,3,4,5,2,4,7,1,2]

}

*# Nested list*

variable "nested\_list" {

type = list(list(number))

default = [[1,2],[3,4],[5,6]]

}

*# Access: var.number\_list[2] → 3*

*# Error if index out of bounds*

**2. Set (Partially Deprecated)**

* Collection of unique values
* Automatically removes duplicates
* Order not guaranteed

variable "example\_set" {

type = set(number)

default = [1,2,3,4,5,2,4,7,1,2] *# Stored as [1,2,3,4,5,7]*

}

**3. Map**

* Key-value pairs
* Keys are always strings
* Values can be constrained

*# Unconstrained map*

variable "untyped\_map" {

type = map

default = {

name = "adi"

id = 123

isactive = true

}

}

*# Typed map*

variable "string\_map" {

type = map(string)

default = {

name = "adi"

id = "123"

isactive = "yes"

}

}

*# Access: var.string\_map["name"] → "adi"*

*# Error if key doesn't exist*

**4. Tuple**

* Fixed-length sequence with specific types
* Position determines type

variable "network\_config" {

type = tuple([string, number, bool])

default = ["192.168.1.0", 24, true]

}

**5. Object**

* User-defined structured type
* Each attribute has specific type
* Combines features of map and struct

variable "user" {

type = object({

name = string

age = number

active = bool

contacts = optional(list(string))

})

default = {

name = "John"

age = 30

active = true

}

}

Type Injection via terraform.tfvars

*# For lists*

number\_list = [1,2,5,6.7]

*# For maps*

user\_map = {

name = "test"

dob = 123

}

**# Primitive Data Type Demo Using local\_file Variables**

# 1. String Variable

variable "filename1" {

default = "abc1.txt"

type = string

}

# 2. String Variable (same as above, but explicitly typed)

variable "filename2" {

default = "abc2.txt"

