Laboratorio

TEORIA

TERRAFORM

CONTROL DE VERSIONES

Elaborado por:	Jonatan Stiven Gutierrez	**No. de Versión:**	1.0.0
Revisado por:		**Fecha de revisión:**	
Aprobado por:		**Fecha de Aprobación:**	

Historia de Modificaciones

No. de Versión	Fecha de Versión	Autor	Revisado por	Aprobado por	Descripción
1.0.0	21/02/2024	Jonatan Stiven Gutierrez			Documento Original

Lista de distribución

Para	Acción*	Empresa	Firma/Medio de Entrega

** Tipos de acción: Aprobar, Revisar, Informar, Archivar, Complementar, Asistir a junta, Otras (por favor especificar)*

Contenido

INTRODUCCION

El siguiente documento proporciona una introducción detallada a Terraform y sus diferentes temas.

PRERREQUISITOS

- Haber cumplido con todo lo previamente visto.

TERRAFORM:

Documentación: https://registry.terraform.io

Terraform es una herramienta de infraestructura como código (IaC) que permite a los equipos de operaciones y desarrollo definir y gestionar la infraestructura de manera programática. Desarrollado por HashiCorp, Terraform simplifica el proceso de aprovisionamiento, configuración y gestión de recursos en la nube y en otros entornos de infraestructura.

Características Principales:

1. **Declarativo y Legible:** Terraform utiliza un enfoque declarativo para describir la infraestructura deseada, lo que permite a los usuarios definir recursos utilizando una sintaxis sencilla y legible.

2. **Soporte para Múltiples Proveedores:** Terraform es compatible con una amplia variedad de proveedores de nube, incluyendo AWS, Azure, Google Cloud Platform, y más. Esto permite a los equipos gestionar recursos en diferentes entornos de manera coherente.

3. **Estado de la Infraestructura:** Terraform mantiene un estado de la infraestructura gestionada, lo que facilita la realización de cambios incrementales y la gestión de recursos de manera eficiente.

4. **Modularidad y Reutilización:** Terraform fomenta la modularidad y la reutilización del código mediante el uso de módulos, variables y plantillas, lo que permite evitar la duplicación de configuraciones y promover la coherencia del código.

Beneficios:

1. **Agilidad:** Terraform permite aprovisionar y configurar recursos de manera rápida y eficiente, lo que acelera el ciclo de desarrollo y despliegue de aplicaciones.

2. **Consistencia:** Al definir la infraestructura como código, Terraform garantiza que los entornos de desarrollo, pruebas y producción sean coherentes y reproducibles.

3. **Escalabilidad:** Con su soporte para múltiples proveedores y su capacidad para gestionar infraestructuras de cualquier tamaño y complejidad, Terraform es altamente escalable.

