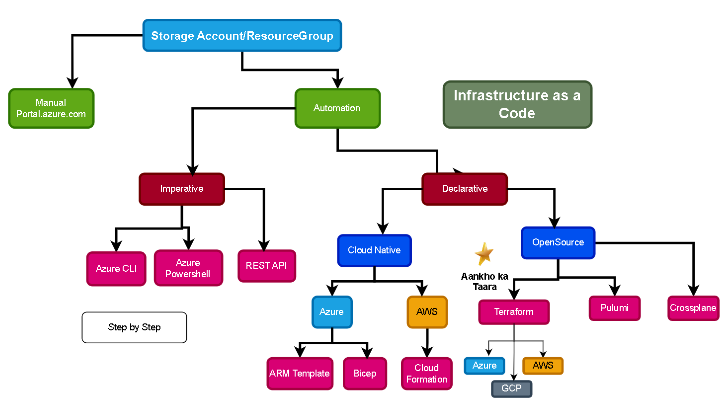


Terraform

Terraform is an open-source tool that helps you manage and provision infrastructure using code created by HashiCorp, it’s also known as Infrastructure as Code (IAC).

* IT using HCL (HashiCorp language) or optionally (JSON)
* IT support (**.tf** -terraform) format.



Download and installation steps of Terraform: -

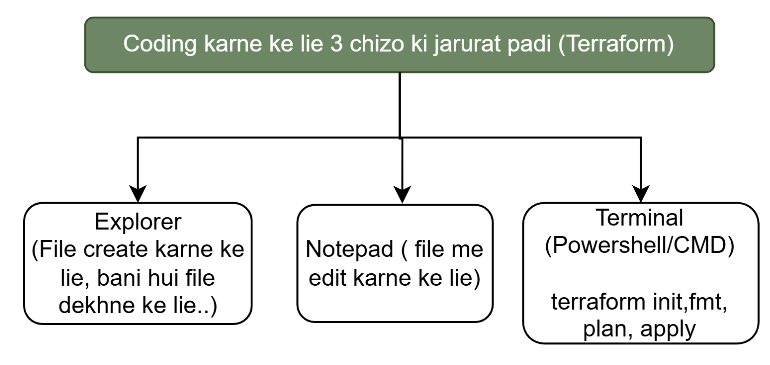
* <https://www.terraform.io/downloads.html> use this link to download terraform
* Unzip the downloaded file to extract terraform.exe
* Add a path for Ex
* Move terraform.exe to a directory of your choice (e.g., C:\Terraform).
* Open the Start Menu, search for “Environment Variables” and select "Edit the system environment variables".
* In the System Properties window, click on **Environment Variables**.
* Under **System variables**, select **Path** and click **Edit...**.
* Click **New** and add the path to the directory containing terraform.exe (e.g., C:\Terraform).

Verify that Terraform has been installed or not: -

* Open cmd or powershell >> terraform - -version >> version will be shown

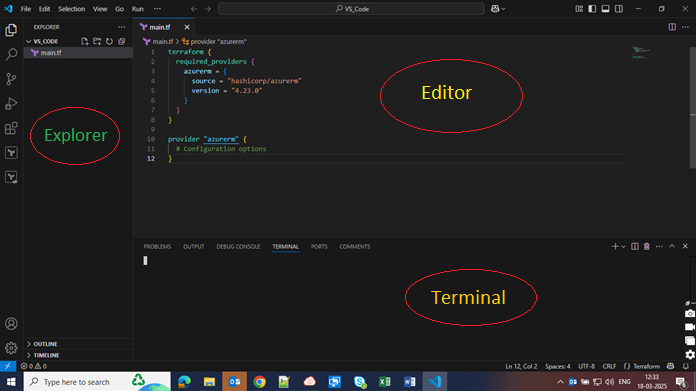
Vs code

Visual Studio Code (VSCode) is a powerful and lightweight code editor that integrates several features to enhance the development experience. One of its key strengths is how it integrates the **Explorer**, **Editor**, and **Terminal** into a unified workspace. Here’s how these components work together:



