INTRODUCCIÓN A TERRAFORM

Creando grandes infraestructuras

José Manuel Redondo López Proyecto "*L-62 Princesa de Asturias*" v1.2




Fuente: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

ÍNDICE

< Ir al Índice

Providers >

Introducción y Terminología

Conceptos básicos de Terraform

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿Qué vamos a ver en este apartado?

José Manuel
Redondo López

- Este tema final de la asignatura es una modesta introducción a Terraform
- Se trata de que entiendas las bases del propósito de esta tecnología
  - Sin necesidad de infraestructura externa y que pueda tener un coste
  - Coordinada con el resto del curso
  - Para poder experimentar con ella en local
- El objetivo es que entiendas para qué sirve Terraform, puedas decidir si te es de aplicación y estudiar más a partir de aquí si lo necesitas
- En este primero apartado veremos los conceptos y propósitos principales de la tecnología, para familiarizarnos con ellos
  - Especialmente el gran peso que tienen los providers en Terraform

HashiCorp

https://www.hashicorp.com/

José Manuel
Redondo López

- **Empresa que distribuye una serie de productos para gestionar infraestructuras en una gran variedad de escenarios**
- **Tanto cloud como locales**
  - Packer (https://www.packer.io/): Visto al final del tema de Vagrant
    - Ejemplos con Docker y AWS: https://learn.hashicorp.com/packer
    - Uso de Packer con imágenes Docker: https://www.packer.io/docs/builders/docker
  - Vagrant: Visto en el tema correspondiente
  - Nomad (https://www.nomadproject.io/): Es un **orquestador** para desplegar aplicaciones en diferentes entornos
    - Clouds públicas, privadas...
  - Consul (https://www.consul.io/): Usado para descubrir servicios, entre otros
  - Vault (https://www.vaultproject.io/): Herramienta para guardar secretos de aplicaciones
  - Terraform: Lo que vamos a ver en esta presentación

Introducción

José Manuel
Redondo López

- **Terraform es otra herramienta de Infrastructure as Code (IaC)**
  - https://www.terraform.io/
  - Permite construir, cambiar y versionar **infraestructuras de MVs o contenedores** eficientemente
  - Tanto en despliegues locales como en nube
  - Incluso trabajando con **varios proveedores de nube** en una misma infraestructura
    - ¡Entornos multi-cloud!
- **Además, permite ver lo que se va a hacer antes de hacerlo**
  - Por lo que esas operaciones **son más seguras**
  - Tiene en cuenta que los recursos en nube pueden tener un coste elevado
    - Tener la opción de revisar lo que se va a desplegar en ellos antes de hacerlo es muy importante
- **Incluye además otros elementos de las infraestructuras**
  - **De bajo nivel**: instancias de computación, almacenamiento, redes, etc.
  - **De alto nivel**: DNS, Software as a Service, etc.
- **Trabaja con proveedores existentes y permite desarrollar propios (extensible)**

Características

José Manuel
Redondo López

- **Infrastructure as Code:** Describe la infraestructura a construir usando su propio lenguaje de configuración de alto nivel (**HCL**)
  - Es sencillo de entender: https://www.terraform.io/docs/language/index.html
  - Como con **Vagrant**, hace muy sencillo compartir y reutilizar infraestructura y hacer nuevas versiones
  - **HCL** será el lenguaje futuro de otros productos **Hashicorp**, **Vagrant** incluido
    - https://www.hashicorp.com/blog/toward-vagrant-3-0
- **Execution Plans:** Planes para describir lo que se va a hacer antes de hacerlo,
  - Pidiendo **confirmación al usuario** antes de hacer los cambios
  - Esto permite **revisar los cambios** antes que se cree, actualice o borre algo, a diferencia de **Vagrant**
- **Resource Graph:** Construye un gráfico de los recursos a crear
  - Con él puede crear en paralelo aquellos que no dependen de otros, **aumentando la eficiencia**
- **Change Automation:** Los cambios causados por los ficheros que configuran la infraestructura se hacen de forma incremental
  - Respetando las dependencias y con **mínima interacción humana**

Terraform vs Vagrant

José Manuel
Redondo López

- **Vagrant está orientado a crear entornos de desarrollo**
  - Implementa una serie de características de alto nivel que Terraform no implementa
    - Carpetas compartidas, redes automatizadas, túneles HTTP...
  - Está pensado para trabajar **en entornos locales** con un número limitado de MV
- **Terraform se pensó para crear infraestructuras**
  - Permite describir infraestructuras complejas que existirán de manera local o remota
  - Está pensado para que esa infraestructura **cambie y evolucione** con el tiempo
  - Y también para trabajar con proveedores de nube como AWS
  - Permitiendo manejar infraestructuras enormes que incluso usen los servicios de varios proveedores
- **También está relacionado con Ansible**
  - https://intellipaat.com/blog/terraform-vs-ansible-difference/

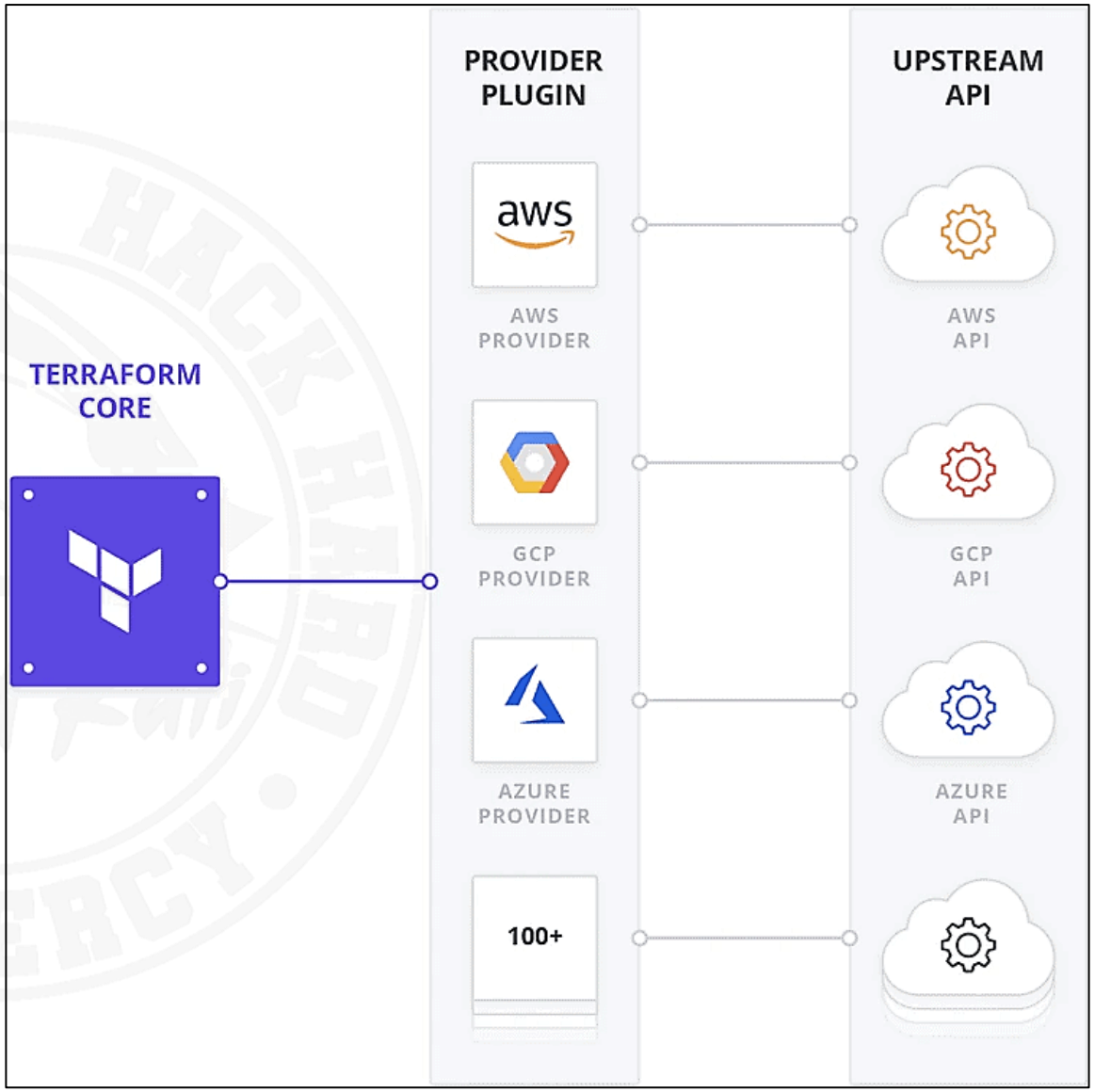
Providers de Terraform

Los elementos que permiten realmente trabajar con él

PROVIDERS

José Manuel
Redondo López

- **Terraform solo puede hacer operaciones a través de plugins llamados Providers**
  - Con ellos puede interactuar con proveedores de nube varios, proveedores SaaS, APIs...
- **Cada configuración de Terraform debe declarar 1º que proveedores necesita**
  - Con ello Terraform los instalará y podrá usar los elementos que definen
- **Algunos necesitan además configuración adicional**
  - Ej.: URL o región de la nube en donde van a trabajar



Fuente: K21academy.com

¿Qué hacen los Providers?

