Terraform

kubernetes

Administrador de Sistemas Informáticos

Alejandro Domínguez Alcocer

PROYECTO - FCT

ÍNDICE

1. Objetivos Básicos

En este proyecto ampliaremos nuestra habilidades y conocimientos de Terraform y aprenderemos a realizar el uso de esta herramienta.

Existen muchas de estas Herramientas IaC, como Ansible, Chef y Pupped. Terraform es un poco diferente, porque permite realizar el seguimiento de todos los recursos desplegados, cosa que también veremos.

Aplicaremos las mejores practicas que HashiCorp recomienda, ya que los usuarios suelen buscar información fuera de su wiki, esto puede suponer un mal uso de la herramienta.

Como utilizarla en el ámbito empresarial, ya que puede beneficiar a los equipos de trabajo a ser mas eficientes y optimizados. También veremos todos los proveedores que contiene Terraform.

Realizaremos una practica donde pongamos todos estos conocimientos en funcionamiento. Para esta practica utilizaremos un clúster de Kubernetes como aprovisionamiento, y algunas herramientas que se explicaran a lo largo del proyecto. Todo esto claro esta controlado por Terraform.

2. Conceptos y Fundamentos teóricos

2.1 Infraestructura como código (IAC)

La infraestructura como código es escribir lo que quieres desplegar como código legible por humanos. En el caso de Terraform el código es muy fácil de leer y a la hora de buscar errores, cuando tienes grandes cantidades de código escrito.

En cambio, si estuviésemos buscando un error en una consola de AWS tendríamos que hacer clic por la diferentes ventanas, esperando el tiempo de carga para cada una de ellas, esto nos podría llevar a errores en los clics y desplegar recursos extras que no nos interesan.

Trabajar con código nos beneficia a la hora de rastrear las versiones del mismo, trabajando por ejemplo con GIT. Esto nos permite trabajar con un equipo y tener mas visibilidad a la hora de desplegar proyectos.

Esto se traduce en mas velocidad, menos coste para la empresa y la reducción del riesgo al desplegar la infraestructura ya que tenemos menos intervención humana, durante el despliegue de las mismas.

2.2 Ejemplo código IaC (Terraform)

El siguiente código que vamos a ver esta escrito en HashiCorp (HCL).

```
#Declaramos el proveedor
provider "aws" {}

#Creamos la VPC en es-east-2
resource "aws_instance" "example"{
    cidr_block = "192.168.1.0/24"
    tags = {
        Name = "vpc-example"
    }
}
```

Como vemos no necesitamos saber demasiado sobre los diferentes lenguajes de programación. En nuestro caso, estamos escribiendo un lenguaje mucho mas declarado, mas fácil de leer y entender.

Ahora mismo no he explicado nada sobre AWS o cualquier infraestructura en la nube, pero como podemos ver en el código, vamos a desplegar en la nube de AWS una VPC, y estamos declarando toda la configuración de forma sencilla.

En este caso estamos configurando Terraform y definiendo el rango de la dirección IP de la VPC y el nombre para poder identificarla.

Ahora, si por ejemplo, queremos desplegar la misma VPC en Google Cloud, simplemente deberemos cambiar el proveedor en el código, y a la hora de crearla, tendríamos la misma configuración. Esto es un pequeño ejemplo para que se entienda fácilmente.

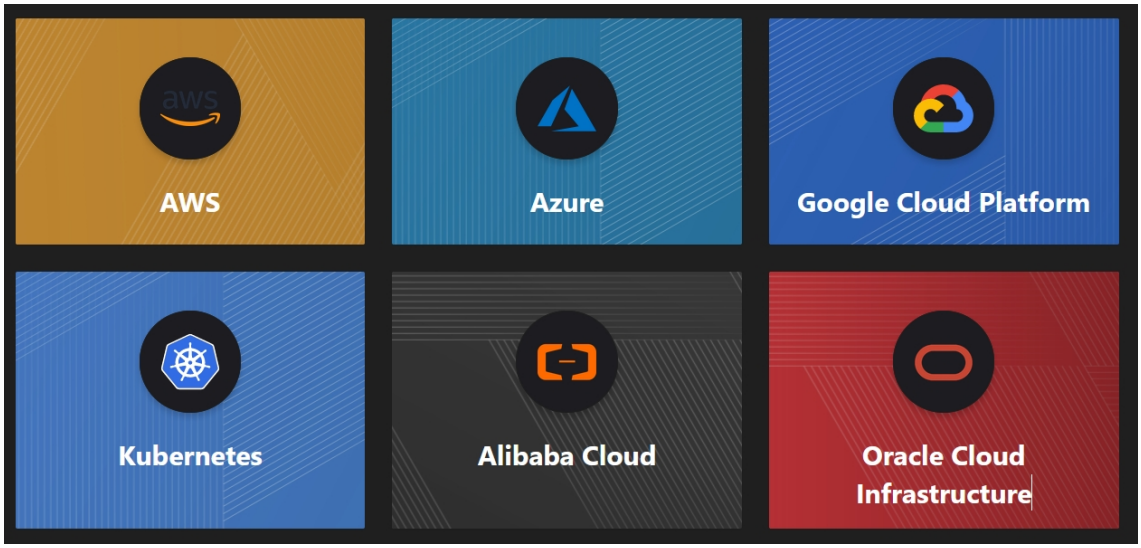
2.3 ¿Porque elegir Terraform?