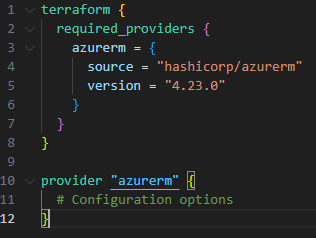
**Install Visual Studio Code & Add extensions:**

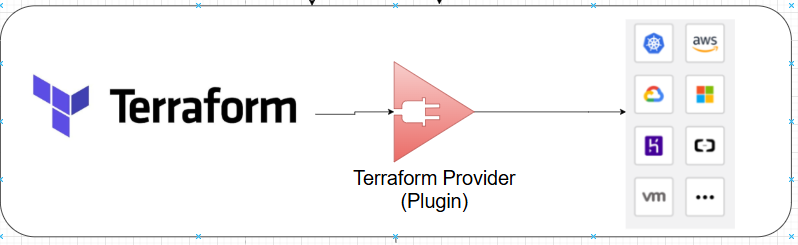
* Download <https://code.visualstudio.com/download> >> install
* Extension >> HashiCorp Terraform >> install
* Extension >> githubcopilot >> install



**Terraform Providers ()**

* **What is a Terraform Provider?**
  + A **provider** is a plugin that enables Terraform to interact with APIs from cloud providers or other services like AWS, Azure, Google Cloud, etc.
  + Providers manage resources like creating, modifying, and deleting them.
* **How Terraform Providers Work**
  + You specify the provider in your configuration files, and it handles the communication with the respective API.



Terraform uses **providers** to interact with various cloud platforms, SaaS services, and other APIs. A **provider** is a plugin that Terraform uses to manage infrastructure resources on a specific platform like AWS, Azure, GCP, Kubernetes, etc.

Each provider communicates with the respective cloud provider’s API to perform operations like creating, updating, and deleting resources.

#### ****Common Cloud Providers and Their APIs****

1. **AWS Provider (hashicorp/aws)**
   * Uses AWS APIs like EC2, S3, IAM, Lambda, etc.
   * Authentication via **Access Key, Secret Key, IAM Roles**
   * Example:
   * provider "aws" {
   * region = "us-east-1"

}

1. **Azure Provider (hashicorp/azurerm)**
   * Uses Azure REST APIs to manage resources like VMs, Storage, Networking, etc.
   * Authentication via **Service Principal, Managed Identity**
   * Example:
   * provider "azurerm" {
   * features {}

}

1. **Google Cloud Provider (hashicorp/google)**
   * Uses Google Cloud APIs for Compute Engine, Cloud Storage, Kubernetes, etc.
   * Authentication via **Service Account Key JSON**
   * Example:
   * provider "google" {
   * project = "my-gcp-project"
   * region = "us-central1"

}

1. **Kubernetes Provider (hashicorp/kubernetes)**
   * Uses Kubernetes API Server to manage resources like Pods, Deployments, Services, etc.
   * Authentication via **kubeconfig**
   * Example:
   * provider "kubernetes" {
   * config\_path = "~/.kube/config"

}

### ****What Does****terraform init****Do?****

terraform init initializes a Terraform working directory. It performs the following tasks:

1. **Downloads Providers**
   * Fetches provider plugins specified in the provider block.
   * Downloads them from the Terraform Registry and places them in the .terraform directory.
2. **Configures Backend**
   * Sets up the backend (e.g., local, S3, Azure Storage, GCS) to store the Terraform state.
3. **Checks and Verifies Modules**
   * Downloads external Terraform modules specified in the configuration.
4. **Prepares for Execution**
   * Ensures everything is properly set up for running Terraform commands like plan and apply.