type = string

}

# 3. Bool Variable – it’s a Boolean type

variable "filename3" {

default = true

type = bool

}

# 4. Number Variable

variable "filename4" {

default = 15

type = number

}

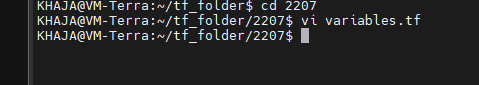
# Content can be of any type

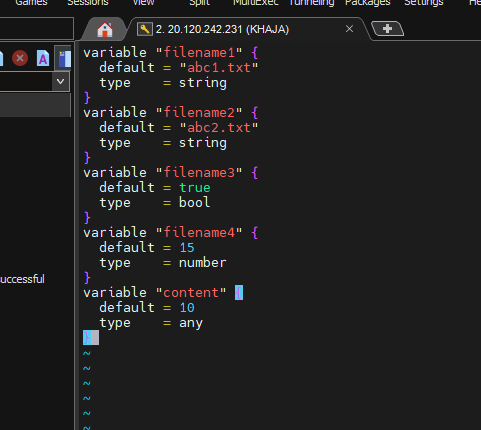
variable "content" {

default = 10

type = any

}





**Resources are**

resource "local\_file" "f1" {

filename = var.filename1

content = var.content

}

resource "local\_file" "f2" {

filename = var.filename2

content = var.content

}

resource "local\_file" "f3" {

filename = var.filename3

content = var.content

