Terraform Task

**Name:Shaik Khaja Basha**

**Batch : Batch 11**

**Date : 23.07.2025**

**Task : datatypes**

1. **Define all datatypes using random provider**

**Ans :**

Create a file name variable.tf using vi variable.tf and enter the details

**Primitive data types**

# 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

}

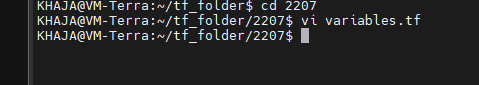
# Content can be of any type

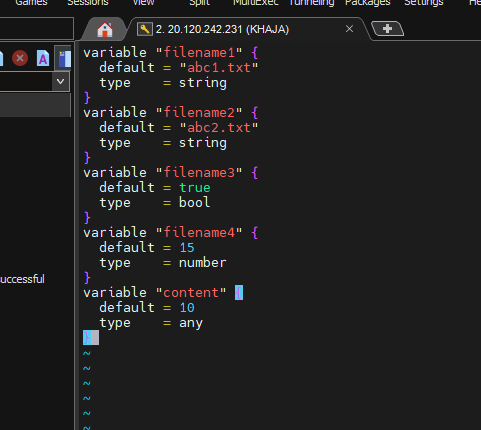
variable "content" {

default = 10

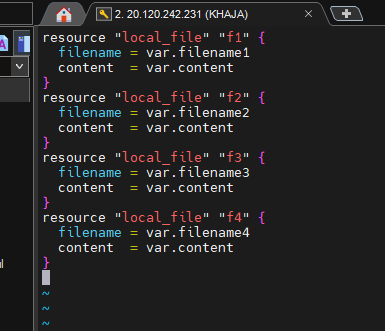
type = any

}

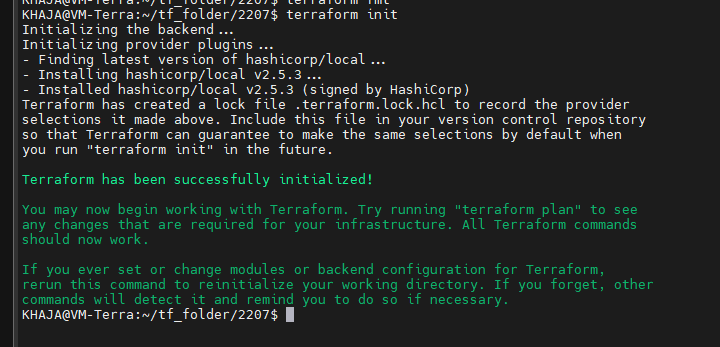




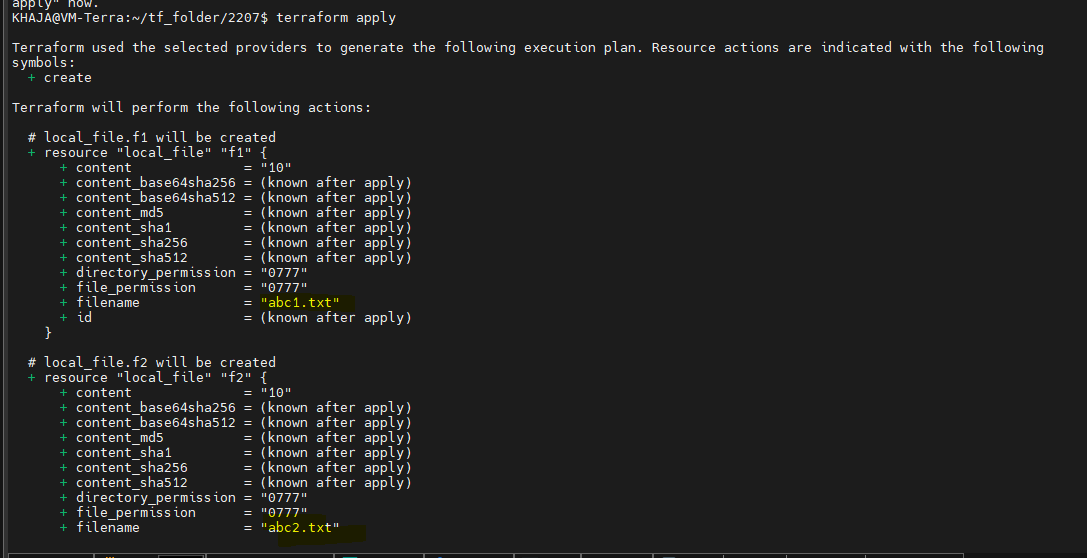
**Resources are**

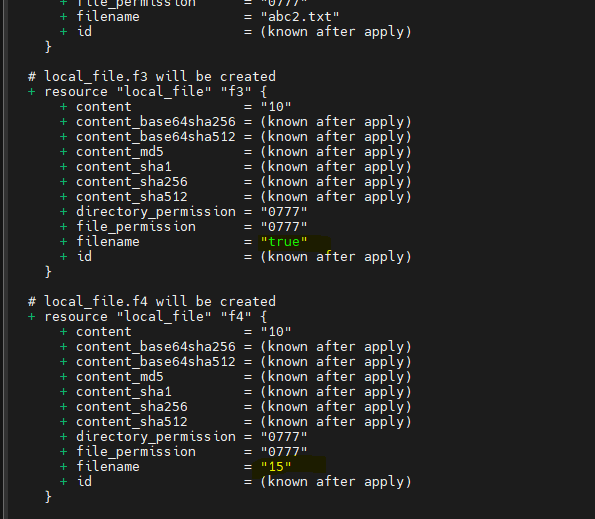


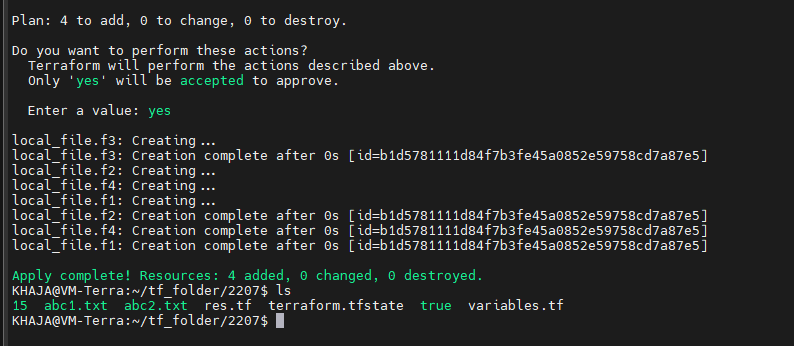
Execute the command terraform init



Execute terraform apply







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

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

**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 = {

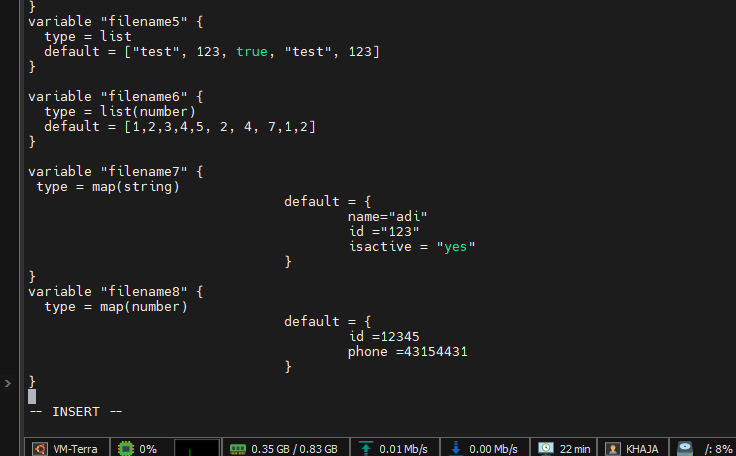
id =12345

phone =43154431

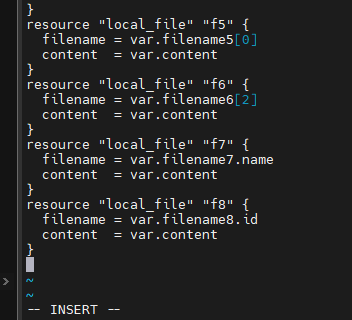
}

}

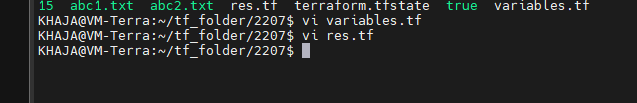
**Variable are mapped to variable file like vi variable.tf**