Uno de los usos mas populares, es la manera tan fácil que tienen los desarrolladores de desplegar redes privadas virtuales, y trabajar con ellas. Esto ocurre, porque Terraform se encarga de traducir tu código y interactuar con las APIs de cada proveedor en la nube, o incluso programadores de recursos como Kubernetes.

Igual que ocurría antes a la hora de crear una VPC, también podemos definir las redes y utilizar la misma configuración en los diferentes proveedores.

Ademas Terraform, esta disponible para una gran cantidad de proveedores en internet, esto significa que podemos tener alta disponibilidad de un mismo servicio, utilizando dos proveedores diferentes, por si en algún momento falla podemos seguir gestionando nuestros recursos sin problemas.

Estos son los siguientes proveedores mas importantes hoy en día:



AWS, Alibaba, Azure, Vmware, Oracle, Google, estos serían los principales proveedores. También existen proveedores menos conocidos, pero no por ellos no vamos a nombrarlos. Estos serian DigitalOcean, Fastly, Gridscale, Heroku.

También dentro de los proveedores menos conocidos, en el ámbito de las bases de datos tenemos por ejemplo, MYSQL y InfluxDB. Esto le da una gran ventaja a Terraform, para ser utilizado por una gran mayoría de SysAdmin y DevOps, en multitud de infraestructuras y empresas.

Dentro de la comunidad de Terraform encontramos usuarios que se encargan de seguir creando interfaces con los diferentes proveedores, en el caso de que no tengan soporte.

Para seguir con las cosas buenas de Terraform tenemos el seguimiento de estado de los recursos, eliminación y creación de los mismos de una manera muy sencilla y la capacidad de automatizar el proceso.

No tenemos que preocuparnos de hacer cambios en el Terraform desplegado, cambiándolo directamente en el mismo archivo de configuración y aplicando los cambios, el solo se encarga de ponerlo todo en marcha. Esto se traduce en un ahorro de tiempo y dinero. Evitándonos posibles errores humanos, y tareas manuales laboriosas.

2.4 Terraform CLI Basic Commands

Vamos a ver la interfaz de línea de comandos de Terraform y la sintaxis de los comandos, que utilizaremos en la infraestructura durante la configuración de Terraform.

El comando **terraform** es el componente principal en la interfaz de linea de comandos, este acepta una variedad de sub-comandos. Podemos ver la lista de los principales comandos en su pagina principal:

```
init          Prepare your working directory for other commands
validate      Check whether the configuration is valid
plan          Show changes required by the current configuration
apply         Create or update infrastructure
destroy       Destroy previously-created infrastructure
```

Terraform permite el auto completado de comandos, para ellos deberemos ejecutar lo siguiente:

```
terraform -install-auto-complete
```

La mayoría de las veces se ejecutaría el comando terraform dentro de tu directorio raíz de módulos, donde están todos los archivos de configuración de Terraform. Pero si queremos ejecutar Terraform dentro de un script, deberíamos utilizar la opción **-chdir=ruta donde encontramos los módulos,** y luego el sub-comando que vayamos a ejecutar.

```
terraform init
```

Esto se utiliza para inicializar un directorio de trabajo que contiene los archivos de configuración de Terraform.

Es el comando que se debe ejecutar después de escribir la nueva configuración de Terraform, con este comando también podemos clonar una configuración desde el control de versiones.

```
terraform plan
```

Este comando es un simulacro antes de crear la configuración. Se mostrara por pantalla y podremos ver exactamente lo que módulos y servicios estará creando para poder tener un control antes de crear nada.

```
terraform apply <nombre_del_plan>
```

Después de comprobar la configuración, podremos ejecutar el comando **terraform apply**, esto aplicara todos los cambios en nuestra configuración. También podemos ejecutar un plan especifico con el mismo nombre.

```
Terraform destroy
```

En el caso de que nuestro entorno sea temporal o simplemente queremos borrarlo utilizaremos **Terraform destroy,** esto destruirá cualquiera infraestructura que fuese montada por el comando **Terraform apply**.

Ahora pasaremos a nombrar etiquetas u opciones durante la aplicación de un plan o su eliminación.

```
terraform plan -out <nombre_del_plan>
```

Esto hace que configuración salida por pantalla, se vuelque en un archivo para poder verificar la configuración de una manera mas fácil y sencilla, en el caso de que tengamos muchas lineas de código.

```
terraform plan -destroy
```

Con el siguiente comando podemos hacer un simulacro del borrado del plan, para asegurarnos de que los recursos borrados son los correctos.

```
terraform apply -target=nombre_del_recurso
```

Podemos también aplicar un cambio solo a un recurso especifico, en una configuración ya creada y no a toda la infraestructura.

```
terraform apply -var variable=<variable>
```

Con el siguiente comando especificamos el nombre de una variable y la variable para pasar a través de la linea de comandos.

```
terraform providers
```

Podemos sacar la lista de proveedores que esta utilizando en la configuración.

