jmcastillo Documentation

Versión 3.0

José María Castillo Cotán

20 de julio de 2017

Índice general

CAPÍTULO 1

Introducción

Terraform es una herramienta para construir, combinar y poner en marcha de manera segura y eficiente la infraestructura. Desde servidores físicos a contenedores hasta productos SaaS (Software como un Servicio), Terraform es capaz de crear y componer todos los componentes necesarios para ejecutar cualquier servicio o aplicación.

Describe su infraestructura completa como código, incluso si se extiende a múltiples proveedores de servicios. Sus servidores pueden estar de Openstack, su DNS de AWS, y la base de datos de MySQL. Terraform construirá todos estos recursos a través de todos estos proveedores en paralelo.

Terraform codifica el conocimiento sobre su infraestructura a diferencia de cualquier otra herramienta, y proporciona el flujo de trabajo y herramientas para cambiar y actualizar la infraestructura con seguridad.

Esta herramienta le permite crear, cambiar y mejorar de manera segura y previsible la infraestructura de producción. Es de código abierto que codifica APIs en archivos de configuración que pueden ser compartidos entre miembros del equipo, tratados como código, editados, revisados y versionados.

CAPÍTULO 2

Características

Infraestructura como Código

La infraestructura se describe utilizando una sintaxis de configuración de alto nivel. Esto permite que un modelo de su centro de datos sea versionado y tratado como lo haría con cualquier otro código. Además, la infraestructura puede ser compartida y reutilizada.

Planes de Ejecución

Terraform tiene un paso de "planificación" donde genera un plan de ejecución . El plan de ejecución muestra lo que hará cuando ejecutemos terraform plan. Esto le permite evitar sorpresas cuando Terraform manipule la infraestructura. Por ejemplo, después de crear un servidor web, podemos cambiar el tamaño de la memoria RAM del servidor de DigitalOcean. El símbolo [~] antes del Droplet significa que Terraform actualizará el recurso, en lugar de destruir o recrear.

```
$ Terraform plan
~ Digitalocean_droplet.web
    Tamaño: "512mb" => "1gb"
```

Representación gráfica de recursos

Terraform construye un gráfico de todos sus recursos y paraleliza la creación y modificación de cualquier recurso no dependiente. Debido a esto, construye la infraestructura lo más eficientemente posible, y los operadores obtienen una visión sobre las dependencias en su infraestructura.

Automatización de los cambios

Los conjuntos de cambios complejos se pueden aplicar a su infraestructura con una mínima interacción humana. Con el plan de ejecución y el gráfico de recursos antes mencionados, usted sabe exactamente lo que cambiará Terraform y en qué orden, evitando muchos posibles errores humanos.

CAPÍTULO 3

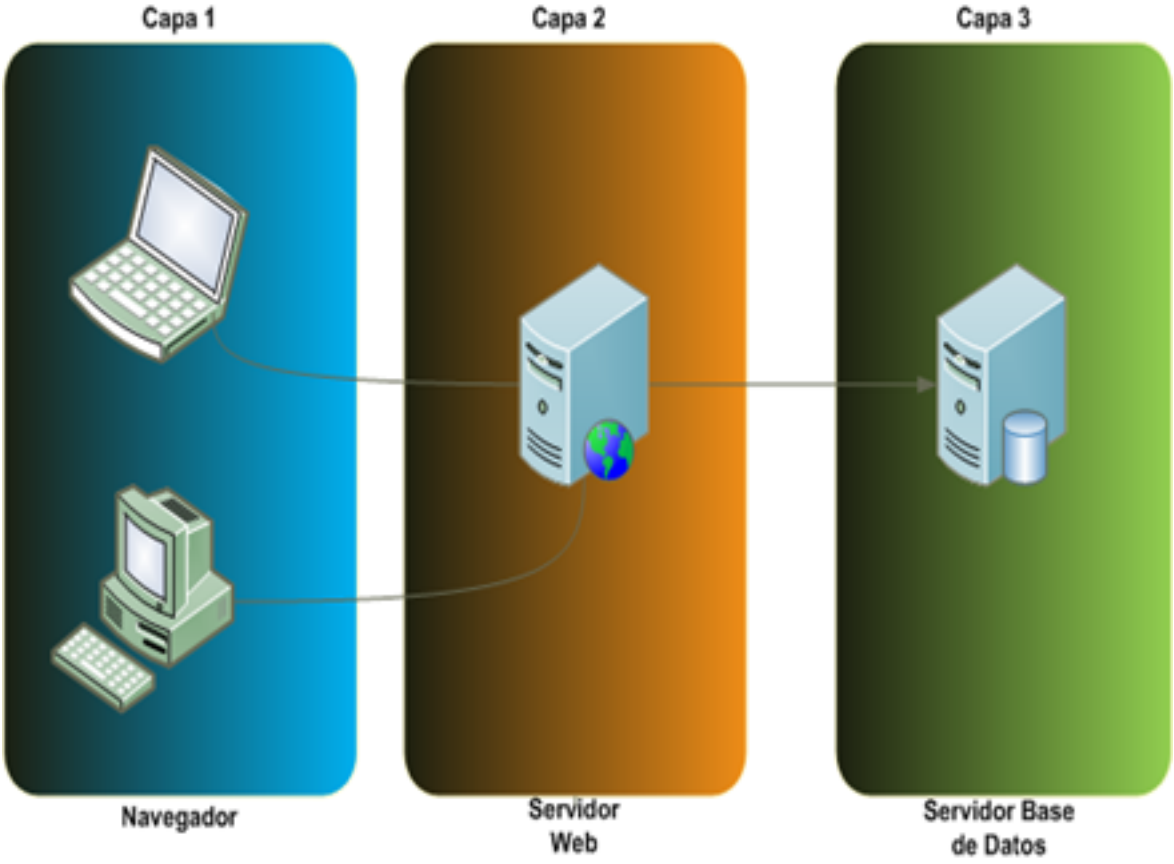
Casos de uso

Terraform se puede usar para codificar la configuración requerida para una aplicación, asegurando que todos los complementos necesarios estén disponibles, pero puede ir aún más lejos: configurar un DNSimple para configurar un CNAME o configurar Cloudflare como CDN para la aplicación. Lo mejor de todo es que puede hacer todo esto en menos de 30 segundos sin usar una interfaz web.

Aplicaciones de varios niveles

Un patrón muy común es la arquitectura N-tier. La arquitectura de dos niveles más común es un grupo de servidores web que utilizan un nivel de base de datos. Se añaden niveles adicionales para servidores API, servidores de almacenamiento en caché, mallas de enrutamiento, etc. Este patrón se utiliza porque los niveles se pueden escalar de forma independiente y proporcionar una separación de los servicios.

Arquitectura N-tier:



Terraform es una herramienta ideal para construir y gestionar estas infraestructuras. Cada nivel se puede describir como una colección de recursos y las dependencias entre cada nivel se manejan automáticamente; Terraform garantizará que el nivel de base de datos esté disponible antes de que se inicien los servidores web y que los equilibradores de carga tengan conocimiento de los nodos web. Cada nivel se puede escalar fácilmente usando Terraform modificando un único valor de configuración de count. Debido a que la creación y el aprovisionamiento de un recurso está codificado y automatizado, la escala elástica con carga se vuelve trivial.

Clústers de autoservicio

Con un cierto tamaño de organización, se vuelve muy difícil para un equipo de operaciones centralizado gestionar una infraestructura grande y creciente. En cambio, se vuelve más atractivo crear una infraestructura de "autoservicio", lo que permite a los equipos de productos gestionar su propia infraestructura utilizando herramientas proporcionadas por el equipo de operaciones centrales.

Utilizando Terraform, el conocimiento de cómo construir y escalar un servicio puede ser codificado en una configuración. Las configuraciones de Terraform pueden ser compartidas dentro de una organización, permitiendo a los equipos de los clientes utilizar la configuración como una caja negra y utilizar Terraform como una herramienta para administrar sus servicios.

Demos de Software

El software moderno está cada vez más conectado y distribuido. Aunque existen herramientas como Vagrant para crear entornos virtualizados para demostraciones, todavía es muy difícil de encontrar software en infraestructura real que se adapte más a los entornos de producción.

Los programadores de software pueden proporcionar una configuración de Terraform para crear, suministrar e iniciar una demostración de proveedores en la nube como AWS. Esto permite a los usuarios finales probar fácilmente el software en su propia infraestructura e incluso permite ajustar parámetros como el tamaño de clúster para probar más rigurosamente las herramientas a cualquier escala.

Entornos de Prueba

Es una práctica común tener un entorno de producción o QA(quality assurance). Estos entornos son clones más pequeños del entorno de producción, pero se utilizan para probar nuevas aplicaciones antes de liberar en el entorno de producción. A medida que el entorno de producción crece y se vuelve más complejo, resulta cada vez más difícil de mantener actualizado. Con Terraform, el entorno de producción puede ser codificado y luego compartido con QA o dev. Estas configuraciones pueden utilizarse para hacer girar rápidamente nuevos entornos para probar y luego ser eliminados fácilmente. Esta herramienta puede ayudar a manejar la dificultad de mantener ambientes paralelos, y los hace práctico para crearlos y destruirlos elásticamente.