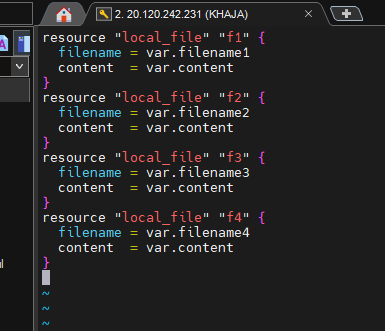
}

resource "local\_file" "f4" {

filename = var.filename4

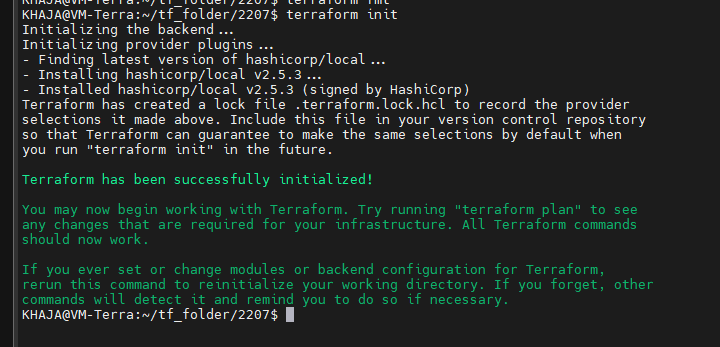
content = var.content

}

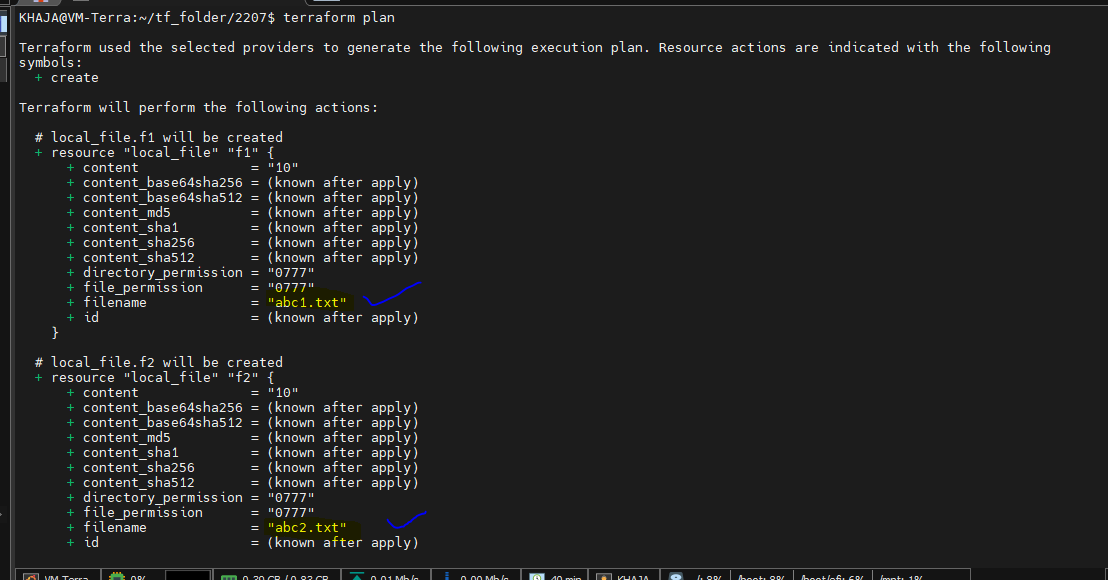


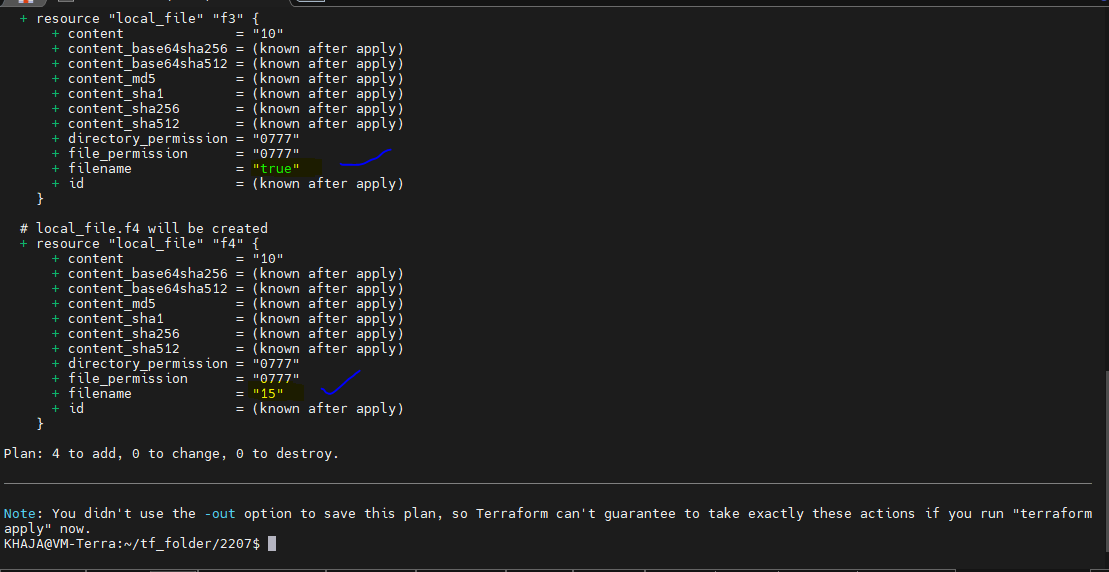
| **Variable Name** | **Type** | **Default Value** | **Usage (as filename)** |
| --- | --- | --- | --- |
| filename1 | string | "abc1.txt" | Directly used |
| filename2 | string | "abc2.txt" | Directly used |
| filename3 | bool | true | Converted to "true.txt" |
| filename4 | number | 15 | Converted to "15.txt" |
| content | any | 10 | Can be any type, here a number |

