Terraform:
=========
Terraform is a tool for IAC.

IAC --> Insfrastructure as a code
Terraform is an Open source tool.
Terraform is developed by HashiCorp.

Cloudformation is also an IAC tool.

CF vs Terraform:
--> CF is used/restricted only to Aws.
--> Terraform is used to provision on multiple cloud providers.

Terraform uses HCL language (HCL --> hashi corp language)

How to install Terraform:?

1) Download Terraform
https://www.terraform.io/downloads.html
2) Unzip the terraform package
Extract the downloaded zip file, after extracting the zip file you can see terraform.exe file.
Extracted path can be Example
"C:\Users\devops\Downloads\terraform_0.12.23_windows_amd64"
3) Configure environment variables for terraform
This PC(MyComputer)—>properties —>advanced system settings->environment variables—>system variables—>path-edit->new
paste the path
"C:\Users\devops\Downloads\terraform_0.12.23_windows_amd64"
Click on Ok.
4) Verify terraform version
Open gitbash and enter
> terraform version
  Terraform v0.12.23

How a terraform configuration looks like?

```
<block> <resource_type> {
 option1
 option2
}
```

.tf --> extension for the terrform resource files.


```
resource "local_file" "my_pet" {
filename = "pets.txt"
content = "I love pets!"
}
```

Resource --> block

local --> provider
type --> type of resource
my_pet --> name for terraform
filename and content --> attributes used for the resource

Terraform Provider --> is a complete package of api calls to communicate with our resource.
idempotent

3 types of providers are available in terraform:
1) official --> provided by terraform
2) partner --> provided by third party vendors
3) community --> individual who can create.

Configuration Directory:
========================
Main.tf   -->main configuration file containing resource definition
variables.tf --> contains varibles declaration
output.tf   --> contains outputs from resources
provider.tf   --> contains the provider definition

Terraform mutable vs immutable infrastructure:
==
Terraform as a IAC tool uses immutable infrastructure stratgey.
Immutable means deleting the older infra and creating a newer one with new update.
Mutable means using the existing infra and updating the system with newer version.

Lifecycle rules:
================
create_before_destroy
prevent_destroy
ignore_changes

Variables:
==========
```
variable "filename" {
    default =
    type = string
    description = This is optional (Used to user understanding)
}
```

Type	Example
String	I love pets
Number	1
bool	true/false
any	default value
list	["cat","dog"]
map	pet1= cat pet2=dog
object	complex data structure
tuple	complex data structure

Using of variables:
==========
1) By using varibles.tf file
2) By using interactive mode (This will get activated if we dont pass default value in variable.tf file)
3) Command line flags
--> terraform apply - var "filename=/root/pets.txt" -var "prefix=MR"
4) Environment variables
--> export TF_VAR_filename="/root.pets.txt"
--> export TF_VAR_prefix= "MR"
--> Set-Item -Path env:TF_VAR_filename -Value 'wild.txt'
terraform apply
5) varibale definition file (Should be end with terraform.tfvars/terraform.tfvars.json)
--> for automatically loaded file name *.auto.tfvars/*.auto.tfvars.json
--> if we are saving the file with other name like varible.tfvars then we need to pass this in CLI
--> terraform apply -var-file varibale.tfvars

Varible definition precedence:
==========
If we use multiple ways to define varibles for the same file then terraform uses varible definition precedence

Example:
--> main.tf

```
resource local_file pet {
filename = var.filename
}
```

--> variable.tf

```
variable filename {
type = string
}
```

--> export TF_VAR_filename="/root/cat.txt"
--> Set-Item -Path env:TF_VAR_filename -Value 'wild.txt'
--> terraform.tfvars
filename = "/root/pets.txt"
--> varible.auto.tfvars
filename = "/root/mypet.txt"

--> terraforma apply -var "filename=/root/best-pet.txt"

Precedence order:
==========

in the above example we have passed all the possible varibles,which will terraform laod first and which will override ?

```
Order           Option
1              Environment variables
2              Terraform.tfvars
3              *.auto.tfvars(alphabetical order)
4               -var or -var-file (Command line flags)
```

Resource Attribute reference:
==========
If i want to link two rerouces together by using resource attributes.

main.tf
======
```
resource "local_file" "pet" {
filename = "/root/pets.txt"
content = "My cat is MR.Cat"
}
resource "random_pet" "mypet" {
prefix = "MR"
separator = "."
length = "1"
}
```

When we execute terraform apply it will create random id with pet name, now i want to add this pet name in my content file (using output of one resource as input for another resource).

main.tf
======
```
resource "local_file" "pet" {
filename = "/root/pets.txt"
content = "My cat is ${random_pet.mypet.id}"     (random_pet = resource type,mypet = resource name, id = attribute)
}
resource "random_pet" "mypet" {
prefix = "MR"
separator = "."
length = "1"
}
```

Output variables:
==========
These are used to display the output of the resources.
Ex:
```
resource "random_pet" "mypet" {
prefix = "MR"
separator = "."
length = "1"
}

output my-pet {
value = random_pet.my-pet.id
description = optional name
}
```

when we use terraform apply we can see the id as output.
we can use terraform output command to see the output of the resource.

Terraform state:
================
Terraform state file will have the complete record of the infra created by terraform.
State file is considered as a blue print of all the resources terraform manages.
terraform.tfstate will be the name of the file and this will created only after using terraform apply command.