Redes definidas por software

Las redes definidas por software (SDN) se está convirtiendo cada vez más frecuente en el centro de datos, ya que proporciona más control a los operadores y desarrolladores y permite a la red apoyar mejor las aplicaciones que se ejecutan en la capa superior. La mayoría de las implementaciones SDN tienen una capa de control y una capa de infraestructura.

Terraform se puede utilizar para codificar la configuración de redes definidas por software. Esta configuración puede entonces ser utilizada por Terraform para configurar y modificar automáticamente la configuración al interactuar con la capa de control. Esto permite que la configuración se versione y que los cambios sean automatizados. Como ejemplo, AWS Virtual Private Cloud(VPC) es una de las implementaciones SDN más utilizadas y puede ser configurada por Terraform.

Programadores de recursos

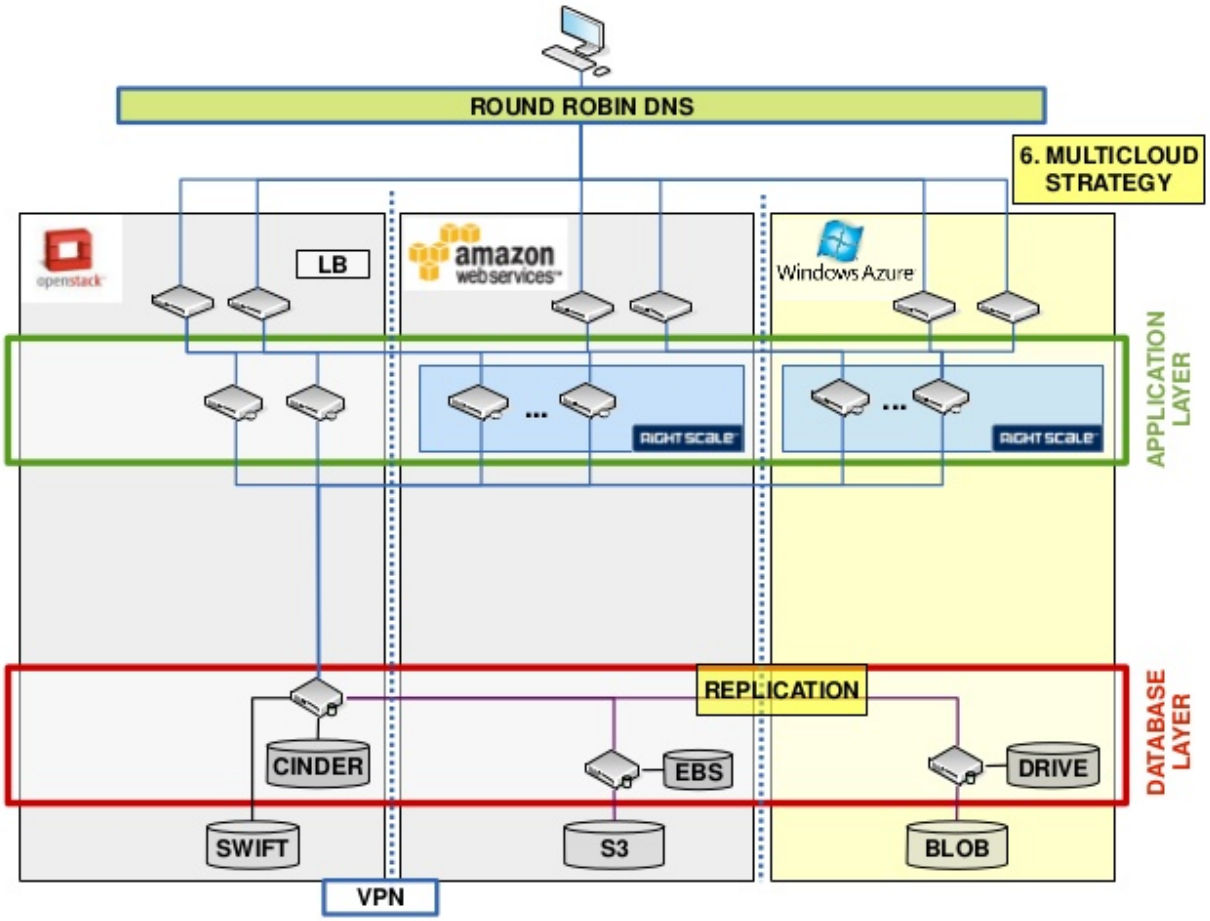
En infraestructuras de gran escala, la asignación estática de aplicaciones a las máquinas se vuelve cada vez más difícil. Para resolver ese problema, hay una serie de orquestadores como Borg, Mesos, YARN y Kubernetes. Éstos se pueden utilizar para programar dinámicamente contenedores Docker, Hadoop, Spark y muchas otras herramientas de software.

Terraform no se limita a proveedores físicos como AWS (Amazon Web Services). Los planificadores de recursos pueden ser tratados como un proveedor, lo que permite a esta herramienta solicitar recursos de ellos. Esto permite que se utilice en capas: para configurar la infraestructura física que ejecuta los planificadores así como el aprovisionamiento en la red programada.

Implementación de multi-cloud

A menudo es necesario extender la infraestructura a través de múltiples nubes para aumentar la tolerancia a fallos. Mediante el uso de una única región o proveedor de la nube, la tolerancia a fallos está limitada por la disponibilidad de ese proveedor. Tener un despliegue multi-cloud permite una recuperación más sencilla ante la pérdida de una región o de un proveedor entero.

Infraestructura multi-cloud:



Realizar implementaciones multi-nube puede ser muy difícil ya que muchas herramientas existentes para la administración de infraestructura son específicas de la nube. Terraform permite que una única configuración se utilice para administrar múltiples proveedores e incluso manejar dependencias entre nubes. Esto simplifica la gestión y la orquestación, ayudando a los operadores a construir infraestructuras multi-cloud a gran escala.

CAPÍTULO 4

Instalación de Terraform

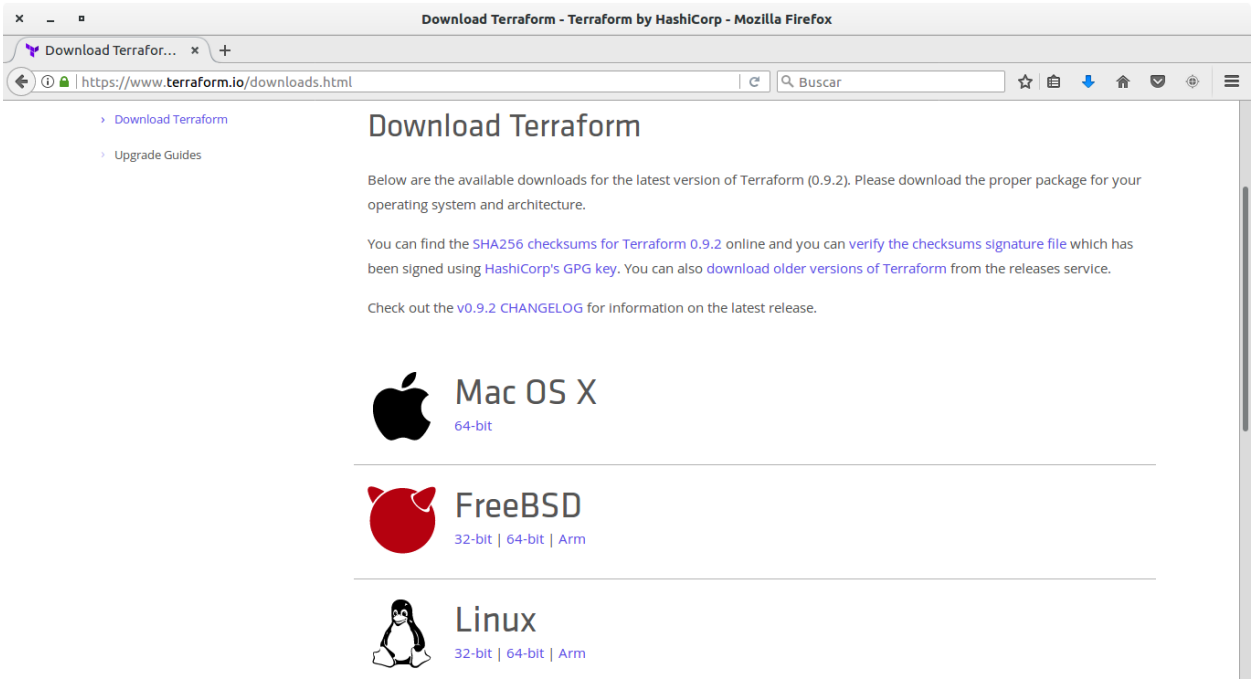
Para instalar Terraform, descargamos el paquete apropiado para nuestro sistema desde la página oficial de Terraform.

