1. **What is Terraform**

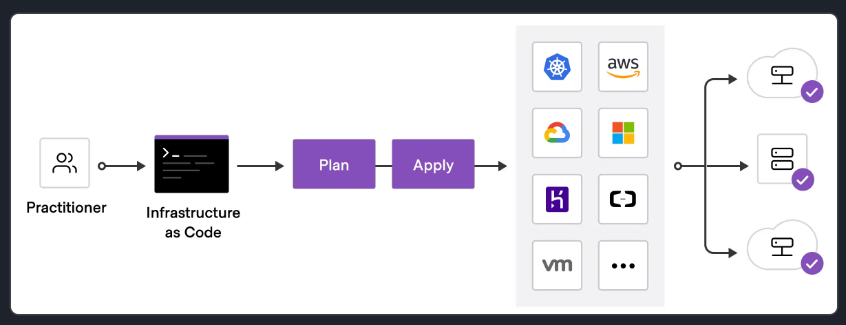
* Is HashiCorp’s infrastructure as code tool.
* It lets you define resources and infrastructure in human-readable, declarative configuration file, and manages your infrastructure’s lifecycle.
* Using terraform has several advantages over manually managing your infrastructure:
  + Can manage infrastructure on multiple cloud platforms.
  + The human-readable configuration language helps you write infrastructure code quickly.
  + Terraform’s state allows you to track resource changes throughout your deployments.
  + You can commit your configurations to version control to safely collaborate on infrastructure.

1. **Infrastructure as Code (IaC)**

* IaC tools allow you to manage infrastructure with configuration files rather than through a graphical user interface.
* IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share.

1. **Standardize your deployment workflow**

* Providers define individual units of infrastructure, for example compute instances or private networks, as resources.
* You can compose resources from different providers into reusable Terraform configurations called modules, and manage them with a consistent language and workflow.
* Terraform’s configuration language is declarative, meaning that is describe the desired end-state for your infrastructure.
* In contrast to procedural programming languages that require step-by-step instructions to perform tasks.
* To deploy infrastructure with Terraform:
  + Scope: Identify the infrastructure for your project.
  + Author: Write the configuration for your infrastructure.
  + Initialize: Install the plugins Terraform needs to manage the infrastructure.
  + Plan: Preview the changes Terraform will make to match your configuration.
  + Apply: Make the planned changes.



1. **Configuration**

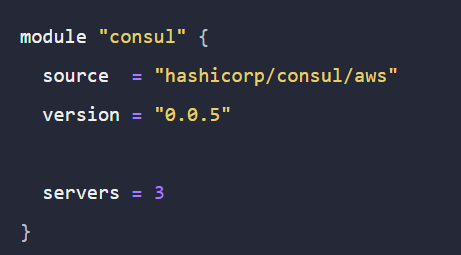
* The set of files used to describe infrastructure in Terraform is known as a **terraform configuration.**
* Each Terraform configuration must be in its own working directory.
* Each Terraform have the following sections/blocks:
  + Terraform Block:
    - Contains Terraform settings include: ***providers*** which each provider has source attribute defines an optional ***hostname***, a ***namespace***, and the ***provider type.***
    - Terraform installs providers from the Terraform Registry by default (the providers are defined in the ***source*** attribute).
    - The ***version*** attribute is optional, but it recommended using it to enforce the provider version. Without it, Terraform will always use the latest version of the provider (which may introduce breaking changes).
  + Providers Block:
    - The provider block configures the specified provider.
    - A provider is a plugin that Terraform uses to create and manage your resources.
    - You can define ***multiple*** provider blocks in a Terraform configuration to manage resources from different providers.
  + Resource block:
    - Use resource blocks to define ***components*** of your infrastructure. A resource might be a ***physical component*** such as a server, or it can be a ***logical resource*** such as a software application.
    - Resource blocks have 2 strings before the block: the ***resource type*** and ***resource name***.
      * *Format: resource “<resource\_type>” “<resource\_name>”*
      * Ex: resource “google\_compute\_network” “vpc\_network” {}
        + The prefix of the resource type maps to the name of the provider (in this case is “google”).
        + Together, the resource type and resource name form a unique ID for the resource => “google\_compute\_network.vpc\_network”
    - Resource blocks contain arguments which used to configure the resource such as: machine sizes, disk image names, or VPC IDs, etc…
  + Data block/Data Source:
    - A data source is accessed via a special kind of resource known as a data resource, declared using a ***data block***.

1. **Module:**
2. **What is module**

* 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.
* Every Terraform configuration has at least one module, know as its ***root module***, which consists of the resources defined in the ‘.tf’ files in the main working directory. And a module that has been called by another module is often referred to as a ***child module***.
* ***Published Module*** is a module that is published to a public module registry (popular is Terraform Registry). These modules are free to use.

1. **Developing/Using Module**

* ***Module blocks*** documents the syntax for calling a child module from a parent module, including meta-arguments like for\_each.
  + ***Calling module*** of the child module (To call a module, we need to defined all required input\_variables of that called module)



* + - The *source* argument defines the source of the child module (Required).
    - The *version* argument is recommended for modules from a registry (Optional).
    - The *input\_variables* are defined by the called module (the *server* argument is the example)
    - The other *meta-arguments* have special meaning:
      * count: creates multiple instances of a module from a single module block.
      * for\_each: creates multiple instances of a module from a single module block
      * providers: passes provider configuration to a child module (the default is that the child module inherits all of the default provider configuration from the calling module).
      * depends\_on: creates explicitly dependencies between the entire module and the listed targets.
* ***Module Sources*** documents what kind of paths, addresses, and URIs can be used in the source argument of a module block.

1. **Conclusion:**

* **Terraform** provisions, updates, and destroys infrastructure resources such as physical machines, VMs, network switches, containers, and more.
* **Configurations** are code written for Terraform, using the human-readable HashiCorp Configuration Language (HCL) to describe the desired state of infrastructure resources.
* **Providers** are the plugins that Terraform uses to manage those resources. Every supported service or infrastructure platform has a provider that defines which resources are available and performs API calls to manage those resources.
* **Modules** are reusable Terraform configurations that can be called and configured by other configurations. Most modules manage a few closely related resources from a single provider.
* **The Terraform Registry** makes it easy to use any provider or module. To use a provider or module from this registry, just add it to your configuration; when you run `terraform init`, Terraform will automatically download everything it needs.

References:

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

<https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/infrastructure-as-code>

<https://www.willianantunes.com/blog/2021/05/the-easiest-way-to-run-a-container-on-gce-with-terraform/>

<https://github.com/terraformed/terraform-docker-gcp>

<https://github.com/terraform-google-modules/terraform-google-container-vm>

Data Source using with template to generate ***cloud-init scripts*** with embed variable using Terraform Interpolation ${…}: <https://kodekloud.com/blog/terraform-template/>