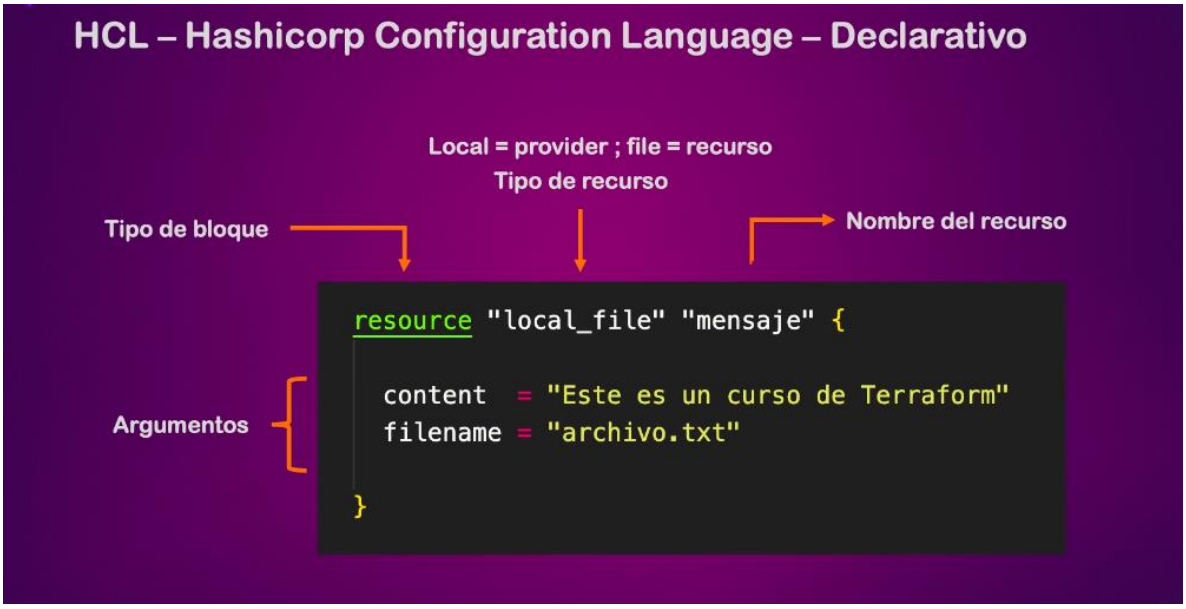
Terraform es una herramienta poderosa que simplifica la gestión de la infraestructura como código, proporcionando una forma eficiente y escalable de automatizar tareas de aprovisionamiento, configuración y gestión de recursos. Con su enfoque declarativo, soporte para múltiples proveedores y capacidad para seguir principios como DRY, Terraform es una opción popular para la automatización de la infraestructura en entornos de nube y de infraestructura.

INTRODUCCION A LAC:

Terraform es una herramienta de infraestructura como código (IaC) desarrollada por HashiCorp que permite definir y gestionar la infraestructura de forma programática. Una de las características clave de Terraform es su lenguaje de configuración de alto nivel (HCL), que proporciona una sintaxis clara y expresiva para definir la infraestructura deseada.

QUE ES HCL:

El Lenguaje de Configuración de Alto Nivel (HCL) es el lenguaje utilizado por Terraform para definir la infraestructura como código. HCL es un lenguaje simple y expresivo que permite a los usuarios describir recursos y configuraciones de manera fácil de entender y mantener. Proporciona una sintaxis clara y concisa para definir recursos, variables, bloques de configuración y más.



Características de HCL:

1. **Declarativo y Expresivo:** HCL utiliza un enfoque declarativo para definir la infraestructura deseada, lo que significa que describe qué recursos deben crearse y cómo deben configurarse, en lugar de especificar pasos detallados para crearlos. Esto hace que las configuraciones sean más legibles y fáciles de entender.

2. **Sintaxis Simple y Legible:** HCL utiliza una sintaxis simple y legible que facilita la escritura y comprensión de las configuraciones. Utiliza bloques de configuración y atributos clave-valor para definir recursos y configuraciones, lo que hace que las configuraciones sean fáciles de estructurar y organizar.

3. **Modularidad y Reutilización:** HCL es modular, lo que significa que puedes definir recursos y configuraciones en módulos separados y luego combinarlos para construir configuraciones más complejas. Esto promueve la reutilización del código y facilita la gestión de configuraciones grandes y complejas.

Ejemplo de Uso de HCL en Terraform:

Supongamos que queremos definir una infraestructura básica en AWS utilizando Terraform y HCL. Aquí hay un ejemplo simple que define una instancia de EC2:

```
# Archivo de configuración main.tf

provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

En este ejemplo:

- Se define el proveedor de AWS y se especifica la región en la que se van a crear los recursos.
- Se define un recurso de instancia de EC2 llamado "example", especificando la AMI y el tipo de instancia.

Este es solo un ejemplo básico, pero muestra cómo puedes utilizar HCL para definir recursos y configuraciones en Terraform. Con HCL, puedes crear configuraciones más complejas y gestionar infraestructuras de cualquier tamaño y complejidad de manera eficiente y escalable.

MULTIPLE PROVIDERS Y DEFINICION DE DRY:

En Terraform, la capacidad de utilizar múltiples proveedores y aplicar el principio DRY (Don't Repeat Yourself) son características esenciales para la gestión eficiente de la infraestructura como código. A continuación, explicaremos qué significan y cómo se aplican en Terraform.

Múltiples Proveedores:

Terraform permite gestionar recursos en varios proveedores de nube, como AWS, Azure, Google Cloud Platform, y otros. Esto proporciona flexibilidad a los equipos de infraestructura para utilizar servicios de diferentes proveedores en un mismo proyecto. Para configurar múltiples proveedores en Terraform, simplemente se define cada proveedor dentro del bloque provider, especificando las credenciales y configuraciones necesarias para cada uno.

```
provider "aws" {
  region = "us-west-2"
}

provider "google" {
  project = "my-google-project"
  region  = "us-central1"
}
```

Con esta configuración, Terraform puede gestionar recursos tanto en AWS como en Google Cloud Platform en un mismo archivo de configuración.

Principio DRY (Don't Repeat Yourself):

El principio DRY es un concepto fundamental en la programación que enfatiza la importancia de la reutilización del código. En Terraform, aplicar el principio DRY implica evitar la duplicación de configuraciones y promover la modularidad del código. Esto se puede lograr utilizando módulos, variables y plantillas para definir recursos y configuraciones comunes una vez y reutilizarlas en múltiples partes del código.

Por ejemplo, en lugar de definir manualmente cada recurso de manera repetitiva, se pueden crear módulos que encapsulen conjuntos de recursos relacionados y luego utilizarlos en diferentes partes del código.

```
module "web_server" {
  source = "./modules/web_server"
}

module "database_server" {
  source = "./modules/database_server"
}
```

En este ejemplo, los módulos web_server y database_server encapsulan la lógica y la configuración necesarias para crear y gestionar servidores web y bases de datos, respectivamente. Luego, estos módulos pueden ser utilizados en diferentes partes de la infraestructura, evitando la duplicación de código y promoviendo la coherencia y la mantenibilidad.

En resumen, utilizar múltiples proveedores y aplicar el principio DRY en Terraform permite crear configuraciones de infraestructura más flexibles, eficientes y mantenibles. Esto facilita la gestión de la infraestructura como código en entornos de nube heterogéneos y complejos.

TIPO DE VARIABLES:

En Terraform, los tipos de variables son esenciales para definir y gestionar la configuración de infraestructura de manera eficiente y estructurada. Estos tipos ayudan a Terraform a comprender los datos que se están utilizando en el código y a garantizar su correcto manejo durante la ejecución. Aquí tienes una introducción a los tipos de variables en Terraform:

Tipos de Variables en Terraform

Terraform admite varios tipos de datos para las variables, que incluyen:

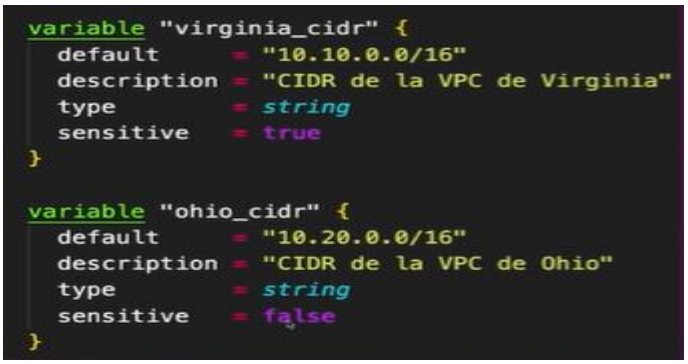
- **any:** puede representar cualquier tipo de dato.
- **string:** Cadena de caracteres.
- **number:** Número entero o de punto flotante.
- **bool:** Valor booleano (true/false).
- **list:** Lista de elementos del mismo tipo.
- **map:** Mapa de pares clave-valor.
- **object:** Objeto que agrupa múltiples atributos relacionados.
- **tuple:** Tupla que puede contener diferentes tipos de datos.
- **set:** un set es una estructura de datos que almacena una colección de elementos únicos y no ordenados.

TIPO DE VARIABLE STRING:

En Terraform, una cadena (string) es una secuencia de caracteres alfanuméricos que se utiliza para representar texto. Las cadenas son uno de los tipos de datos fundamentales y se utilizan ampliamente en las configuraciones de Terraform para representar valores como nombres de recursos, direcciones IP, rutas de archivos, etc.

Ejemplo:

- sensitive es para poder ver su contenido (true), por lo contrario (false)



TIPO DE VARIABLE NUMBER:

En Terraform, un número (number) es un tipo de dato que representa valores numéricos, ya sean enteros o de punto flotante. Los números se utilizan ampliamente en las configuraciones de Terraform para representar cantidades, tamaños, índices y otros valores numéricos.

Ejemplo:

TIPO DE VARIABLE BOOL:

En Terraform, un booleano (bool) es un tipo de dato que representa un valor lógico, es decir, una expresión que puede ser verdadera (true) o falsa (false). Los booleanos se utilizan para representar condiciones, opciones binarias y resultados de comparaciones en las configuraciones de Terraform.

Ejemplo:

- false o true van sin comillas.

TIPO DE VARIABLE LIST:

Una lista en Terraform es una estructura de datos que contiene una secuencia ordenada de elementos del mismo tipo. En Terraform, las listas se utilizan para representar conjuntos de datos homogéneos donde cada elemento tiene una posición específica en la lista.

Las listas admiten elementos repetidos.

Ejemplo:

```
variable "lista_cidrs" {
  default = ["10.10.0.0/16", "10.20.0.0/16"]
#               Posicion 0       Posicion 1
  type    = list(string)
}
```

```
resource "aws_vpc" "vpc_virginia" {
  cidr_block = var.lista_cidrs[0]
  tags = {
    Name = "VPC_VIRGINIA"
    name = "prueba"
    env  = "Dev"
  }
}

resource "aws_vpc" "vpc_ohio" {
  cidr_block = var.lista_cidrs[1]
  tags = {
    Name = "VPC_OHIO"
    name = "prueba"
    env  = "Dev"
  }
  provider = aws.ohio
}
```

TIPO DE VARIABLE MAP:

En Terraform, un map es una estructura de datos que permite asociar valores con claves. Se utiliza para representar conjuntos de datos relacionados donde cada elemento tiene una clave única y un valor asociado. Los mapas son útiles cuando necesitas almacenar y manipular datos de forma estructurada y eficiente.

Ejemplo:

```
variable "map_cidrs" {
  default = {
    "virginia" = "10.10.0.0/16"
    "ohio"     = "10.20.0.0/16"
  }
  type = map(string)
}
```

```
resource "aws_vpc" "vpc_virginia" {
  cidr_block = var.map_cidrs["virginia"]
  tags = {
    Name = "VPC_VIRGINIA"
    name = "prueba"
    env  = "Dev"
  }
}

resource "aws_vpc" "vpc_ohio" {
  cidr_block = var.map_cidrs["ohio"]
  tags = {
    Name = "VPC_OHIO"
    name = "prueba"
    env  = "Dev"
  }
  provider = aws.ohio
}
```

TIPO DE VARIABLE OBJECT:

En Terraform, un object (objeto) es una estructura de datos que permite agrupar múltiples atributos relacionados bajo una sola entidad. A diferencia de un map, que utiliza claves únicas para acceder a sus valores, un objeto organiza sus datos en pares atributo-valor, donde cada atributo tiene un nombre único y un valor asociado.

Ejemplo:

```
variable "virginia" {
  type = object({
    nombre      = string
    cantidad    = number
    cidrs       = list(string)
    disponible  = bool
    env         = string
    owner       = string
  })

  default = {
    cantidad    = 1
    cidrs       = ["10.10.0.0/16"]
    disponible  = true
    env         = "Dev"
    nombre      = "Virginia"
    owner       = "Nazareno"
  }
}
```

```
resource "aws_vpc" "vpc_virginia" {
  cidr_block = var.virginia.cidrs[0]
  tags = {
    Name = var.virginia.nombre
    name = var.virginia.nombre
    env  = var.virginia.env
  }
}
```

TIPO DE VARIABLE TUPLE:

Un tuple en Terraform es una estructura de datos que permite almacenar múltiples valores de diferentes tipos en una sola entidad. A diferencia de una lista, donde todos los elementos deben ser del mismo tipo, un tuple puede contener elementos de diferentes tipos.

Ejemplo:

```
variable "ohio" {
  type    = tuple([string, string, number, bool, string])
  default = ["Ohio", "10.20.0.0/16", 1, false, "Dev"]
}
```

```
resource "aws_vpc" "vpc_ohio" {
  cidr_block = var.ohio[1]
  tags = {
    Name = var.ohio[0]
    name = var.ohio[0]
    env  = var.ohio[4]
  }
  provider = aws.ohio
}
```

TIPO DE VARIABLE SET:

En Terraform, un set es una estructura de datos que almacena una colección de elementos únicos y no ordenados. A diferencia de las listas y los tuples, que permiten elementos duplicados y mantienen un orden específico, los sets garantizan la unicidad de los elementos y no tienen un orden definido.

- Set no admite elementos repetidos.
- No podemos acceder a elementos puntuales.
- No es muy común su uso.

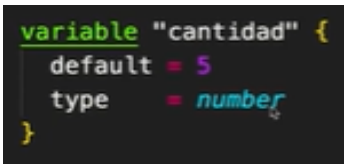
Ejemplo:

```
variable "set_cidrs" {
  default = ["10.10.0.0/16", "10.20.0.0/16"]
  type = set(string)
}
```

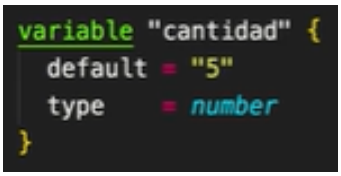
```
resource "aws_vpc" "vpc" {
  for_each   = var.set_cidrs
  cidr_block = each.value
  tags = {
    Name = "VPC_TEST"
    name = "prueba"
    env  = "Dev"
  }
}
```

CONVERSION DE TIPOS DE VARIABLES:

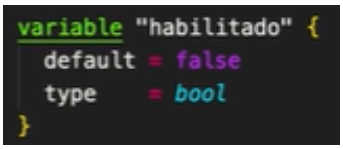
1. Primer ejemplo, podemos ver la primera variable "cantidad", que tiene:
   - default = 5
   - type = number



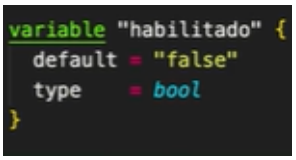
   - En cambio, la segunda variable tiene un error, al escribir el numero entre comillas en teoría sería un string, pero terraform lo corrige porque se da cuenta de lo que quisimos hacer y lo acepta en comillas.



2. Segundo ejemplo podemos ver la primera variable "habilitado", que tiene:
   - default = false
   - type = bool



   - En cambio, la segunda variable tiene un error, al escribir false entre comillas lo toma como string y terraform lo corrige porque se da cuenta de lo que quisimos hacer y lo acepta en comillas.



Nota: Solamente corrige de String a number y viceversa, string a bool y viceversa.

OUTPUTS:

…