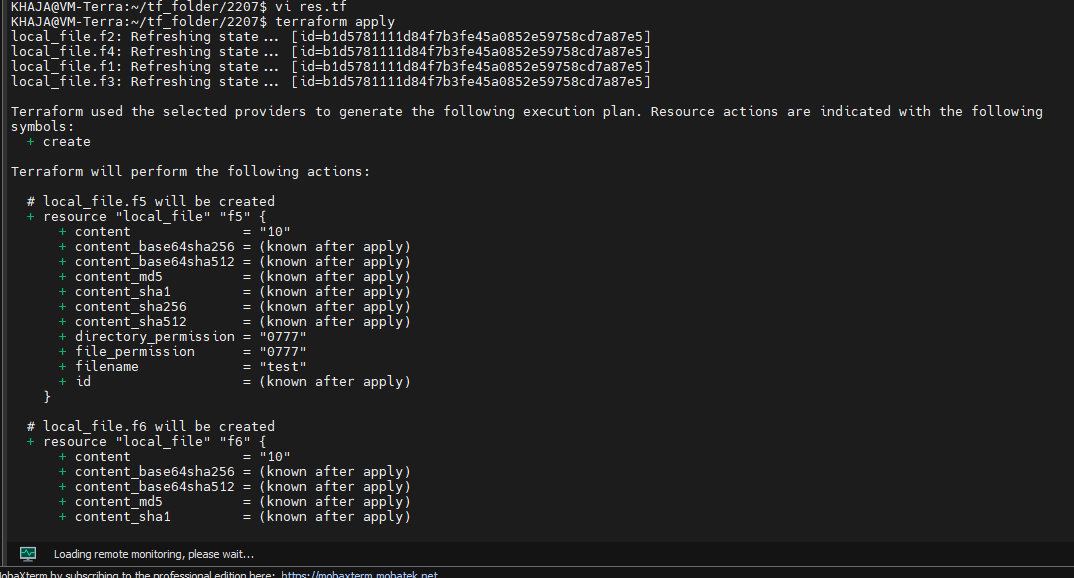
**Adding resources to the resource file “vi res.tf”**

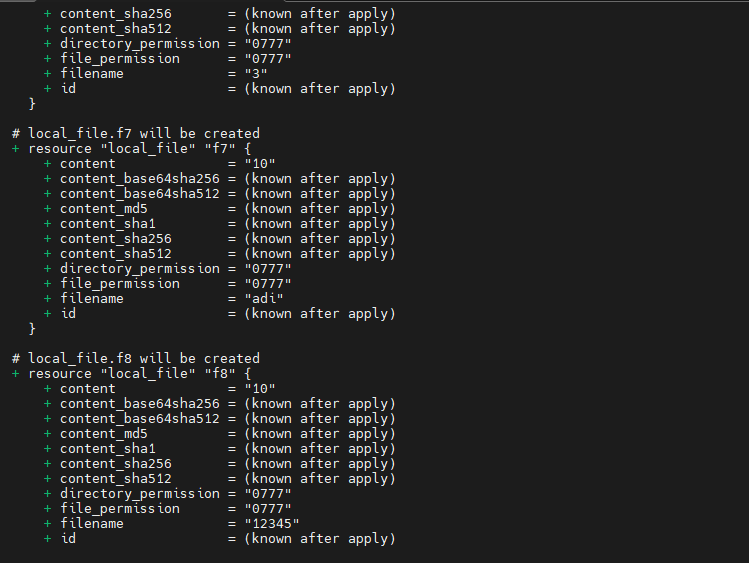


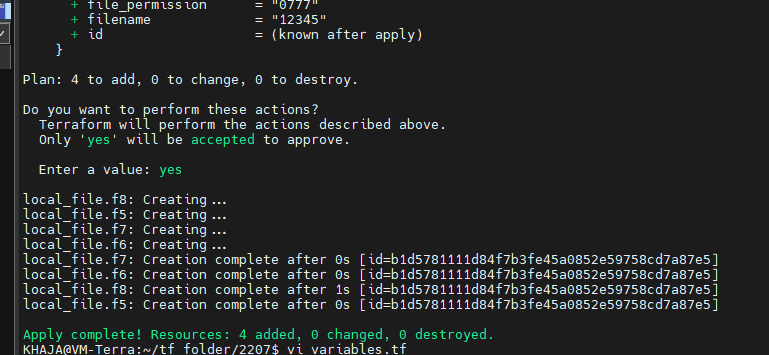
Know we have variable and resources with respect to their files



Execute the command apply







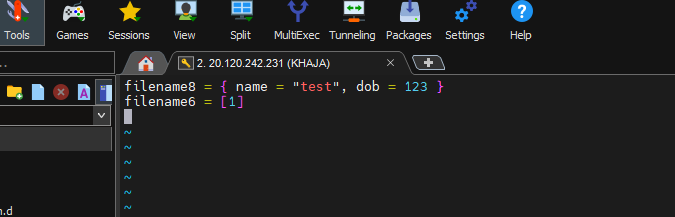
Inject the values by using .tfvar

vi terraform.tfvar

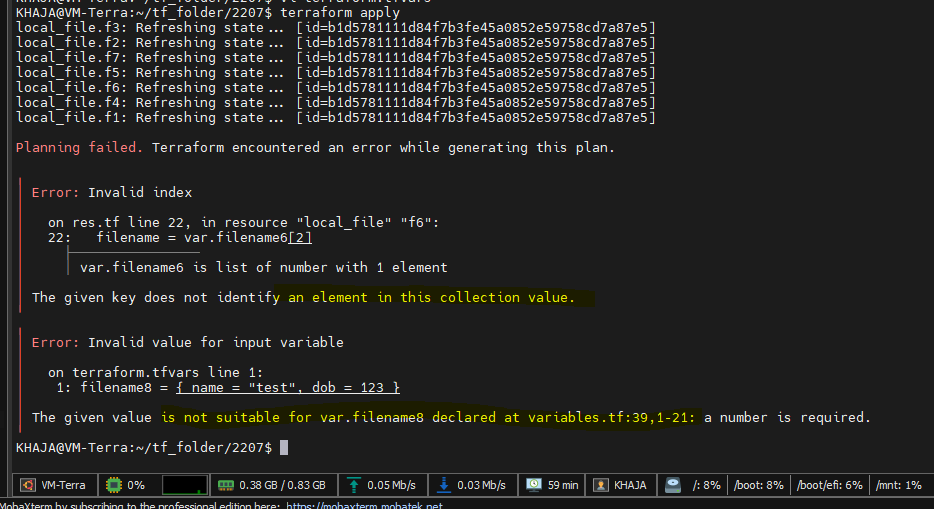
lets inject two values

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

filename6 = [1] # we are injecting one value



When we execute terraform apply



**Tuple :**

A tuple in Terraform is a fixed-length, ordered collection of values where each position has its own specific type. Unlike lists (which must contain elements of the same type), tuples can contain elements of different types.