Sistemas operativos compatibles con Terraform:

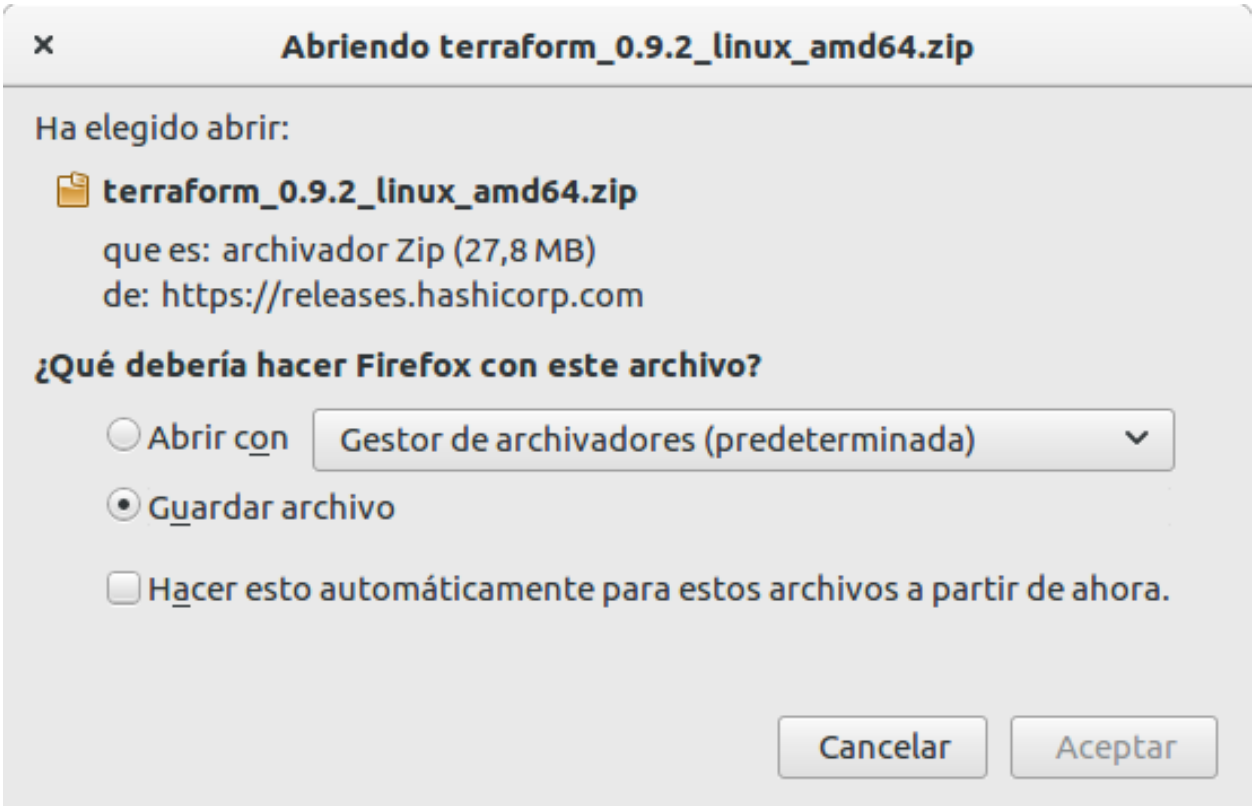
- Linux: 32bit | 64bit | Arm
- Windows: 32bit | 64bit
- Mac OS X: 64bit
- FreeBSD: 32bit | 64bit | Arm
- OpenBSD: 32bit | 64bit
- Solaris: 64bit

Instalación en sistema operativo Linux

Vamos a la página oficial https://www.terraform.io/downloads.html. En ella encontramos los binarios.



Seleccionamos el sistema operativo y la arquitectura, en nuestro caso elegiremos Linux 64bit puesto que lo instalaremos en una maquina Debian jessie.



Creamos un directorio para los binarios de Terraform.

```
root@castillo:/home/jose# mkdir terraform
root@castillo:/home/jose#
```

Moveremos el fichero descargado anteriormente al interior de la carpeta.

```
root@castillo:/home/jose# mv Descargas/terraform_0.9.2_linux_amd64.zip terraform/.
root@castillo:/home/jose#
```

Nos situamos en el directorio terraform.

```
root@castillo:/home/jose# cd terraform/
root@castillo:/home/jose/terraform#
```

Descomprimimos los binarios de Terraform con el comando unzip.

```
root@castillo:/home/jose/terraform# unzip terraform_0.9.2_linux_amd64.zip
Archive:  terraform_0.9.2_linux_amd64.zip
  inflating: terraform
root@castillo:/home/jose/terraform#
```

Exportamos las variables de entorno de Terraform.

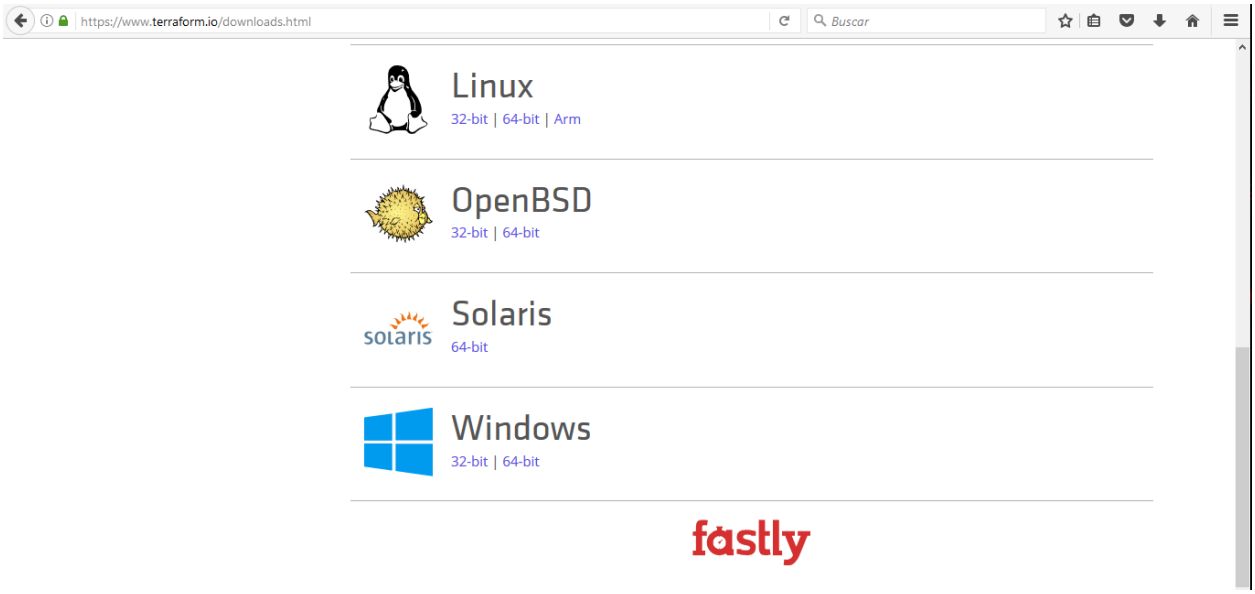
```
root@castillo:/home/jose/terraform# export PATH=/home/jose/terraform:$PATH
root@castillo:/home/jose/terraform#
```

Por último comprobamos que se ha instalado bien ejecutando el comando siguiente:

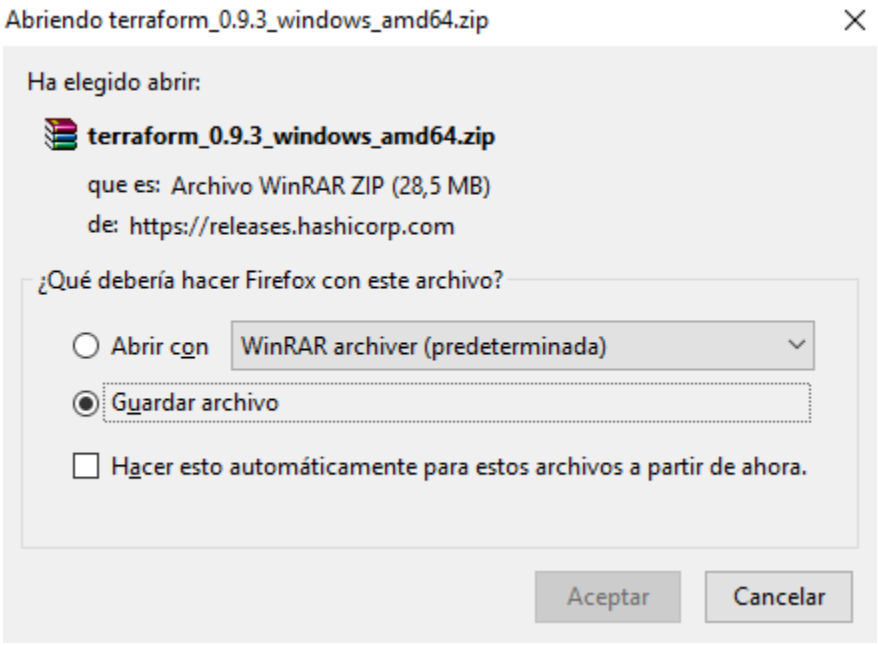
```
root@castillo:/home/jose/terraform# terraform --version
Terraform v0.9.2

root@castillo:/home/jose/terraform#
```

Instalación en sistema operativo Windows

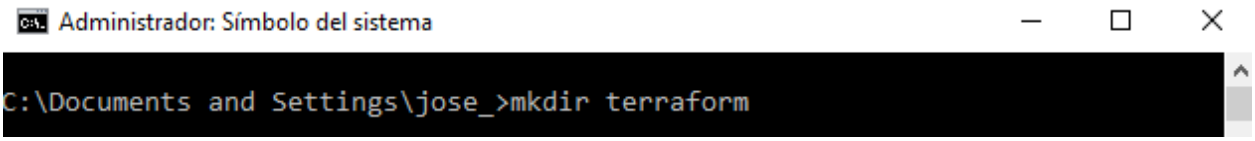
Vamos a la página oficial https://www.terraform.io/downloads.html. En ella encontramos el ejecutable.