José Manuel
Redondo López

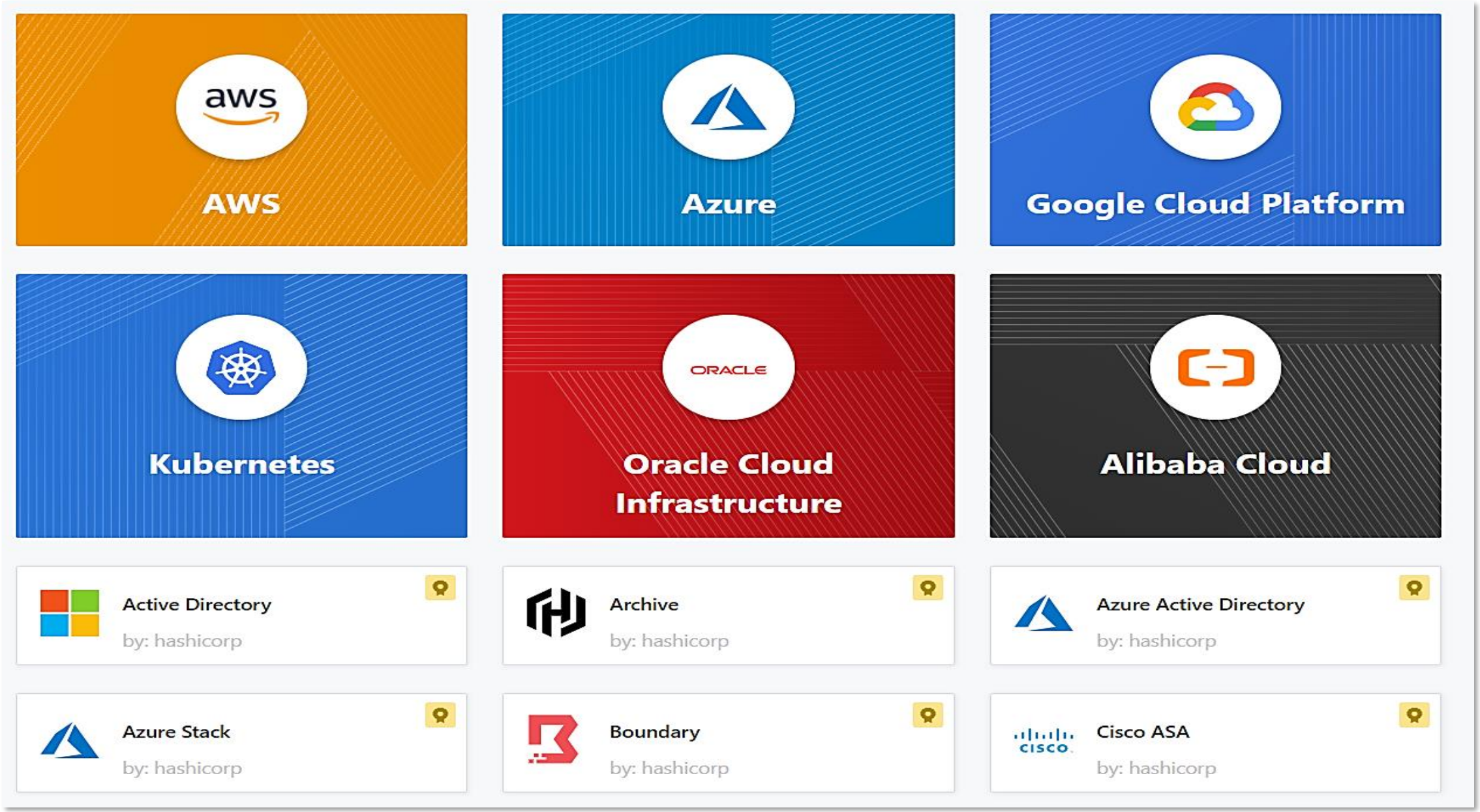
- **Cada plugin fundamentalmente añade dos cosas para usar en el lenguaje HCL**
  - Un conjunto de **tipos de recursos** (resource types)
    - https://www.terraform.io/docs/language/resources/index.html
  - Un conjunto de **orígenes de datos** (data sources) que Terraform puede manejar:
    - https://www.terraform.io/docs/language/data-sources/index.html
- **Sin los providers, Terraform no puede manejar NADA**
  - Es decir, **necesitamos un provider obligatoriamente**
  - Cada provider **te da acceso a recursos y orígenes de datos particulares suyos**
    - Que dependen de los conceptos manejados por la infraestructura subyacente
    - No es lo mismo un provider de Amazon que uno de Google Cloud
      - ¡Ambas nubes manejan conceptos con nombres y propiedades diferentes!
    - Es decir, define **los elementos con los que se puede trabajar** (nombres, clases y propiedades)
      - Ej.: No habrá los mismos resource types en un provider AWS que en uno de Docker
  - Además, pueden incorporar utilidades adicionales para manejar mejor las infraestructuras

Dónde encontrar providers y cómo usarlos

José Manuel
Redondo López

- **Se instalan independientemente de Terraform**
  - Por tanto, **deben descargarse aparte**
- **En el Terraform Registry podemos encontrar providers para las plataformas en nube más conocidas**
  - https://registry.terraform.io/browse/providers
  - Además, también incluyen su **documentación completa**
    - Con todos sus resource types, data sources, parámetros...
  - Los hay directamente **creados por Hashicorp**, de **terceros** o creados por **la comunidad**
  - **La documentación está versionada**
    - Cada versión de un provider tiene una documentación independiente
    - Por si una infraestructura usase **una versión anterior** con un API distinta
    - Lo que ofrecen los providers puede cambiar con las distintas versiones y algunas propiedades y elementos se pueden "deprecar"
  - Por tanto, **es imprescindible consultar la documentación** de un provider para poder usarlo

Terraform Registry



José Manuel
Redondo López

Documentación de los providers más populares

José Manuel
Redondo López

- Hay un plugin para cada proveedor de nube que te puedas imaginar
- AWS
  - https://registry.terraform.io/providers/hashicorp/aws/latest/docs
- Microsoft Azure
  - https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs
- Google Cloud
  - https://registry.terraform.io/providers/hashicorp/google/latest/docs
- Kubernetes
  - https://registry.terraform.io/providers/hashicorp/kubernetes/latest/docs
- Oracle Cloud
  - https://registry.terraform.io/providers/hashicorp/oci/latest/docs

COMO USAR LOS PROVIDERS

José Manuel
Redondo López

- **Para usar un proveedor y sus recursos es necesario aportar la siguiente información, siguiendo las indicaciones que se dan en su documentación**
  - **Requisitos**: Instrucciones acerca de cómo declarar el provider para que Terraform lo instale
    - https://www.terraform.io/docs/language/providers/requirements.html
  - **Configuración**: Cómo modificar las opciones del provider:
    - https://www.terraform.io/docs/language/providers/configuration.html
  - **Archivo de bloqueo de dependencias**: Que indica a Terraform qué versiones concretas del provider debe usar
    - https://www.terraform.io/docs/language/dependency-lock.html
    - **Recuerda**: **Un cambio de versión puede significar un cambio en las API** o recursos que declara
      - Y, por tanto, hacer que configuraciones existentes se vuelvan incompatibles
      - Con esto evitamos este problema
      - Se prioriza la estabilidad de los proyectos existentes
    - Migrar de versión algo en producción no se puede hacer a la ligera
      - Forzar una migración por una actualización de Terraform puede romper un sistema
      - ¡Necesitas planificación!

Instalación de providers

José Manuel
Redondo López

- En la siguiente sección veremos que para crear una infraestructura Terraform se crea un directorio de trabajo
  - Donde estarán **los ficheros que describen la infraestructura**
- En los ficheros de este directorio debe estar **la declaración del proveedor**
  - De acuerdo al procedimiento especificado en su documentación
- Así Terraform podrá descargarlo automáticamente del Terraform Registry
  - O cargarlo localmente si ya se ha descargado alguna vez o si se le da una dirección local
  - Existe una **caché de plugins** para evitar descargas que se puede activar con `plugin_cache_dir`
    - https://www.terraform.io/docs/cli/config/config-file.html
- Más información acerca de instalación de plugins y control de versiones instaladas
  - https://learn.hashicorp.com/tutorials/terraform/provider-versioning?in=terraform/configuration-language

TIPOS DE PROVIDERS SEGÚN QUIEN LOS CREA Y/O MANTIENE

José Manuel
Redondo López

- Uno no debe usar simplemente cualquier **provider** de **Terraform**...

Tipo	Descripción	Namespace
Official	**Providers** oficiales creados y mantenidos por **HashiCorp** (**usar estos preferentemente para evitar malware o errores de seguridad**)	hashicorp
Verified	**Providers** verificados creados y mantenidos por terceros. **Hashicorp** ha verificado la autenticidad de dicho tercero, que además pertenece al programa de **partners** https://www.hashicorp.com/ecosystem/become-a-partner/	El nombre de la organización de terceros, Ej. mongodb/mongodbatlas
Community	**Providers** creados por la comunidad y publicados en el **Terraform Registry** por mantenedores (individuales o en grupo) de la comunidad **Terraform**	Nombre del mantenedor o del grupo de mantenedores
Archived	**Providers** oficiales o verificados que ya no tienen mantenedor por algún motivo (Si tienes algo basado en ellos, vete pensando en migrar...)	**hashicorp** o el nombre de la organización de terceros

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

Introducción

- *¿Entiendes cuál es el propósito de **Terraform** y las principales ventajas que ofrece en un contexto de nube?*
- *¿Crees que has entendido correctamente el concepto de **execution plan** y de **resource graph**?*
- *¿Eres consciente de que los cambios a un fichero **Terraform** hacen cambios incrementales a la infraestructura (se mantiene lo que había si sigue siendo válido, se crea solo lo nuevo)?*
- *¿Tienes claro los distintos escenarios que cubren **Terraform** y **Vagrant**?*