Execute the command terraform init

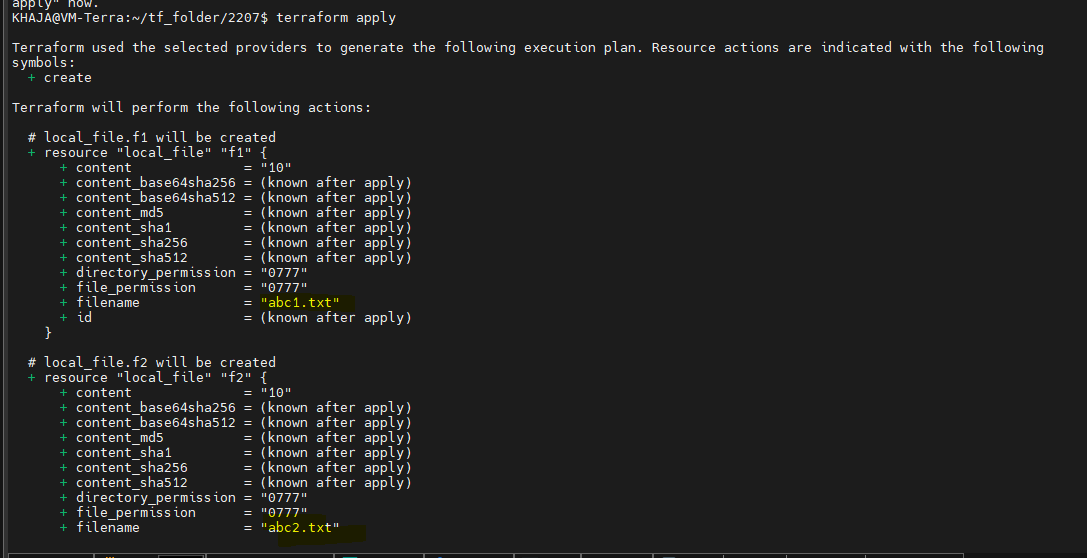


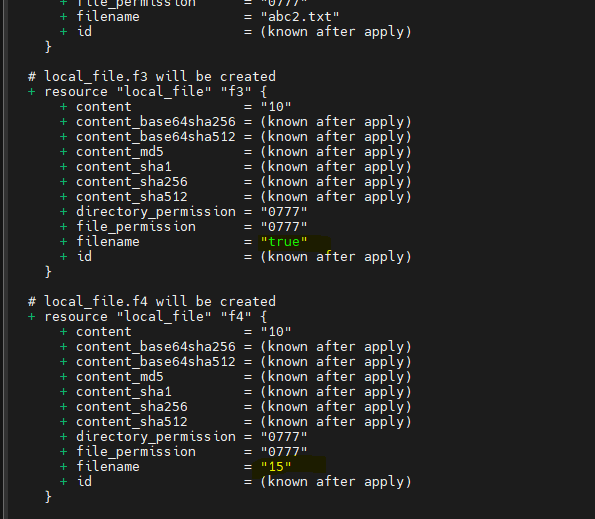
Execute terraform plan

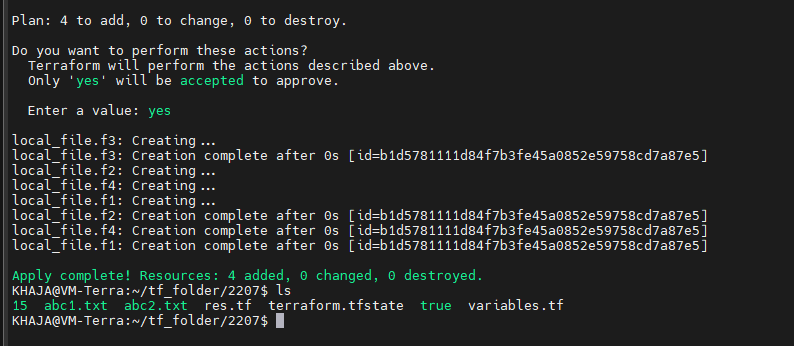




Execute terraform apply







Here we can see four resources which we have given is created

**Terraform Complex / Composite / Advanced Data Types:**

**1. List**

An ordered sequence of values of the same or different type (depending on how you define it).

Syntax & Examples:

* Mixed-type list (type = list):

variable "mixed\_list" {

type = list(any) # default for 'list' without constraint

default = ["test", 123, true, "test", 123]

}

* Homogeneous number list (type = list(number)):

variable "num\_list" {

type = list(number)

default = [1, 2, 3, 4, 5, 2, 4, 7, 1, 2]

}

* Nested list (list of lists):

variable "nested\_list" {

type = list(list(number))

default = [[1, 2], [3, 4], [5, 6]]

}

**Access Elements:**

var.varname[index]

var.num\_list[2] # returns 3

**Error Scenario:**

default = [1, 2]

var.num\_list[2] # Error: index out of range

**tfvars Injection:**

varname= [1, 2, 5, 6.7]

num\_list = [1, 2, 5, 6.7]

**2. Set**

An unordered collection of unique values of the same type. Duplicate values are automatically removed.

Syntax & Example:

variable "unique\_set" {

type = set(number)

default = [1, 2, 3, 4, 5, 2, 4, 7, 1, 2]

}

# Interpreted as: [1, 2, 3, 4, 5, 7]

Notes:

* Indexing like var.unique\_set[0] may not work reliably since set is unordered.
* Treat set as similar to a de-duped list, but avoid accessing via index.

**3. Map**

A key-value pair data structure with keys as strings.

Basic map (type = map):

variable "user\_info" {

type = map(any)

default = {

name = "adi"

id = 123

isactive = true

}

}

**Typed map (type = map(string)):**

variable "user\_string\_map" {

type = map(string)

default = {

name = "adi"

id = "123"

isactive = "yes"

}

}

**Number map:**

variable "numeric\_map" {

type = map(number)

default = {

id = 12345

phone = 43154431

}

}

**Map of list:**

variable "map\_list" {

type = map(list(string))

default = {

devs = ["ram", "raj"]

admins = ["khaja", "admin"]

}

}

**Access:**

var.user\_info["id"]

var.user\_info.id

**Error Scenario:**

var.user\_info.phoneno # Key doesn't exist → error

**tfvars Injection:**

**example :** varname = { name = "test", dob = 123 }

user\_info = {

name = "test"

dob = 123

}

**4. Tuple**

A fixed-length, ordered list with elements of different types. Each element has a specific position and type.

**Syntax:**

variable "my\_tuple" {

type = tuple([string, number, bool])

default = ["hello", 10, true]

}

**Access:**

var.my\_tuple[0] # "hello"

var.my\_tuple[2] # true

Error:

* If you pass wrong type or wrong length → validation error.
* Example:

default = ["test", "wrong", false] # Second element must be number

**5. Object**

A custom user-defined structure with fixed attributes and types — think of it as a strict version of a map.

