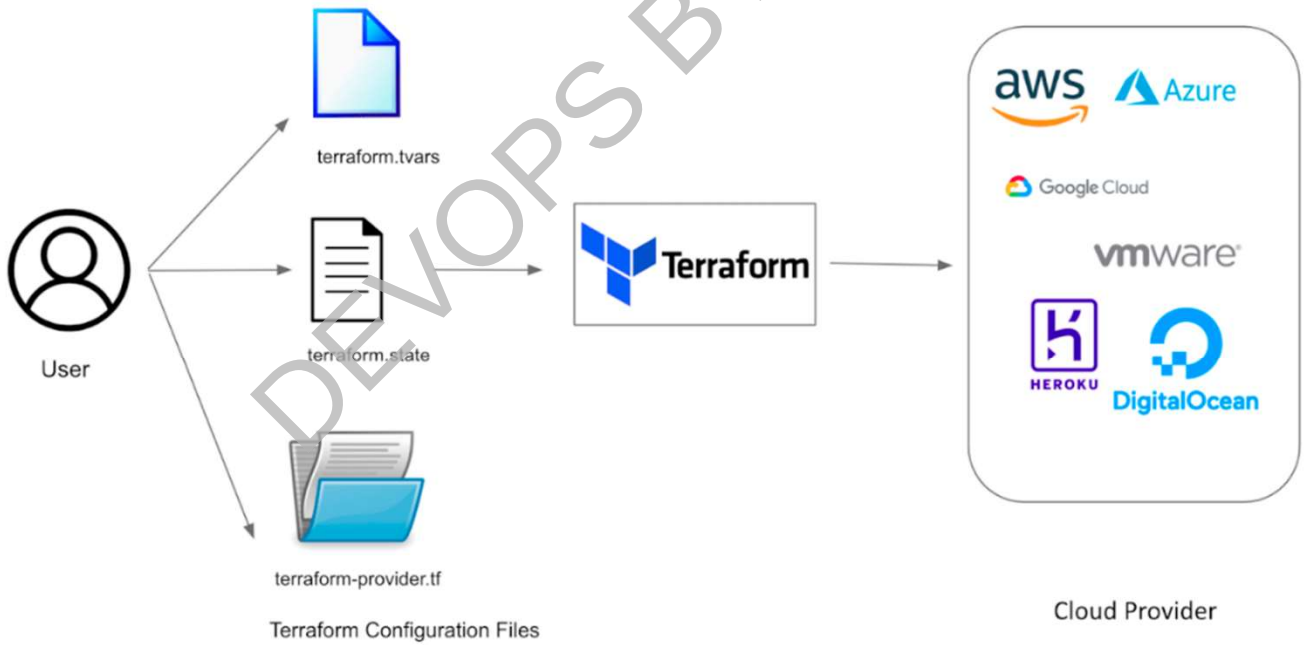
TERRAFORM:

Terraform is an open source “Infrastructure as Code” tool, created by HashiCorp.
A declarative coding tool, Terraform enables developers to use a high-level configuration language called HCL (HashiCorp Configuration Language) to describe the desired “end-state” cloud or on-premises infrastructure for running an application.
Terraform uses a simple syntax, can provision infrastructure across multiple clouds & On premises.
It is Cloud Agnostic it means the systems does not depends on single provider.



IAAC:

Infrastructure as a Code (IaaC) is the managing and provisioning of infrastructure through code instead of through manual processes.
With IaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations.
IaC allows you to meet the growing needs of infrastructure changes in a scalable and trackable manner.
The infrastructure terraform could handle low-level elements like networking, storage, compute instances, also high-level elements like SaaS features, DNS entries, etc.
It is famous for easy to use but not true for complex environments it is not easy.