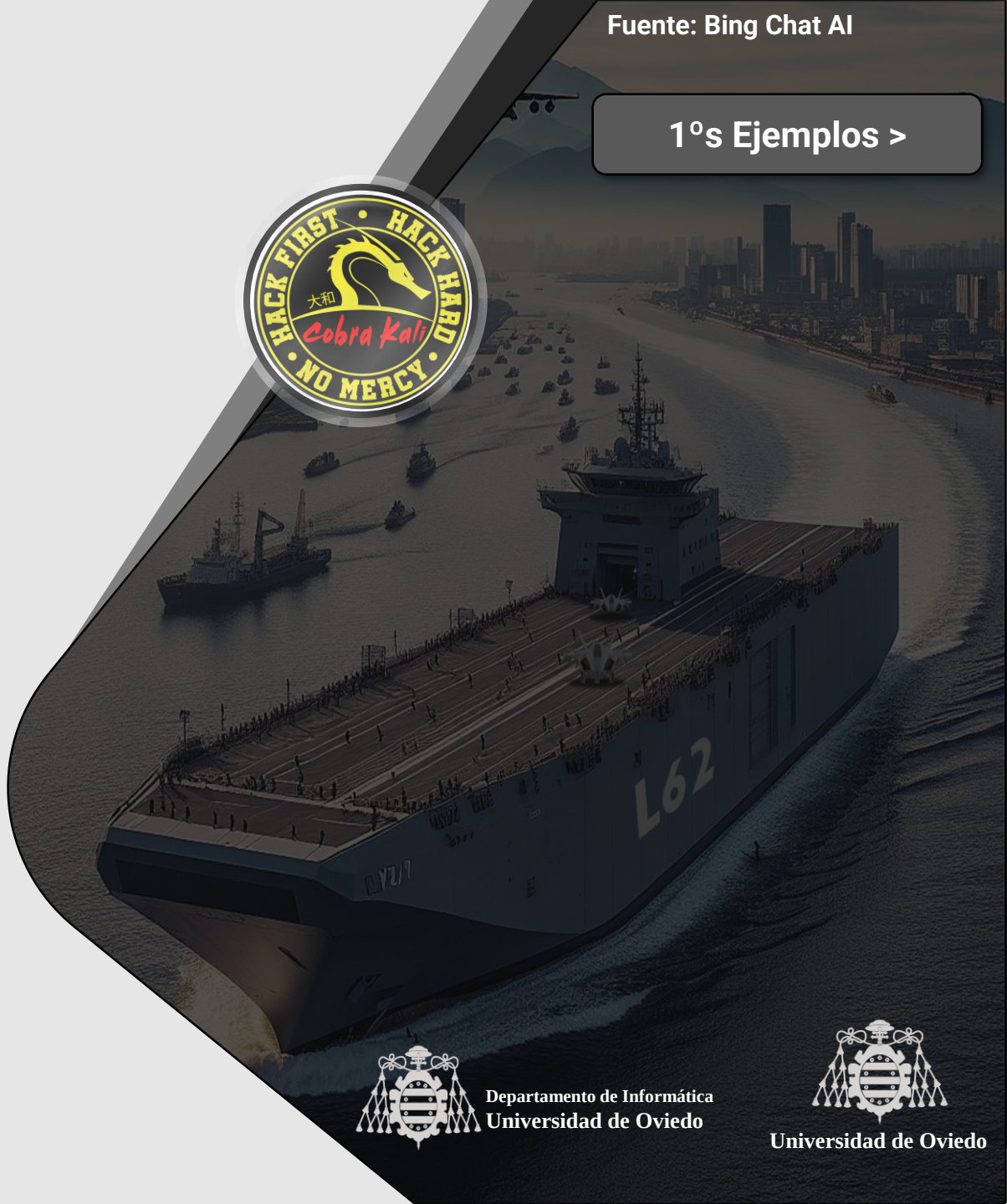
Providers

- *¿Entiendes que **Terraform** hace todo a través de sus distintos **plugins**, llamados **providers**?*
- *¿Entiendes que cada plugin te da acceso a un conjunto de tipos de recursos y orígenes de datos distinto, vinculado con la plataforma de nube o local que representan?*
- *¿Entiendes por qué entonces leer la documentación de cada plugin es imprescindible?*
- *¿Entiendes lo que se puede encontrar en el **Terraform Registry**?*
- *¿Entiendes los tres pasos necesarios para instalar y usar cualquier **provider**?*

< Ir al Índice

Fuente: Bing Chat AI

1ºs Ejemplos >



Terraform y Docker

Puesta en marcha de una infraestructura Terraform en la máquina local

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Esta sección contiene los elementos necesarios para empezar a usar Terraform con un provider Docker**
  - Explicando antes **por qué hemos elegido Docker** y no un sistema de nube modernos
  - Veremos un **equivalente a las operaciones comunes** que vimos en el tema de Docker
  - Para ello veremos **ejemplos funcionales** explicados

¿Por qué probamos con Docker?

José Manuel
Redondo López

- Es cierto que **Terraform** está pensado para trabajar con proveedores de nube
  - Recuerda, incluso varios en la misma infraestructura
- Pero usar proveedores de nube **puede incurrir en gastos innecesarios** para vosotros si no tenéis cuidado a la hora del uso de recursos
  - Muchos obligan a **introducir un medio de pago incluso aunque tengan un "free tier"**
  - Y cobran por tiempo de uso, siendo fácil exceder lo que se oferta en su "free tier" sin darnos cuenta
  - Si no destruimos adecuadamente la infraestructura, **se nos pueden pasar pagos no deseados**
- En esta asignatura no queremos correr riesgos y llevamos una política de gasto adicional cero
- Por ello, para manejar **Terraform** como lo haríamos con un proveedor de nube sin riesgo usaremos **Docker**

Por qué probamos con Docker

José Manuel
Redondo López

- **El uso del provider Docker requiere, como hemos dicho, leer su documentación completa para conocer sus resources y data sources**
  - Podemos encontrarla aquí: https://registry.terraform.io/providers/kreuzwerker/docker/latest/docs
- **Recuerda: Si cambiamos a un proveedor AWS, Google Cloud, etc. tendremos que leer la documentación correspondiente**
  - Para cambiar los elementos de los ficheros escritos en HCL (resource types y data sources) del proveedor antiguo a los equivalentes en el nuevo
  - Los ficheros creados para ambos tendrán elementos distintos...
  - **¡Pero la filosofía de uso será exactamente la que veremos aquí!**

Usar Docker con Terraform

José Manuel
Redondo López

- Para poder usar **Docker** con **Terraform**, necesitamos una instalación de **Docker** operativa
  - Visto en el tema de **Docker**
  - Debemos seguir la documentación oficial
  - **Y hacer los pasos de post instalación para que un usuario no `root` pueda ejecutarlo sin `sudo`**
    - https://docs.docker.com/engine/install/linux-postinstall/
- Nos vale una instalación local de **Docker** o remota
  - En nuestro caso la haremos local

INSTALAR TERRAFORM

José Manuel
Redondo López

- **Terraform está disponible como un paquete snap de Ubuntu**
  - Pero no suele ser la última versión disponible
  - **Lo mejor es instalar la versión más reciente del mismo** para evitar bugs
    - Y tener las características más avanzadas
  - Se puede descargar de aquí, siguiendo las instrucciones que se indican
    - https://www.terraform.io/downloads.html

PUESTA EN MARCHA DE LA INFRAESTRUCTURA

José Manuel
Redondo López

- Ahora creamos un directorio de trabajo y dentro de él un fichero `provider.tf`
- En el declaramos el **provider** para **Docker**
  - `kreuzwerker/docker`
- Todo ello siguiendo su documentación oficial
  - https://registry.terraform.io/providers/kreuzwerker/docker/latest/docs
- Con este fichero **Terraform** podrá descargarse el **provider**
  - Y configurarlo adecuadamente de manera automática

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "2.14.0"
    }
  }
}

provider "docker" {
  host = "unix:///var/run/docker.sock"
}
```

Primeros ejemplos prácticos de uso

Desplegando contenedores con Terraform

Uso de una imagen oficial del registro de Docker

José Manuel
Redondo López

- **Para esta 1ª prueba crearemos una infraestructura simple**
  - Un solo contenedor creado a partir de una imagen oficial de Ubuntu del Docker Hub
- **Para ello configuramos adecuadamente las propiedades de los resources** `docker_image` **y** `docker_container`
  - Elementos definidos por el provider de Docker
  - De nuevo consultando su documentación oficial
- **El fichero adjunto crea un contenedor** `miw_test` **de la imagen oficial Ubuntu**

```
 # Pulls the image
resource "docker_image" "ubuntu" {
  name = "ubuntu:latest"
}


# Create a container
resource "docker_container" "miw_test"
{
  image = docker_image.ubuntu.latest
  name  = "miw_test"
}
```

Pasos de creación de la infraestructura

José Manuel
Redondo López

- Con los dos ficheros creados, lo primero que debemos hacer es `terraform init`
- Con ello **Terraform** instalará el plugin de **Docker**
  - Si no está instalado de antemano, por ejemplo, por una ejecución anterior
- Y además inicializará todo lo necesario para poder crear posteriormente la infraestructura

```
operario@server:~/terraform_test$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "2.14.0"...
- Installing kreuzwerker/docker v2.14.0...
- Installed kreuzwerker/docker v2.14.0 (self-signed, key ID 24E54F214569A8A5)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
```

Pasos de creación de la infraestructura

José Manuel
Redondo López

- El 2º paso es **validar** que la configuración es correcta con `terraform validate`
- Hecho esto debemos ejecutar:
  - `terraform plan`: para que nos detalle lo que va a hacer antes de hacerlo
    - En este caso es muy sencillo, por lo que no da lugar a posibles errores
  - `terraform apply`: para que ejecute lo que le hemos indicado
    - Debemos contestar "yes" al final
- ¡Pero este contenedor morirá tras arrancar!
  - Puesto que no deja ninguna aplicación en ejecución (lo mismo que ya nos pasaba en Docker)
- Veremos el detalle de estas operaciones en un ejemplo que use un servidor web

```
operario@server:~/terraform_test$ terraform validate
Success! The configuration is valid.
```

Ejemplo: Uso de nuestras imágenes locales

José Manuel
Redondo López

- En este ejemplo haremos uso de una de las imágenes que describimos en el tema de contenedores **Docker**
- Se trata del servidor web **Apache 2** simple que se mantiene en ejecución
- Añadimos el uso de `ports` para mapear el puerto 8080 del host al puerto 80 del contenedor

```
terraform {
  required_providers {
    docker = {
      source = "kreuzwerker/docker"
      version = "2.14.0"
    }
  }
}

provider "docker" {
  host = "unix:///var/run/docker.sock"
}

# Pulls the image
resource "docker_image" "mi_servidor_web" {
  name = "ubuntu_apache"
}

# Create a container
resource "docker_container" "apache_test" {
  image = docker_image.mi_servidor_web.latest
  name = "apache_miw_test"