### ****What is****.terraform****Directory?****

* It is a hidden directory created after running terraform init.
* Contains provider binaries, modules, and backend-related files.
* Helps Terraform execute commands faster without redownloading providers.

**Example Contents of .terraform Directory:**

.terraform/

│── providers/

│ ├── registry.terraform.io/

│ │ ├── hashicorp/

│ │ │ ├── aws/

│ │ │ │ ├── 5.0.0/

│ │ │ │ │ ├── terraform-provider-aws\_v5.0.0\_x5

│── modules/

│── plugin-cache/

### ****What is****terraform.lock.hcl****?****

* It is an auto-generated lock file that ensures Terraform always uses the same provider versions.
* Prevents accidental upgrades that might break infrastructure.
* Stored in the root directory of the Terraform project.

Terraform code is basically form with (Block by Block)

Block

Block

What is Terraform Block

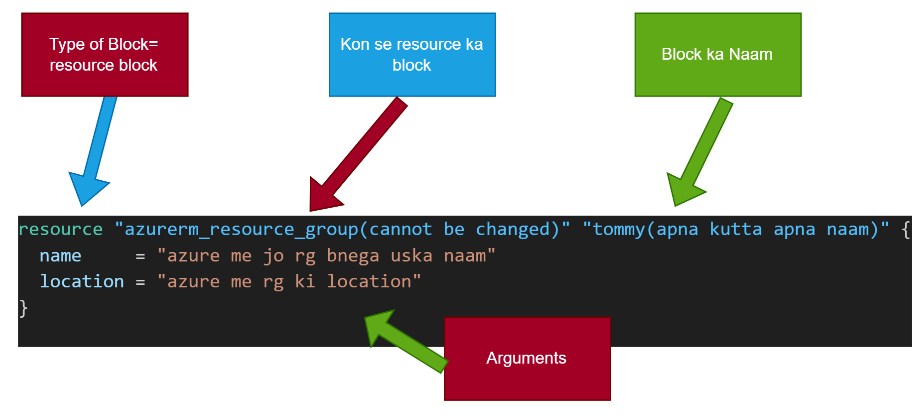
In Terraform, a "block" is a fundamental building block of configuration syntax, acting as a container for other attributes and blocks, used to define resources, data sources, providers, and more.

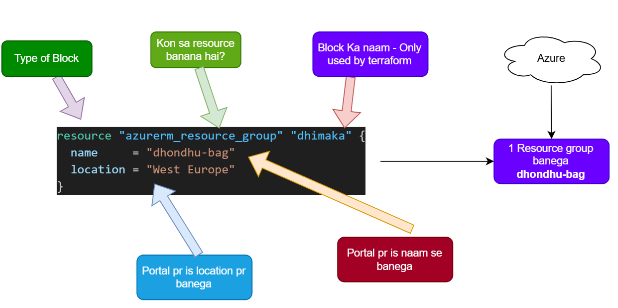
Eg. Terraform {} <https://developer.hashicorp.com/terraform/language/terraform>

<https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs>

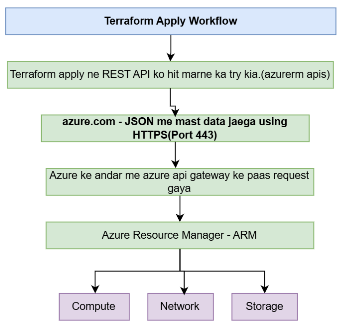
Terraform provider file ko download kr ke .terraform me rkh deta hai

