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 4
PRERREQUISITOS 4
TERRAFORM: 5
INTRODUCCION A LAC: 6
QUE ES HCL: 6
MULTIPLE PROVIDERS Y DEFINICION DE DRY: 9
TIPO DE VARIABLES: 10
TIPO DE VARIABLE STRING: 11
TIPO DE VARIABLE NUMBER: 12
TIPO DE VARIABLE BOOL: 13
TIPO DE VARIABLE LIST: 14
TIPO DE VARIABLE MAP: 15
TIPO DE VARIABLE OBJECT: 16
TIPO DE VARIABLE TUPLE: 17
TIPO DE VARIABLE SET: 18
CONVERSION DE TIPOS DE VARIABLES: 19
OUTPUTS: 20
DEPENDENCIAS: 21
DEPENDENCIAS EXPLICITAS: 21
DEPENDENCIAS IMPLICITAS: 22
TARGET RESOURCES: 22
DIAGRAMAS DE INFRAESTRUCTURA: 24
TERRAFORM STATE: 26
COMANDOS TERRAFORM: 28
LIFECYCLES: 29

INTRODUCCION

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

PRERREQUISITOS

- Haber cumplido con todo lo visto en los anteriores archivos.

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)

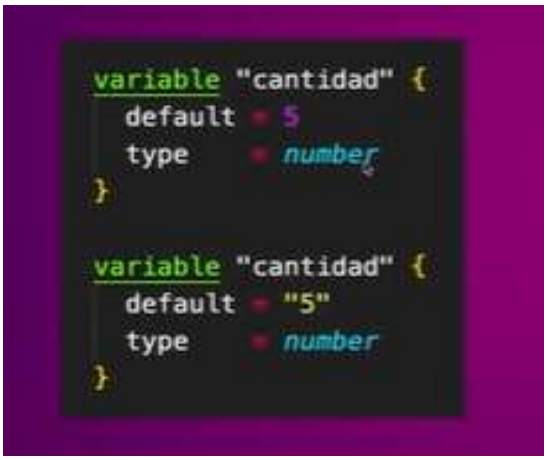
```
variable "virginia_cidr" {
  default     = "10.10.0.0/16"
  description = "CIDR de la VPC de Virginia"
  type        = string
  sensitive   = true
}

variable "ohio_cidr" {
  default     = "10.20.0.0/16"
  description = "CIDR de la VPC de Ohio"
  type        = string
  sensitive   = 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:

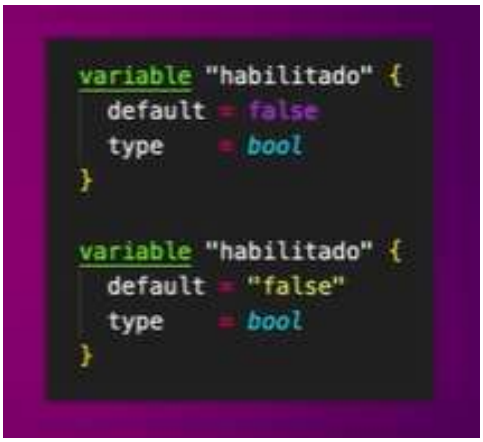


TIPO DE VARIABLE BOOL:

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

```
variable "cantidad" {
  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.

```
variable "cantidad" {
  default = "5"
  type    = number
}
```

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

```
variable "habilitado" {
  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.