Syntax:

variable "employee" {

type = object({

name = string

id = number

isactive = bool

})

default = {

name = "Khaja"

id = 101

isactive = true

}

}

**Access:**

var.employee.name # "Khaja"

var.employee.isactive # true

**Error Scenario:**

var.employee.email # Attribute not defined → error

**creating of resources :**

**# creating resource using set type**

resource "local\_file" "f5" {

filename = var.filename5[0] # (here its using via index)

content = var.content

}

resource "local\_file" "f6" {

filename = var.filename6[2]

content = var.content

}

**# creating resource using map type**

resource "local\_file" "f7" {

filename = var.filename7.name # (here its using via key name)

content = var.content

}

resource "local\_file" "f8" {

filename = var.filename8.id

content = var.content

}

**For Variable Declaration:**

**# list**

variable "filename5" {

type = list

default = ["test", 123, true, "test", 123]

}

variable "filename6" {

type = list(number)

default = [1,2,3,4,5, 2, 4, 7,1,2]

}

**# For Map**

variable "filename7" {

type = map(string)

default = {

name="adi"

id ="123"

isactive = "yes"

}

}

variable "filename8" {

type = map(number)

default = {

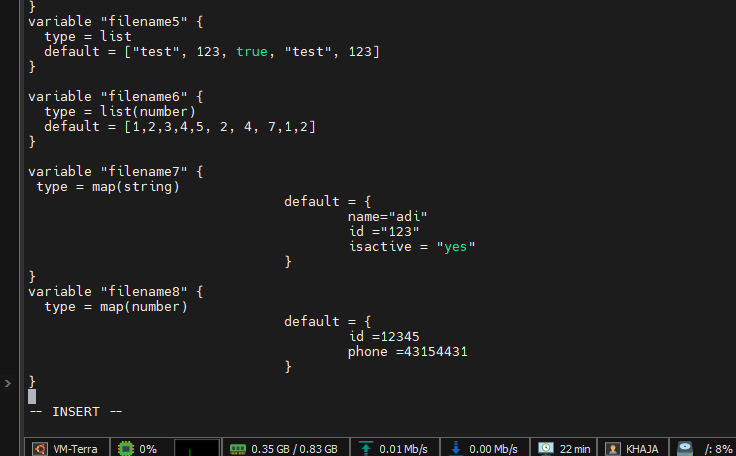
id =12345

phone =43154431

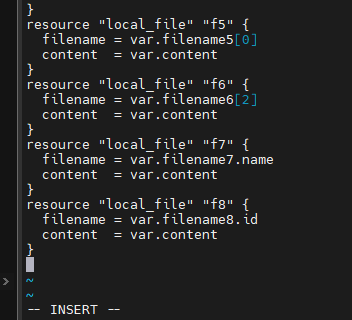
}

}

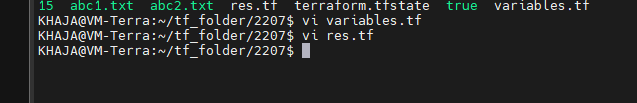
**Variable are mapped to variable file like vi variable.tf**



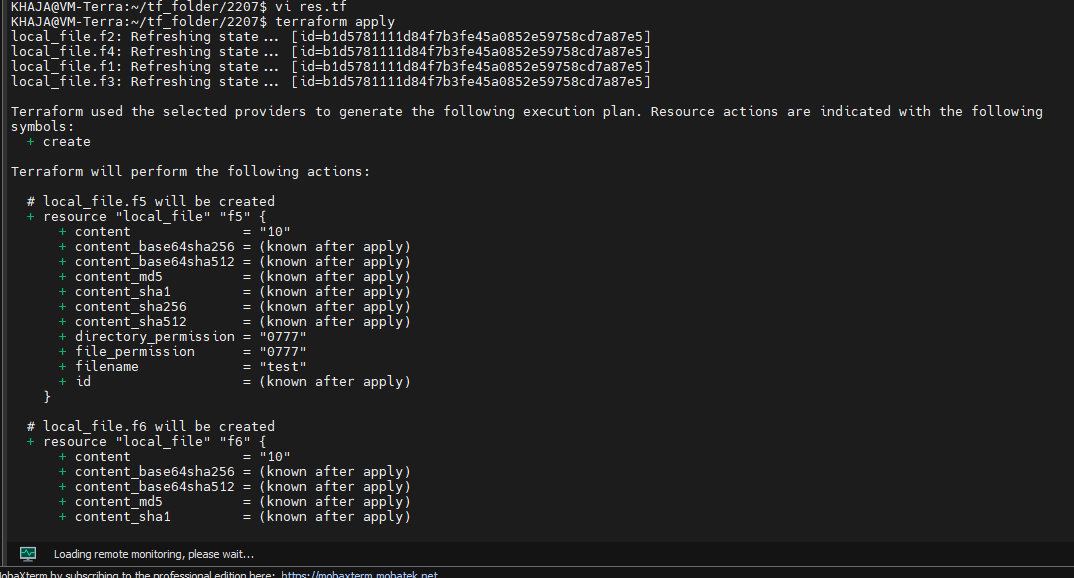
**Adding resources to the resource file “vi res.tf”**