ALTERNATIVES:

AWS -- > CFT (JSON/YAML)

AZURE -- > ARM TEMPLATES (JSON)

GCP -- > CLOUD DEPLOYMENT MANAGER (YAML/ PYTHON)

PULUMI -- (PYTHON, JS, C#, GO & TYPE SCRIPT)

ANSIBLE -- > (YAML)

PUPPET

CHEF

VAGRANT

CROSSPLANE

ADVANTAGES:

Readable code.

Dry run.

Importing of Resources is easy.

Creating of multiple resources.

Can create modules for repeatable code.

DIS ADVANTAGES:

Currently under development. Each month, we release a beta version.

There is no error handling

There is no way to roll back. As a result, we must delete everything and re-run code.

A few things are prohibited from import.

Bugs

TERRAFORM SETUP:

wget
https://releases.hashicorp.com/terraform/1.1.3/terraform_1.1.3_linux_amd64.zip
sudo apt-get install zip -y
Unzip terraform
mv terraform /usr/local/bin/
terraform version
cd ~
mkdir terraform & vim main.tf
write the basic code
Go to IAM andcreate a user called terraform and give both access give admin access.

TERRAFORM LIFECYCLE:

TERRAFORM INIT:

It initializes the provider, module version requirements, and backend configurations.

TERRAFORM PLAN:

Determines the state of all resources and compares them with real or existing infrastructure. It uses terraform state file data to compare and provider API to check.

TERRAFORM APPLY:

Executes the actions proposed in a Terraform plan.

TERRAFORM DESTROY:

It will destroy terraform-managed infrastructure or the existing enviornment

NOTE: We can use -auto-approve command for Apply and Destroy phases.

CREATING EC2:

```
provider "aws" {
 region          = "ap-south-1"
 access_key      = "AKIAWW7WL2JMJKCCMORC"
 secret_key      = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "example" {
  ami           = "ami-0af25d0df86db00c1"
  instance_type = "t2.micro"

  tags = {
    name = "web-server"
  }
}
```

TERRAFORM VARIABLE TYPES:

```
variable "<YOUR_VARIABLE_NAME>" {
  description  = "Instance type t2.micro"
  type         = string
  default      = "t2.micro"
}
```

Meaning full description

Ex - *string, number, bool, list, set, map..*

variable default value

Input Variables serve as parameters for a Terraform module, so users can customize behavior without editing the source.

Output Values are like return values for a Terraform module.

Local Values are a convenience feature for assigning a short name to an expression.

STRING: a sequence of Unicode characters representing some text, like "hello".

```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0767046d1677be5a0"
   instance_type =  var.instance_type

   tags = {
           Name = "Terraform EC2"
   }
}

variable "instance_type" {
   description = "Instance type t2.micro"
   type        = string
   default     = "t2.micro"
}
```

NUMBERS: The number type can represent both whole numbers and fractional values .

```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   count  =  var.instance_count

   tags = {
           Name = "Terraform EC2"
   }
}

variable "instance_count" {
   description = "Instance type count"
   type        = number
   default     = 2
}
```

BOOL: It is a boolean value, either true or false

```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   count  = 1
   associate_public_ip_address = var.enable_public_ip

   tags = {
           Name = "Terraform EC2"
   }
}

variable "enable_public_ip" {
   description = "Enable public IP"
   type        = bool
   default     = true
}
```

LIST/TUPLE: a sequence of values, like ["user1", "user2", "user3"] Identified by index

```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   count  = 1

   tags = {
           Name = "Terraform EC2"
   }
}

resource "aws_iam_user" "example" {
  count = length(var.user_names)
  name  = var.user_names[count.index]
}

variable "user_names" {
   description = "IAM USERS"
   type        = list(string)
   default     = ["user1", "user2", "user3"]
}
```

MAP/OBJECT: a group of values identified by named labels, like {project = "project-plan", environment = “dev”}.

```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}
resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type =  "t2.micro"

   tags = var.project_environment

}


variable "project_environment" {
  description = "project name and environment"
  type        = map(string)
  default     = {
    project     = "project-alpha",
    environment = "dev"
  }
}
```

FOR LOOP:

The for loop is pretty simple and if you have used any programming language before then I guess you will be pretty much familiar with the for loop.

