**Terraform resource structure:**

* variables
* Providers
* Resources >>> contains arguments and attributes.
* Outputs

**Terraform Directory Structure:**

* Variables.tf
* Resource.tf
* Outputs.tf
* Terraform.tfvars>>>if you are writing modules no need of tfvars file.

(ec2 resource contains connection and provisioners to use)

**Terraform interpolation:**

* Variable interpolation

Var.variable

* Resource interpolation

“${type.name.attribute}”

* Module interpolation

“${module.modulename.output}”

* Datasource interpolation

“${data.type.name.attribute}”

**Terraform Funtions:**

* Element>>
* Concat
* Length
* Index
* Merge
* Lookups
* File

Reference: <https://www.terraform.io/docs/configuration/functions.html>

**Terraform conditionals:**

As terraform is declarative language it becomes difficult to give certain loops as it is not a programming laungauge so terraform has given certain conditionals so that you can create some if and for statements:

* Using count for for loop
* Using ? for if conditions

**Examples:**

**For condition example:**

resource "aws\_instance" "example" {  
 count = 3  
 ami = "ami-2d39803a"  
 instance\_type = "t2.micro"

tags {  
 Name = "example-${count.index}"  
 }  
}

 “count.index” to get the index of each “iteration” in the “loop”:

**If condition can be done as following:**

resource "aws\_instance" "web" {

subnet **=** "${var.env == "production" ? var.prod\_subnet : var.dev\_subnet}"

}

Reference: <https://www.terraform.io/docs/configuration-0-11/interpolation.html#conditionals>

Terraform plan>>>you can send the output to a file

and use that file while applying here you no need to give variables etc.

**Backend:** multiple people executing the script it should create the same resource.

To achieve this we configure backend in which we will store the state file.

There are many backends available.

Whenever two people are working on the same module or tf file it should say that you can’t do the plan or apply because someone else is using the module or tf file.

**Backend types:**

[Enhanced Backends](https://www.terraform.io/docs/backends/types/remote.html#docs-backends-types-enhanced)

* [local](https://www.terraform.io/docs/backends/types/local.html)
* [remote](https://www.terraform.io/docs/backends/types/remote.html)

[Standard Backends](https://www.terraform.io/docs/backends/types/remote.html#docs-backends-types-standard)

* [artifactory](https://www.terraform.io/docs/backends/types/artifactory.html)
* [azurerm](https://www.terraform.io/docs/backends/types/azurerm.html)
* [consul](https://www.terraform.io/docs/backends/types/consul.html)
* [etcd](https://www.terraform.io/docs/backends/types/etcd.html)
* [etcdv3](https://www.terraform.io/docs/backends/types/etcdv3.html)
* [gcs](https://www.terraform.io/docs/backends/types/gcs.html)
* [http](https://www.terraform.io/docs/backends/types/http.html)
* [manta](https://www.terraform.io/docs/backends/types/manta.html)
* [oss](https://www.terraform.io/docs/backends/types/oss.html)
* [pg](https://www.terraform.io/docs/backends/types/pg.html)
* [s3](https://www.terraform.io/docs/backends/types/s3.html)
* [swift](https://www.terraform.io/docs/backends/types/swift.html)
* [terraform enterprise](https://www.terraform.io/docs/backends/types/terraform-enterprise.html)

**Backend configuration:**

* Complete configation within backend section in tf file
* Partial configuration to avoid some sensitive info we pass secrets from command line

**Examples:**

terraform {

backend "consul" {}

}

A backend configuration file has the contents of the backend block as top-level attributes, without the need to wrap it in another terraform or backend block:

address **=** "demo.consul.io"

path **=** "example\_app/terraform\_state"

scheme **=** "https"

The same settings can alternatively be specified on the command line as follows:

$ terraform init \

-backend-config="address=demo.consul.io" \

-backend-config="path=example\_app/terraform\_state" \

-backend-config="scheme=https"

**Reference:** <https://www.terraform.io/docs/backends/config.html>

Azure rm refecrenc: <https://docs.microsoft.com/en-us/azure/terraform/terraform-backend>

**Remote state:** Retrieves state data from a [Terraform backend](https://www.terraform.io/docs/backends/index.html). This allows you to use the root-level outputs of one or more Terraform configurations as input data for another configuration.

data "terraform\_remote\_state" "vpc" {

backend **=** "atlas"

config **=** {

name **=** "hashicorp/vpc-prod"

}

}

# Terraform >= 0.12

resource "aws\_instance" "foo" {

# ...

subnet\_id **=** data.**terraform\_remote\_state**.**vpc**.**outputs**.**subnet\_id**

}

**Scenario:**

(Assume two users in two terminalsand do plan and apply with same tf files.)