2.5 Explicación básica del lenguaje

Veremos como se compone el lenguaje de configuración de Terraform.

La parte principal del lenguaje de Terraform es declarar recursos. Estos representan objetos de infraestructura en la configuración. Los demás componentes del lenguaje solo existen para hacer la definición de los recursos mas flexible y para nuestra conveniencia.

Cuando hablamos de la configuración de Terraform, nos referimos a un documento completo en el lenguaje de Terraform que dice como gestionar la infraestructura. Puede depender de múltiples archivos y directorios igual que ocurre a la hora de configurar **Ansible** por ejemplo.

Podemos ver como es un bloque de código:

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

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

La sintaxis del lenguaje de configuración de Terraform, consiste en solo tres elementos básicos. En el primer lugar, tenemos los bloques, que son contenedores de objetos como los recursos. A continuación tenemos los argumentos, que asignan un valor a un nombre.

Estos aparecen dentro del bloque. Y al final, tenemos expresiones, que representan un valor, ya sea literal o mediante la referencia y combinación de otros valores.

Las expresiones, aparecen como valores de argumentos o dentro de otras expresiones El lenguaje de configuración de Terraform es declarativo, describe un objetivo, en lugar de los pasos para alcanzar el objetivo

Terraform se componen de una serie de archivos con su respectiva extensión. Tenemos dos tipos diferentes de extensiones. La extensión de archivo .tf que son los archivos principales del lenguaje y también podemos tener la combinación de .tf.json, que es una variante JSON del lenguaje. La codificación del texto de estos archivos debe ser UTF-8.

La configuración de Terraform se compone de directorios y módulos. Un modulo se compone unicamente de los archivos del directorio. El directorio de trabajo se considera el módulo raíz. Aquí es donde Terraform se ejecuta. La configuración de Terraform consiste en un módulo raíz y algunos módulos hijos. A la hora de ejecutarse busca dentro del directorio y luego cada uno va definiendo el conjunto distinto de objetos de configuración.

3. Instalación de Kubectl

1. Actualizamos el índice de paquetes de apt e instalamos los paquetes necesarios para usar Kubectl con el repositorio apt:

```
sudo apt update

sudo apt-get install -y apt-transport-https ca-
certificates curl
```

2. Descargamos la clave de firma pública de Google Cloud:

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes archive-
keyring.gpg https://packages.cloud.google.com/apt/doc/apt-
key.gpg
```

3. Agregamos el repositorio de Kubectl a apt:

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-
archive-keyring.gpg] https://apt.kubernetes.io/
kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
```

4.Actualizamos el índice de paquetes de apt con el nuevo repositorio e instale kubectl:

```
sudo apt-get update

sudo apt-get install -y kubectl
```

4. Instalación de Terraform

1. Descargamos el paquete binario Terraform mediante el comando wget:

```
wget -c
https://releases.hashicorp.com/terraform/0.13.5/terraform_
0.13.5_linux_amd64.zip
```

2. Descomprimimos el archivos descargado:

```
unzip terraform_0.13.5_linux_amd64.zip
```

3. Colocamos el binario de Terraform en el directorio raiz del sistema.

```
sudo mv terraform /usr/sbin/
```

4. Verificamos la versión que tenemos instalada:

terraform versión

```
servidor@server:~/terraform$ terraform version
Terraform v1.3.5
on linux_amd64
+ provider registry.terraform.io/hashicorp/kubernetes v2.16.0

Your version of Terraform is out of date! The latest version
is 1.3.6. You can update by downloading from https://www.terraform.io/downloads.html
servidor@server:~/terraform$
```

5. Instalación de Docker

1. Para poder crear los servicios necesarios en Kubernetes, necesitamos los contenedores que proporciona Docker, para ello procedemos a instalarlo. Ejecutamos el siguiente comando para añadir la clave GPG para Docker.

```
curl -fsSL https://download.docker.com/linux/debian/gpg |
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-
keyring.gpg
```

2. Después, añadimos el repositorio de Docker, utilizando el siguiente comando.

```
echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg]
https://download.docker.com/linux/debian \
 $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

3. Ahora ejecuta el comando ‘apt update‘ para actualizar/refrescar todos los repositorios disponibles.

```
apt update
```

4. Ahora estás listo para instalar Docker en Debian 11 Bullseye.

```
sudo apt install docker-ce docker-ce-cli containerd.io
```

5. Habilitamos el servicio de docker.