# Especificamos el puerto del contenedor que vamos a exponer
  ports {
    internal = 80
    external = 8080
    }
}
```

ApacheNet.tf

Ejemplo: Uso de nuestras imágenes locales

José Manuel
Redondo López

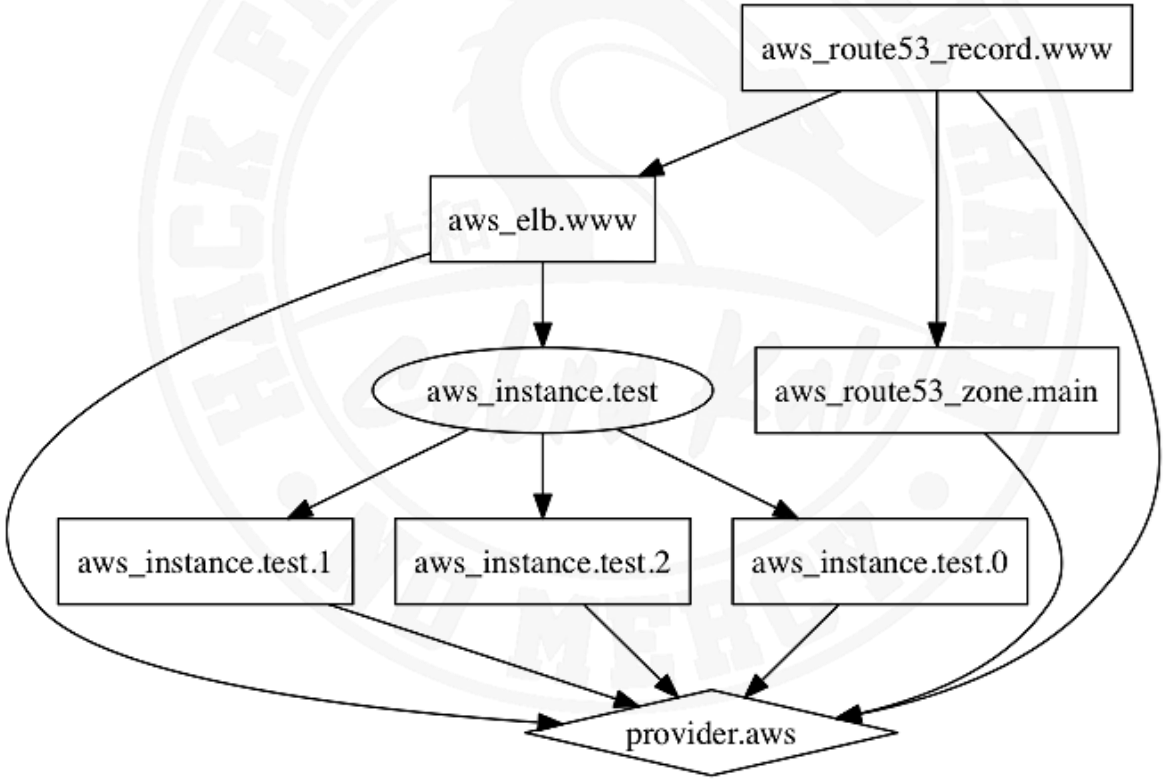
- **Como hemos dicho, Terraform construye un grafo de recursos**
  - Para paralelizar las operaciones de creación de infraestructuras si es posible
- **Podemos consultarlo con `terraform graph`**
- **En este caso es muy sencillo**
  - Pero se vuelve mucho más interesante en infraestructuras más complejas ☺

```
operario@server:~/terraform_test$ terraform graph
digraph {
        compound = "true"
        newrank = "true"
        subgraph "root" {
                "[root] docker_container.apache_test2a (expand)" [label = "docker_container.apache_test2a", shape = "box"]
                "[root] docker_container.apache_test2b (expand)" [label = "docker_container.apache_test2b", shape = "box"]
                "[root] docker_image.mi_servidor_web (expand)" [label = "docker_image.mi_servidor_web", shape = "box"]
                "[root] docker_network.private_network (expand)" [label = "docker_network.private_network", shape = "box"]
                "[root] provider[\"registry.terraform.io/kreuzwerker/docker\"]" [label = "provider[\"registry.terraform.io/kreuzwerker/docker\"]", shape = "diam
ond"]
                "[root] docker_container.apache_test2a (expand)" -> "[root] docker_image.mi_servidor_web (expand)"
                "[root] docker_container.apache_test2b (expand)" -> "[root] docker_image.mi_servidor_web (expand)"
                "[root] docker_image.mi_servidor_web (expand)" -> "[root] provider[\"registry.terraform.io/kreuzwerker/docker\"]"
                "[root] docker_network.private_network (expand)" -> "[root] provider[\"registry.terraform.io/kreuzwerker/docker\"]"
                "[root] meta.count-boundary (EachMode fixup)" -> "[root] docker_container.apache_test2a (expand)"
                "[root] meta.count-boundary (EachMode fixup)" -> "[root] docker_container.apache_test2b (expand)"
                "[root] meta.count-boundary (EachMode fixup)" -> "[root] docker_network.private_network (expand)"
                "[root] provider[\"registry.terraform.io/kreuzwerker/docker\"] (close)" -> "[root] docker_container.apache_test2a (expand)"
                "[root] provider[\"registry.terraform.io/kreuzwerker/docker\"] (close)" -> "[root] docker_container.apache_test2b (expand)"
                "[root] provider[\"registry.terraform.io/kreuzwerker/docker\"] (close)" -> "[root] docker_network.private_network (expand)"
                "[root] root" -> "[root] meta.count-boundary (EachMode fixup)"
                "[root] root" -> "[root] provider[\"registry.terraform.io/kreuzwerker/docker\"] (close)"
        }
}
```

Grafos de Terraform

José Manuel
Redondo López

- Cuando la infraestructura es más compleja, el uso del grafo tiene más sentido
- Se puede visualizar a partir de la información generada muy fácilmente con `terraform graph`
  - https://developer.hashicorp.com/terraform/cli/commands/graph



Ejemplo: Uso de nuestras imágenes locales

José Manuel
Redondo López

- Hacemos `terraform plan` para que nos detalle qué se va a crear y destruir
  - Y así revisar que todo es correcto

```
operario@server:~/terraform_test$ terraform plan
docker_image.ubuntu: Refreshing state... [id=sha256:c29284518f497b8c5f49933e74e43ca5221e69c8251e780427f7d12f7166
25ffubuntu:latest]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create
  - destroy

Terraform will perform the following actions:

  # docker_container.apache_test will be created
  + resource "docker_container" "apache_test" {
      + attach           = false
      + bridge           = (known after apply)
      + command          = (known after apply)
      + container_logs   = (known after apply)
      + entrypoint       = (known after apply)
      + env              = (known after apply)
      + exit_code        = (known after apply)
      + gateway          = (known after apply)
      + hostname         = (known after apply)
      + id               = (known after apply)
      + image            = (known after apply)
      + init             = (known after apply)
      + ip_address       = (known after apply)
      + ip_prefix_length = (known after apply)
      + ipc_mode         = (known after apply)
      + log_driver       = "json-file"
      + logs             = false
      + must_run         = true
      + name             = "apache_miw_test"
      + network_data     = (known after apply)
      + read_only        = false
      + remove_volumes   = true
      + restart          = "no"
      + rm               = false
      + security_opts    = (known after apply)
      + shm_size         = (known after apply)
      + labels {
          + label = (known after apply)
          + value = (known after apply)
        }

      + ports {
          + external = 8080
          + internal = 80
          + ip       = "0.0.0.0"
          + protocol = "tcp"
        }
    }

  # docker_image.mi_servidor_web will be created
  + resource "docker_image" "mi_servidor_web" {
      + id          = (known after apply)
      + latest      = (known after apply)
      + name        = "ubuntu_apache"
      + output      = (known after apply)
      + repo_digest = (known after apply)
    }

  # docker_image.ubuntu will be destroyed
  - resource "docker_image" "ubuntu" {
      - id          = "sha256:c29284518f497b8c5f49933e74e43ca5221e69c8251e780427f7d12f716625ffubuntu:latest" ->
null
      - latest      = "sha256:c29284518f497b8c5f49933e74e43ca5221e69c8251e780427f7d12f716625ff" -> null
      - name        = "ubuntu:latest" -> null
      - repo_digest = "ubuntu@sha256:b3e2e47d016c08b3396b5ebe06ab0b711c34e7f37b98c9d37abe794b71cea0a2" -> null
    }

Plan: 2 to add, 0 to change, 1 to destroy.

------------------------------------------------------------------------

Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
```

Ejemplo: Uso de nuestras imágenes locales

José Manuel
Redondo López

- **Si estamos de acuerdo, hacemos `terraform apply` para ejecutar las operaciones**
  - **Recuerda**: Solo las hará si al final contestamos literalmente "`yes`"

```
operario@server:~/terraform_test$ terraform apply
docker_image.ubuntu: Refreshing state... [id=sha256:c29284518f497b8c5f49933e74e43ca5221e69c8251e780427f7d12f7166
25ffubuntu:latest]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create
  - destroy

Terraform will perform the following actions:

  # docker_container.apache_test will be created
  + resource "docker_container" "apache_test" {
      + attach           = false
      + bridge           = (known after apply)
      + command          = (known after apply)
      + container_logs   = (known after apply)
      + entrypoint       = (known after apply)
      + env              = (known after apply)
      + exit_code        = (known after apply)
      + gateway          = (known after apply)
      + hostname         = (known after apply)
      + id               = (known after apply)
      + image            = (known after apply)
      + init             = (known after apply)
      + ip_address       = (known after apply)
      + ip_prefix_length = (known after apply)
      + ipc_mode         = (known after apply)
      + log_driver       = "json-file"
      + logs             = false
      + must_run         = true
      + name             = "apache_miw_test"
      + network_data     = (known after apply)
      + read_only        = false
      + remove_volumes   = true
      + restart          = "no"
      + rm               = false
      + security_opts    = (known after apply)
      + shm_size         = (known after apply)
      + labels {
          + label = (known after apply)
          + value = (known after apply)
        }

      + ports {
          + external = 8080
          + internal = 80
          + ip       = "0.0.0.0"
          + protocol = "tcp"
        }
    }

  # docker_image.mi_servidor_web will be created
  + resource "docker_image" "mi_servidor_web" {
      + id          = (known after apply)
      + latest      = (known after apply)
      + name        = "ubuntu_apache"
      + output      = (known after apply)
      + repo_digest = (known after apply)
    }

  # docker_image.ubuntu will be destroyed
  - resource "docker_image" "ubuntu" {
      - id          = "sha256:c29284518f497b8c5f49933e74e43ca5221e69c8251e780427f7d12f716625ffubuntu:latest" ->
null
      - latest      = "sha256:c29284518f497b8c5f49933e74e43ca5221e69c8251e780427f7d12f716625ff" -> null
      - name        = "ubuntu:latest" -> null
      - repo_digest = "ubuntu@sha256:b3e2e47d016c08b3396b5ebe06ab0b711c34e7f37b98c9d37abe794b71cea0a2" -> null
    }