**Tuple Characteristics:**

* Fixed number of elements (length is part of the type definition)
* Each position has a specific type
* Defined using the syntax tuple([type1, type2, ...])

Example:

variable "example\_tuple" {

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

default = ["hello", 42, true, ["a", "b", "c"]]

}

Accessing Tuple Elements:

var.example\_tuple[0] *# "hello" (string)*

var.example\_tuple[1] *# 42 (number)*

var.example\_tuple[2] *# true (bool)*

var.example\_tuple[3] *# ["a", "b", "c"] (list)*

**Random Provider**

The **Random provider** generates random values during Terraform operations. It's useful for creating unique names, passwords, or other values that need to be unpredictable.

**initialize the random provider:**

terraform {

required\_providers {

random = {

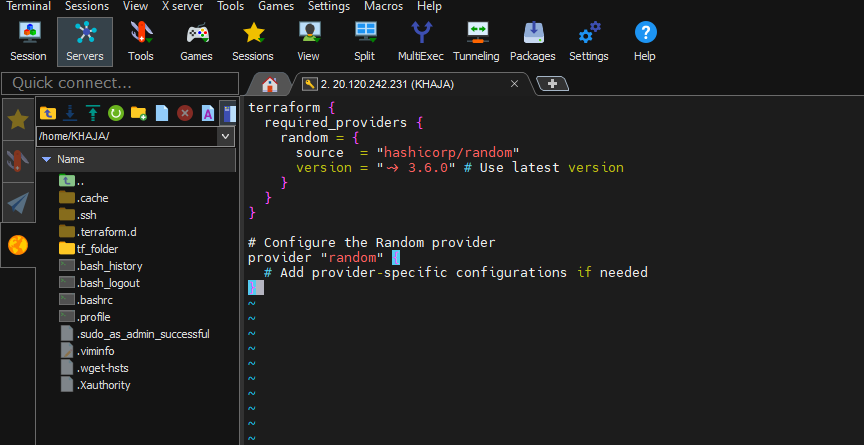
source = "hashicorp/random"

version = "~> 3.0"

}

}

}



**variables.tf - Declare of variables**

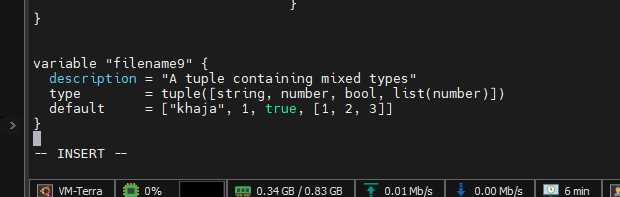
variable "filename9" {

description = "A tuple containing mixed types"

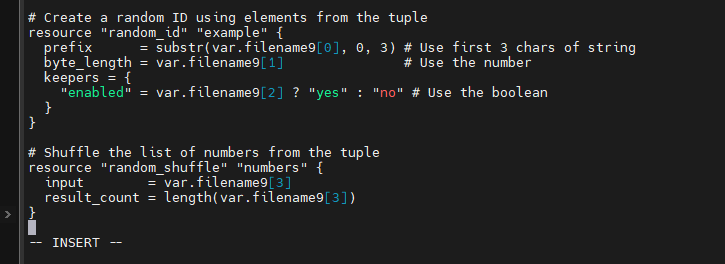
type = tuple([string, number, bool, list(number)])

default = ["khaja", 1, true, [1, 2, 3]]

}

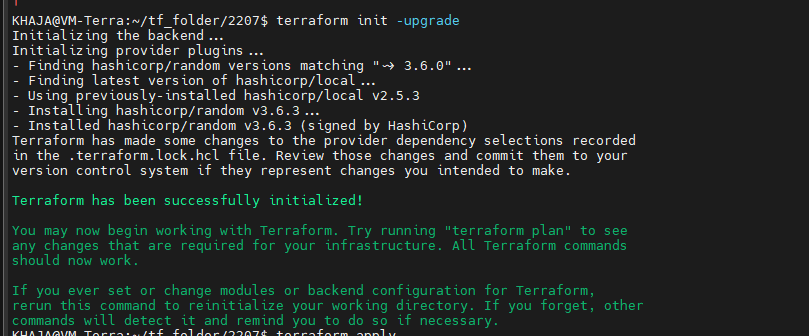


**res.tf - Define resources using Random provider**

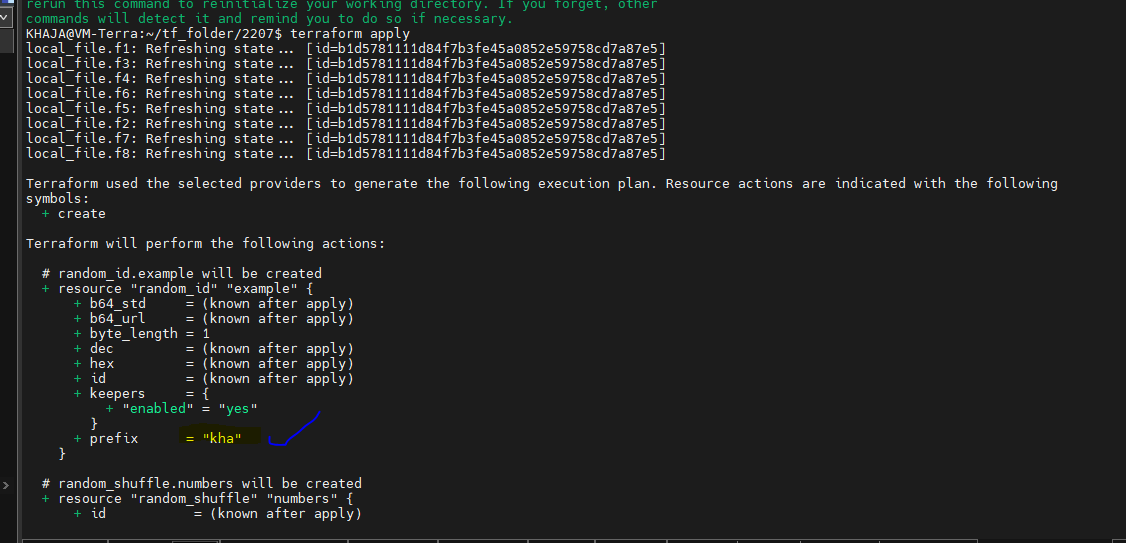


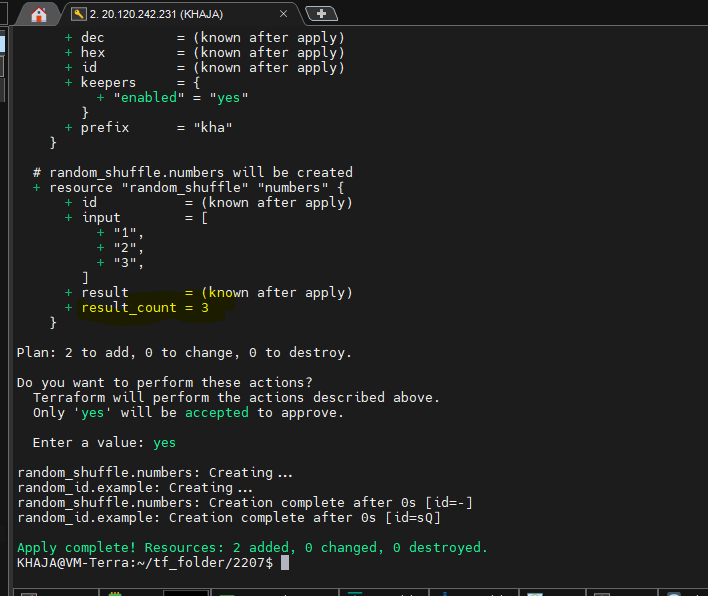
Added the resources to the existing res.tf