```
sudo systemctl is-enabled docker
```

6. Habilitamos el servicio.

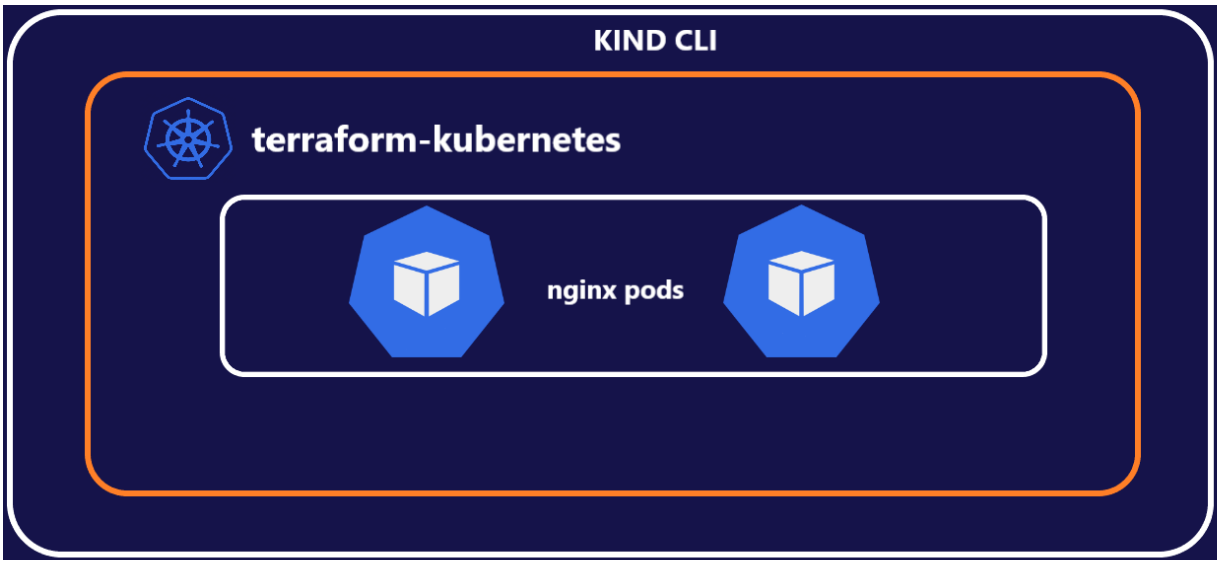
```
sudo systemctl is-enabled containerd
```

7. Con este comando verificamos el estatus del servicio docker.

```
sudo systemctl status docker containerd
```

6. Gestión de un clúster local de Kubernetes utilizando Kind y Terraform

Durante la siguiente practica, crearemos un nuevo clúster local de Kubernetes utilizando Kind y Terraform para aprovisionar un servicio nginx. En el siguiente diagrama, tenemos un clúster de Kubernetes con 2 nodos, que están ejecutando el servicio de NGINX. Esto sera creado desde Terraform y el cluster sera gestionado por KIND.



El primer paso sera crear un directorio donde trabajar:

```
servidor@servidor:~$ mkdir terraform-kubernetes
servidor@servidor:~$ cd terraform-kubernetes/
servidor@servidor:~/terraform-kubernetes$ ls -l
total 0
servidor@servidor:~/terraform-kubernetes$
```

Ahora tendremos que instalar KIND, para poder crear el cluster. Lo descargaremos desde su propio repositorio:

```
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.17.0/kind-linux-amd64
```

```
servidor@servidor:~/terraform-kubernetes$ curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.17.0/kind-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    97  100    97    0     0    203      0 --:--:-- --:--:-- --:--:--   203
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 6766k  100 6766k    0     0  2585k      0  0:00:02  0:00:02 --:--:-- 3634k
servidor@servidor:~/terraform-kubernetes$ ls -l
total 6768
-rw-r--r-- 1 servidor servidor 6929103 dic 14 22:17 kind
servidor@servidor:~/terraform-kubernetes$
```

Al binario descargado le damos permiso de ejecución.

```
chmod +x ./kind
```

```
servidor@servidor:~/terraform-kubernetes$ chmod +x ./kind
servidor@servidor:~/terraform-kubernetes$ ls -lh
total 6,7M
-rwxr-xr-x 1 servidor servidor 6,7M dic 14 22:17 kind
servidor@servidor:~/terraform-kubernetes$
```

Movemos el binario dentro de la raíz del sistema.

```
sudo mv ./kind /usr/local/bin/kind
```

```
servidor@servidor:~/terraform-kubernetes$ sudo mv ./kind /usr/local/bin/kind
servidor@servidor:~/terraform-kubernetes$ kind help
kind creates and manages local Kubernetes clusters using Docker container 'nodes'

Usage:
  kind [command]

Available Commands:
  build       Build one of [node-image]
  completion  Output shell completion code for the specified shell (bash, zsh or fish)
```

El siguiente paso seria descargar el archivo de configuración de Kind, que tenemos guardado en nuestro repositorio Git. Descargamos el directorio completo y luego movemos el archivo que nos interesa.

