TERRAFORM:

**Defenition:**Terraform is an infrastructure as a code tool that allows you to build, change, and version infrastructure safely and efficiently.

**Infrastructure as Code** (IaC) is the managing and provisioning of infrastructure through code instead of through manual processes.

**Components of terraform (integral part of terraform):**

1. Providers
2. Resources,
3. Variables,
4. Statefile,
5. Provisioners,
6. Backends,
7. Modules,
8. Data sources,
9. Service principals.

**Terraform Providers** :

Providers are essentially plugins that Terraform installs to interact with the remote systems i.e. Azure/AWS/Google Cloud/ VMware and a lot of other vendors devices.

**Ex:**

# main.tf

# Configure the AWS provider

provider "aws" {

  access\_key = "YOUR\_AWS\_ACCESS\_KEY"

  secret\_key = "YOUR\_AWS\_SECRET\_KEY"

  region     = "us-west-2"

}

**Resources:**

Resources are the most important part of Terraform. Resources are defined by resource blocks. A resource can define one or more infrastructure resource objects, such as VPCs, virtual machines, or DNS records, Consul key-value pair data, etc.

**Resource Syntax:**

Resources are defined through resource blocks, and I will first explain the scenario of defining a single resource object through resource blocks.

**EX:**

resource "aws\_instance" "web\_server" {   
  ami           = "ami-a1b2c3d4"   
  instance\_type = "t2.micro"   
}

Resource blocks documents the syntax for declaring resources.

[Resource Behavior](https://developer.hashicorp.com/terraform/language/resources/behavior) explains in more detail how Terraform handles resource declarations when applying a configuration.

**Variables:**

By using variables in terraform configuration makes our deployment more dynamic.

A separate file with the name “variable.tf” needs to be created in the working directory to store all the variables for our used in main.tf configuration file.

**Ex:**

variable "image\_id" {

  type = string

}

variable "availability\_zone\_names" {

  type    = list(string)

  default = ["us-west-1a"]

}

**Input Variables:**

[Input Variables](https://developer.hashicorp.com/terraform/language/values/variables) serve as parameters for a Terraform module, so users can customize behavior without editing the source.

Input variables let you customize aspects of Terraform modules without altering the module's own source code. This functionality allows you to share modules across different Terraform configurations, making your module composable and reusable.

* Input variables are like function arguments.

**Ex:**

variable "image\_id" {

  type = string

}

variable "availability\_zone\_names" {

  type    = list(string)

  default = ["us-west-1a"]

}

variable "docker\_ports" {

  type = list(object({

    internal = number

    external = number

    protocol = string

  }))

  default = [

    {

      internal = 8300

      external = 8300

      protocol = "tcp"variable "image\_id" {

  type = string

}

variable "availability\_zone\_names" {

  type    = list(string)

  default = ["us-west-1a"]

}

variable "docker\_ports" {

  type = list(object({

    internal = number

    external = number

    protocol = string

  }))

  default = [

    {

      internal = 8300

      external = 8300

      protocol = "tcp"

    }

  ]

}

    }

  ]

}

**Output Variables:**

[Output Values](https://developer.hashicorp.com/terraform/language/values/outputs) are like return values for a Terraform module. Output values make information about your infrastructure available on the command line, and can expose information for other Terraform configurations to use. Output values are similar to return values in programming languages.

Output values have several uses:

* A child module can use outputs to expose a subset of its resource attributes to a parent module.
* A root module can use outputs to print certain values in the CLI output after running terraform apply.
* When using [remote state](https://developer.hashicorp.com/terraform/language/state/remote), root module outputs can be accessed by other configurations via a [terraform\_remote\_state data source](https://developer.hashicorp.com/terraform/language/state/remote-state-data" \t "_blank).

Resource instances managed by Terraform each export attributes whose values can be used elsewhere in configuration. Output values are a way to expose some of that information to the user of your module.

Ex:

output "instance\_ip\_addr" {

  value = aws\_instance.server.private\_ip

}

**Local Variable:**

A local value assigns a name to an [expression](https://developer.hashicorp.com/terraform/language/expressions), so you can use the name multiple times within a module instead of repeating the expression.

If you are familiar with traditional programming languages, it can be useful to compare Terraform modules to function definitions:

* [Input variables](https://developer.hashicorp.com/terraform/language/values/variables) are like function arguments.
* [Output values](https://developer.hashicorp.com/terraform/language/values/outputs) are like function return values.
* Local values are like a function's temporary local variables.

**Ex:**

locals {

  # Ids for multiple sets of EC2 instances, merged together

  instance\_ids = concat(aws\_instance.blue.\*.id, aws\_instance.green.\*.id)

}

locals {

  # Common tags to be assigned to all resources

  common\_tags = {

    Service = local.service\_name

    Owner   = local.owner

  }

}

locals {

  # Ids for multiple sets of EC2 instances, merged together

  instance\_ids = concat(aws\_instance.blue.\*.id, aws\_instance.green.\*.id)

}

locals {

  # Common tags to be assigned to all resources

  common\_tags = {

    Service = local.service\_name

    Owner   = local.owner

  }

}

**State file:**

 This state file keeps track of resources created by your configuration and maps them to real-world resources.

Terraform compares your configuration with the state file and your existing infrastructure to create plans and make changes to your infrastructure. When you run terraform apply or terraform destroy against your initialized configuration, Terraform writes metadata about your configuration to the state file and updates your infrastructure resources accordingly.

**Provisioners:**

Provisioning mainly deals with configuration activities that happen after the resource is created. It may involve some file operations, executing CLI commands, or even executing the script. Once the resource is successfully initialized, it is ready to accept connections. These connections help Terraform log into the newly created instance and perform these operations.

**The local-exec Provisioner:**

The local-exec provisioner works on the Terraform host – where Terraform configuration is applied/executed. It is used to execute any shell command. It is used to set or read environment variables, details about the resource which is created, invoke any process or application, etc.

**The File Provisioner:**

The file provisioner is a way to copy certain files or artifacts from the host machine to target resources that will be created in the future. This is a very handy way to transport certain script files, configuration files, artifacts like .jar files, binaries, etc. when the target resource is created and boots for the first time.

**Backend Configuration:**

A backend defines where Terraform stores its [state](https://developer.hashicorp.com/terraform/language/state) data files.

Terraform backend is like the repository of instruction to load and run the terraform state especially when the operation like terraform applies executed. Backend is very important when you are developing the infrastructure with a big team. So, Understanding the Terraform Backend is very important.

**Modules:**

*Modules* are containers for multiple resources that are used together. A module consists of a collection of .tf and/or .tf.json files kept together in a directory.Modules are the main way to package and reuse resource configurations with Terraform.

**Data sources:**

Data sources allow Terraform to use the information defined outside of Terraform, defined by another separate Terraform configuration, or modified by functions.

A data source is accessed via a special kind of resource known as a *data resource*, declared using a data block.

Ex:

data "aws\_ami" "example" {

  most\_recent = true

  owners = ["self"]

  tags = {

    Name   = "app-server"

    Tested = "true"

  }

}