Execute the terraform init with update

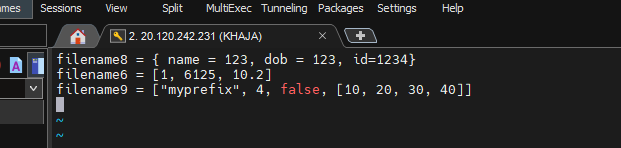


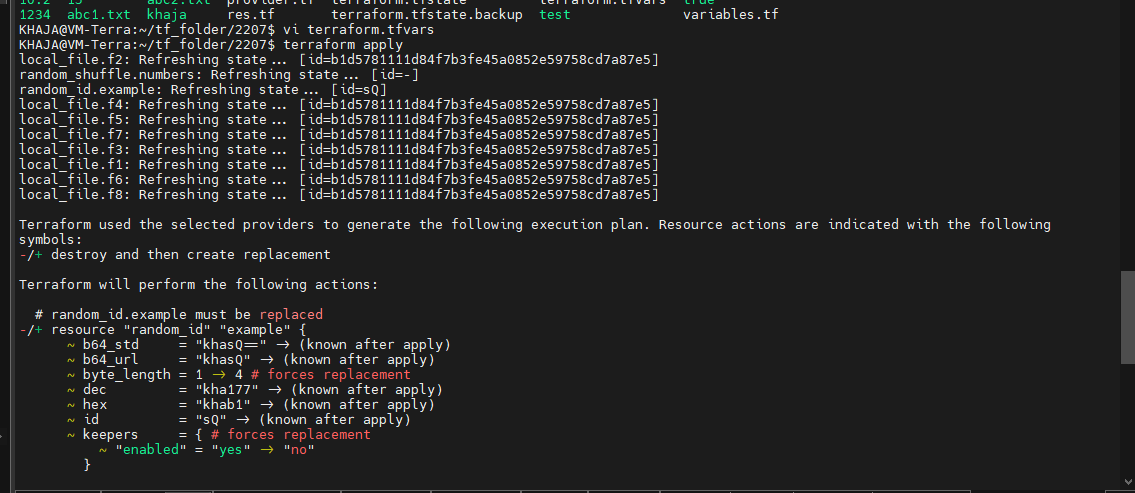
Use terraform apply to make the changes

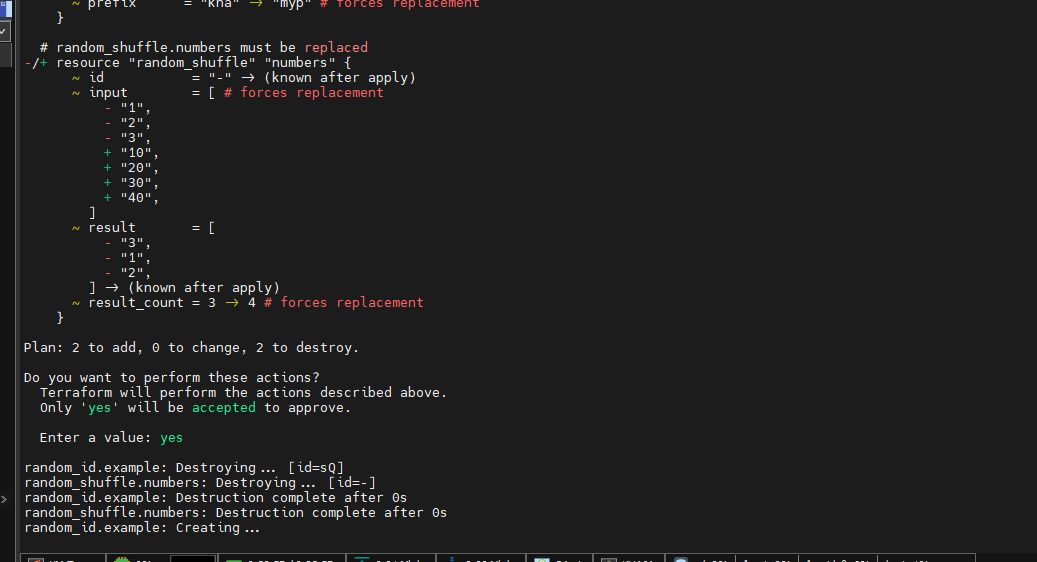


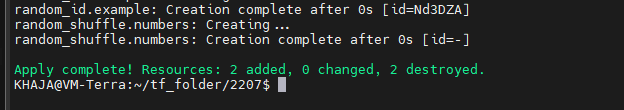


**terraform.tfvars - Provide variable values (optional)**









Key Benefits of This Structure:

1. **Separation of Concerns** - Each file has a clear purpose
2. **Better Maintainability** - Easier to find and modify components
3. **Reusability** - Provider configuration can be shared across modules
4. **Clear Dependencies** - Obvious where each component is defined

**object :**

user defind datatype -> wrapper on map, it declares the key structure and type

type = object({

name = string

id = number

address = list(string)

})

default = {

name = "adi"

id = 123

address = ["marathalli","bangalore","560037"]

}

**variables.tf - Object Variable Definition**

variable "filename10" {

type = object({

name = string

id = number

address = list(string)

})

default = {

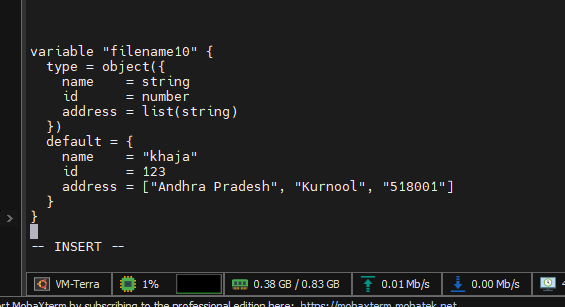
name = "khaja"

id = 123

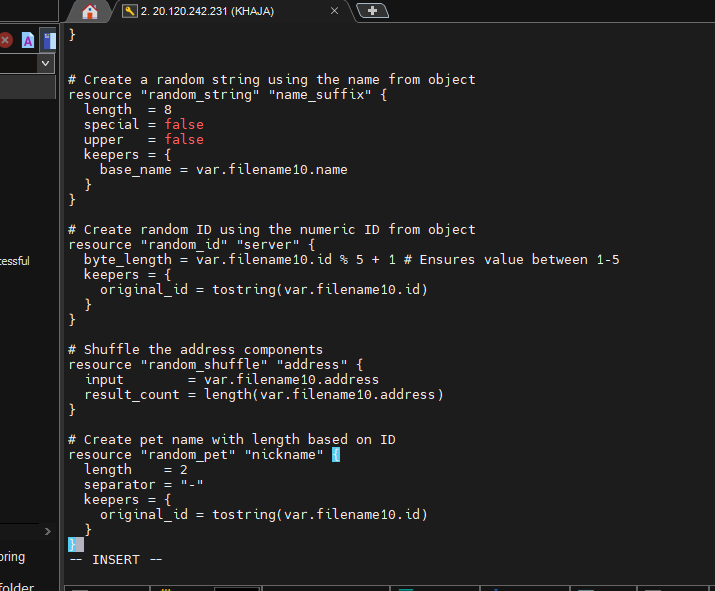
address = ["Andhra Pradesh", "Kurnool", "518001"]