Plan: 2 to add, 0 to change, 1 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
```

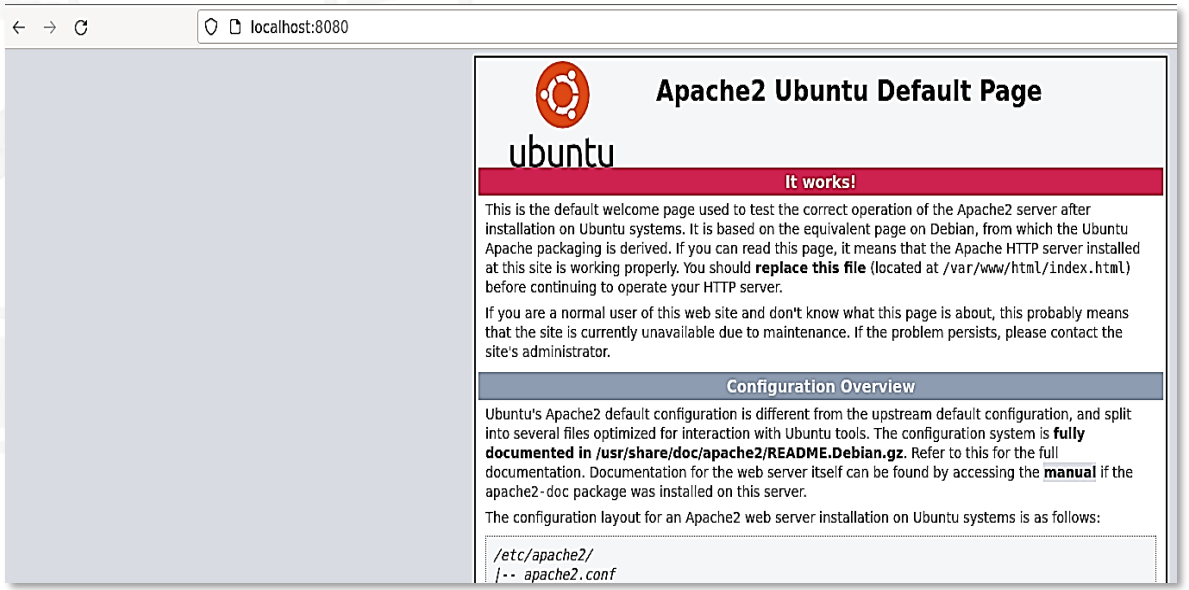
Ejemplo: Uso de nuestras imágenes locales

José Manuel
Redondo López

```
operario@server:~/terraform_test$ docker images
REPOSITORY      TAG       IMAGE ID       CREATED             SIZE
ubuntu_apache   latest    5c9ef6207ad2   12 minutes ago      195MB
ubuntu_base     latest    95e8c00e05cb   15 minutes ago      102MB
ubuntu          focal     c29284518f49   7 days ago          72.8MB
hello-world     latest    d1165f221234   4 months ago        13.3kB
operario@server:~/terraform_test$
```

- **Hecho esto tenemos**
  - Un contenedor con Apache 2 creado
  - La imagen correspondiente descargada
    - Si no lo estaba ya de una ejecución anterior ☺
  - El servidor operativo mapeado en el puerto 8080 del host

```
Apply complete! Resources: 2 added, 0 changed, 1 destroyed.
operario@server:~/terraform_test$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                    NAME
S
409806e578c3   5c9ef6207ad2   "./container_init.sh"    6 seconds ago   Up 4 seconds   0.0.0.0:8080->80/tcp     apac
he_miw_test
operario@server:~/terraform_test$
```



Más opciones...

José Manuel
Redondo López

- **En general las opciones del provider de Docker mapean todas las opciones que vimos en el tema de Docker**
  - Eso quiere decir que, consultando la documentación, Terraform puede usar todos los elementos vistos (volúmenes, etc.) en el tema de contenedores
  - Es cuestión de buscar los equivalentes
    - https://registry.terraform.io/providers/kreuzwerker/docker/latest/docs/resources/container
  - Esto te permite sacar una conclusión clara
    - Si tienes conocimientos de una plataforma de nube concreta (AWS, Google Cloud, etc.) estarás familiarizado con **sus conceptos** (tipos de instancia, regiones, etc.)
    - Vas a encontrar un **mapeo de esos conceptos tal cual** en el provider correspondiente de Terraform
    - Usar Terraform teniendo ya conocimientos del proveedor que hay por debajo **es una transición sencilla**
    - ¡Y más ahora, que ya has visto el workflow de cómo se trabaja con cualquier proyecto Terraform! ☺
- **Más información**
  - https://www.devopsschool.com/pdf/terraform/slides/Complete-Terraform-documentation/docker.pdf

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

- **Docker y Terraform**
  - *¿Entiendes por qué hemos elegido Docker para las pruebas?*
  - *¿Comprendes que la misma filosofía de trabajo que te muestro aquí se puede migrar fácilmente a cualquier proveedor de nube posteriormente?*
  - *¿Recuerdas que para poder usar Terraform y Docker necesitas una instalación funcional de ambos sistemas (local o remoto)?*
- **Ejemplos prácticos**
  - *¿Entiendes como declarar el provider Docker en un fichero .tf?*
  - *¿Crees que puedes crear un contenedor con un servidor web a partir de una imagen del Docker Hub con Terraform con un fichero .tf?*
  - *¿Entiendes la secuencia típica de uso de un fichero Terraform (init - validate - plan - apply)?*
  - *¿Entiendes la utilidad de terraform graph?*
  - *¿Comprendes que un proveedor de Docker mapea todas las opciones de Docker sobre Terraform y, por tanto, basta con conocer bien Docker para, leyendo la documentación, explotar el sistema adecuadamente?*
  - *¿Entiendes que lo mismo ocurre con cualquier proveedor de cualquier sistema?*

< Ir al Índice

Fuente: Bing Chat AI

Ejemplos locales >

Ejemplos en nube >

Ejemplos adicionales de uso de Terraform

Poniendo la tecnología en marcha de forma sencilla

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- En esta sección vamos a mostrar más ejemplos de Terraform para entender mejor su uso
- Coordinando estos ejemplos con el resto del curso
  - Ampliando lo visto con Docker
  - Usando Terraform y LXD
  - Usando Terraform y Vagrant
  - Usando Terraform y K8s

Ejemplos locales

Practicando con Terraform y Docker

EJEMPLO: CREAR UNA RED ENTRE CONTENEDORES

José Manuel
Redondo López

- Veamos un ejemplo donde dos servidores **Apache 2** se comunican entre sí por una red privada local
- Dicha red se crea de forma como vimos en **Docker**
  - Con máscara `172.56.0.0/16`
- Solo 1 de los contenedores servidor web será accesible desde el host
  - Puerto 8081->80
- El otro solo será accesible desde este primer contenedor servidor web

```
terraform {
  required_providers {
    docker = {
      source = "kreuzwerker/docker"
      version = "2.14.0"
    }
  }
}

provider "docker" {
  host = "unix:///var/run/docker.sock"
}

# Pulls the image
resource "docker_image" "mi_servidor_web" {
  name = "ubuntu_apache"
}

resource "docker_network" "private_network" {
  name = "custom_private_network"
  driver = "bridge"
  ipam_config {
      subnet = "172.56.0.0/16"
  }
}
```

```
# Create a container
resource "docker_container" "apache_test2a" {
  image = docker_image.mi_servidor_web.latest
  name = "apache_miw_test_server1"

  networks_advanced {
    name = "custom_private_network"
  }

# Especificamos el puerto del contenedor que vamos a exponer
  ports {
    internal = 80
    external = 8081
    }
}

# Create a container
resource "docker_container" "apache_test2b" {
  image = docker_image.mi_servidor_web.latest
  name = "apache_miw_test_server2"

  networks_advanced {
      name = "custom_private_network"
  }
}
```

ApacheNet.tf

EJEMPLO: CREAR UNA RED ENTRE CONTENEDORES

José Manuel
Redondo López

- **Si volvemos a ejecutar `terraform plan` y `terraform apply` veremos que al final tenemos dos contenedores en funcionamiento**
  - Con el mapeo de puertos especificado
- **Por supuesto, las imágenes correspondientes se habrán descargado previamente si no lo están ya**

```
operario@server:~/terraform_test$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                    NA
MES
a4ffa464a55f   5c9ef6207ad2   "./container_init.sh"   10 seconds ago   Up 9 seconds    0.0.0.0:8081->80/tcp     ap
ache_miw_test_server1
01e1365a6552   5c9ef6207ad2   "./container_init.sh"   55 seconds ago   Up 54 seconds   80/tcp                   ap
ache_miw_test_server2
```

EJEMPLO: CREAR UNA RED ENTRE CONTENEDORES

José Manuel
Redondo López

- Si accedemos a la consola de este primer contenedor, vemos que tiene una IP asignada automáticamente en el rango dado
- También que su servidor es el que está mapeado en el puerto `8081` del host

```
operario@server:~/terraform_test$ docker exec -it apache_miw_test_server1 /bin/bash
root@a4ffa464a55f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.56.0.3  netmask 255.255.0.0  broadcast 172.56.255.255
        ether 02:42:ac:38:00:03  txqueuelen 0  (Ethernet)
        RX packets 40  bytes 5223 (5.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 8365 (8.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

EJEMPLO: CREAR UNA RED ENTRE CONTENEDORES

José Manuel
Redondo López

- Desde el 1er contenedor podemos acceder al 2º
- Con esto hemos logrado la infraestructura que deseábamos
  - A partir del fichero `.tf` visto anteriormente
- Ahora podemos replicarla en cualquier otro host con **Terraform** y **Docker** instalado
- Siempre que tenga forma de acceder o construir las imágenes **Docker** usadas

```
root@a4ffa464a55f:/# curl 172.56.0.2

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitio
nal.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <!--
    Modified from the Debian original for Ubuntu
    Last updated: 2016-11-16
    See: https://launchpad.net/bugs/1288690
  -->
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Apache2 Ubuntu Default Page: It works</title>
    <style type="text/css" media="screen">
  * {
    margin: 0px 0px 0px 0px;
    padding: 0px 0px 0px 0px;
  }

  body, html {
    padding: 3px 3px 3px 3px;

    background-color: #D8DBE2;

    font-family: Verdana, sans-serif;
    font-size: 11pt;
    text-align: center;
  }
```