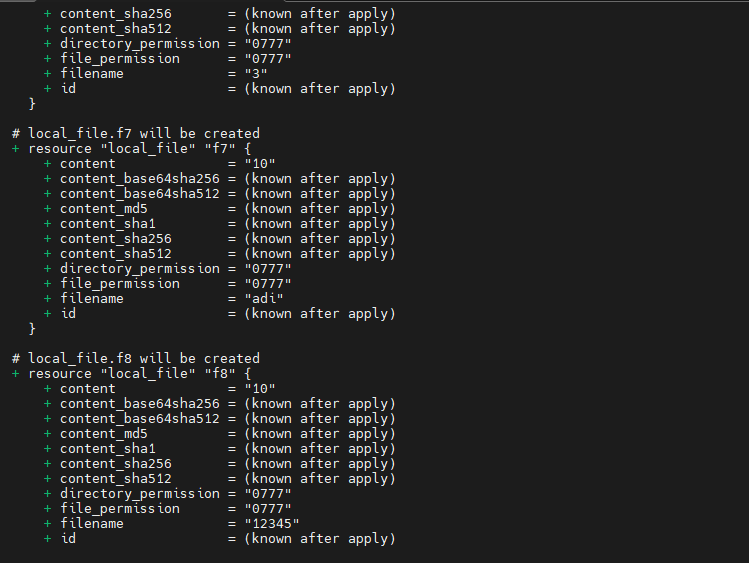


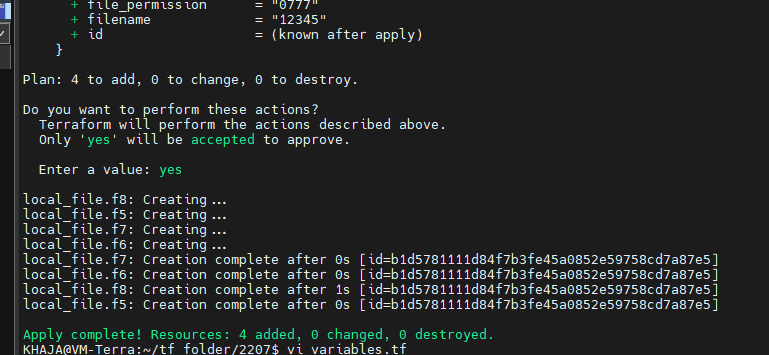
Know we have variable and resources with respect to their files