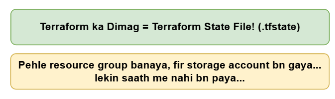
Terraform init



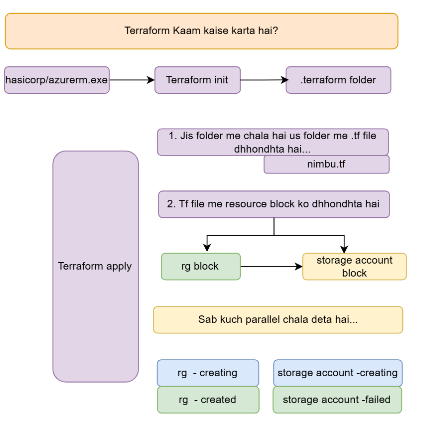


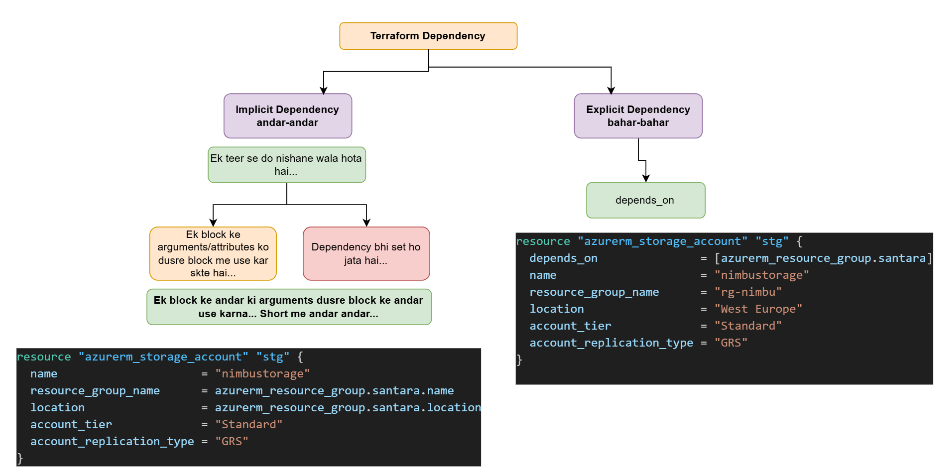
**Terraform workflow :-**











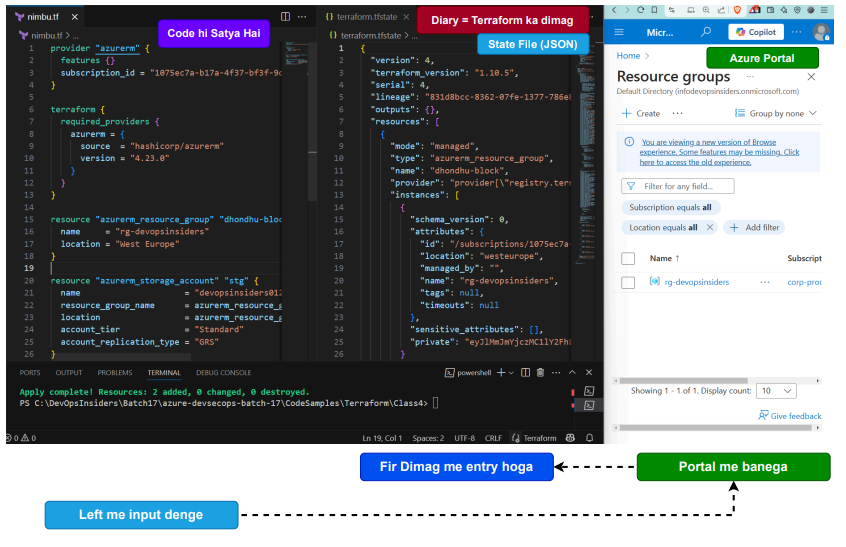
Note: Depends\_on is a meta arguments

Terraform state file

Terraform state file will be form after running terraform apply command.