}

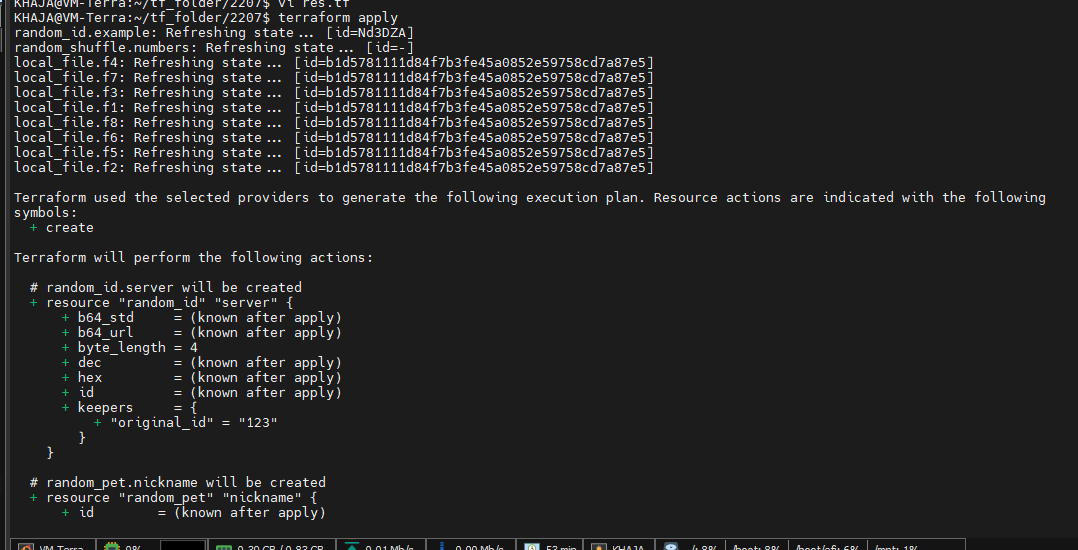
}

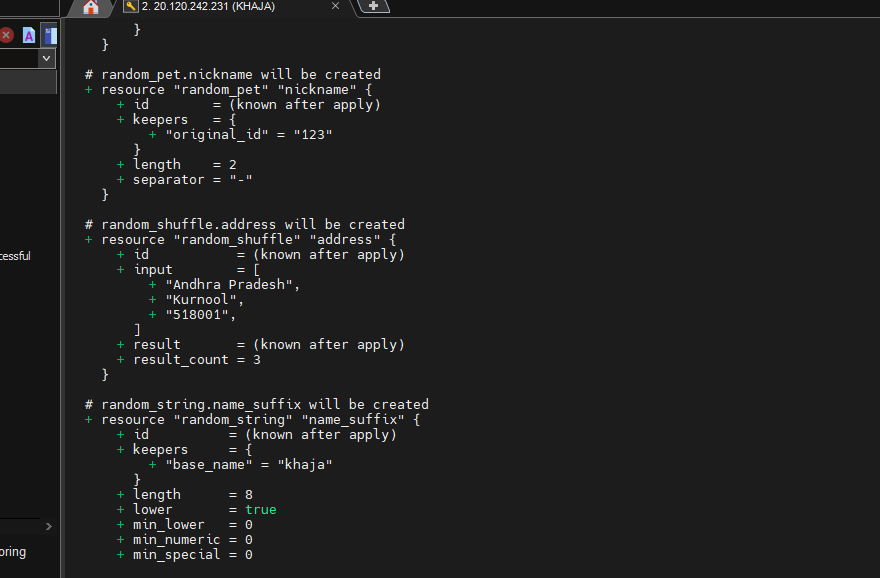


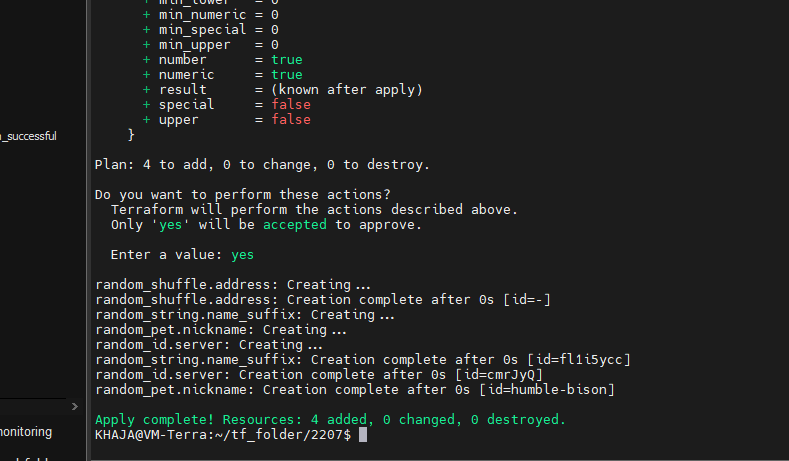
**res.tf - Random Provider Resources Using the Object**