```
Git clone https://github.com/Domin01/terraform-kubernetes
```

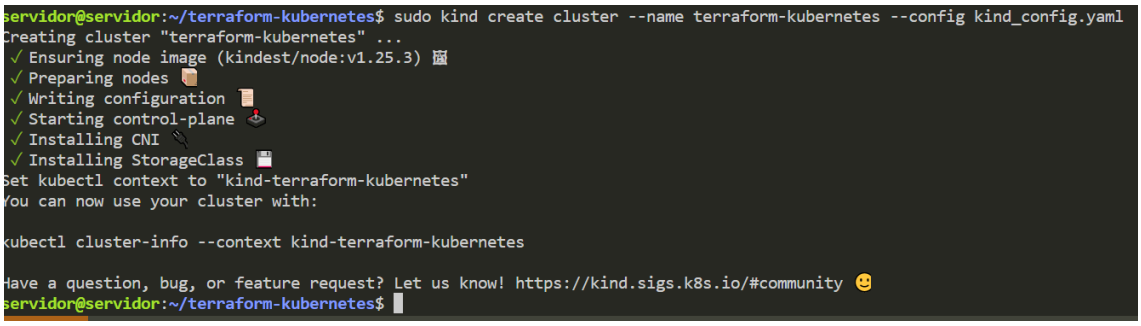
```
servidor@servidor:~/terraform-kubernetes$ git clone https://github.com/Domin01/terraform-kubernetes.git
Clonando en 'terraform-kubernetes'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 18 (delta 5), reused 0 (delta 0), pack-reused 0
Recibiendo objetos: 100% (18/18), 4.56 KiB | 4.56 MiB/s, listo.
Resolviendo deltas: 100% (5/5), listo.
servidor@servidor:~/terraform-kubernetes$ ls -L
terraform-kubernetes
servidor@servidor:~/terraform-kubernetes$ mv terraform-kubernetes/kind_config.yaml .
servidor@servidor:~/terraform-kubernetes$ ls -L
kind_config.yaml  terraform-kubernetes
servidor@servidor:~/terraform-kubernetes$
```

Si abrimos el archivo, podemos ver que es una configuración para nuestro cluster. Simplemente lo que estamos haciendo es crear el cluster y definir el puerto del mismo, ademas de una regla para que escuche en todas las dirección.

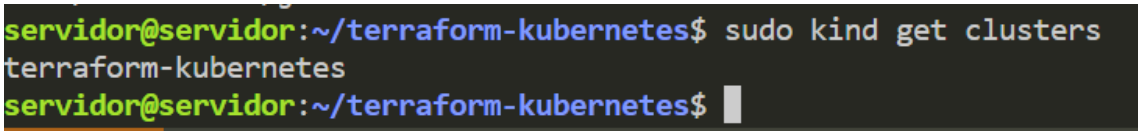
```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 30201
    hostPort: 30201
    listenAddress: "0.0.0.0"
```

Vamos a proceder a crear el cluster con el siguiente comando:

```
sudo kind create cluster --name terraform-kubernetes --
config kind_config.yaml
```

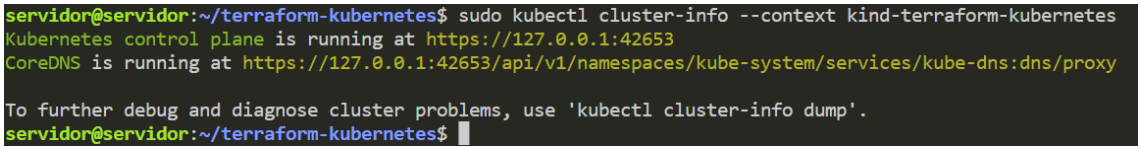


Para verificar que se ha creado correctamente ejecutamos el siguiente comando para ver el estado del cluster.



Ahora verificamos que kubectl puede ver nuestro cluster, ejecutamos el siguiente comando:

```
sudo kubectl cluster-info --context kind-terraform-
kubernetes
```



Nuestro siguiente paso sera trabajar con las credenciales que tiene el cluster incorporadas, para ello bajamos el archivo de configuración donde estarán las variables, para que Terraform pueda crear los objetos dentro del cluster.

```
servidor@servidor:~/terraform-kubernetes$ mv terraform-kubernetes/kubernetes.tf .
servidor@servidor:~/terraform-kubernetes$ ls -L
kind_config.yaml  kubernetes.tf  terraform-kubernetes
servidor@servidor:~/terraform-kubernetes$
```

Dentro de este archivo estamos declarando nuestro proveedor, que en este caso es Kubernetes.

```
terraform {
  required_providers {
    kubernetes = {
      source = "hashicorp/kubernetes"
    }
  }
}
```

También estamos declarando variables, como host, certificado de cliente, nuestra clave del cliente y nuestro certificado CA de cluster.

```
variable "host" {
  type = string
}

variable "client_certificate" {
  type = string
}

variable "client_key" {
  type = string
```

```
}

variable "cluster_ca_certificate" {
  type = string
}
```

Justo mas abajo del documento se declaran las variables, que se recogeran en el documento que creamos mas adelante.

