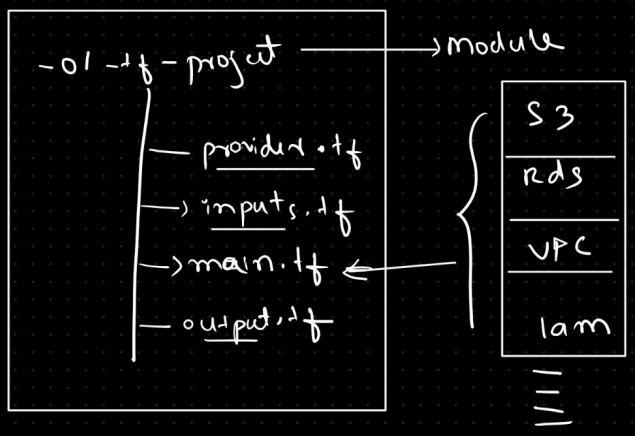
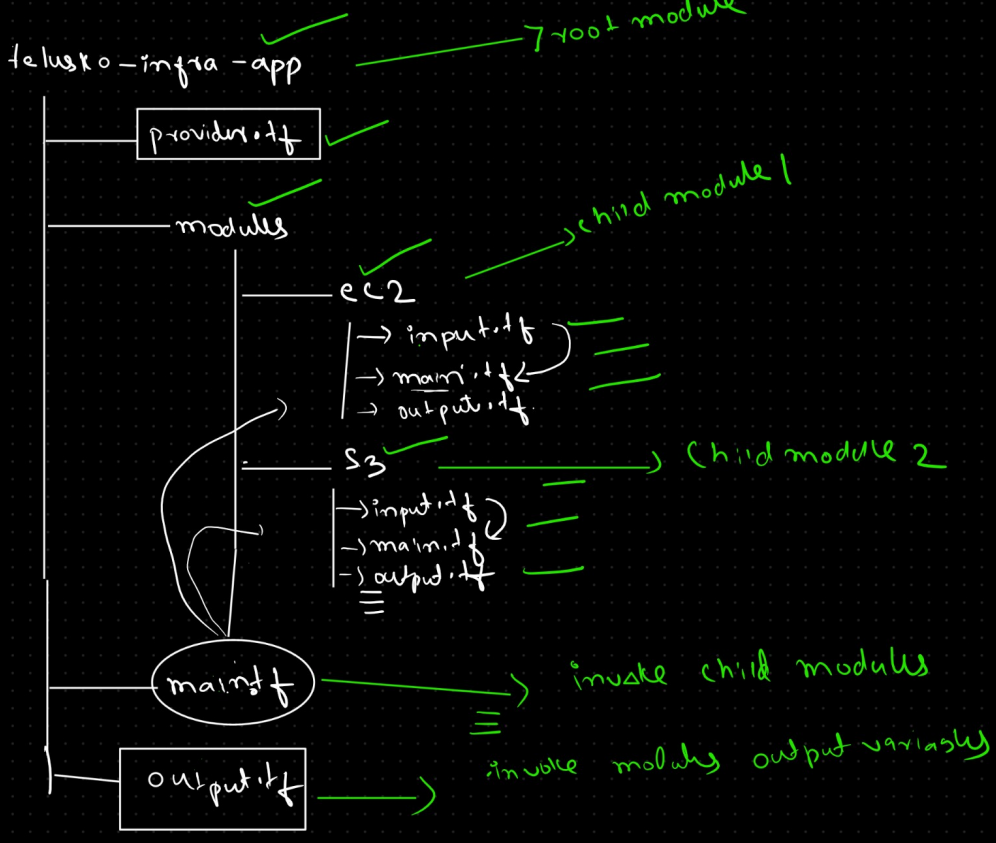
=> Terraform =>

terraform init, terraform fmt,
terraform validate, terraform plan, terraform apply, terraform destroy

inputs.tf, main.tf, provider.tf, output.tf & ——

=> Terraform Modules :-





A terraform module is a set of Terraform Configuration files in a single directory

Single Directory with one or more .tf files even for a simple configurations can be considered as a module