(This is also called the brain of the terraform)

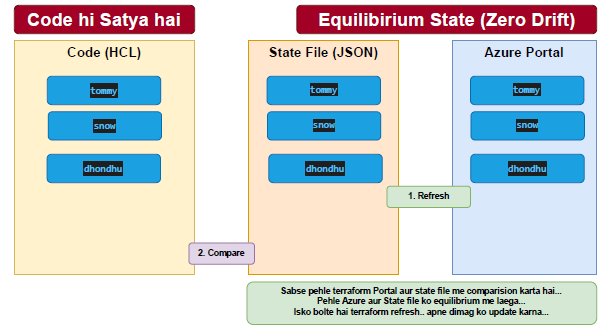
* In given below scenario when we run the terraform apply command in terminal section after adding some blocks in configuration file, terraform state file will be form
* When we adding any blocks inside the configuration file (.tf file), after terraform apply command it will scan the whole directory and and then scan (nimbu.tf) file and found that there is two resources which will be added inside the portal.
* After adding in the portal, it will save the details inside the (terraform.tfstate) file in JSON format.



Equilibrium or Zero drift condition

When we run terraform plan command after adding the resource in configuration file (.tf)

What will be happen:



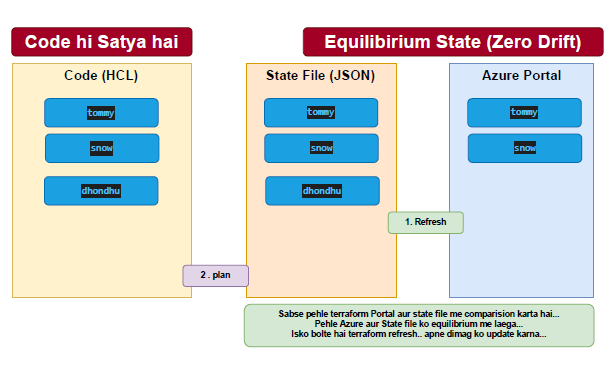
* After applying terraform plan command it will compare the (.tf file) means configuration file and state file (terraform.tfstate) and found that there is two resource block is present inside the configuration file and and not available inside the state file so it will understand and prepare plan that two resources will be added.
* Terraform apply = Terraform refresh + Terraform plan + Terraform apply
* After Terraform apply command it will refreshing the state file and Azure portal and makes both in equilibrium state or zero drift state, and compare again the configuration file and state file and prepare the plan and same Terraform apply command will add two resources inside the Azure portal
* After adding two resources in the Azure portal it will add the relevant information in state file in the JSON format.
* This is complete workflow.

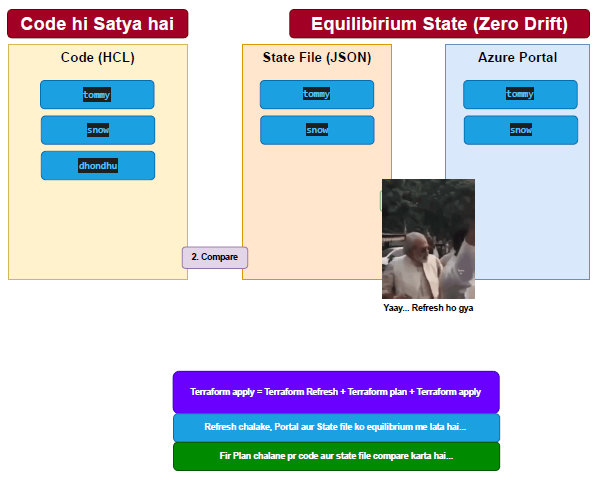
In below scenario there are three resource block is present in configuration file and state file,

If suppose someone deleted the resource from the portal or from the configuration file then what will be happens.

Terraform Refresh:- It will refresh the both side portal and state file and found that only two resource is present inside the portal and three resource present inside the state file then refresh command will remove the one block from the state file which is not found inside the portal.