Only the difference you will notice over here is the syntax in Terraform.

We are going to take the same example by declaring a list(string) and adding three users to it - user1, user2, user3

Use the above ec2 block if you want

```
output "print_the_names" {
  value = [for name in var.user_names : name]
}

variable "user_names" {
  description = "IAM usernames"
  type        = list(string)
  default     = ["user1", "user2", "user3"]
}
```

FOR EACH:

The for each is a little special in terraforming and you can not use it on any collection variable.

Note : - It can only be used on set(string) or map(string).

The reason why for each does not work on list(string) is because a list can contain duplicate values but if you are using set(string) or map(string) then it does not support duplicate values.

```
resource "aws_iam_user" "example" {
  for_each = var.user_names
  name  = each.value
}

variable "user_names" {
  description = "IAM usernames"
  type        = set(string)
  default     = ["user1", "user2", "user3"]
}
```

LOOPS WITH COUNT:

we need to use count but to use the count first we need to declare collections inside our file.

```
resource "aws_iam_user" "example" {
  count = length(var.user_names)
  name  = var.user_names[count.index]
}

variable "user_names" {
  description = "IAM usernames"
  type        = list(string)
  default     = ["user1", "user2", "user3"]
}
```

VARIABLE.TF:

A variables.tf file is used to define the variables type and optionally set a default value.

```
root@ip-172-31-17-121:~/terraform# ls *.tf
main.tf  variable.tf
root@ip-172-31-17-121:~/terraform# cat main.tf
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type =  var.instance_type

   tags = {
           Name = "Terraform EC2"
   }
}
root@ip-172-31-17-121:~/terraform# cat variable.tf
variable "instance_type" {
   description = "Instance type t2.micro"
   type        = string
   default     = "t2.micro"
}
```

TERRAFORM WORKSPACE:

Terraform Workspace is something that you can use when you want to work on multiple environments at same time.

To create a new workspace : terraform workspace new workspace_name
To list the workspace : terraform workspace list
To show current workspace : terraform workspace show
To switch workspace : terraform workspace select workspace_name

```
root@ip-172-31-17-121:~/terraform# terraform workspace list
* default
```

```
root@ip-172-31-17-121:~/terraform# terraform workspace list
* default

root@ip-172-31-17-121:~/terraform# terraform workspace new dev
Created and switched to workspace "dev"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
root@ip-172-31-17-121:~/terraform# terraform workspace new test
Created and switched to workspace "test"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
root@ip-172-31-17-121:~/terraform# terraform workspace list
  default
  dev
* test
```

DYNAMIC BLOCK:

it is for loop which is going to iterate over and will help you to create a dynamic resource. With the help of dynamic blocks you can create nested repeatable blocks such as settings, ingress rules etc...

```
provider "aws" {
  region          = "ap-south-1"
  access_key      = "AKIAWW7WL2JMJKCCMORC"
  secret_key      = "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"
}

locals {
   ingress_rules = [{
      port        = 443
      description = "Ingress rules for port 443"
   },
   {
      port        = 80
      description = "Ingree rules for port 80"
   }]
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   vpc_security_group_ids = [aws_security_group.main.id]
}
```

```
resource "aws_security_group" "main" {
  egress = [
    {
      cidr_blocks      = [ "0.0.0.0/0" ]
      description      = ""
      from_port        = 0
      ipv6_cidr_blocks = []
      prefix_list_ids  = []
      protocol         = "-1"
      security_groups  = []
      self             = false
      to_port          = 0
    }
  ]
 dynamic "ingress" {
      for_each = local.ingress_rules

      content {
        description = ingress.value.description
        from_port   = ingress.value.port
        to_port     = ingress.value.port
        protocol    = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
      }
  }

  tags = {
    Name = "AWS security group dynamic block"
  }
}
```