```
variable "habilitado" {
  default = "false"
  type    = bool
}
```

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

OUTPUTS:

Los outputs en Terraform son una forma de obtener información útil y relevante sobre los recursos que has creado o modificado utilizando tu configuración de Terraform. Estos outputs te permiten recuperar datos específicos de tus recursos y utilizarlos en otros pasos de tu infraestructura o en tu proceso de implementación.

A continuación, te explico cómo funcionan los outputs en Terraform:

1. **Definición de outputs:** Los outputs se definen en tu archivo de configuración de Terraform utilizando el bloque output. Dentro de este bloque, especificas el nombre del output y la expresión que quieres evaluar para obtener el valor deseado. Por ejemplo:

   ```
   output "instance_ip" {
     value = aws_instance.example.public_ip
   }
   ```

   En este ejemplo, estamos creando un output llamado instance_ip que recupera la dirección IP pública de una instancia EC2 creada con la configuración de Terraform.

2. **Evaluación de expresiones:** La expresión definida en el bloque value se evalúa cuando se ejecuta Terraform. Puedes utilizar cualquier expresión válida de Terraform para recuperar datos de tus recursos. Esto puede incluir referencias a otros recursos, valores de atributos, funciones, etc.

3. **Visualización de outputs:** Después de ejecutar terraform apply, Terraform mostrará los valores de los outputs en la salida estándar. Esto te permite ver rápidamente la información recuperada de tus recursos.

4. **Uso de outputs:** Puedes utilizar los valores de los outputs en otros pasos de tu infraestructura, como scripts de aprovisionamiento, configuración de servicios, etc. Por ejemplo, podrías utilizar la dirección IP recuperada en el ejemplo anterior para configurar un balanceador de carga o actualizar registros DNS.

5. **Referencia de outputs**: También puedes referenciar los outputs en otros archivos de configuración de Terraform utilizando la función terraform_remote_state. Esto te permite compartir información entre diferentes configuraciones de Terraform.

En resumen, los outputs en Terraform te permiten recuperar información específica de tus recursos y utilizarla de manera dinámica en otros pasos de tu infraestructura. Son una herramienta poderosa para automatizar y gestionar tu infraestructura de manera eficiente.

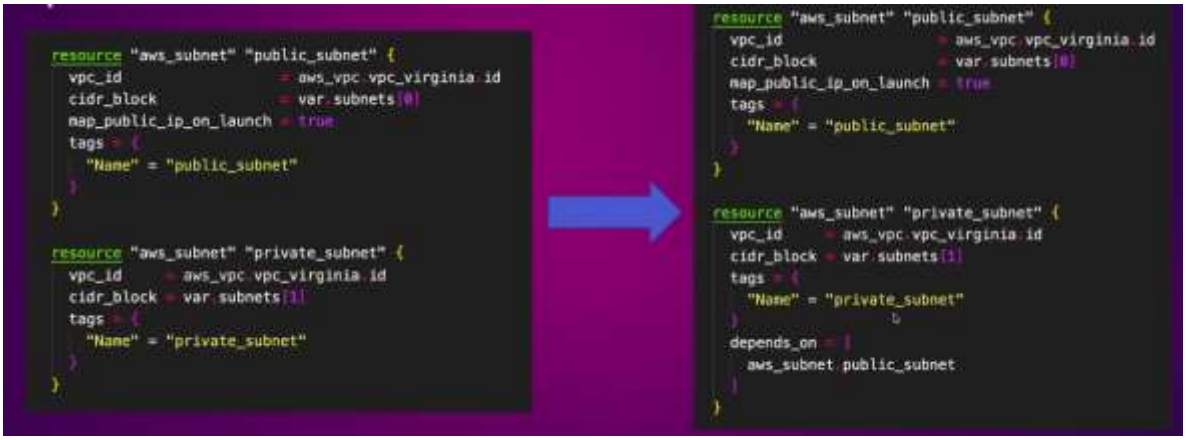
DEPENDENCIAS:

En Terraform, las dependencias se refieren a cómo los recursos se relacionan entre sí y en qué orden deben crearse, actualizarse o destruirse. Las dependencias pueden ser explícitas o implícitas.

DEPENDENCIAS EXPLICITAS:

Son las relaciones establecidas explícitamente mediante los bloques de '**depends_on**' en el código de Terraform. Esto indica a Terraform que un recurso depende de otro y debe esperar a que se complete antes de ejecutar una acción en él.

Ejemplo:

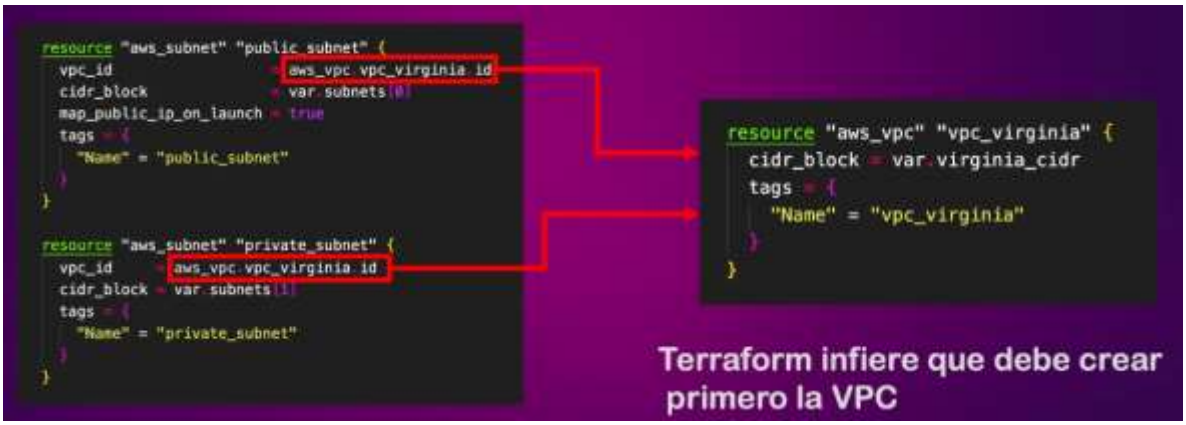


- Depends_on, nos indica que primero debe crear la subnet publica, y ahí si crea la privada.

DEPENDENCIAS IMPLICITAS:

