TASKS ON TERRAFORM

- **NAME: S.MUZZAMMIL HUSSAIN**
- **DATE: 28/07/2025**
- **BATCH: 11**
- **NO.OF TASKS: 1**

Task 1: What are modules explain with example

A **Terraform module** is a collection of Terraform configuration files (typically .tf files) organized in a single directory that define a set of resources to be created and managed together. Even a simple directory with one or more Terraform files is considered a module. The module acts as a reusable, logical container grouping resources that serve a specific purpose or infrastructure function.

Modules help us organize, abstract, and encapsulate infrastructure code, allowing us to reuse it efficiently across different environments or projects. Every Terraform configuration contains at least one module called the **root module** (the code in your current working directory), and can call other modules inside it as **child modules**.

In short, modules enable:

- **Reusability**: Write infrastructure code once and use it many times.
- **Abstraction**: Hide complex resource groups behind simpler interfaces.
- **Consistency**: Deploy infrastructure components consistently.
- **Modularity**: Break large infrastructure into manageable, logical parts.

Detailed Notes on Terraform Setup

Overview:

We have a Terraform root module (**main.tf**) that calls multiple instances of the **config** module (located locally or remotely). Each instance is passed different variable values (f1, c1). The **config** module creates a local file resource based on these inputs and exposes an output.

1. Root Module (main.tf)

What it does:

- Defines four modules:
  - m1 and m2: Use a local module ./config with different input values.
  - m3: Uses a module from a relative path ../2707
  - m4: Uses a remote module from GitHub (github.com/muzzammil-hamdu/Terraform-module-test.git).

```
module m1 {
  source = "./config"
  f1     = "123.txt"
  c1     = "content from module"
}

module m2 {
```

```
  source = "./config"
  f1     = "456.txt"
  c1     = "content from module 2"
}

module m3 {
  source = "../2707"
}

module m4 {
  source = "github.com/muzzammil-hamdu/Terraform-module-test.git"
}
```

- Outputs (m1f1 and m2f1) expose resource info from m1 and m2 modules, specifically the fa1 output from within the module instances.

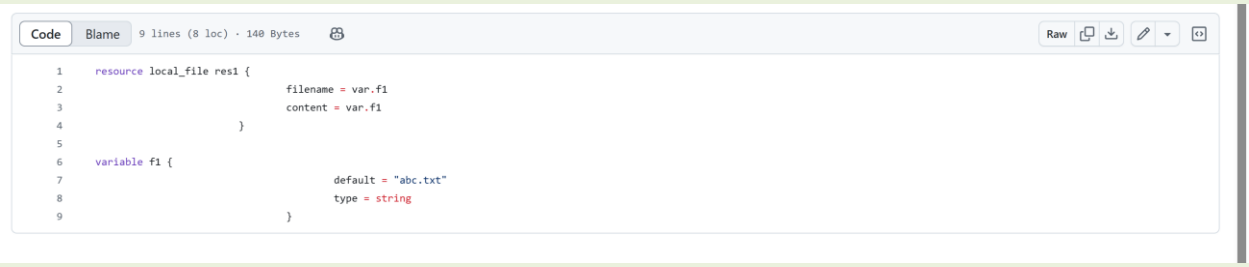
```
output m1f1 {
  value = module.m1.fa1
}

output m2f1 {
  value = module.m2.fa1
}
```

```
module "m1" {
  source = "./config"
  f1     = "123.txt"
  c1     = "content from module"
}
module "m2" {
  source = "./config"
  f1     = "456.txt"
  c1     = "content from module 2"
}

module "m3" {
  source = "../2707"

}

module "m4" {
  source = "github.com/muzzammil-hamdu/Terraform-module-test.git"

}

output "m1f1" {
  value = module.m1.fa1
}

output "m2f1" {
  value = module.m2.fa1
}
```