```
provider "kubernetes" {
  host = var.host
  client_certificate     = base64decode(var.client_certificate)
  client_key             = base64decode(var.client_key)
  cluster_ca_certificate =
base64decode(var.cluster_ca_certificate)
}
```

Con el siguiente comando sacamos las credenciales del cluster, esta información la guardaremos en un archivo llamado terraform.tfvars. Donde se guardara el host, certificado de cliente, nuestra clave del cliente y nuestro certificado CA de cluster.

```
sudo kubectl config view --minify --flatten --
context=kind-terraform-kubernetes
```

```
servidor@servidor:~/terraform-kubernetes$ sudo kubectl config view --minify --flatten --context=kind-terraform-kubernetes
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMvakNDQWVhZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBR
RHVnpNQjRYRFRJeU1USXhORE14TkRFeE4xb1hEVE15TVRJeE1USXhOREV4TjFvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh
TGVQCmovblY3WWNiOXhHUWwwYVR0NUpmckZSc1NTYVUxc3VVSlBIRzNQUGl1S1pOb041azN4NUFLOGJNNHFjMGx4eWQKWGV5dEwvbGJMMzY4QW5nL0h2cHJnVlk3R
pSTUQxUysWTWhaYwpDTE9yQjZ0WC9pVDFxUldOY3BWeVp6OXlseUxCbTgwNi9relgwVllrUndQV1VoU2dxM09JZlBNaS91SzVhelRSCkdPOW1kYnVwaU1MTkRiOXH
```

Crearemos el fichero **terraform.tfvars** con las siguientes variables y la información recopilada anteriormente.

En primer lugar, vamos a crear la variable host = “”. Y vamos a copiar nuestra dirección de host. A continuación, vamos a crear el client_certificate = “”. Vamos a pegar los datos de nuestro cliente. Después vamos a crear la variable client_key = “”. Pegamos la información extraída. Y por último, creamos la variable cluster_ca_certificate = “” con su información correspondiente.

```
1  #Variables Terraform
2  host = "https://127.0.0.1:42653"
3  client_certificate = "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURJVENDQWdt
4  client_key = "LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NB
5  cluster_ca_certificate ="LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMvakNDQ
```

Guardamos el documento. Ahora deberíamos tener dos archivos uno con la configuración principal de Terraform llamado **kubernetes.tf** y otro archivo con las variables llamado **terraform.tfvars**.

Ahora vamos a formatear los archivos de configuración al formato legible por terraform.

```
terraform fmt
```

```
servidor@servidor:~/terraform-kubernetes$ terraform fmt
servidor@servidor:~/terraform-kubernetes$
```

Después inicializaremos el directorio, ejecutando el comando **terraform init.**

```
* hashicorp/kubernetes: version = "~> 2.16.1"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
servidor@servidor:~/terraform-kubernetes$
```

El siguiente paso sera empezar a programar el despliegue. Creamos el cluster con Terraform, descargamos del repositorio el archivo **resource_nginx.tf.** Dentro de este archivo encontramos el despliegue de Kubernetes.

```
resource "kubernetes_deployment" "nginx" {
 metadata {
  name = "nginx"
  labels = {
   App = "Nginx"
  }
 }
```

Mas abajo aparecen las replicas especificadas, en este caso serán 2 y el puerto por donde mandara información fuera del contenedor, puerto 80 y por supuesto la imagen del contenedor como hemos dicho antes utilizaremos nginx.

```
 spec {

    replicas = 2
    selector {
      match_labels = {
        App = "Nginx"
      }
    }
    template {
      metadata {
        labels = {
          App = "Nginx"
        }
      }
      spec {
        container {
```

```
            image = "nginx"
            name  = "proyecto-nginx"

            port {
              container_port = 80
            }
          }
        }
      }
    }
}
```

A continuación, aplicaremos la configuración para programar el despliegue de NGINX. Pero antes que nada verificamos si la configuración esta bien establecida con el siguiente comando:

```
terraform validate
```

```
servidor@servidor:~/terraform-kubernetes$ terraform validate
Success! The configuration is valid.

servidor@servidor:~/terraform-kubernetes$
```

Ahora realizaremos el despliegue con el comando **terraform apply**.

```
        }
      }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Ponemos el parámetro **"yes"** y empezara a crearse automáticamente.
Verificamos que se ha creado correctamente.

```
kubernetes_deployment.nginx: Creating...
kubernetes_deployment.nginx: Creation complete after 7s [id=default/nginx]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
servidor@servidor:~/terraform-kubernetes$ sudo kubectl get deployments
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   2/2     2            2           18s
servidor@servidor:~/terraform-kubernetes$
```

Ahora para poder acceder desde fuera necesitamos un recurso para nuestro desligue de NGINX, en este caso el recurso NodePort que expondrá nuestro cluster al exterior con el numero de puertos 30201.