So this is called Equilibrium condition or Zero drift condition

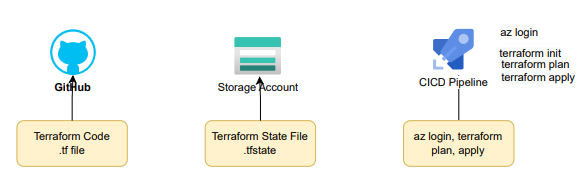


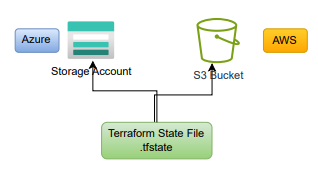
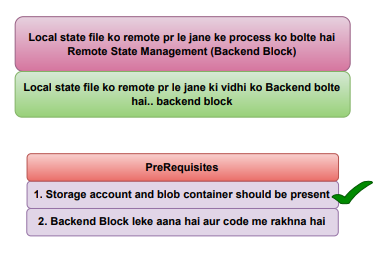


Terraform Remote Backend

1. When Terraform state file was stored in local (own laptop/another laptop), suppose he will left the organization or going on the leave so how organization and terraform will run smoothly
2. In any team there will be multiple teammates then how it could be possible to run terraform using the same state file

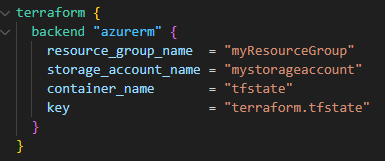
This was the problem to store the state file in local computer, so we have implemented that the state file will be stored inside the somewhere in remote location so it will be accessible to all teammates and will be more secured.



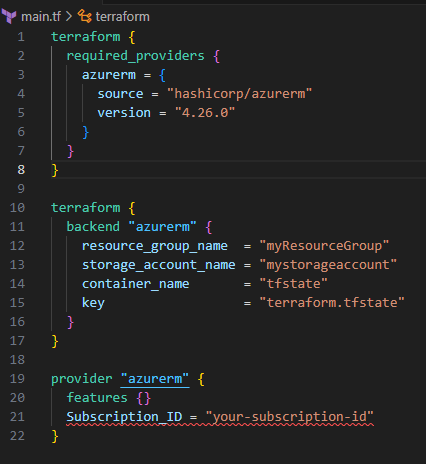
 

Google search:- azurerm terraform backend block configuration

<https://developer.hashicorp.com/terraform/language/backend>



After applying the terraform init command the state file will be stored in the remote backend (azure storage container).



Note: - we need to create storage account and container manually or by using azure cli after that we need to give their name and values inside the backend block then the state file will go to that given path or name.

We can access the state file by using below command and also we can download the state file.

* Terraform state list (To check list of the resources)
* Terraform state show [copy and paste the list of the resource]
* Terraform state pull [which name you wanted to give the file] eg:- sumantra.tfstate
* Terraform state pull Sumantra.tfstate

Terraform state Locking

Terraform Variables (Code)

Terraform variables Using Optional attributes validation block-condition

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Variables.tf\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

variable "rg\_name" {

type = string

validation {

condition = contains (["guru", "kalu", "pilu"], var.rg\_name)

error\_message = "name must be one of the following: guru, kalu, pilu"

}

}

---------------------------------------------main.tf-----------------------------------------------------------

resource "azurerm\_resource\_group" "rg01" {

name = "va.rg\_name"

location = "central india"

}

\*\*\*\*\*\*\*\*\*\*Default attributes\*\*\*\*\*\*\*\*\*\*\*\*usecase\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*variable.tf\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

variable "rg\_name1" {

description = "Name of the resource group"

type = string

default = "rg\_p1"

}

variable "rg\_name2" {

description = "Name of the resource group"

type = string

default = "rg\_p2"

}

variable "rg\_name3" {

description = "Name of the resource group"

type = string

default = "rg\_p3"

}

variable "rg\_name4" {

description = "Name of the resource group"

type = string

default = "rg\_p4"

}

variable "rg\_name5" {

description = "Name of the resource group"

type = string

default = "rg\_p5"