Terraform y LXD

José Manuel
Redondo López

- Con **Terraform** se pueden desplegar contenedores de SO en una instalación LXD existente
- Gracias al **provider** `terraform-lxd/lxd`
  - https://registry.terraform.io/providers/terraform-lxd/lxd/latest/docs
- Define **resource types** y **data sources** para las entidades LXD que hemos visto
  - Storage pools, volumes, contenedores, devices, profiles...
- Usamos las imágenes que vimos en LXD
  - Con su mismo convenio de nombrado

```
terraform {
  required_providers {
    lxd = {
      source = "terraform-lxd/lxd"
      version = "1.5.0"
    }
  }
}
resource "lxd_container" "soubuntu" {
  name      = "soubuntu"
  image     = "ubuntu:20.04"
  ephemeral = false

  config = {
    "boot.autostart" = true
  }

  limits = {
    cpu = 1
  }
}
```

Terraform y Vagrant

José Manuel
Redondo López

- **Terraform puede desplegar MV creadas con Vagrant**
  - Leyendo su configuración de Vagrantfiles y en una instalación de Vagrant existente
- **Gracias al provider** `bmatcuk/vagrant`
  - https://github.com/bmatcuk/terraform-provider-vagrant
- **Debemos pasarle la ruta donde está la Vagrantfile a leer**
  - `vagrantfile_dir`
- **Define resource types y data sources para las entidades Vagrant que hemos visto**

```
terraform {  required_providers {
    vagrant = {
      source  = "bmatcuk/vagrant"
      version = "~> 4.0.0"
    }
  }
}

resource "vagrant_vm" "my_vagrant_vm" {
  vagrantfile_dir = "."
  env = {
    BACKEND_IP = "192.168.50.10",
  }
  get_ports = true
}
```

Terraform y Kubernetes

José Manuel
Redondo López

- **Terraform** también puede desplegar en instalaciones de clúster **Kubernetes** existentes
  - Eso incluye el `minikube` que usamos en esta asignatura
  - Gracias al **provider** `hashicorp/kubernetes`
  - https://registry.terraform.io/providers/hashicorp/kubernetes/latest
- **Define resource types y data sources para las entidades Kubernetes que hemos visto**
  - Namespaces, deployments, services, labels...

```
terraform {
  required_providers {
    kubernetes = {
      source = "hashicorp/kubernetes"
      version = "2.11.0"
    }
  }
}

provider "kubernetes" {
  config_path    = "~/.kube/config"
  config_context = "minikube"
}
```

Terraform en la nube

Ejemplo de uso de un proveedor de nube

¿Cómo hago el paso a la nube?

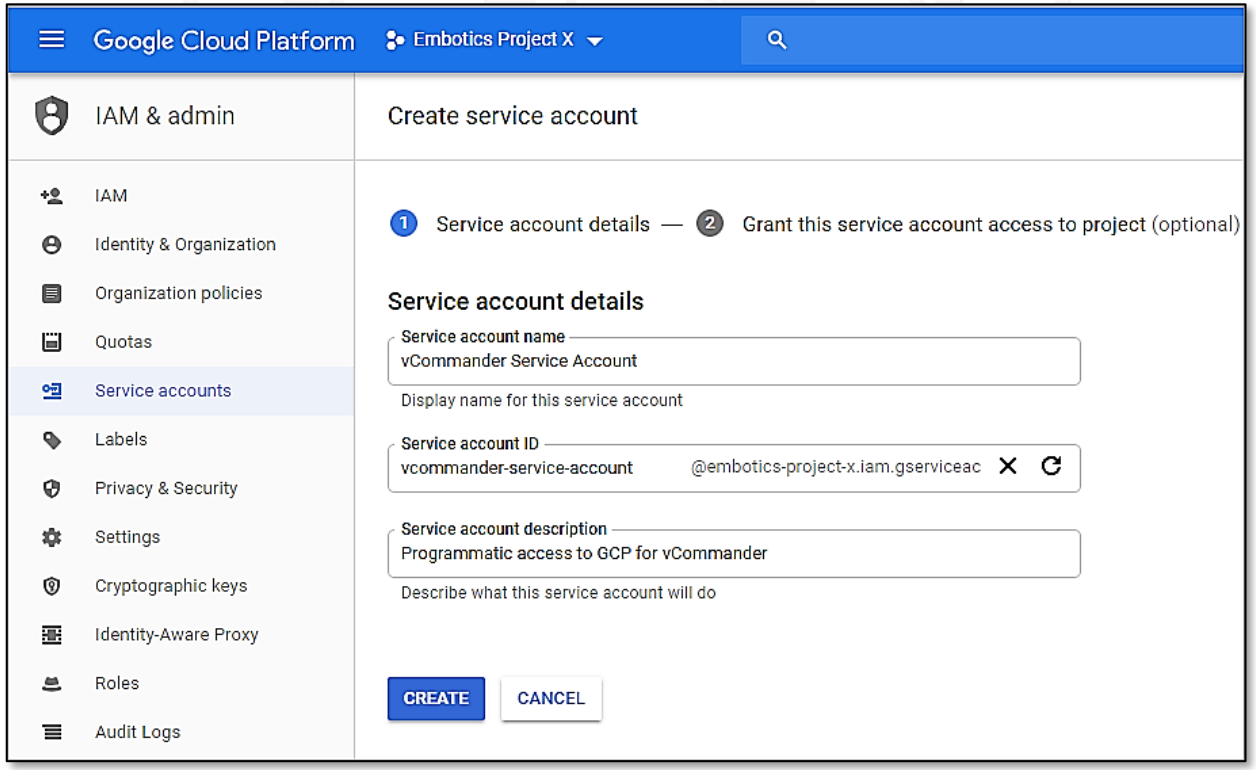
José Manuel
Redondo López

- **Como hemos dicho, tenemos que leer la documentación del proveedor correspondiente a la plataforma de nube que vayamos a usar**
  - Y seguir las instrucciones que se nos indican
  - Además, hay que asegurarse de
    - Tener cuenta en el proveedor de nube en vigor
    - Tener **un plan de pagos activo**
    - Dar correctamente la configuración de dicha cuenta a Terraform para que pueda crear los recursos adecuadamente
  - Luego, hacer un `terraform init`, `terraform validate`, `terraform plan` y `terraform apply` como hemos visto
- **Ejemplos**
  - https://blog.gruntwork.io/an-introduction-to-terraform-f17df9c6d180
  - https://alissonmachado.com.br/terraform-azure-criando-uma-infraestrutura-basica
- **Documentación**
  - https://cloud.google.com/docs/terraform?hl=es-419

Ejemplo con Google Cloud

José Manuel
Redondo López

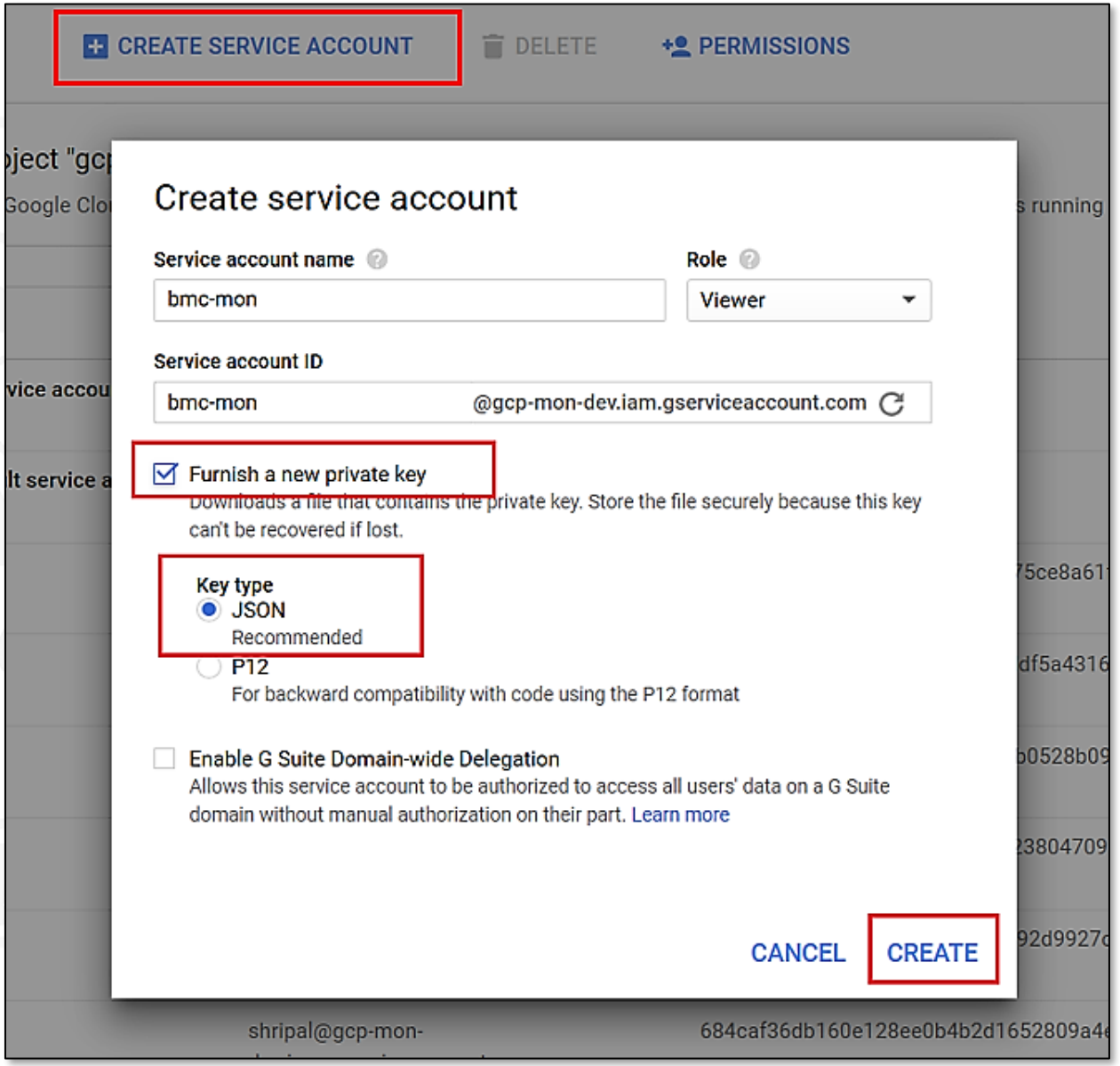
- Con nuestra cuenta de Google activa y un plan de pagos establecido, lo primero es ir a la Google Cloud Console
  - https://console.cloud.google.com
- Dentro de la sección IAM & Admin, opción Create service account, crear una cuenta llamada Terraform