**Common arguments to resources:**

* [depends\_on, for specifying hidden dependencies](https://www.terraform.io/docs/configuration/resources.html#depends_on-explicit-resource-dependencies)
* [count, for creating multiple resource instances](https://www.terraform.io/docs/configuration/resources.html#count-multiple-resource-instances)
* [provider, for selecting a non-default provider configuration](https://www.terraform.io/docs/configuration/resources.html#provider-selecting-a-non-default-provider-configuration)
* [lifecycle, for lifecycle customizations](https://www.terraform.io/docs/configuration/resources.html#lifecycle-lifecycle-customizations)
* [provisioner and connection, for taking extra actions after resource creation](https://www.terraform.io/docs/configuration/resources.html#provisioner-and-connection-resource-provisioners)

reference: <https://www.terraform.io/docs/configuration/resources.html>

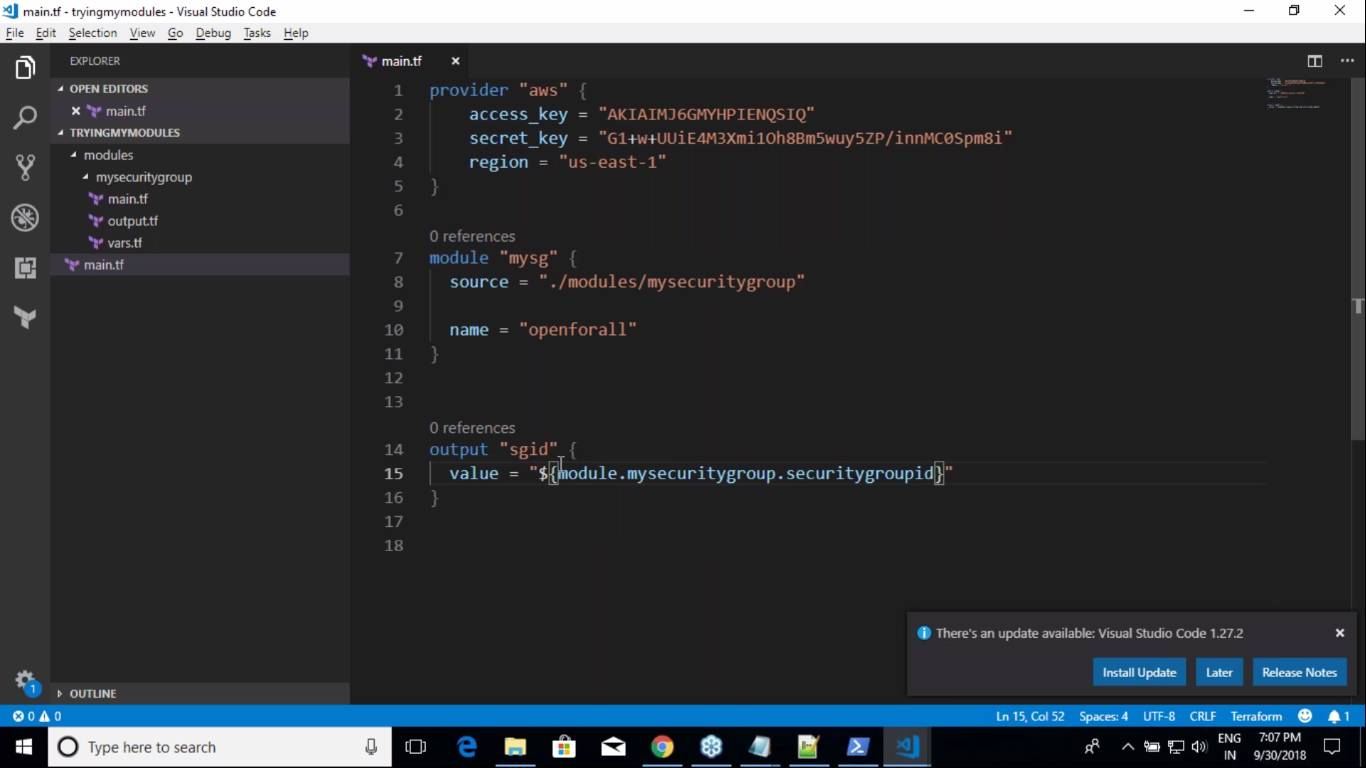
**Count argument: For every resouce**

* you define multiple resources using count argument.
* If you want to turn off a particular resource or module you can use count.

**Terraform Modules:**

* If our company has two teams let us say hrms and ecommerce
* You have provided terraform templates to hrms nad ecommerce teams
* What happens if you want to fix a bug then you should fix in both the places.
* Instead of that if you have solution which you can reuse for both teams that is exactly what modules are.
* So here rather than giving templates to teams you will give same terraform module to multiple teams. If you fixed any bug in that module just inform teams there is a new version you can take it.
* Terraform Module registry: you can find the reusable modules here. You can find github location here you can find examples folder which tells how to use the module.
* Modules can be store in terraform registry or [Terraform Enterprise's private module registry](https://www.terraform.io/docs/enterprise/registry/index.html) or git source or in local systems

**Key-Points:**

* Parameterize the terraform scripts more.
* Variables of standard modules becomes inputs to the root module
* Outputs of standard module becomes outputs of the root module
* Standard modules are refered in root module using “source” argument.
* You can access one module resource attribute(output) in another module via outputs i.e if you define outputs for that particular attribute then only you can access that attribute in other module with the syntax: “${module.modulename.outputname}”