Execute the command apply







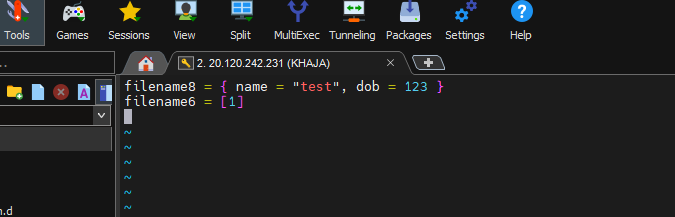
Inject the values by using .tfvar

vi terraform.tfvar

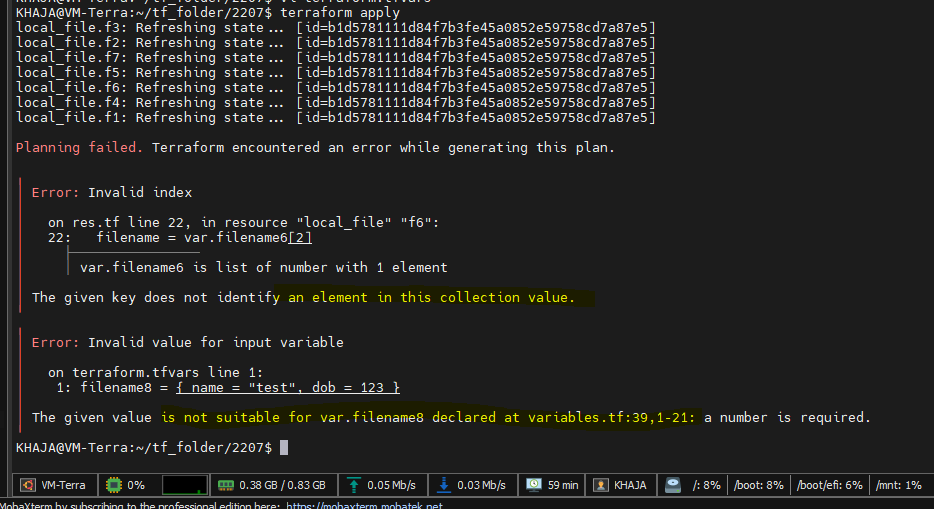
lets inject two values

filename8 = { name = "test", dob = 123 }

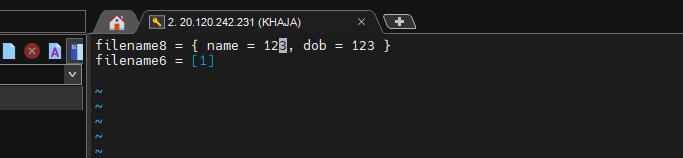
filename6 = [1] # we are injecting one value



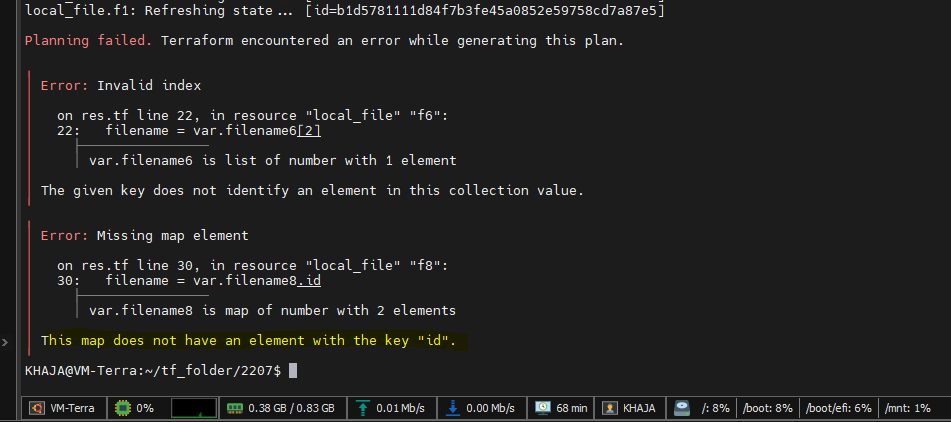
When we execute terraform apply



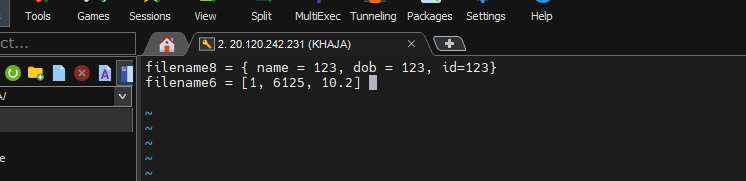
* It show an error because here filename8 we use number
* In filename6 given values is [2] but here we have only one element