}

-------------------------------------------main.tf----------------------------------------------------------

resource "azurerm\_resource\_group" "rg01" {

name = var.rg\_name1

location = "central india"

}

resource "azurerm\_resource\_group" "rg02" {

name = var.rg\_name2

location = "central india"

}

resource "azurerm\_resource\_group" "rg03" {

name = var.rg\_name3

location = "central india"

}

resource "azurerm\_resource\_group" "rg04" {

name = var.rg\_name4

location = "central india"

}

resource "azurerm\_resource\_group" "rg05" {

name = var.rg\_name5

location = "central india"

}

-----------------------------------------Terrarom.tfvars------------------------------------------------------

rg\_name1 = "rg-rondu1" # passing the value in string format

rg\_name2 = "rg-rondu2" # passing the value in string format

rg\_name3 = "rg-rondu3" # passing the value in string format

rg\_name4 = "rg-rondu4" # passing the value in string format

rg\_name5 = "rg-rondu5" # passing the value in string format

=================================================================

In all above scenario there are the issue that we are copying the resource block multiple time so that to avoid this we will use the "List\_atribute"

=================================================================

-------------------------------------------Variable.tf----------------------------------------------------------

variable "rg\_name" {

description = "Name of the resource group"

type = list(string)

default = ["rg-rondu1", "rg-rondu2", "rg-rondu3", "rg-rondu4", "rg-rondu5"] #List me attributes paas kiye hai

}

------------------------------------------------main.tf--------------------------------------------------------

resource "azurerm\_resource\_group" "rg" {

count = 5

name = var.rg\_name[count. index]

location = "central india"

}

----------------------------------------------------------------------------------------------------------------

Meta argument = 1. Count, 2. For\_each

----------------------------------------------------------------------------------------------------------------

-----------main.tf---------simple structure to understand the meta argument [count]--------------------

resource "azurerm\_resource\_group" "rg" {

count = 5

name = "rg-dhondhu"

location = "central india"

}

#it will create the same resource group with the name of same rg-name but it will add indexing from [0 1 2 3 4 5]

resource "azurerm\_resource\_group" "rg" {

count = 5

name = count.index

location = "central india"

}

#Now it has start the count indexing from [0 1 2 3 4 5] inside the rg-name

------------------------------------------main.tf-------------------------------------------------------------

resource "azurerm\_resource\_group" "rg" {

count = 5

name = "rg-dhondhu${count.index}" # String interpolation

location = "central india"

}

=========================Terraform\_variables\_Count======================

count + List = we will see that what is the issue with this then we will switch on

For-each + List = will use and see what is the problem in this and then we will jump on the next

For-each + MAP = This is the best and final solution

----------------------------------------------------------------------------------------------------------------

Agenda = 5RG required in same location then we will create with below use

----------------------------------------------------------------------------------------------------------------

----------------------------------------------main.tf----------------------------------------------------------

resource "azurerm\_resource\_group" "rg" {

count = 0 # No changes -- No resource group will be created

name = "rg-dhondhu"

location = "central india"

}

----------------------------------------------------------------------------------------------------------------

-----------------------------------------------main.tf---------------------------------------------------------

resource "azurerm\_resource\_group" "rg" {

count = 5

name = "rg-dhondhu-count.index"

location = "central india"

}

#It will create the five resource group but the problem is it will create the name of the RG is rg-dhondhu-count.index so we will use string intercolation

-------------------------------------------

resource "azurerm\_resource\_group" "rg" {

count = 5

name = "rg-dhondhu-${count.index}" # String interpolation

location = "central india"

}

# Now it will create the resources with the name of rg-dhondhu-0, rg-dhondhu-1, rg-dhondhu-2, rg-dhondhu-3, rg-dhondhu-4

# but this is the problem that name of the resource group came in form of o 1 2 3 4 5, so to resolve this we will see the next concept

----------------------------------------------------------------------------------------------------------------

