# Terraform Configuration Language
# OR
# HashiCorp Configuration Language (HCL)

# Terraform Configuration Language

- *The Terraform language is Terraform's primary user interface.* In every edition of Terraform, a configuration written in the Terraform language is always at the heart of the workflow.

- The main purpose of the Terraform language is declaring resources, which represent infrastructure objects.

- A *Terraform configuration* is a complete document in the Terraform language that tells Terraform how to manage a given collection of infrastructure. A configuration can consist of multiple files and directories.

# Terraform Configuration Language

- The syntax of the Terraform language consists of only a few basic elements

```
resource "aws_vpc" "main" {
  cidr_block = var.base_cidr_block
}

<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```

- Blocks are containers for other content and usually represent the configuration of some kind of object, like a resource. Blocks have a block type, can have zero or more labels, and have a body that contains any number of arguments and nested blocks. Most of Terraform's features are controlled by top-level blocks in a configuration file.

- Arguments assign a value to a name. They appear within blocks.

- Expressions represent a value, either literally or by referencing and combining other values. They appear as values for arguments, or within other expressions.

# Terraform Configuration Language / HCL – Files

- Code in the Terraform language is stored in plain text files with the **.tf file extension**. There is also a JSON-based variant of the language that is named with the **.tf.json file extension**.

- Files containing Terraform code are often called *configuration files.*

# Terraform Configuration Language / HCL – Directories & Modules

- A *module* is a collection of **.tf and/or .tf.json files kept together in a directory.**

- A Terraform module only consists of the top-level configuration files in a directory; **nested directories are treated as separate modules and are not automatically included in the configuration.**

- Terraform evaluates all of the configuration files in a module, ***effectively treating the entire module as a single document.*** Separating various blocks into different files is purely for the convenience of readers and maintainers and has no effect on the module's behaviour.

- A Terraform module can use module calls to explicitly include other modules into the configuration. These child modules can come from local directories (nested in the parent module's directory, or anywhere else on disk), or from external sources like the Terraform Registry.

# Terraform Configuration Language / HCL – The Root Module

Terraform always runs in the context of a **single *root module***. A complete *Terraform configuration* consists of a root module and the tree of child modules (which includes the modules called by the root module, any modules called by those modules, etc.).

***In Terraform CLI, the root module is the working directory where Terraform is invoked***. (You can use command line options to specify a root module outside the working directory, but in practice this is rare. )

More Info: https://www.terraform.io/docs/language/index.html

# Terraform Configuration Language / HCL – Configuration Syntax

- Terraform language is a rich language designed to be relatively easy for humans to read and write. Terraform language can also be expressed in JSON syntax, which is harder for humans to read and edit but easier to generate and parse programmatically.

- It is not necessary to know all of the details of HCL syntax in order to use Terraform

- Terraform language syntax is built around two key syntax constructs: ***arguments and blocks***

# Configuration Syntax– Arguments

An *argument* assigns a value to a particular name:

***name = "xyz234"***

The identifier before the equals sign is the *argument name*, and the expression after the equals sign is the argument's value

# Configuration Syntax– Blocks

A *block* is a container for other content:

```
resource "aws_instance" "example" {
  ami = "abc123"

  network_interface {
    # ...
  }
}
```

- A block has a *type* (**resource** in this example). Each block type defines how many *labels* must follow the type keyword.
- The resource block type expects two labels, which are **aws_instance** and **example** in the example above.
- A particular block type may have any number of required labels, or it may require none as with the nested network_interface block type.
- After the block type keyword and any labels, the **block *body* is delimited by the { and } characters**. Within the block body, **further arguments and blocks may be nested**, creating a hierarchy of blocks and their associated arguments.
- The Terraform language uses a **limited number of *top-level block types*,** which are blocks that can appear outside of any other block in a configuration file.

# Configuration Syntax– Comments

- The Terraform language supports three different syntaxes for comments:

  **#** begins a single-line comment, ending at the end of the line.

  **//** also begins a single-line comment, as an alternative to #.

  **/*** and ***/** are start and end delimiters for a comment that might span over multiple lines.

- The # single-line comment style is the default comment style and should be used in most cases.

- Automatic configuration formatting tools may automatically transform // comments into # comments, since the double-slash style is not idiomatic.

# Terraform Configuration Language / HCL – Resources

- *Resources* are the most important element in the Terraform language. Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components such as DNS records.

## Resources Syntax:

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

- A resource block declares a resource of a given type ("aws_instance") with a given local name ("web"). The name is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside that module's scope.

- The resource type and name together serve as an identifier for a given resource and so must be unique within a module.

# Resources Types:

- Each resource is associated with a **single *resource type***, which determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports.
- Each resource type is implemented by a **provider, which is a plugin for Terraform** that offers a collection of resource types.
- A **provider** usually provides resources to manage a single cloud or on-premises infrastructure platform
- **Providers** are distributed separately from Terraform itself but **Terraform can automatically install most providers when initializing a working directory**.
- In order to manage resources, a Terraform module must specify which **providers** it requires.
- Additionally, **most providers need some configuration in order to access their remote APIs**, and the root module must provide that configuration.
- Terraform usually automatically determines which provider to use based on a resource type's name

# Terraform Configuration Language / HCL – Providers

- **Terraform relies on plugins called "providers"** to interact with remote systems.
- Terraform configurations must declare which providers they require so that Terraform can install and use them. Additionally, some providers require configuration (like endpoint URLs or cloud regions) before they can be used.
- Each provider adds a set of resource types and/or data sources that Terraform can manage.
- Every resource type is implemented by a provider; **without providers, Terraform can't manage any kind of infrastructure.**
- Providers are distributed separately from Terraform itself, and **each provider has its own release cadence and version numbers.**
- The **Terraform Registry** is the main directory of publicly available Terraform providers, and hosts providers for most major infrastructure platforms.

# Terraform Configuration Language / HCL

## configuration files may include

- Variables & Outputs
- Data Sources
- Expressions
- Functions
- Terraform Setting
- Modules

More Info can be found here:

https://www.terraform.io/docs/language/index.html