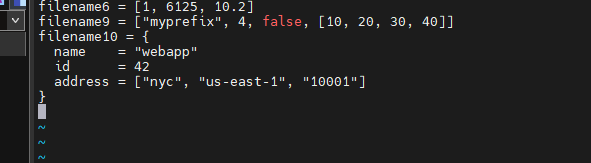
Execute the command terraform apply



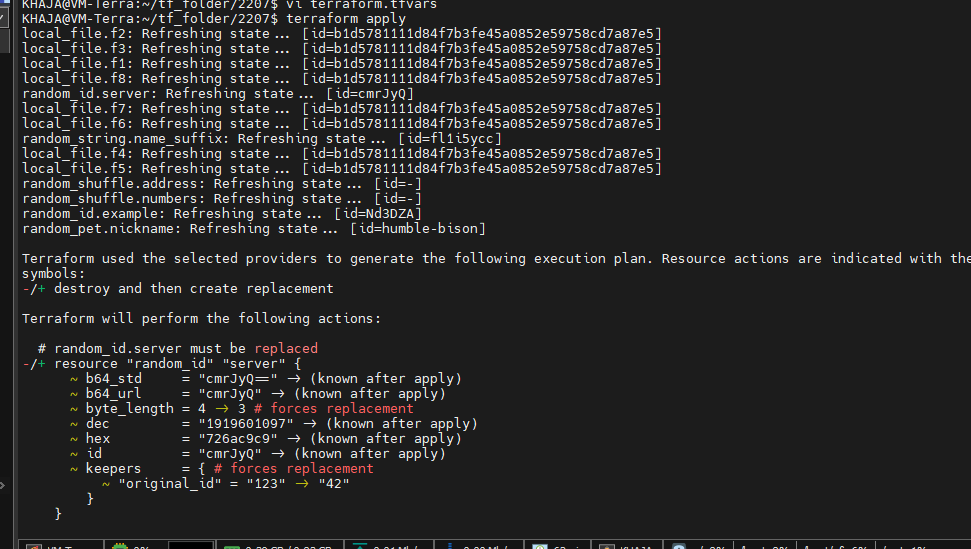


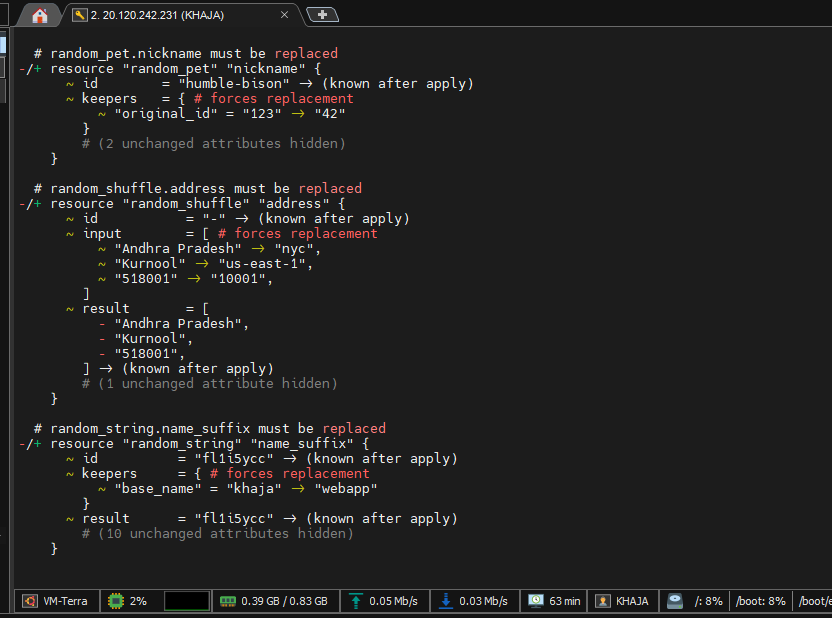


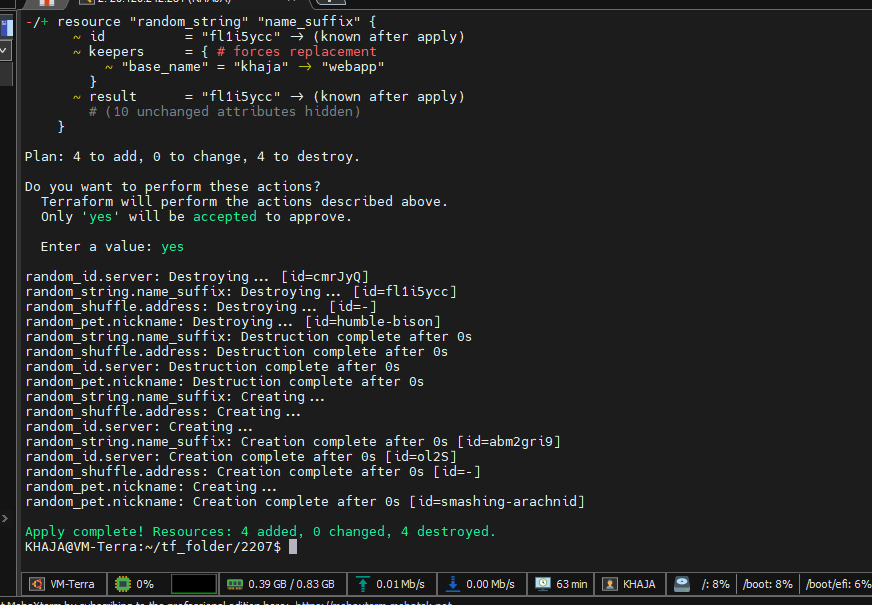
**terraform.tfvars - Provide variable values (optional)**



Again execute the terraform apply with the changes







**Null provider**

A provider for creating "no-op" resources that don't provision real infrastructure.  
**Primary Use Cases:**

* Dependency management
* Debugging variables
* Triggering side effects

**Example:**

resource "null\_resource" "debug" {

triggers = {

timestamp = timestamp() *# Re-runs when this changes*

}

provisioner "local-exec" {

command = "echo 'Triggered at ${self.triggers.timestamp}'"

}

}

**Using Null Provider with Tuple**

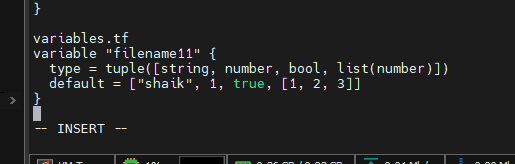
variables.tf

variable "filename11" {

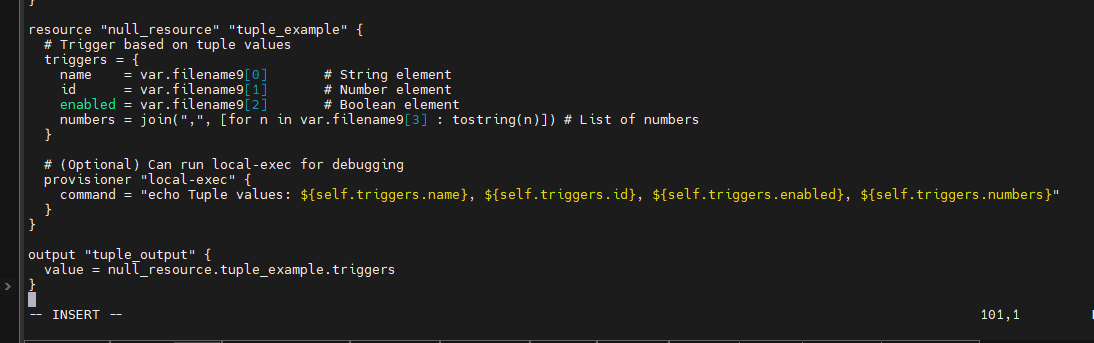
type = tuple([string, number, bool, list(number)])

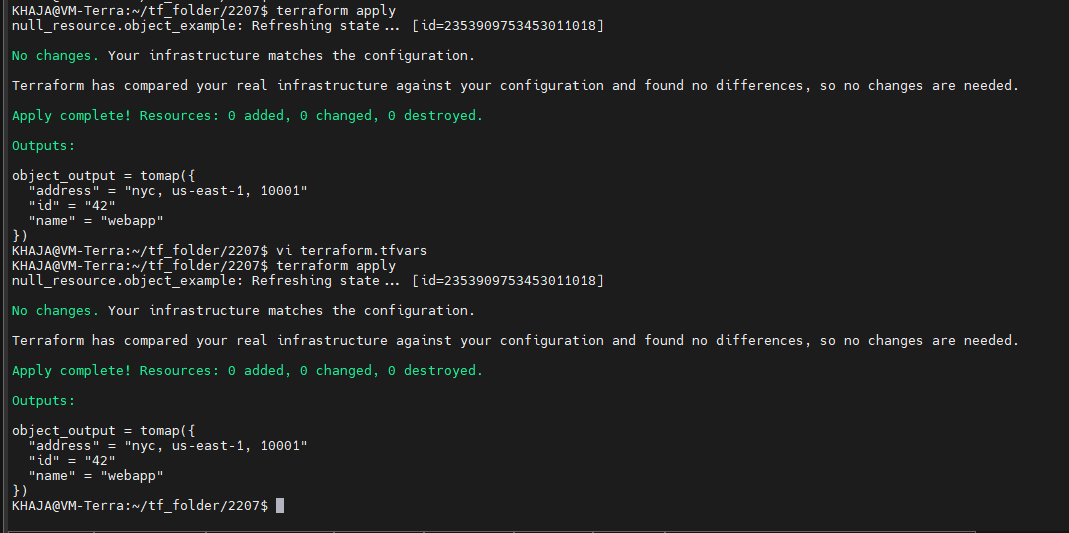
default = ["shaik", 1, true, [1, 2, 3]]