For module **m3** we have given location of main.tf as follows(copy the path to provide in module m3)

```
locals {
        filename_upper =[for value in var.filename: upper(value)]
        map_keys = [ for key, value in  var.filnamemap : upper(key) ]
        map_values = [ for key, value in  var.filnamemap : upper(value) ]
        map_upper = { for key, value in  var.filnamemap : key => upper(value) }
       }


output a1{
         value = local.filename_upper
}
output a2{
         value = local.map_keys
}
output a3{
         value = local.map_values
}
output a4{
         value = local.map_upper
}


variable "filename" {
  type    = list(string)
  default = ["a", "b", "c"]
}
resource "local_file" "f8" {
  count    = length(local.filename_upper)
  filename = local.filename_upper[count.index]
  content  = "test"
}

variable filnamemap {
  type    = map(string)
  default = {
        name ="a"
        address = "b"
        }
}
```

The github repository consists file with configuration as follows(this repository URL is given in module **m4**



```
resource local_file res1 {
                                filename = var.f1
                                content = var.f1
                        }

variable f1 {
                                        default = "abc.txt"
                                        type = string
                                }
```

2. config Module

Create a directory named **config** and create a file in it named **res.tf** using **vi res.tf**

The module located in **./config** is used by **m1** and **m2**. It contains:

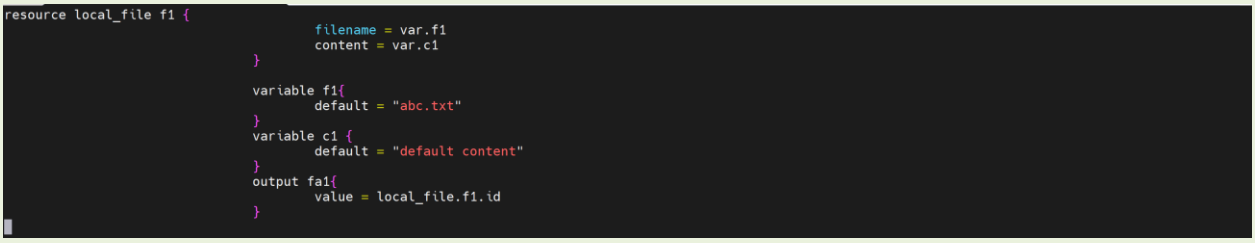
res.tf (Terraform resource and outputs)

```
resource "local_file" "f1" {
  filename = var.f1
  content  = var.c1
}

output "fa1" {
  value = local_file.f1.id
}
```

- **local_file resource**: Uses the local_file resource to create a file on the local filesystem.
  - filename: Comes from var.f1 (variable "f1").
  - content: Comes from var.c1 (variable "c1").
- **output 'fa1'**: Returns the ID of the created local file resource, which is one of the outputs exposed to the root module.

variables.tf (variables for the module)

```
variable "f1" {
  default = "abc.txt"
}

variable "c1" {
  default = "default content"
}
```



```
resource local_file f1 {
                                filename = var.f1
                                content = var.c1
                        }

                        variable f1{
                                default = "abc.txt"
                        }
                        variable c1 {
                                default = "default content"
                        }
                        output fa1{
                                value = local_file.f1.id
                        }
```

- Two variables declared:
  - f1: filename, defaults to "abc.txt".
  - c1: file content, defaults to "default content".

Both can be overridden when calling the module, as done in main.tf.

Run the command **terraform init**

```
mujju@terravm:~/b11/2807$ terraform init
Initializing the backend...
Initializing modules...
- m1 in config
- m2 in config
- m3 in ../2707
Downloading git::https://github.com/muzzammil-hamdu/Terraform-module-test.git for m4...
- m4 in .terraform/modules/m4
Downloading registry.terraform.io/Azure/vnet/azurerm 5.0.1 for m5...
- m5 in .terraform/modules/m5
Initializing provider plugins...
- Finding latest version of hashicorp/local...
- Finding hashicorp/azurerm versions matching "~> 4.0"...
- Finding azure/modtm versions matching "0.3.2"...
- Finding hashicorp/random versions matching ">= 3.3.2, < 4.0.0"...
- Installing hashicorp/local v2.5.3...
- Installed hashicorp/local v2.5.3 (signed by HashiCorp)
- Installing hashicorp/azurerm v4.37.0...
- Installed hashicorp/azurerm v4.37.0 (signed by HashiCorp)
- Installing azure/modtm v0.3.2...
- Installed azure/modtm v0.3.2 (signed by a HashiCorp partner, key ID 6F0B91BDE98478CF)
- Installing hashicorp/random v3.7.2...
- Installed hashicorp/random v3.7.2 (signed by HashiCorp)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://developer.hashicorp.com/terraform/cli/plugins/signing
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.
mujju@terravm:~/b11/2807$
```