* For the same module also you can use the same syntax.
* You can deploy into multi cloud also using providers.
* Once you are using modules you will not define the providers in standard module you will define the providers in the root module.
* 

**Modules Directory Structure:**

* Main.tf>>>variables becomes inputs to the modules and values to the inputs comes from either hardcorded variables.tf file
* Variables.tf>>variables of resources becomes variables to modules with entirely different name you an configure because you will this variable as values to the inputs(varibales) in modules.
* Terraform.tfvars>>>define values to variables.tf file you can maintain different tfvars for different evinronment. Ex: dev.tfvars, uat.tfvars, prod.tfvars and you can call them while terraform apply.
* Provider.tf>>>>>instead of defining providers in modules tf files you can define providers in this file and you can reference this provider in main.tf file.

**Standard Modules or child module**

* Standard Modules are the place where you will define the resources in a specific structure.

**Root Module:**

* Where you will call the standard modules.

**Module sources:**

The module installer supports installation from a number of different source types, as listed below.

* [Local paths](https://www.terraform.io/docs/modules/sources.html#local-paths)
* [Terraform Registry](https://www.terraform.io/docs/modules/sources.html#terraform-registry)
* [GitHub](https://www.terraform.io/docs/modules/sources.html#github)
* [Bitbucket](https://www.terraform.io/docs/modules/sources.html#bitbucket)
* Generic [Git](https://www.terraform.io/docs/modules/sources.html#generic-git-repository), [Mercurial](https://www.terraform.io/docs/modules/sources.html#generic-mercurial-repository) repositories
* [HTTP URLs](https://www.terraform.io/docs/modules/sources.html#http-urls)
* [S3 buckets](https://www.terraform.io/docs/modules/sources.html#s3-bucket)

Reference: <https://www.terraform.io/docs/modules/sources.html>

**Complete Module structure:**

linode\_stackscripts/

├── main.tf

├── outputs.tf

├── secrets.tfvars

├── terraform

├── terraform.tfvars

├── variables.tf

└── modules/

    ├── linodes/

    │   ├── main.tf

    │   ├── variables.tf

    │   └── outputs.tf

    └── stackscripts/

        ├── main.tf

        ├── variables.tf

        └── outputs.tf

**Terraform plan action symbols:**

1. + : means adding a resource
2. - : means deleting a resource
3. +/- : destroy and recreate the resource (or) create before destroy if lifecycle is used.
4. ~ : updating