}

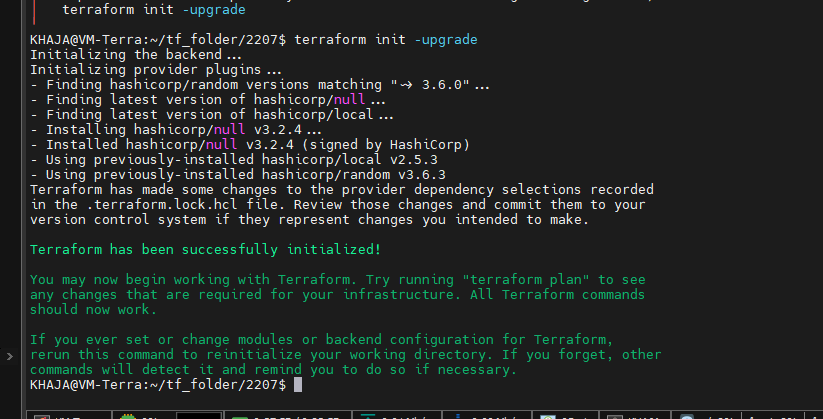


res.tf for declaring reources

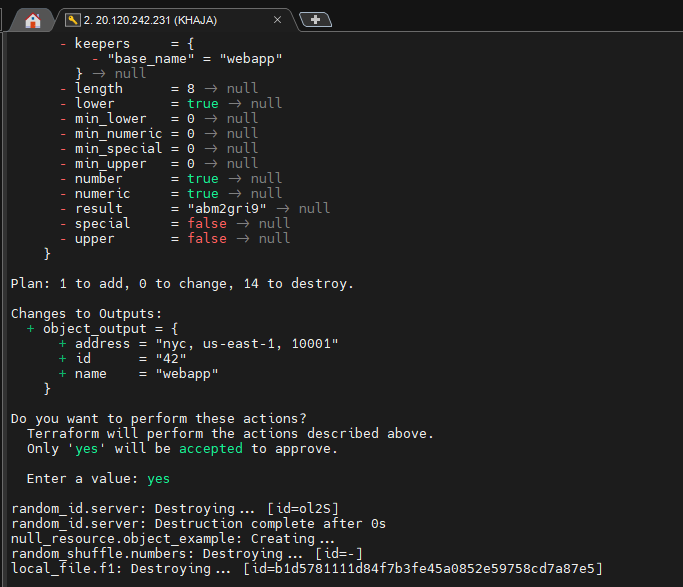




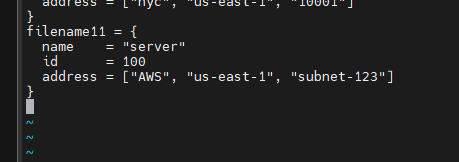
Execute the command terraform init



Execute the command terraform apply



terraform.tfvars (Optional)



**Using Null Provider with Object**

variables.tf

variable "filename12" {

type = object({

name = string

id = number

address = list(string)

})

default = {

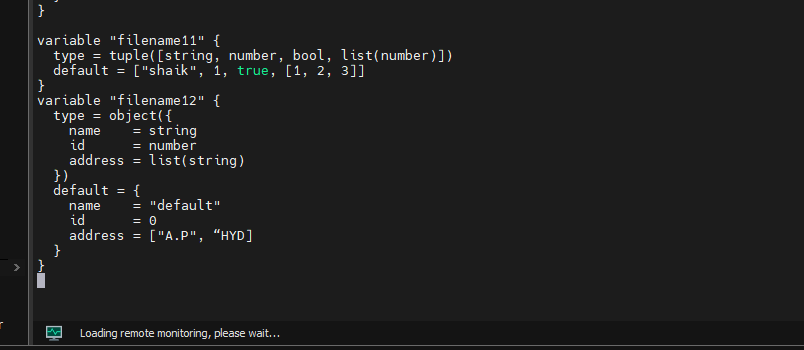
name = "default"

id = 0

address = ["A.P", “HYD]

}

}



**res.tf for declaring reources**

resource "null\_resource" "object\_example" {

triggers = {

# Extract all object fields

name = var.filename12.name

id = var.filename12.id

address = join(", ", var.filename12.address)

}

**# (Optional) Debug output**

provisioner "local-exec" {

command = "echo Object values: ${self.triggers.name}, ${self.triggers.id}, ${self.triggers.address}"