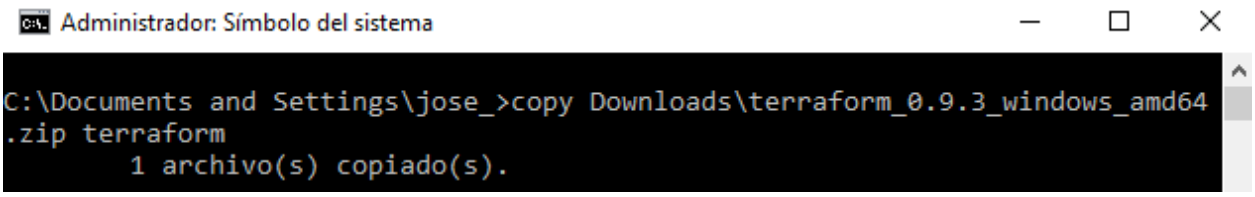
Seleccionamos el sistema operativo y la arquitectura, en nuestro caso elegiremos Windows 64bit puesto que lo instalaremos en una maquina Windows 10.



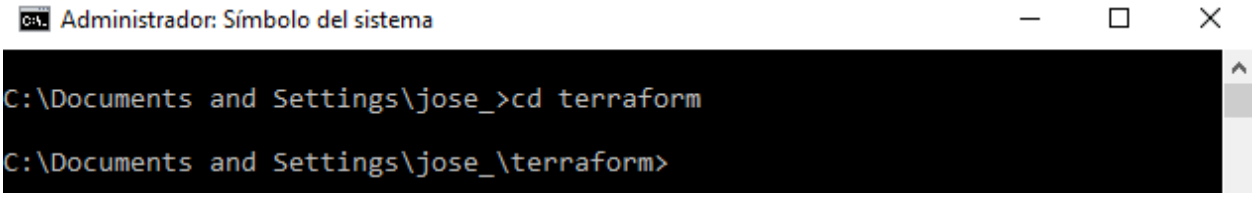
Creamos un directorio para el ejecutable de Terraform.



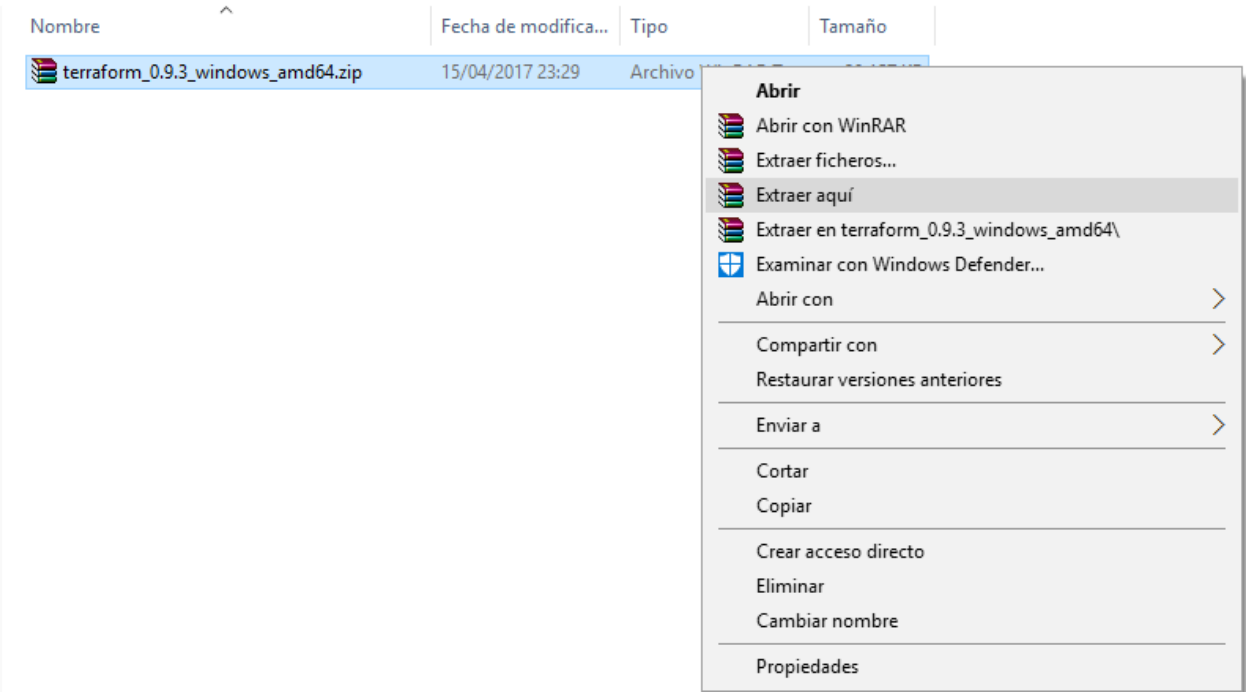
Copiamos el fichero descargado anteriormente al directorio terraform.



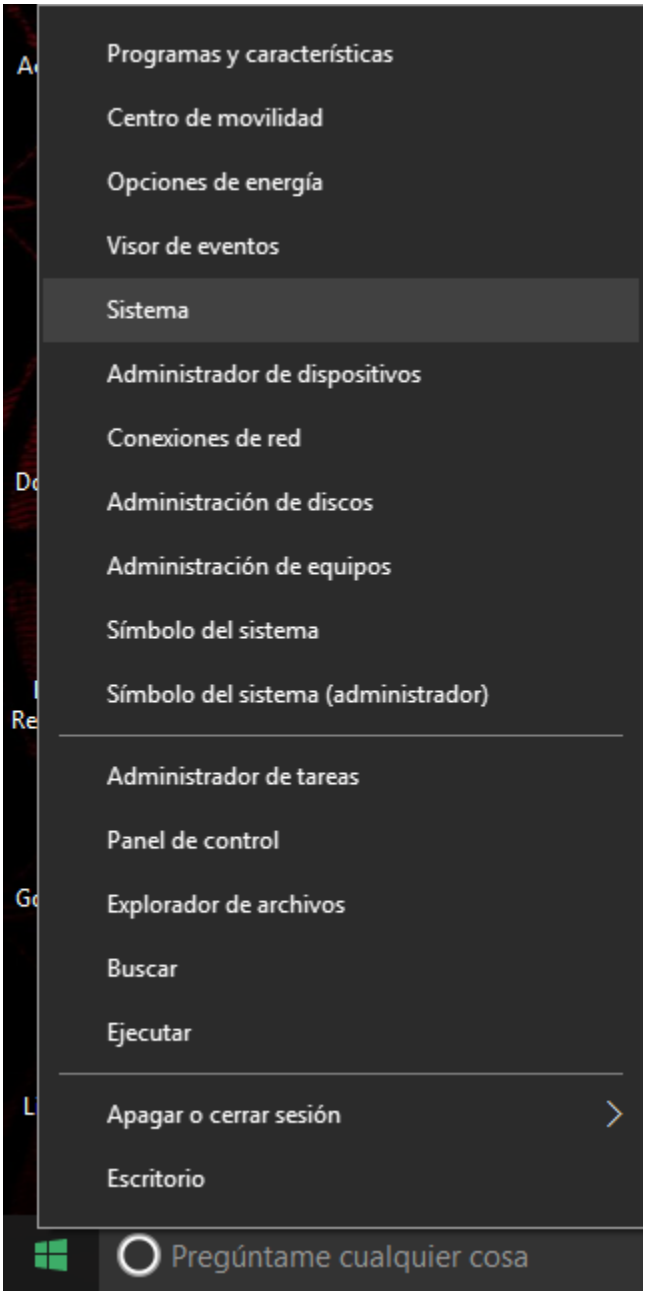
Nos situamos en el directorio terraform.