When we excute terraform apply then terraform will check for the state file config and main.tf configuration and make the changes.
If both the files are in sync and we are again trying to execute terraform apply then terraform will not make the changes but show "Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed."

Each resource created by terraform will have the unique ID.
State files also capture the Metadata of the configuration file.
State file will helps for better performance because of the cache of the data.
state file benifits in collborating with different team members.
State files should be shared in the remote backend place so that team can access the state file.
State files also store the sensitive data so not recommended to store in public repo's like github,gitlab.
Terraform state is a json format file,never try to edit the state file manually.

Version Constraints:
====================
Changing in terraform providers version may get us into incompatability issues.
By default terraform will always try to downlaod the latest version of provider available on registry.

To make sure to use the specific version provider we can add the provider block in configuration.

Example:
=======
```
terraform {
  required_providers {
    local = {
      source = "hashicorp/local"
      version = "2.3.0"
    }
  }
}

resource "local_file" "my-pet" {
```

```
filename = "pets.txt"
content = "I love cats!"
}
=========
version = "2.3.0" --> download the exact version
version = "!=2.3.0" --> will not use the mentioned version
version = "< 2.3.0"  --> lesses than the mention version
version = "> 2.3.0"  --> greater than the given version
version = "~> 2.3.0"  --> specific version or higher version.

Data sources:
===========
Apart from terraform we have multiple other tools where the infra can be
created.
Ex: ansible,salt,puppet,bash script,manual process.

Data sources are used to read the content of the infrastructure

for example if we want terraform to read the content of the file which
has been created by any other tool.

create a file called in dogs.txt in the same terrafrom working directory.

main.tf
======

resource "local_file" "my-pet" {
filename = "pets.txt"
content = data.local_file.dog.content
}
data "local_file" "dog" {
filename = "dogs.txt"
}


Difference between resources and data ?

Resources starts with keyword resource
resource are used to create,modify,delete the infra

Data source start with keyword data.
data sources are used to read the infrastructure.

Meta-Arguments:
==============
Meta arguments are used if we want to create multiple resources.
Meta arguments can be used within any resource block to change the
behaviour of the resources.
Examples for meta arguments:

1) Depends_on
2) lifecycle rules
3) Count
4) For_each
```

Example of count:
==========
If we use count as 3 then it will create 3 files with
pet[0],pet[1],pet[2]

```
resource "local_file" "my-pet" {
filename = "pets.txt"
content = "I love cats!"
count = 3
}
```

This is not the idela way to use because these are getting replaced.

```
resource "local_file" "my-pet" {
filename = var.filename[count.index]
content = "I love cats!"
count = 3
}
```

variables.tf

```
variable "filename" {
default = [
"pets.txt"
"cats.txt"
"dogs.txt"
]
}
```

Still we have problem in the above configuration,if in future the list of varibles then we need to change the count value manually.
to avoid this we can use the inbuilt lenth function in terraform.

```
resource "local_file" "my-pet" {
filename = var.filename[count.index]
content = "I love cats!"
count = lenght(var.filename)
}
```

variables.tf

```
variable "filename" {
default = [
"pets.txt"
"cats.txt"
"dogs.txt"
]
}
```

But when we want to update/destroy any one file then we will see un wanted results in count as count will store the output in list and works on index number.

to overcome the issue we have for_each meta argument.

Example of for_each:
====================
main.tf:

```
resource "local_file" "pet" {
filename = each.value
for_each = var.filename
}
```

variables.tf
============

```
variable "filename" {
type=set(string)    --> list type will throw error for_each argument.
default = [
"pets.txt"
"cats.txt"
"dogs.txt"
]
}
```

or if you want to use list varibale then we can change the main.tf with toset inbuilt function.

main.tf:

```
resource "local_file" "pet" {
filename = each.value
for_each = toset(var.filename)
}
```

variables.tf
============

```
variable "filename" {
default = [
"pets.txt"
"cats.txt"
"dogs.txt"
]
}
```

Count will store the output as list and identified based on indexnumber
foreach store the output as map and identified based on filename.

Terraform with AWS:
==================
--> first we need to create a secret key and access key and configure then in laptop.

Example script to create aws iam:
==========

```
resource "aws_iam_user" "Admin-user" {
name = "lucy"
tags = {
  "description" = "Technical Team Lead"
}
}
```

Example script to create iam user with policy attached to the user:
==========

```
resource "aws_iam_user" "Admin-user" {
  name = "lucy"
  tags = {
    "description" = "Technical Team Lead"
  }
}
resource "aws_iam_policy" "adminuser" {
  name   = "AdminUsers"
  policy = <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "1234567890",
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
EOF
}

resource "aws_iam_user_policy_attachment" "lucy-admin-access" {
  user       = aws_iam_user.Admin-user.name
  policy_arn = aws_iam_policy.adminuser.arn
}
```

In the above example we have added the json policy using "heredoc syntax" and delimeters "EOF --> End of file" inside the main.tf.

we can also use them by saving the template in seperate file and call that file in our main.tf.

Example:
=======
admin-policy.json
==========

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "1234567890",
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

main.tf:
======

```
resource "aws_iam_user" "Admin-user" {
  name = "lucy"
  tags = {
    "description" = "Technical Team Lead"
  }
}
resource "aws_iam_policy" "adminuser" {
  name   = "AdminUsers"
  policy = file("admin-policy.json")
}

resource "aws_iam_user_policy_attachment" "lucy-admin-access" {
  user       = aws_iam_user.Admin-user.name
  policy_arn = aws_iam_policy.adminuser.arn
}
```