Descargamos el archivo de configuración llamado **service_nginx.tf.**

```
resource "kubernetes_service" "nginx" {
  metadata {
    name = "nginx-example"
  }
  spec {
    selector = {
      App =
kubernetes_deployment.nginx.spec.0.template.0.meta
data[0].labels.App
    }
    port {
      node_port   = 30201
      port        = 80
      target_port = 80
    }

    type = "NodePort"
  }
}
```

En la configuración podemos ver que el puerto que esta reenviando, es el puerto 80 al puerto 30201. Lo siguiente seria aplicar la configuración desde Terraform. Utilizando **terraform apply**.

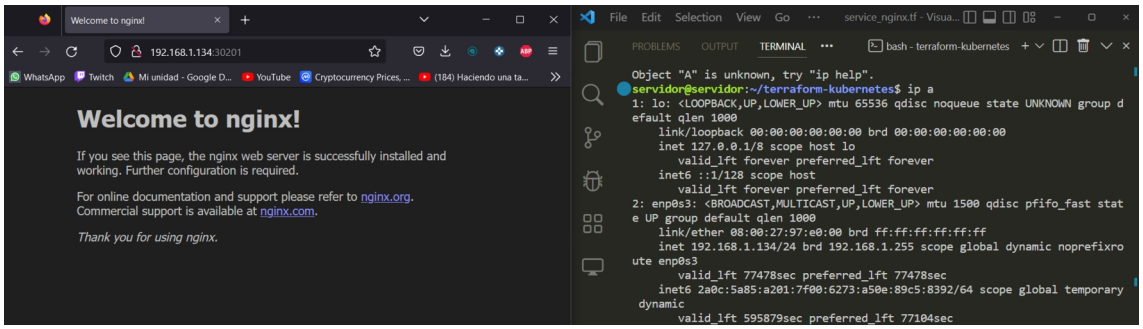
```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

kubernetes_service.nginx: Creating...
kubernetes_service.nginx: Creation complete after 0s [id=default/nginx-example]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
servidor@servidor:~/terraform-kubernetes$
```

Ahora verificamos que el servicio esta creado correctamente, ejecutando:

```
kubectl get services
```

```
servidor@servidor:~/terraform-kubernetes$ sudo kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        20h
nginx-example   NodePort    10.96.72.109    <none>        80:30201/TCP   3m14s
servidor@servidor:~/terraform-kubernetes$
```

Para verificar que todo esta funcionando correctamente entraremos en el navegador y pondremos la dirección IP de la maquina con el numero de puertos.



Ahora para comprobar que con el recurso desplegado, se puede cambiar la configuración, vamos a cambiar el numero de despliegues a 6 y vamos a aplicar los cambios.

```
spec {
    replicas = 6
    selector {
      match_labels = {
        App = "Nginx"
      }
    }
```

Aplicamos los cambios y como vemos en la descripción del despliegue aparece lo siguiente:

```
  ~ spec {
        min_ready_seconds         = 0
        paused                    = false
        progress_deadline_seconds = 600
      ~ replicas                  = "2" -> "6"
        revision_history_limit    = 10
```

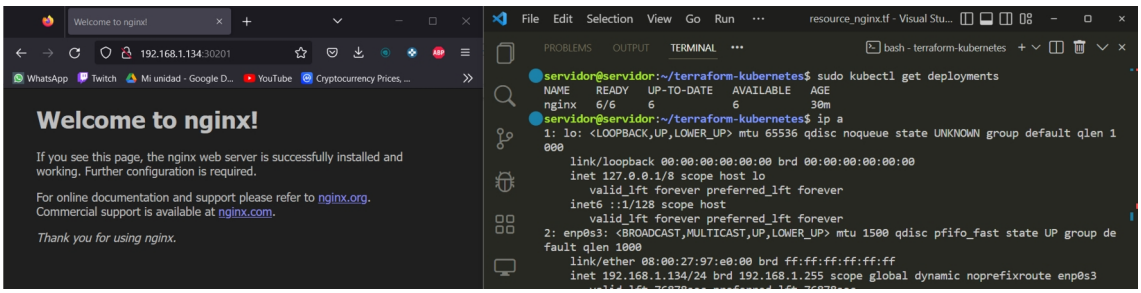
Nos avisa que el numero de replicas a cambiado y ahora el plan que tenemos no es para añadir, es para realizar cambios en la configuración.

```
Plan: 0 to add, 1 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

kubernetes_deployment.nginx: Modifying... [id=default/nginx]
kubernetes_deployment.nginx: Modifications complete after 7s [id=default/nginx]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

Verificamos que realmente el numero de replicas a cambiado.

```
servidor@servidor:~/terraform-kubernetes$ sudo kubectl get deployments
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   6/6     6            6           27m
servidor@servidor:~/terraform-kubernetes$
```

Volvemos a verificar que el recurso sigue funcionando:



El ultimo paso seria eliminar el proyecto y ver como se elimina por partes, el cluster y los servicios creados, utilizamos el siguiente comando y escribimos "**yes**", para confirmar el borrado.