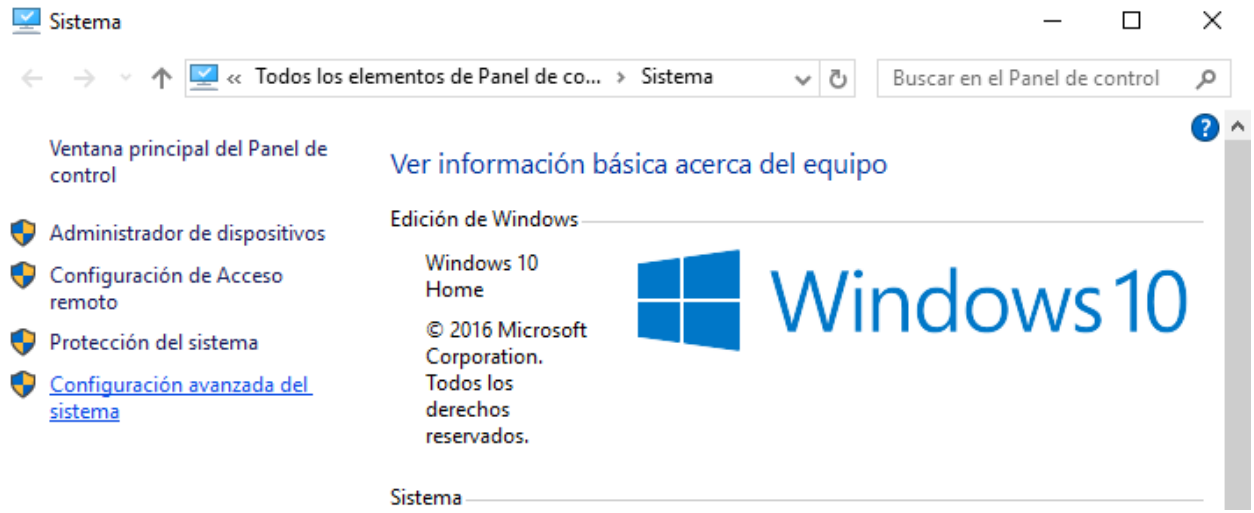
Descomprimimos el ejecutable en el interior del directorio creado anteriormente.



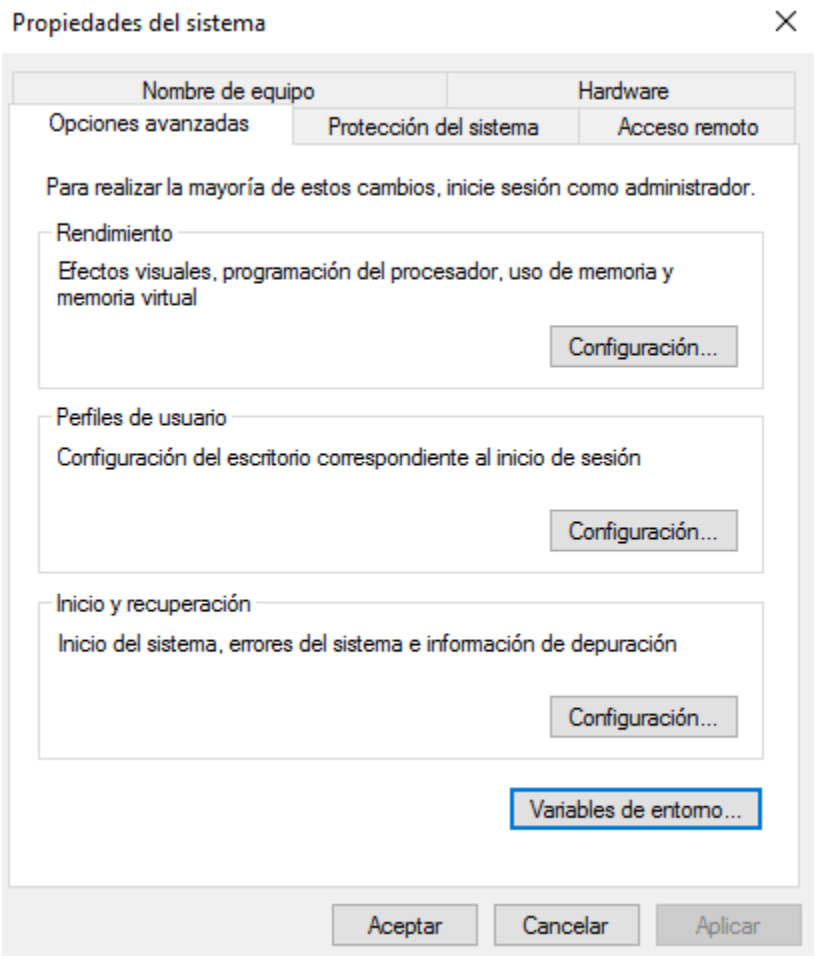
Ahora añadimos la variable de entorno de Terraform para el usuario, para ello en el menú inicio de Windows hacemos clic derecho en sistema.



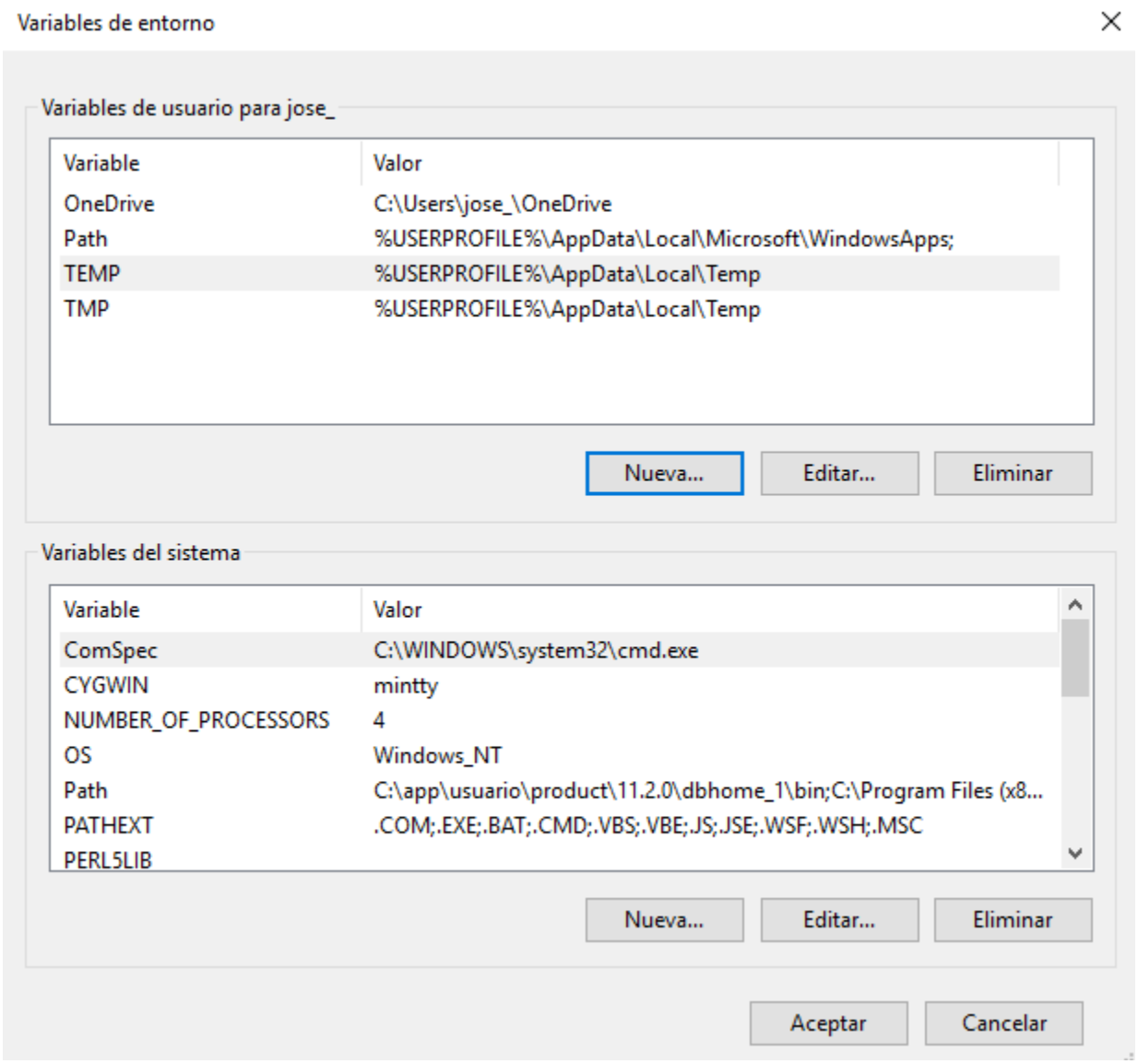
Hacemos clic en Configuración avanzada del sistema.



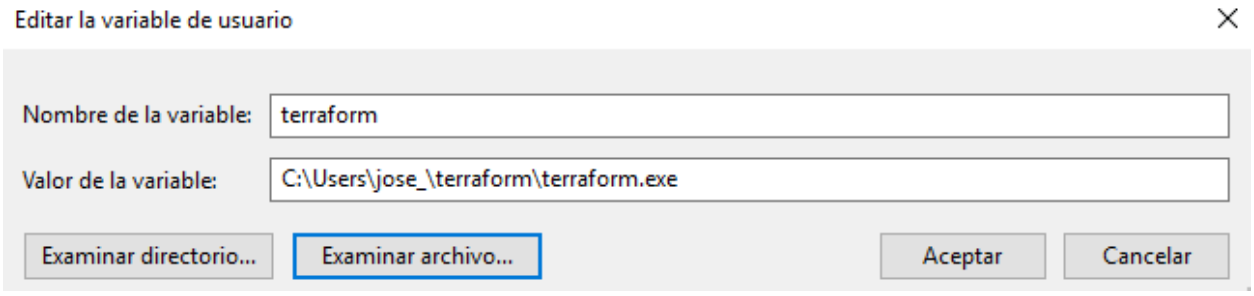
Hacemos clic en Variables de entorno.