--------------------------------------------main.tf------------------------------------------------------------

variable "rg\_name" {}

type = string

default = "rg-apple"

resource "azurerm\_resource\_group" "rg" {

count = 4

name = "$(var.rg-\_name)-${count.index}" # String interpolation # we are calling here varible name

location = "central india"

}

# RG will be created rg-apple-0, rg-apple-1, rg-apple-2, rg-apple-3,

----------------------------------------------------------------------------------------------------------------

Variable = 1--Premitive = String, Number, bool = Singular = koi bhi ek value rkhen ke kaam aata hai

= 2--Advance = List/Array, Map = List is a collection of element of same types = multiple chizo ko store kr payegi

List/ Array = []

biwi\_ke\_gahne = ["rg-jhumka", "rg-kangan", "rg-mangalsutra", "rg-payal", "rg-baali"]

----------------------------------------------main.tf----------------------------------------------------------

variable "biwi\_ke\_gahne" {}

type = List(string)

default = ["rg-jhumka", "rg-kangan", "rg-mangalsutra", "rg-payal", "rg-baali"]

#0 1 2 3 4 5

resource "azurerm\_resource\_group" "rg" {

count = 4

name = var.biwi\_ke\_gahne[3] # var.biwi\_ke\_gahne is a list(string)\_var.biwi\_ke\_gehne it will through the error but if we use [3] then it will identify the value of index variable

location = "central india

# rg-payal will be created using [3]

# but here is the problem that all resource groups are not creating which is mentioned in list format then we will user count.index

-----------------------------------------------variabl.tf--------------------------------------------------------

variable "biwi\_ke\_gahne" {}

type = List(string)

default = ["rg-jhumka", "rg-kangan", "rg-mangalsutra", "rg-payal", "rg-baali"]

resource "azurerm\_resource\_group" "rg" {

count = 5

name = var.biwi\_ke\_gahne[count.index] # count.index will create the copies as per the count value means 5 copies will be created

location = "central india

# rg-jhumka, rg-kangan, rg-mangalsutra, rg-payal, rg-baali

----------------------------------------------------------------------------------------------------------------

---------------------------------------------terraform.tfvars--------------------------------------------------

rg\_name = ["rg-ram", "rg-shyam", "rg-kalu", "rg-pilu", "rg-nilu"]

#0 1 2 3 4

----------------------variabl.tf---------------------------------------------------

variable "rg\_name" {

description = "Name of the resource group"

type = list(string)

default = ["rg-rondu1", "rg-rondu2", "rg-rondu3", "rg-rondu4", "rg-rondu5"]

}

-------------------------------------------main.tf-------------------------------------------------------------

resource "azurerm\_resource\_group" "rg" {

count = 5 # count = length(var.rg\_name)

name = var.rg\_name[count.index]

location = "central india"

}

# rg-ram, rg-shyam, rg-kalu, rg-pilu, rg-nilu

#IF we deleted one value from terraform.tfvars then what will be happend

default = ["rg-rondu1",

"rg-rondu2",

#"rg-rondu3",

"rg-rondu4",

"rg-rondu5"]

**=========================(List +For-each)=======================**

------------------terraform.tfvars-----------------------

rg\_name = ["rg-01", "rg-02", "rg-03", "rg-04", "rg-05"]

------------------variabl.tf-----------------------------

variable "rg\_name" {

description = "Name of the resource group"

type = list(string)

default = ["rg-rondu1", "rg-rondu2", "rg-rondu3", "rg-rondu4", "rg-rondu5"]

}

------------------main.tf--------------------------------

resource "azurerm\_resource\_group" "rg" {

for\_each = toset (var.rg\_name)

name = each.key

location = "central india"

}

# rg-01, rg-02, rg-03, rg-04, rg-05 will be created

----------------------------------------------------------------------------------------------------------------

Multiple resources will be made in same location so this is the proble to resolve this proble introduced (for\_each + MAP)