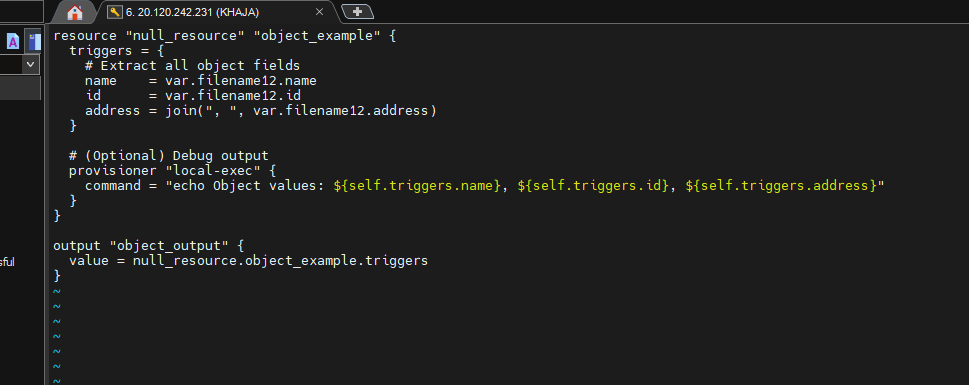
}

}

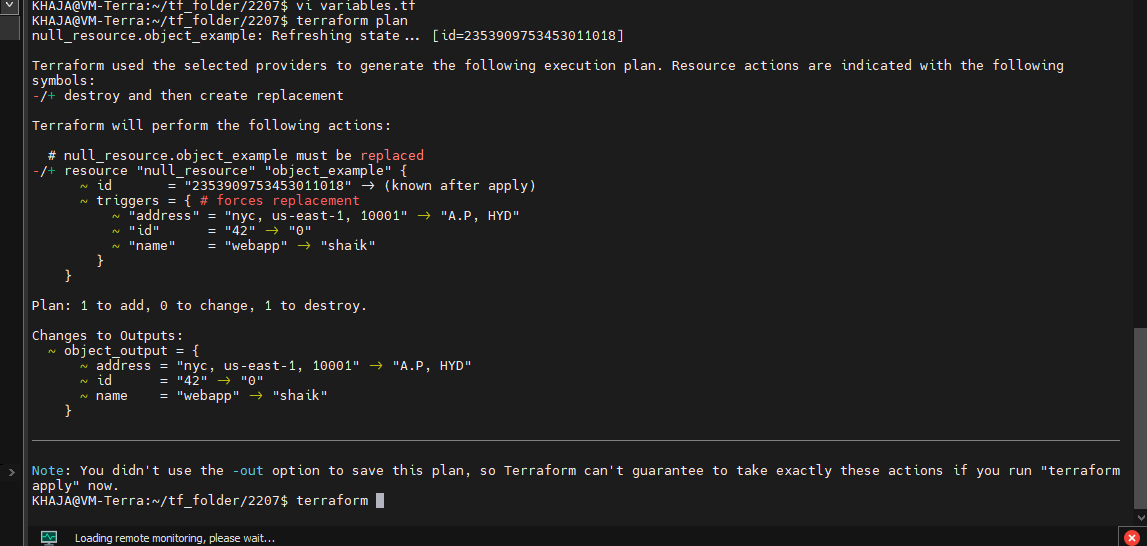
output "object\_output" {

value = null\_resource.object\_example.triggers

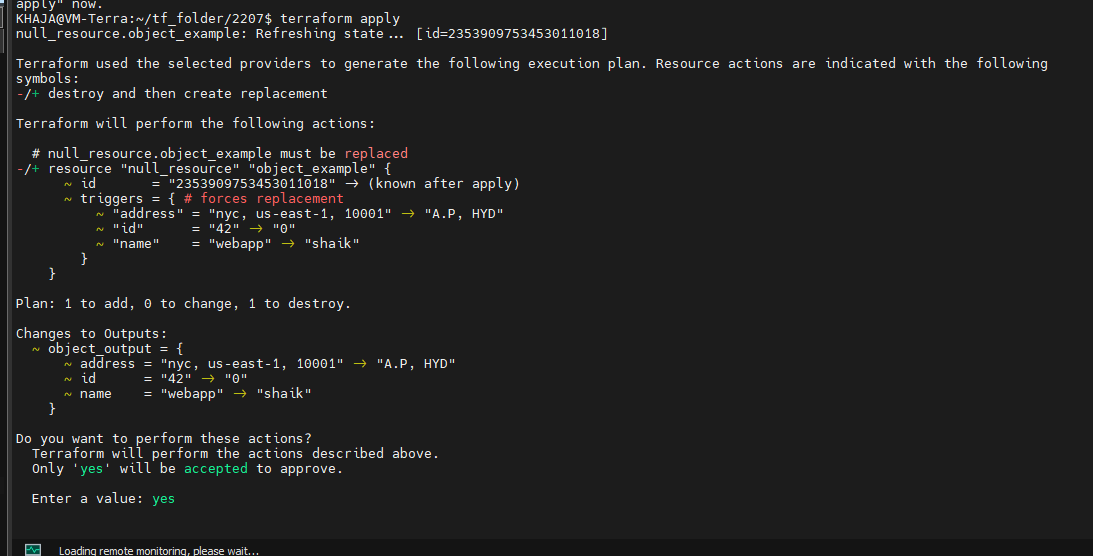
}

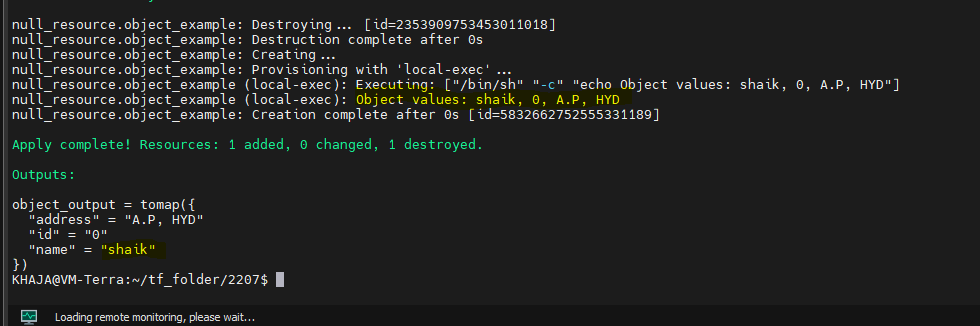


Execute the command terraform plan

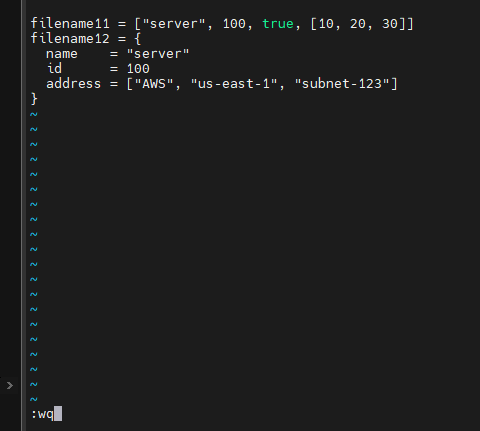


Run the command terraform apply for actual change

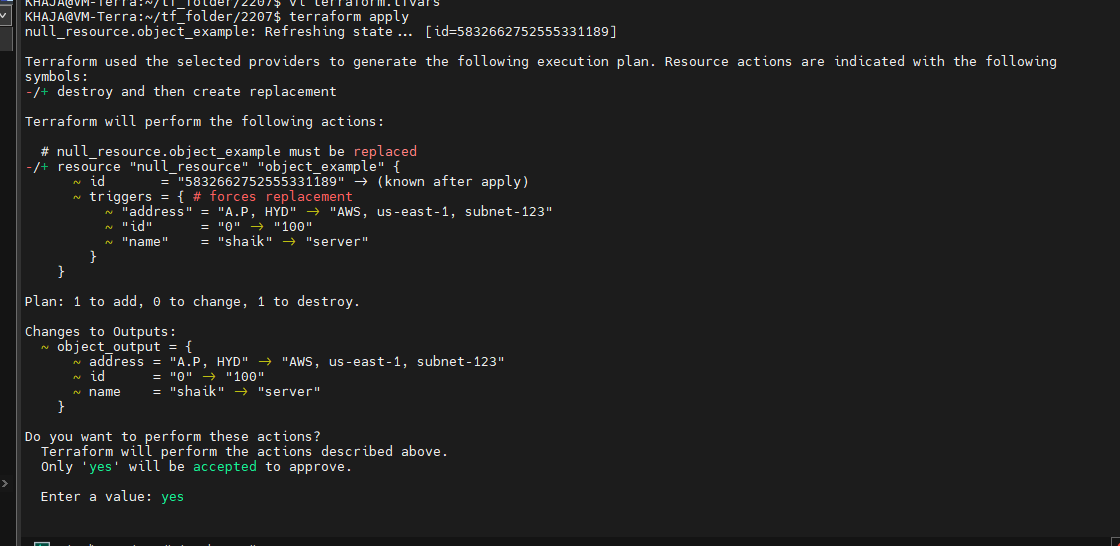




**terraform.tfvars (Optional)**



Run the command terraform apply





**Key Use Cases for Null Provider with Tuples & Objects**

1. **Dependency Management**
   * Ensures resources wait for variable processing before execution.
2. **Debugging Variables**
   * Use local-exec to print variable values during terraform apply.
3. **Triggering Other Actions**
   * Useful with local-exec or remote-exec provisioners when variables change.
4. **Conditional Logic**
   * Combine with count or for\_each to control resource creation.