Son determinadas por Terraform automáticamente en función de las referencias entre recursos en el código. Terraform detecta automáticamente estas dependencias y las gestiona internamente. Por ejemplo, si un recurso hace referencia a otro en su configuración, Terraform comprende que hay una dependencia entre ellos. No necesitas declarar estas dependencias explícitamente.

Ejemplo:



TARGET RESOURCES:

En Terraform, el concepto de "Target Resources" se refiere a la capacidad de limitar las operaciones de Terraform a un conjunto específico de recursos dentro de tu configuración. Esto es útil cuando deseas realizar operaciones como la creación, actualización o destrucción solo en un subconjunto de recursos en lugar de todos los recursos definidos en tu configuración.

Puedes especificar los recursos de destino de diferentes maneras según la versión de Terraform que estés utilizando:

- **Terraform 0.12 y versiones posteriores:** En versiones más recientes de Terraform (0.12 y posteriores), puedes utilizar el argumento -target junto con los comandos apply, plan, destroy, etc., para especificar los recursos de destino.

  Ejemplo:

  - terraform apply -target=aws_instance.web

  Esto indica a Terraform que solo aplique cambios en el recurso aws_instance.web, ignorando todos los demás recursos definidos en tu configuración.

- **Versiones anteriores de Terraform:** En versiones anteriores de Terraform, no existe un soporte nativo para especificar recursos de destino. Sin embargo, puedes lograr un resultado similar utilizando la opción terraform refresh seguida de un terraform apply.

  Ejemplo:

  - terraform refresh
  - terraform apply

  El comando terraform refresh actualizará el estado de Terraform con la información actual de la infraestructura, y luego puedes ejecutar terraform apply para aplicar los cambios solo en los recursos que se han actualizado en comparación con el estado anterior.

Recuerda que es importante usar la opción -target con precaución, ya que puede tener implicaciones en la integridad de tu infraestructura si se usa incorrectamente. Es recomendable utilizarlo solo cuando sea absolutamente necesario y comprender completamente las consecuencias de aplicar cambios solo a un subconjunto de recursos.

DIAGRAMAS DE INFRAESTRUCTURA:

Para crear diagramas de infraestructura de Terraform en draw.io, puedes seguir estos pasos:

1. **Accede a draw.io:** Ve al sitio web de draw.io o abre la aplicación si ya la tienes instalada.

2. **Selecciona la Plantilla de Infraestructura:** En el menú lateral izquierdo, elige la opción "Plantillas" y busca una plantilla que se adapte a tu diagrama de infraestructura de Terraform. Puedes seleccionar una plantilla genérica de infraestructura o buscar plantillas específicas de AWS, Azure, Google Cloud, etc., dependiendo de la nube que estés utilizando.

3. **Arrastra y Suelta Elementos de Terraform:** Una vez que hayas seleccionado la plantilla adecuada, arrastra y suelta los elementos de Terraform desde la biblioteca de formas de draw.io al lienzo de trabajo. Por ejemplo, puedes agregar recursos como instancias EC2, grupos de seguridad, VPCs, subredes, etc., dependiendo de tu infraestructura.