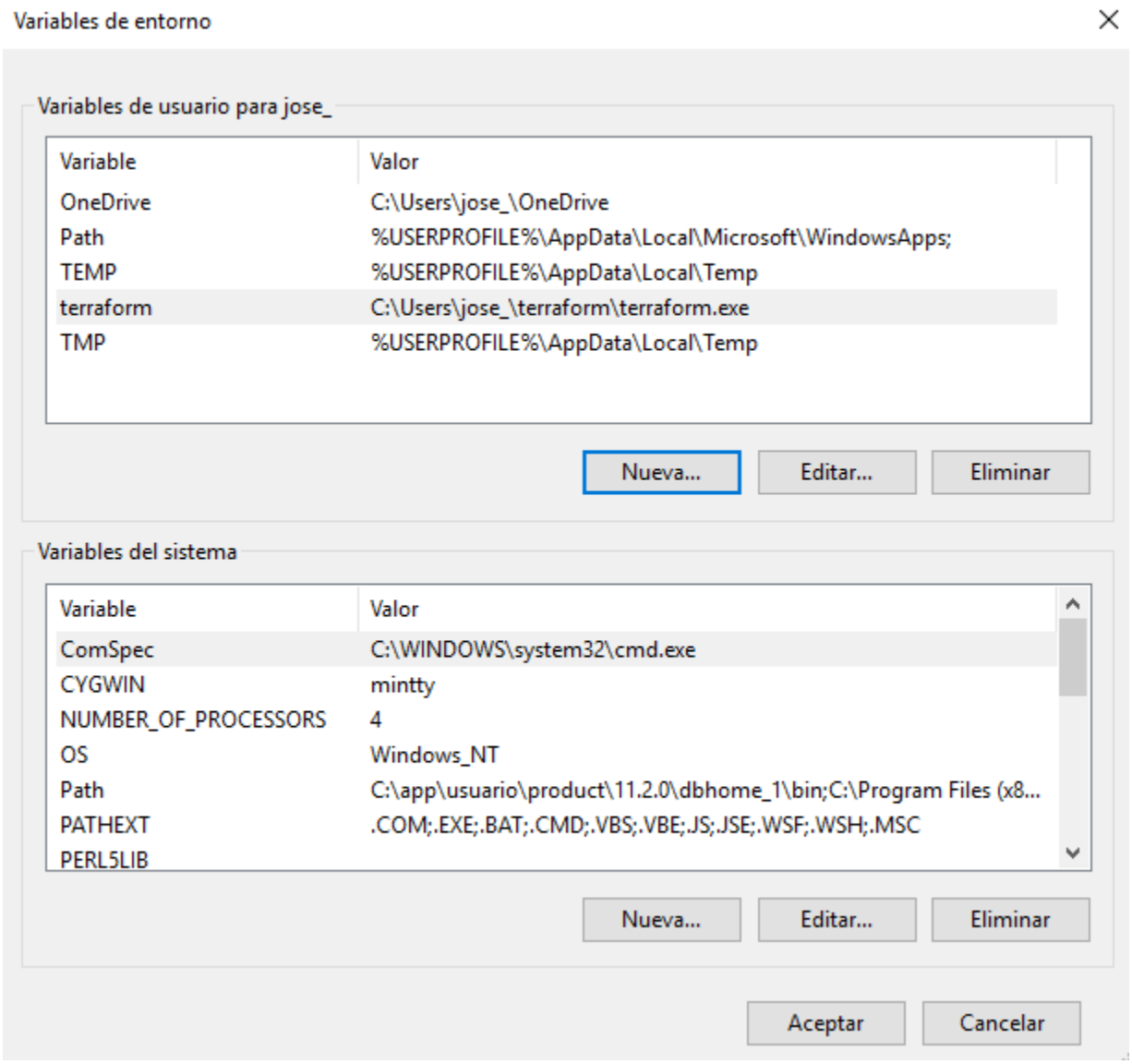
En la siguiente pantalla pulsamos el botón Nueva.



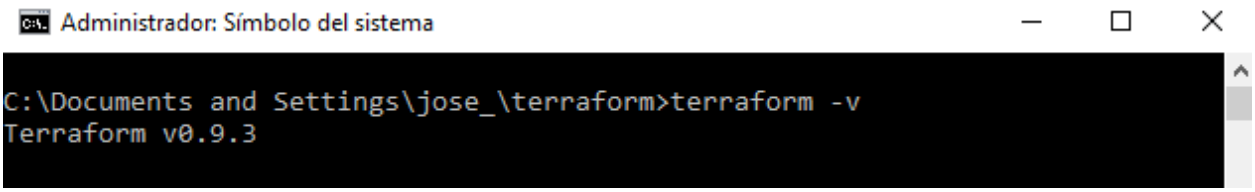
Asignamos un nombre a la variable y en el botón Examinar archivo buscamos la ruta del ejecutable de Terraform.



Aceptamos y comprobamos que se ha guardado la variable de entorno.



Ahora desde una ventana de símbolo de sistema comprobamos que Terraform funciona.



CAPÍTULO 5

Comandos de Terraform

Terraform se controla a través de una interfaz de línea de comandos muy fácil de usar. Para ver una lista de los comandos disponibles en cualquier momento, simplemente ejecutamos terraform sin argumentos:

```
jose@castillo:~$ terraform
Usage: terraform [--version] [--help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Environment management
    fmt                Rewrites config files to canonical format
    force-unlock       Manually unlock the terraform state
    get                Download and install modules for the configuration
    graph              Create a visual graph of Terraform resources
    import             Import existing infrastructure into Terraform
    init               Initialize a new or existing Terraform configuration
    output             Read an output from a state file
    plan               Generate and show an execution plan
    push               Upload this Terraform module to Atlas to run
    refresh            Update local state file against real resources
    show               Inspect Terraform state or plan
    taint              Manually mark a resource for recreation
    untaint            Manually unmark a resource as tainted
    validate           Validates the Terraform files
    version            Prints the Terraform version

All other commands:
    debug              Debug output management (experimental)
    state              Advanced state management
```

Para obtener ayuda para cualquier comando específico, introducimos el parámetro -h o –help al subcomando correspondiente. Por ejemplo, para ver la ayuda sobre el subcomando console.

```
jose@castillo:~$ terraform -h console
Usage: terraform console [options] [DIR]

  Starts an interactive console for experimenting with Terraform
  interpolations.

  This will open an interactive console that you can use to type
  interpolations into and inspect their values. This command loads the
  current state. This lets you explore and test interpolations before
  using them in future configurations.

  This command will never modify your state.

  DIR can be set to a directory with a Terraform state to load. By
  default, this will default to the current working directory.

Options:

  -state=path            Path to read state. Defaults to "terraform.tfstate"

  -var 'foo=bar'         Set a variable in the Terraform configuration. This
                         flag can be set multiple times.

  -var-file=foo          Set variables in the Terraform configuration from
                         a file. If "terraform.tfvars" is present, it will be
                         automatically loaded if this flag is not specified.
```

Comando apply

El comando terraform apply se utiliza para aplicar los cambios necesarios para alcanzar el estado deseado de la configuración, o el conjunto predefinido de acciones generado por un plan ejecución.

    terraform apply [options] [dir-or-plan]

De forma predeterminada, apply examina el directorio actual de la configuración y aplica los cambios de forma adecuada. Sin embargo, se puede proporcionar una ruta a otra configuración o un plan de ejecución. Los planes de ejecución se pueden utilizar para ejecutar solamente un conjunto predeterminado de acciones.

Los indicadores de línea de comandos son todos opcionales. La lista de parámetros disponibles es:

- -backup=path - Ruta de acceso al archivo de copia de -backup=path . El valor predeterminado es -state-out con la extensión ”.backup”.
- -lock=true - Bloquea el archivo de estado cuando se admite el bloqueo.
- -lock-timeout=0s - Duración de volver a intentar un bloqueo de estado.
- -input=true - Solicita entrada para variables si no se establece directamente.
- -no-color - Desactiva la salida con color.
- -parallelism=n - Limita el número de operaciones concurrentes cuando Terraform recorre el gráfico .
- -refresh=true - Actualiza el estado de cada recurso antes de planificar y aplicar. Esto no tiene efecto si un archivo del plan se da directamente para aplicar.
- -state=path - Ruta al archivo de estado. El valor predeterminado es “terraform.tfstate”. Se ignora cuando se utiliza el estado remoto .

- -state-out=path - Ruta para escribir el archivo de estado actualizado. Por defecto
- -state usará la ruta. Se ignora cuando se utiliza el estado remoto .
- -target=resource - Una dirección de recursos para el destino. La operación se limitará a este recurso y a sus dependencias. Este indicador se puede utilizar varias veces.
- -var 'foo=bar' - Establece una variable en la configuración de Terraform. Este indicador se puede establecer varias veces. Los valores de las variables se interpretan como Lenguaje de Configuración HashiCorp (HCL) , por lo que los valores de lista y mapa se pueden especificar mediante este indicador.
- -var-file=foo - Establece variables en la configuración de Terraform desde un archivo de variable . Si "terraform.tfvars" está presente, se cargará automáticamente primero. Cualquier archivo especificado por -var-file reemplaza cualquier valor en un "terraform.tfvars". Este indicador se puede utilizar varias veces.

Comando console

El comando terraform console crea una consola interactiva.

```
terraform console [options] [dir]
```

Esto es útil para probar antes de usarlas en configuraciones, así como interactuar con un estado existente.