Ejemplo con Google Cloud

José Manuel
Redondo López

- Hecho esto, darle a create y seleccionar Role Compute Admin
- Este rol permite a la cuenta del servicio manejar los recursos del Google Compute Engine (nuestras VM)
- Para poder usar Terraform, debemos crear una clave en formato JSON (`terraform_sa.json`)



Ejemplo con Google Cloud

José Manuel
Redondo López

- **Este fichero lo usaremos en la configuración de Terraform que veremos a continuación**
  - Además, incluiremos una serie de secciones "output" que nos den los datos necesarios para hacer la conexión a la máquina creada
  - Concretamente la IP, para poder hacer `ssh` a `redondo@<IP proporcionada>`
- **Las máquinas creadas en Google Cloud no admiten conexiones vía SSH con password por temas de seguridad**
  - Debemos crear un par de claves pública-privada para conectarse vía SSH
  - **Recuerda**: hay que suministrar también el fichero `id_rsa.pub` que habitualmente se crea en `~/.ssh`

Ejemplo con Google Cloud

José Manuel
Redondo López

Google.tf

```
provider "google" {
   credentials = file ("terraform_sa.json")
   project = "redondo-270721"
   region = "us-east1"
}
resource "google_compue_instance""terraformmiw" {
   name = "terraformmiw-instance"
   machine_type = "g1-small"
   zone = "us-east1-b"
   boot_disk {
      initialize_params {
         image = "ubuntu-1804-bionic-v20200414"
      }
   }
   network_interface {
      network = "default"
      access_config {}
   }
   metadata = {
      sshKeys = join("",["redondo:",file("id_rsa.pub")])
   }
}
output "name" {
   value = google_compute_instance.terraformmiw.name
}
output "size" {
   value = google_compute_instance.terraformmiw.machine_type
}
output " ip " {
   value = google_compute_instance.terraformmiw.network_interface[0].access_config[0].nat_ip
}
```

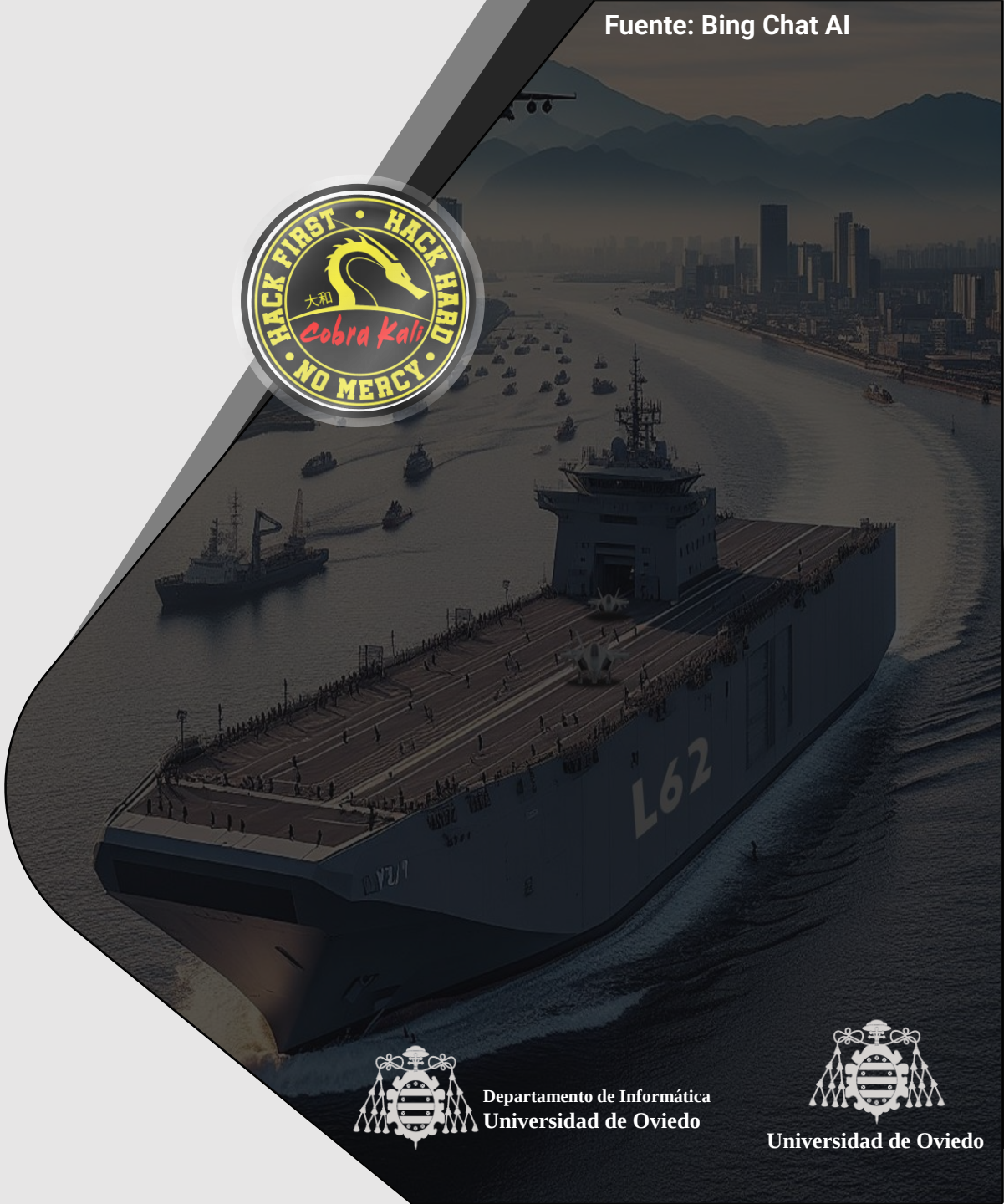
¿CREES QUE LO HAS ENTENDIDO TODO? ¡AUTOEVALÚATE!

José Manuel
Redondo López

?

- *¿Entiendes cómo usar **Terraform** para hacer una red privada entre contenedores, mapeando lo que ya se vio con **Docker**?*
- *¿Has visto como empezar a usar **LXD** junto con **Terraform** (proveedor y ejemplo sencillo inicial)?*
- *¿Has visto como empezar a usar **Vagrant** con **Terraform** (proveedor y ejemplo sencillo inicial)?*
- *¿Has visto como empezar a usar **K8s** junto con **Terraform** (proveedor y ejemplo sencillo inicial)?*
- *¿Has comprendido cómo pasar a usar **Terraform** en nube con un proveedor **Google Cloud**?*
- *En este sentido, ¿Te das cuenta de cómo el proveedor cambia los elementos a usar en los ficheros **.tf**?*

< Ir al Índice



Para ampliar

Otros conceptos

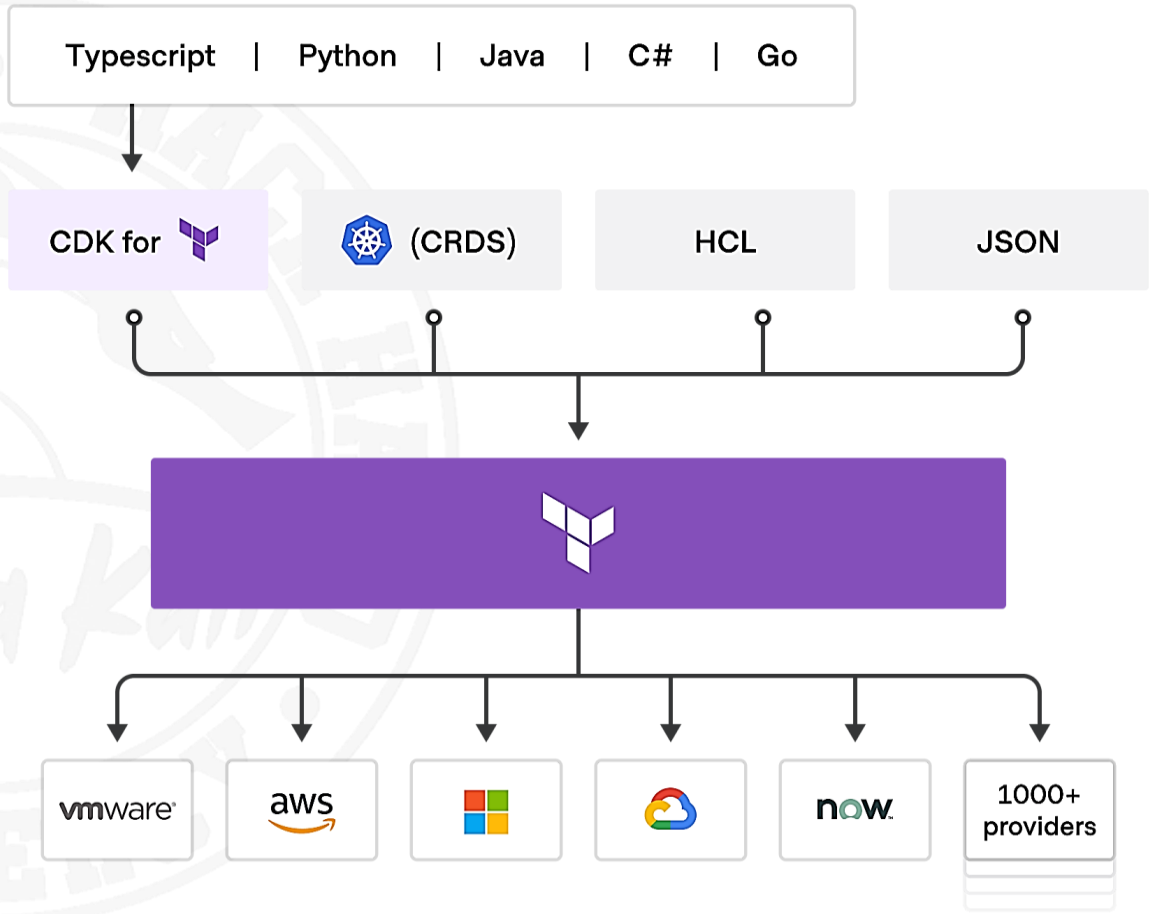
Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