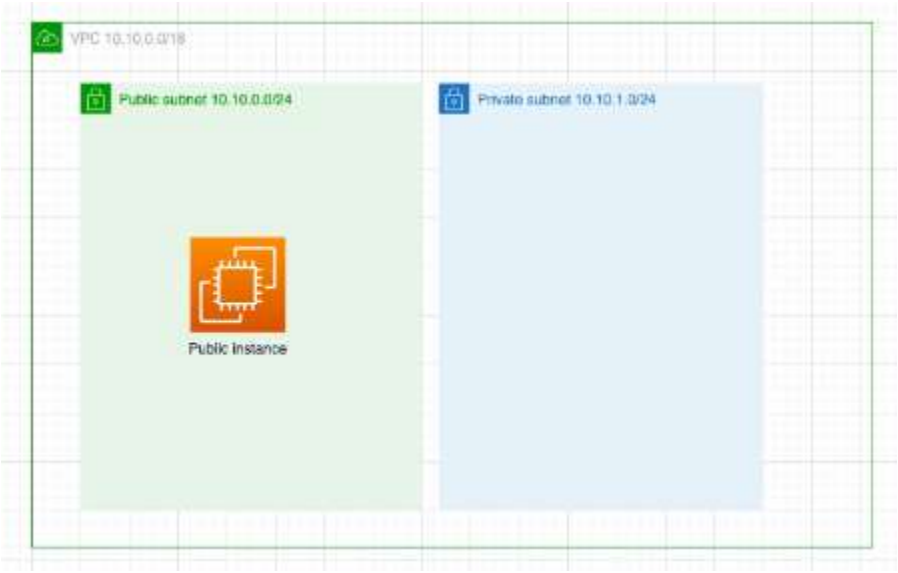
4. **Conecta los Elementos y Representa la Estructura:** Utiliza las herramientas de conexión (flechas, líneas, etc.) para conectar los elementos y representar la estructura de tu infraestructura de Terraform. Por ejemplo, puedes conectar una instancia EC2 con un grupo de seguridad y una subred.

5. **Personaliza los Elementos de Terraform:** Haz clic derecho en cada elemento para personalizar sus propiedades, como el nombre, las etiquetas, los atributos específicos de Terraform, etc. Puedes cambiar colores, tamaños y estilos de texto según sea necesario.

6. **Añade Etiquetas y Texto Explicativo:** Utiliza la herramienta de texto para añadir etiquetas y descripciones a los elementos del diagrama. Esto te ayudará a identificar y explicar cada parte de la infraestructura de Terraform.

7. **Agrega Notas o Comentarios:** Si es necesario, puedes agregar notas o comentarios al diagrama para proporcionar información adicional o aclaraciones sobre ciertos elementos o decisiones de diseño.

8. **Guarda y Exporta:** Una vez que hayas terminado de crear tu diagrama de infraestructura de Terraform en draw.io, guarda tu trabajo en el formato deseado (por ejemplo, en el almacenamiento local, Google Drive, Dropbox, etc.). También puedes exportar el diagrama en diferentes formatos, como PNG, PDF, SVG, etc., para compartirlo con otros miembros del equipo o incluirlo en documentación.



Recuerda que draw.io es una herramienta flexible y te permite crear diagramas de infraestructura de Terraform personalizados según tus necesidades específicas. Experimenta con las diferentes opciones y herramientas para crear un diagrama claro y completo de tu infraestructura basada en Terraform.

TERRAFORM STATE:

El estado (state) de Terraform es un archivo que almacena información sobre la infraestructura gestionada por Terraform. Este archivo mantiene un registro del estado actual de los recursos creados y administrados por Terraform, como instancias de servidores, redes, almacenamiento, entre otros.

El archivo de estado de Terraform puede estar en diferentes formatos, como JSON o HCL (HashiCorp Configuration Language). Algunos de los detalles que se almacenan en el estado incluyen:

1. **IDs de Recursos**: Terraform asigna identificadores únicos a los recursos creados, y estos IDs se almacenan en el estado para realizar el seguimiento de los recursos.

2. **Atributos de Recursos:** El estado también incluye los atributos y configuraciones de cada recurso, como direcciones IP, nombres, configuraciones de seguridad, etc.

3. **Relaciones entre Recursos:** Terraform también mantiene información sobre las relaciones y dependencias entre recursos, como la conexión entre una instancia EC2 y un grupo de seguridad en AWS.

4. **Versiones de Configuración:** El estado incluye información sobre la versión de la configuración de Terraform utilizada para crear los recursos, lo que permite mantener un historial de cambios y versiones.

Es importante entender la gestión y el manejo del estado en Terraform, ya que el estado es la fuente de verdad para Terraform y se utiliza para planificar y aplicar cambios en la infraestructura. Aquí hay algunos conceptos clave relacionados con el estado en Terraform:

a. **Backend de State:** Terraform ofrece diferentes backends de estado para almacenar y gestionar el archivo de estado, como el almacenamiento remoto en AWS S3, Azure Blob Storage, Google Cloud Storage, entre otros.

b. **Bloque de Estado Remoto:** En la configuración de Terraform, puedes especificar un backend de estado remoto para almacenar el archivo de estado fuera de la máquina local y permitir la colaboración en equipos.

c. **Locking de Estado:** Terraform utiliza el locking de estado para evitar conflictos cuando múltiples usuarios o procesos intentan modificar el estado al mismo tiempo.

d. **Manejo de Estado en Equipo:** Es importante establecer prácticas y políticas para el manejo del estado en equipos, como el uso de backends de estado remoto, controles de acceso y revisión del estado.

En resumen, el estado de Terraform es fundamental para la gestión de la infraestructura como código y proporciona una visión completa y actualizada del estado actual de los recursos gestionados por Terraform.

COMANDOS TERRAFORM:

Los comandos son esenciales para trabajar con Terraform y gestionar la infraestructura como código de manera eficiente y segura. Se recomienda comprender el propósito y la funcionalidad de cada comando antes de utilizarlos en entornos de producción.

Te doy algunos comandos no muy usados frecuentemente, pero útiles:

N°	COMANDO	EXPLICACION
1	Terraform graph | dot -Tsvg > graph.svg	Es utilizado en Terraform para generar un gráfico visual de la infraestructura definida en tu configuración.
2	Terraform state list	Se utiliza en Terraform para mostrar una lista de todos los recursos que están registrados en el estado de Terraform
3	Terraform state show aws_instance.public_instance	se utiliza en Terraform para mostrar información detallada sobre un recurso específico en el estado de Terraform aws_instance.public_instance es el recurso de ejemplo.
4	Terraform providers	Se utiliza para mostrar información sobre los proveedores (providers) de Terraform que están configurados en tu entorno de trabajo
5	Terraform output	Se utiliza para mostrar los valores de salida (outputs) definidos en tus archivos de configuración de Terraform.

6	Terraform show -json	Se utiliza en Terraform para mostrar el estado actual de la infraestructura gestionada por Terraform en formato JSON
7	Terraform validate	Se utiliza en Terraform para validar la sintaxis y la estructura de los archivos de configuración
8	Terraform fmt	Este comando formateará automáticamente tus archivos de configuración de Terraform para seguir las convenciones de estilo recomendadas por HashiCorp.

LIFECYCLES:

Los lifecycles (ciclos de vida) se refieren a las fases y acciones que Terraform puede llevar a cabo durante el ciclo de vida de un recurso. Estas fases y acciones se definen en los bloques de ciclo de vida dentro de la configuración de Terraform y pueden incluir acciones como la creación, actualización o eliminación de recursos, así como acciones específicas de provisionamiento o gestión de estado.