Si no existe un archivo de estado, la consola todavía funciona y se puede utilizar para experimentar con funciones compatibles.

El dir se puede utilizar para abrir una consola para un directorio de configuración de Terraform específico. Esto cargará cualquier estado de ese directorio, así como la configuración. El valor predeterminado es el directorio de trabajo actual. El comando console no requiere que el estado de Terraform o la configuración funcione.

La lista de parámetros disponible es:

- -state=path - Ruta al archivo de estado. De forma terraform.tfstate a terraform.tfstate . No es necesario que exista un archivo de estado. Puede cerrar la consola con el comando exit o mediante Control-C o Control-D.

Comando destroy

El comando terraform destroy se utiliza para destruir la infraestructura gestionada por Terraform.

```
terraform destroy [options] [dir]
```

La infraestructura gestionada por Terraform será destruida. Esto pedirá confirmación antes de destruir. Este comando acepta todos los argumentos y parámetros que acepta el comando apply , con la excepción de un argumento de archivo de plan.

Si se establece -force , entonces la confirmación de destruir no se mostrará.

El indicador -target , en lugar de afectar a las "dependencias", también destruirá los recursos que dependan de los objetivos especificados.

El comportamiento de cualquier comando de destrucción de terraform se puede previsualizar en cualquier momento con un comando equivalente terraform plan -destroy.

Comando env

El comando terraform env se utiliza para administrar entornos. Este comando es un subcomando anidado, lo que significa que tiene subcomandos adicionales.

terraform env [nombre]

- list: El comando mostrará todos los entornos creados.
- select: El comando terraform env select se utiliza para seleccionar un entorno diferente ya creado.
- new: El comando terraform env new se utiliza para crear un nuevo entorno de estado.
- delete: El comando terraform env delete se utiliza para eliminar un entorno existente.

Comando fmt

El comando terraform fmt se utiliza para reescribir los archivos de configuración de Terraform a un formato y estilo normalizado.

terraform fmt [options] [dir]

De forma predeterminada, fmt explora el directorio actual para los archivos de configuración. Si se proporciona el argumento dir , entonces escaneará ese directorio dado. Si dir es un solo guión (-) entonces fmt leerá desde entrada estándar (STDIN).

- -list=true - Lista de archivos cuyo formato difiere (desactivado si se utiliza STDIN).
- -write=true - Escribe el resultado en el archivo fuente en lugar de STDOUT (deshabilitado si utiliza STDIN).
- -diff=false - Mostrar diferencias de cambios de formato.

Comando force-unlock

Este comando elimina el bloqueo del estado para la configuración actual. El comportamiento de este bloqueo depende del backend que se utiliza. Los archivos de estado local no pueden desbloquearse por otro proceso. Este comando no modificará la infraestructura.

terraform force-unlock [dir]

Opciones:

- -force – Fuerza la eliminación del bloqueo del estado para la configuración actual.

Comando get

El comando terraform get se utiliza para descargar y actualizar módulos.

terraform get [options] [dir]

Los módulos se descargan en una carpeta .terraform local. Esta carpeta no debe asignarse al control de versiones. La carpeta .terraform se crea en relación con su directorio de trabajo actual, independientemente del argumento dir dado a este comando.

Opciones:

- -update - Con la opción update a los módulos descargados comprueba si tienen actualizaciones y se las descarga si esta disponible.

Comando graph

El comando terraform graph se utiliza para generar una representación visual de una configuración o plan de ejecución.

terraform graph [options] [dir]

Emite el gráfico de dependencia visual de los recursos de Terraform según los archivos de configuración en dir (o el directorio actual si se omite).

El gráfico se emite en formato DOT. El programa típico que puede leer este formato es GraphViz, pero muchos servicios web también están disponibles para leer este formato.

Opciones:

- -draw-cycles - Destaque cualquier ciclo en el gráfico con bordes coloreados. Esto ayuda al diagnosticar errores de ciclo.
- -no-color - Si se especifica, la salida no contendrá ningún color.
- -type=plan - Tipo de gráfico de salida. Puede ser: plan, plan-destroy, apply, legacy.

Comando import

El comando de terraform import se utiliza para importar recursos a Terraform. terraform import [options] ADDRESS ID

La importación buscará el recurso existente del ID y lo importará en la ADDRESS dada.

ADDRESS debe ser una dirección de recurso válida. Dado que cualquier dirección de recursos es válida, el comando import puede importar recursos en módulos.

ID depende del tipo de recurso importado. Por ejemplo, para instancias de AWS es el ID i-abcd1234 (i-abcd1234), pero para las zonas de AWS Route53 es el ID de zona (Z12ABC4UGMOZ2N). Cada proveedor tiene su formato de ID.

Opciones:

- -backup=path - Ruta de acceso al archivo de estado -backup=path . La opción predeterminada es la ruta -state-out con la extensión ".backup". Establecer en "-" para deshabilitar las copias de seguridad.
- -config=path - Ruta al directorio de archivos de configuración de Terraform que configuran el proveedor para la importación. El valor predeterminado es su directorio de trabajo. Si este directorio no contiene archivos de configuración de Terraform, el proveedor debe configurarse a través de entradas manuales o variables.
- -input=true - Si desea solicitar información para la configuración del proveedor.
- -lock=true - Bloquea el archivo de estado cuando se admite el bloqueo.
- -lock-timeout=0s - Duración de volver a intentar un bloqueo de estado.
- -no-color - Si se especifica, la salida no contendrá ningún color.
- -provider=provider - Proveedor especificado para usar para la importación. Se utiliza para especificar alias de proveedor, como "aws.eu". Por defecto, el proveedor normal se basa en el prefijo del recurso que se está importando. Por lo general, no es necesario especificar esto.
- -state=path - La ruta de acceso para leer y guardar archivos de estado (a menos que se especifique estado-out).
- -state-out=path - Ruta para escribir el archivo de estado final. De forma predeterminada, esta es la ruta de estado.

- -var 'foo=bar' - Establece una variable en la configuración de Terraform. Este indicador se puede establecer varias veces. Los valores de las variables se interpretan como HCL , por lo que los valores de lista y mapa se pueden especificar mediante este indicador.
- -var-file=foo - Establece variables en la configuración de Terraform desde un archivo de variable . Si "terraform.tfvars" está presente, se cargará automáticamente primero. Cualquier archivo especificado por -var-file valor en un "terraform.tfvars".

Comando init

El comando terraform init se utiliza para inicializar una configuración de Terraform. Este es el primer comando que se debe ejecutar para cualquier configuración de Terraform nueva o existente.

    terraform init [options] [SOURCE] [PATH]

Opciones:

- -backend=true - Inicializa el backend para este entorno.
- -backend-config=value - El valor puede ser una ruta a un archivo HCL o una cadena en el formato de 'key = value'. Esto especifica la configuración adicional para combinar para el backend. Esto se puede especificar varias veces. Los indicadores especificados más adelante en la línea anulan los especificados anteriormente si entran en conflicto.
- -get=true - Descargue cualquier módulo para esta configuración.
- -input=true - Solicite información interactiva si es necesario. Si esto es falso y la entrada es init , init error.
- -lock=true - Bloquea el archivo de estado cuando se admite el bloqueo.
- -lock-timeout=0s - Duración de volver a intentar un bloqueo de estado.
- -no-color - Si se especifica, la salida no contendrá ningún color.
- -force-copy - Suprime las indicaciones sobre la copia de datos de estado. Esto es equivalente a proporcionar un "sí" a todos los mensajes de confirmación.

Comando output

El comando terraform output se utiliza para extraer el valor de una variable de salida de un fichero de estado.

    terraform output [options] [name]

Opciones:

- -json - Si se especifica, las salidas se formatean como un objeto JSON, con una clave por salida. Si se especifica [name] , sólo se devolverá la salida especificada. Esto se puede canalizar en herramientas tales como jq para el procesamiento adicional.
- -state=path - Ruta al archivo de estado. El valor predeterminado es "terraform.tfstate". Se ignora cuando se utiliza el estado remoto .
- -module=module_name - La ruta del módulo que ha -module=module_name salida. Por defecto es la ruta raíz. Otros módulos se pueden especificar mediante una lista separada por períodos. Ejemplo: "foo" haría referencia al módulo "foo" pero "foo.bar" haría referencia al módulo "bar" en el módulo "foo".