Run the command **terraform apply**

```
mujju@terravm:~/b11/2807$ terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # module.m1.local_file.f1 will be created
  + resource "local_file" "f1" {
      + content              = "content from module"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "123.txt"
      + id                   = (known after apply)
    }

  # module.m2.local_file.f1 will be created
  + resource "local_file" "f1" {
      + content              = "content from module 2"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "456.txt"
      + id                   = (known after apply)
    }
```

```
  # module.m3.local_file.f8[0] will be created
  + resource "local_file" "f8" {
      + content              = "test"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "A"
      + id                   = (known after apply)
    }

  # module.m3.local_file.f8[1] will be created
  + resource "local_file" "f8" {
      + content              = "test"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "B"
      + id                   = (known after apply)
    }

  # module.m3.local_file.f8[2] will be created
  + resource "local_file" "f8" {
      + content              = "test"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "C"
      + id                   = (known after apply)
    }
```

```
  # module.m4.local_file.res1 will be created
  + resource "local_file" "res1" {
      + content              = "abc.txt"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "abc.txt"
      + id                   = (known after apply)
    }

Plan: 6 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + m1f1 = (known after apply)
  + m2f1 = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

module.m4.local_file.res1: Creating...
module.m3.local_file.f8[0]: Creating...
module.m3.local_file.f8[1]: Creating...
module.m3.local_file.f8[2]: Creating...
module.m1.local_file.f1: Creating...
module.m2.local_file.f1: Creating...
module.m1.local_file.f1: Creation complete after 0s [id=23a5190ba385ff85f9c4a03049c7a023e5c43605]
module.m2.local_file.f1: Creation complete after 0s [id=47917037cef863f0daf025504b2feb631d1d1a60]
module.m3.local_file.f8[2]: Creation complete after 0s [id=a94a8fe5ccb19ba61c4c0873d391e987982fbbd3]
module.m4.local_file.res1: Creation complete after 1s [id=060e4b697d1846e3a7bec5a486b38a177b0542d4]
module.m3.local_file.f8[0]: Creation complete after 1s [id=a94a8fe5ccb19ba61c4c0873d391e987982fbbd3]
module.m3.local_file.f8[1]: Creation complete after 1s [id=a94a8fe5ccb19ba61c4c0873d391e987982fbbd3]

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

Outputs:

m1f1 = "23a5190ba385ff85f9c4a03049c7a023e5c43605"
m2f1 = "47917037cef863f0daf025504b2feb631d1d1a60"
mujju@terravm:~/b11/2807$
```

Summary of Workflow:

1. **Root module** calls m1 and m2 modules, overriding f1 and c1 values.
2. Each **config module** instance:
   - Creates a **local file** with the filename and content passed in.
   - Returns the **local file resource ID** as fa1.
3. Root module outputs m1f1 and m2f1 fetch those file resource IDs from each module instance and print/send them as outputs after terraform apply.

Important Notes:

- The local_file resource creates or updates files on the **local machine** Terraform is running on (usually your workstation or CI environment).
- Filenames (f1) like "123.txt" and "456.txt" are relative paths and will be created relative to where you run Terraform.
- The outputs expose the resource IDs (usually file paths) so you can reference or check the files after apply.
- Modules m3 and m4 are included but not detailed; they seem unrelated to the file-writing task but indicate your setup supports multiple sources.

- When m1 runs, it creates a file named 123.txt with content: "content from module".
- When m2 runs, it creates 456.txt with content: "content from module 2".