Here I change name=123 it’s a number after apply

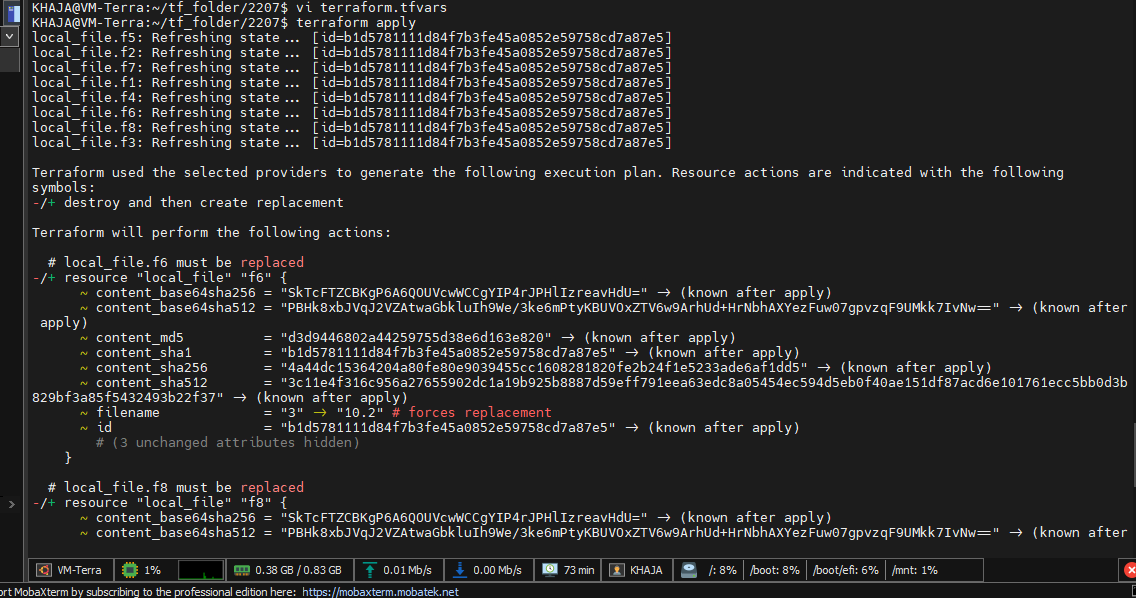


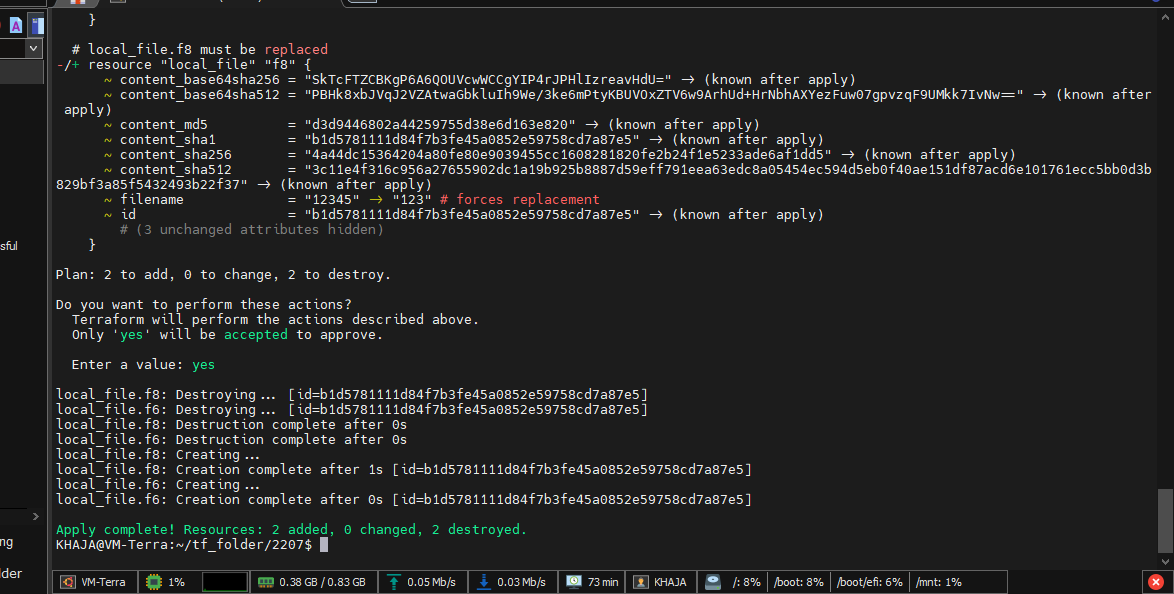
Again its show me an error does not have an element



Know I have change the value id =123 a random number

Execute terraform apply know





We can also terraform apply -auto-approve

Automatically **applies the Terraform plan** without prompting for **manual confirmation**.