One root module can have any number of child modules in terraform
example: Inside one project or module we can create multiple child modules ec2, s3, rds, iam as a child modules

Note : We will run terraform commands from root module and root module will invoke child modules for execution

Terraform project setup with modules
==========

Step - 1 : Create a project directory

$ mkdir 05-terraform-modules-project

2 .create a provider.tf file

```
provider "aws" {
  region = "us-east-1"
}
```

3.create a "modules" directory inside main project directory

$ mkdir 05-terraform-modules-project/modules

4.Inside modules directory we need to create ec2,s3 directories

$ mkdir 05-terraform-modules-project/modules/ec2

$ mkdir 05-terraform-modules-project/modules/s3

5.Need to add access key, secret access key to the project

```
export AWS_ACCESS_KEY_ID="YOUR_ACCESS_KEY_ID"
export AWS_SECRET_ACCESS_KEY="YOUR_SECRET_ACCESS_KEY"
```

6.Create files in ec2 directory with names input.tf , main.tf , output.tf

```
touch input.tf,
main.tf,
output.tf
```

7.Create files in s3 directory with names input.tf , main.tf , output.tf

```
touch input.tf
,main.tf,
output.tf
```

8.create main.tf , output.tf files under main directory

9.edit ec2 dir main.tf , input.tf , output.tf

$ vi input.tf

```
variable "ami" {
     description = "Amazon vm image value"
     type=string
}
```

```
variable "instance_type" {
      description = "Represents the type of instance"
      default = "t2.micro"
}

$ vi output.tf
output "ec2_vm_public_ip" {
  value= aws_instance.linux_vm.public_ip
}
output "ec2_vm_private_ip" {
  value= aws_instance.linux_vm.private_ip
}


$ vi main.tf
resource "aws_instance" "linux_vm" {
 ami             = var.ami
 instance_type   = var.instance_type
 key_name        = "terraform"
 security_groups = ["default"]
 tags = {
   Name = "modules-Linux_VM"
 }
}

cd ..
cd s3

10 edit main.tf and other files s3 as required


resource "aws_s3_bucket" "telusko_bucket" {

       bucket = "telusko4455"
       acl = "private"
}
cd ..
cd ..
11 .in root directory

11. edit main.tf
vi main.tf
module "my_ec2"
    source = "./modules/ec2
    ami = "ami value"
module "my_s3"
    source = "./modules/s3
}
 12 . edit output file to access child modules related output
output "ec2_vm_public_ip" {
 value = module.my_ec2.ec2_vm_public_ip
}
output "ec2_vm_private_ip" {

 value = module.my_ec2.ec2_vm_private_ip
}
```

terraform init
terraform fmt
terraform validate
terraform plan
terraform apply --auto-approve
terraform destroy --auto-approve

Working with Terraform in windows machine , Lock file, statefile, tain and untaint in terraform

Working with Terraform in Windows Machine
==========
Step - 1 : Download terraform for windows and extract the zip file

after extracting we can use terraform.exe file

Step - 2 : Set path for terraform in system variable -> environmental variables

Step - 3 : Configure AWS Credentials in system environment variables

Step - 4 : Download install Vs code

Taint & Untaint in Terraform:

Taint: You can "taint" a resource using terraform taint to mark it as needing to be re-created during the next terraform apply. It's like telling Terraform that a resource is "bad" or needs re-deployment.

$ terraform taint aws_instance.linux_vm ---> terraform taint resource name

Untaint: Using terraform untaint, you can remove the taint from a resource, so it will not be re-created and will stay as is during the next apply.

$ terraform untaint aws_instance.linux_vm ---> terraform untaint resource name

State file in Terraform :

The state file (terraform.tfstate) is where Terraform stores the current state of your infrastructure. It keeps track of all the resources that Terraform manages, so it knows what to create, update, or delete. This file is essential for Terraform to understand what is already deployed.

Lock file in terraform :

A lock file (.terraform.lock.hcl) is used to lock the versions of provider plugins that Terraform uses.
This ensures that your team or CI/CD pipelines are using the same versions of providers across all environments.
It's automatically generated by Terraform to avoid unexpected changes due to provider updates.