DESARROLLAR PARA LA NUBE

José Manuel
Redondo López

- **Existen en Internet ejemplos de ficheros Terraform para infraestructuras comunes**
  - No hace falta partir de cero a la hora de construir una propia
  - Ej.: https://github.com/futurice/terraform-examples
- **Otra opción es usar el Cloud Development Kit for Terraform (CDKTF)**
  - Te permite **usar lenguajes de programación populares** para definir infraestructuras con ellos
    - Typescript, Python, Java, C#, Go
  - Imagina crear infraestructuras con los mismos tipos (clases, objetos) que creas la funcionalidad de la aplicación
  - https://github.com/hashicorp/terraform-cdk



Ejemplo CDTKF

José Manuel
Redondo López

- Como ejemplo, este código Node .js es equivalente al fichero .tf de abajo
  - https://developer.hashicorp.com/terraform/tutorials/cdktf/cdktf-install
- La configuración se puede manipular con clases de Node .js
  - Qué clases y sus miembros se obtienen importando los paquetes del `cdktf` para Node

```
resource "docker_image" "nginx" {
  name         = "nginx:latest"
  keep_locally = false
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.name
  name  = "tutorial"
  ports {
    internal = 80
    external = 8000
  }
}
```

```
import { Construct } from "constructs";
import { App, TerraformStack } from "cdktf";
import { DockerProvider } from "@cdktf/provider-docker/lib/provider";
import { Image } from "@cdktf/provider-docker/lib/image";
import { Container } from "@cdktf/provider-docker/lib/container";

class MyStack extends TerraformStack {
  constructor(scope: Construct, name: string) {
    super(scope, name);

    new DockerProvider(this, "docker", {});

    const dockerImage = new Image(this, "nginxImage", {
      name: "nginx:latest",
      keepLocally: false,
    });

    new Container(this, "nginxContainer", {
      name: "tutorial",
      image: dockerImage.name,
      ports: [
        {
          internal: 80,
          external: 8000,
        },
      ],
    });
  }
}
const app = new App();
new MyStack(app, "learn-cdktf-docker");
app.synth();
```

OpenTF

https://opentf.org/

José Manuel
Redondo López

- **Quiere ser una implementación alternativa a la que hemos visto**
  - Creada a raíz del cambio de licencia que el creador de Terraform hizo al producto a un modelo de licencia no open source
  - La nueva licencia podría comprometer las infraestructuras hechas con Terraform en el futuro,
  - Si HashiCorp percibe un producto como “competencia” de sus propios productos o vuelve a cambiar las condiciones
- **Para evitar tener problemas en el futuro, la iniciativa OpenTF tiene dos líneas de acción planificadas**
  - Pedirle a HashiCorp volver a un modelo de licencia open source
  - En caso negativo , crear una nueva implementación completamente open-source de Terraform
    - El objetivo es **que sea un drop-in replacement**: Todo lo que se haya hecho hasta ahora con Terraform funcionará igualmente con OpenTF
- **Es algo a mirar en el futuro, por si aparecen problemas de licencias con algo que hagamos con Terraform**

TERRAGRUNT

https://terragrunt.gruntwork.io/

José Manuel
Redondo López

- **Es una capa de software que se integra con Terraform**
  - Te permite usar los mismos comandos, pero con ventajas adicionales
- **Proporciona una herramienta para crear y gestionar infraestructuras de Terraform siguiendo buenas prácticas**
- **Las que se enuncian en el libro "Terraform: Up and Running" como, por ejemplo**
  - Usar módulos versionados y reutilizables para diferentes entornos
  - Ejecutar código recursivamente en subdirectorios
  - Definir hooks para el ciclo de vida
  - Simplificar la configuración del estado remoto
- **Requiere un conocimiento más profundo de Terraform, pero puede ser una herramienta interesante a usar si profundizas en el mismo**
- **Más información**
  - https://github.com/gruntwork-io/terragrunt
  - Ejemplo con AWS: https://terragrunt.gruntwork.io/docs/getting-started/quick-start/

EJEMPLOS PARA SEGUIR PRACTICANDO

José Manuel
Redondo López

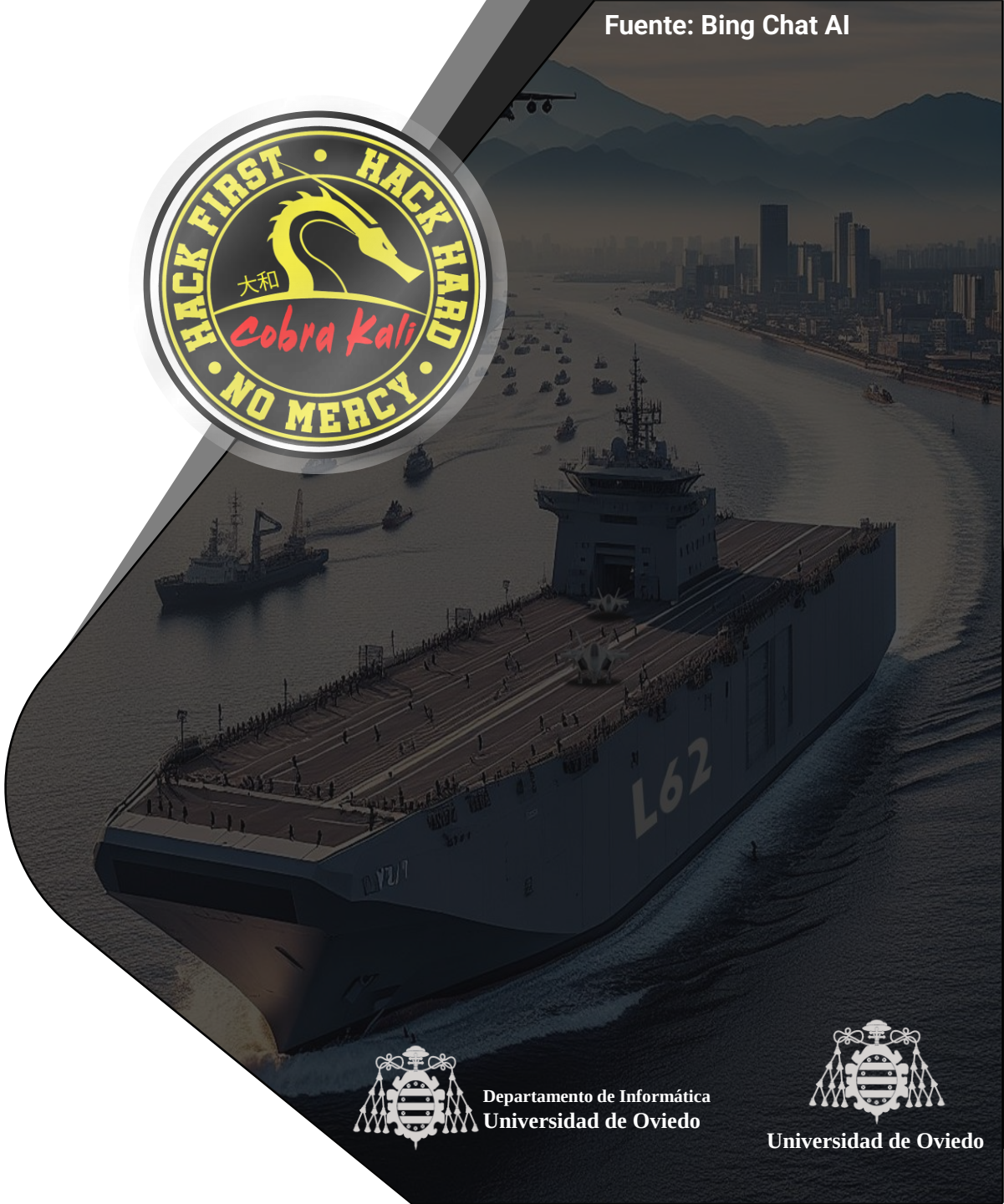
- **Terraform AWS Example - Create EC2 instance with Terraform**
  - https://www.middlewareinventory.com/blog/terraform-aws-example-ec2/
- **How To Structure a Terraform Project**
  - https://www.digitalocean.com/community/tutorials/how-to-structure-a-terraform-project
- **Terraform best practices**
  - https://www.terraform-best-practices.com/
- **Muestras de recursos de Terraform**
  - https://cloud.google.com/docs/terraform/samples?hl=es-419
- **Futurice Terraform examples**
  - https://github.com/futurice/terraform-examples
- **How to create reusable infrastructure with Terraform modules**
  - https://blog.gruntwork.io/how-to-create-reusable-infrastructure-with-terraform-modules-25526d65f73d?gi=a65269acf60a

Curso completo de Terraform complementario

José Manuel
Redondo López

- **Si quieres seguir profundizando en Terraform, puedes hacerte el curso gratuito de 2.5 horas de Sid Palas para complementar este tema**
  - Curso en YouTube: https://www.youtube.com/watch?v=7xngnjfIlK4
  - GitHub: https://github.com/sidpalas/devops-directive-terraform-course
  - Discord: https://discord.devopsdirective.com

INTRODUCCIÓN A TERRAFORM



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Fin de la asignatura

José Manuel
Redondo López

- Gracias por cursar esta asignatura, dónde he tratado de transmitiros todo lo que considero adecuado para empezar con buen pie en el mundo de la IaC
  - **Automatiza todo el despliegue de infraestructuras con código**, por lo que te permiten crear dónde va a desplegarse tu aplicación en el mismo pipeline que usas para el código (DevOps)
- Esta asignatura es una evolución del esfuerzo inicial que se hizo por llevar la IaC y la filosofía DevOps al MIW y a Uniovi (el "R-11 Príncipe de Asturias")
- ¡Seguiremos evolucionando y mejorando este producto con los años! ☺