```
terraform destroy
```

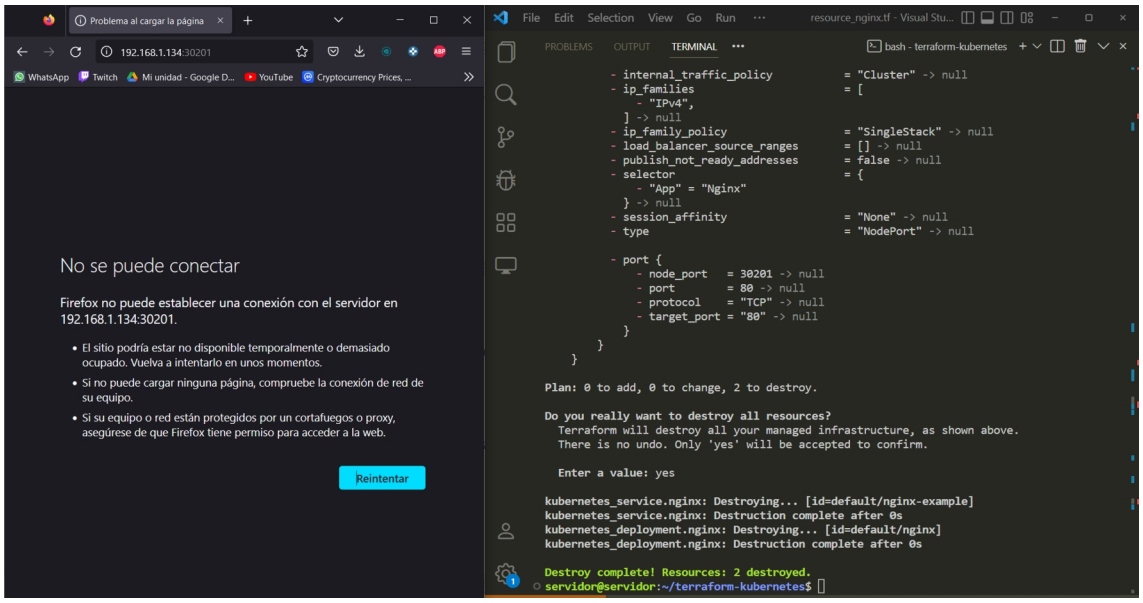
```
        }
    }

Plan: 0 to add, 0 to change, 2 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

Se eliminara el servicio y los deployments.

```
# kubernetes_service.nginx will be destroyed
- resource "kubernetes_service" "nginx" {
```

```
# kubernetes_deployment.nginx will be destroyed
- resource "kubernetes_deployment" "nginx" {
```

Verificamos que después de destruir el plan, ya no podríamos acceder al recurso web.



7.Conclusiones y Propuestas

En conclusión, Terraform es una buena herramienta para trabajar en el sector de la nube hoy en día, ofrece mas posibilidades que los competidores y se puede desplegar en multitud de proveedores de la nube.

Su lenguaje es simple y se puede entender echando un vistazo al código, existe muchísima información en la pagina principal de la herramienta y la comunidad que tiene es muy extensa. Esto asegura un buen ecosistema para buscar ayuda, en caso de tener problemas en un futuro. Podemos encontrar una gran cantidad de videos, cursos, paginas web, que ofrecen muchísima información sin ser oficial.

Esta herramienta soluciona los problemas derivados por trabajar en la nube, te ayuda a mejorar tu infraestructura, los errores que pueden ocurrir durante el despliegue de los servicios y controlar las versiones de los planes que lanzamos, haciendo posible seguir y documentar las lineas de producción de una manera fácil y sencilla.

Para concluir como posible mejora, seria recomendable preparar un buen escenario, en una de las nubes mas famosas en internet, como podrían ser Google Cloud, Amazon Web Service o Microsoft Azure. Esta practica se realizo utilizando pocos recursos, porque se quería enseñar la simpleza de la herramienta y como crear un recurso web en sencillos pasos.

8. Bibliografía

(Proveedores) https://registry.terraform.io/browse/providers

(Configuración Proveedores)https://developer.hashicorp.com/terraform/language/providers

(Terraform CLI) https://developer.hashicorp.com/terraform/cli

(Blog) https://ernestovazquez.es/posts/terraformaws/

(Crear una instancia)https://www.adictosaltrabajo.com/2020/06/19/primeros-pasos-con-terraform-crear-instancia-ec2-en-aws/

(Comandos Básicos)https://developer.hashicorp.com/terraform/cli/commands

(GitHub) https://github.com/Domin01/terraform-kubernetes

(Instalar Kind) https://kind.sigs.k8s.io/docs/user/quick-start/#installation

(Instalación Kubernetes)https://kubernetes.io/es/docs/tasks/tools/included/install-kubectl-linux/

(Instalación docker) https://aprendiendoavirtualizar.com/instalar-docker-en-debian-11/