Comando plan

El comando terraform plan se utiliza para crear un plan de ejecución. Terraform realiza una actualización, a menos que se deshabilite explícitamente y, a continuación, determina qué acciones son necesarias para lograr el estado deseado especificado en los archivos de configuración.

terraform plan [options] [dir-or-plan]

Opciones:

- -destroy - Si se establece, genera un plan para destruir todos los recursos conocidos.
- -detailed-exitcode - Devuelve un código de salida -detailed-exitcode cuando sale el comando. Cuando se proporciona, este argumento cambia los códigos de salida y sus significados para proporcionar información más granular sobre lo que el plan resultante contiene:

  0 = Sin cambios 1 = Error 2 = Cambios presentes
- -input=true - Solicitar entrada para variables si no se establece directamente.
- -lock=true - Bloquea el archivo de estado cuando se admite el bloqueo.
- -lock-timeout=0s - Duración de volver a intentar un bloqueo de estado.
- -module-depth=n - Especifica la profundidad de los módulos a mostrar en la salida. Esto no afecta al plan en sí, sólo la salida mostrada. De forma predeterminada, esto es -1, lo que ampliará todo.
- -no-color - Desactiva la salida con color.
- -out=path - La ruta para guardar el plan de ejecución generado. Este plan se puede utilizar con terraform apply para terraform apply de que solo se aplican los cambios que se muestran en este plan. Lea la advertencia sobre los planes guardados a continuación.
- -parallelism=n - Limita el número de operaciones concurrentes cuando Terraform recorre el gráfico .
- -refresh=true - Actualiza el estado antes de verificar las diferencias.
- -state=path - Ruta al archivo de estado. El valor predeterminado es “terraform.tfstate”. Se ignora cuando se utiliza el estado remoto .
- -target=resource - Una -target=resource al destino. La operación se limitará a este recurso ya sus dependencias. Este indicador se puede utilizar varias veces.
- -var ‘foo=bar’ - Establece una variable en la configuración de Terraform. Este indicador se puede establecer varias veces. Los valores de las variables se interpretan como HCL , por lo que los valores de lista y mapa se pueden especificar mediante este indicador.
- -var-file=foo - Establece variables en la configuración de Terraform desde un archivo de variable . Si “terraform.tfvars” está presente, se cargará automáticamente primero. Cualquier archivo especificado por -var-file valor en un “terraform.tfvars”. Este indicador se puede utilizar varias veces.

Comando push

El comando terraform push carga su configuración de Terraform para que sea gestionada por Terraform Enterprise. Al cargar su configuración en Terraform Enterprise, puede ejecutar Terraform automáticamente, guardará todas las transiciones de estado, planes y mantendrá un historial de todas las ejecuciones de Terraform.

Esto hace que sea mucho más fácil usar Terraform como un equipo: los miembros del equipo modifican las configuraciones de Terraform localmente y continúan usando el control de versión normal. Cuando las configuraciones de Terraform están listas para ser ejecutadas, son enviadas a Terraform Enterprise, y cualquier miembro de su equipo puede ejecutar Terraform con solo pulsar un botón.

terraform push [options] [path]

Opciones:

- -atlas-address= - Una dirección alternativa a una instancia.
- -upload-modules=true - Si es true (-upload-modules=true), entonces los módulos que se usan están bloqueados en su comprobación actual y cargados completamente.
- -name= - Nombre de la configuración de la infraestructura en Terraform Enterprise. El formato de este es: "nombre de usuario / nombre" para que pueda cargar configuraciones no sólo a su cuenta, sino a otras cuentas y organizaciones.
- -no-color - Desactiva la salida con color
- -overwrite=foo - Marca una variable -overwrite=foo . Normalmente, si ya está definida una variable, Terraform no enviará el valor local (aunque sea diferente). Esto obliga a enviar el valor local a Terraform Enterprise.
- -token= - El token de API de Terraform Enterprise que se utiliza para autorizar la subida. Si está en blanco o no especificado, se utilizará la variable de ATLAS_TOKEN ATLAS_TOKEN.
- -var='foo=bar' - -var='foo=bar' el valor de una variable para la configuración de Terraform.
- -var-file=foo - Establece el valor de las variables usando un archivo de variable. Este indicador se puede utilizar varias veces.
- -vcs=true - Si es true (por defecto), entonces Terraform detectará si un VCS está en uso, como Git, y solo subirá archivos que estén comprometidos con el control de versiones. Si no se detecta ningún sistema de control de versiones, Terraform cargará todos los archivos en path (parámetro al comando).

Comando refresh

El comando terraform refresh se utiliza para actualizar el fichero de estado usando la infraestructura real. Esto se puede utilizar para detectar cualquier diferencia desde el último estado conocido.

terraform refresh [options] [dir]

Opciones:

- -backup=path - Ruta de acceso al archivo de copia de -backup=path . El valor predeterminado es -state-out con la extensión ".backup". Deshabilitado ajustando a "-".
- -no-color - Desactiva la salida con color
- -input=true - Solicitar entrada para variables si no se establece directamente.
- -lock=true - Bloquea el archivo de estado cuando se admite el bloqueo.
- -lock-timeout=0s - Duración de volver a intentar un bloqueo de estado.
- -no-color - Si se especifica, la salida no contendrá ningún color.
- -state=path - Ruta de acceso para leer y escribir el archivo de estado en. El valor predeterminado es "terraform.tfstate". Se ignora cuando se utiliza el estado remoto .
- -state-out=path - Ruta para escribir el archivo de estado actualizado. De forma -state , se -state ruta -state . Se ignora cuando se utiliza el estado remoto .
- -target=resource - Una -target=resource al destino. La operación se limitará a este recurso ya sus dependencias. Este indicador se puede utilizar varias veces.

- -var 'foo=bar' - Establece una variable en la configuración de Terraform. Este indicador se puede establecer varias veces. Los valores de las variables se interpretan como HCL , por lo que los valores de lista y mapa se pueden especificar mediante este indicador.
- -var-file=foo - Establece variables en la configuración de Terraform desde un archivo de variable . Si "terraform.tfvars" está presente, se cargará automáticamente primero. Cualquier archivo especificado por -var-file valor en un "terraform.tfvars". Este indicador se puede utilizar varias veces.

Comando show

El comando terraform show se utiliza para proporcionar una salida legible para humanos desde un archivo de estado o de plan.

terraform show [options] [path]

Opciones:

- -module-depth=n - Especifica la profundidad de los módulos a mostrar en la salida. De forma predeterminada, esto es -1, lo que ampliará todo.
- -no-color - Desactiva la salida con color.

Comando state

El comando terraform state se utiliza para la administración avanzada del estado. A medida que su uso se vuelve más avanzado, hay casos en los que puede ser necesario modificar el estado de Terraform. En lugar de modificar el estado directamente, los comandos terraform state se pueden utilizar en muchos casos en su lugar.

Todos los terraform state que modifican el estado escriben archivos de copia de seguridad. La ruta de acceso de estos archivos de copia de seguridad se puede controlar con -backup .

Subcomandos que son de sólo lectura no escriben ningún archivo de copia de seguridad ya que no están modificando el estado.

Tenga en cuenta que las copias de seguridad para la modificación de estado no se pueden desactivar. Debido a la sensibilidad del archivo de estado, Terraform obliga a cada comando de modificación de estado a escribir un archivo de copia de seguridad. Tendrá que quitar estos archivos manualmente si no desea mantenerlos alrededor.

terraform state [options] [args]

Subcomandos:

- list: Enumera los recursos dentro de un estado de Terraform.
- mv: Mueve elementos en un estado de Terraform .
- pull: El comando terraform state pull se utiliza para descargar y enviar manualmente el estado desde el estado remoto.
- push: Carga manualmente un archivo de estado local en estado remoto.
- rm: Elimina elementos del estado Terraform.
- show: Muestra los atributos de un solo recurso en el estado Terraform.

Comando taint

El comando terraform taint marca manualmente un recurso gestionado por Terraform como corrupto, forzándolo a ser destruido y recreado en la siguiente aplicación.

Este comando no modificará la infraestructura, sino que modificará el archivo de estado para marcar un recurso como corrupto. Una vez que un recurso está marcado como corrupto, el siguiente plan mostrará que el recurso será destruido y recreado y el siguiente implementará este cambio.

```
terraform taint [options] name
```

El argumento name es el nombre del recurso para marcar como corrupto. El formato de este TYPE.NAME es TYPE.NAME , como aws_instance.foo

Opciones:

- -allow-missing - Si se especifica, el comando tendrá éxito (código de salida 0) aunque falte el recurso.
- -backup=path - Ruta de acceso al archivo de copia de -backup=path . El valor predeterminado es -state-out con la extensión ".backup".
- -lock=true - Bloquea el archivo de estado cuando se admite el bloqueo.
- -lock-timeout=0s - Duración para volver a intentar un bloqueo de estado.
- -module=path - La ruta del módulo donde existe el recurso corrupto. Por defecto es la ruta raíz.
- -no-color - Desactiva la salida con color
- -state=path - Ruta de acceso para leer y escribir el archivo de estado. El valor predeterminado es "terraform.tfstate".
- -state-out=path - Ruta para escribir el archivo de estado actualizado. De forma -state , se -state ruta -state .

Comando validate

El comando terraform validate se utiliza para validar la sintaxis de los ficheros con extensión ".tf" de terraform. Realiza una comprobación de sintaxis en todos los archivos del directorio y mostrará un error si alguno de los archivos no se valida.

```
terraform validate [dir]
```

Comando untaint

El comando terraform untaint desmarca manualmente un recurso gestionado por Terraform como corrupto, restaurándolo como la instancia primaria en el estado.

```
terraform untaint [options] name
```

Opciones:

- -allow-missing - Si se especifica, el comando tendrá éxito (código de salida 0) aunque falte el recurso.
- -backup=path - Ruta de acceso al archivo de copia de -backup=path . El valor predeterminado es -state-out con la extensión ".backup".
- -index=n - Selecciona una única instancia corrupta cuando hay más de una instancia corrompida presente en el estado de un recurso dado. Este indicador es necesario cuando hay varias instancias infectadas.

- -lock=true - Bloquea el archivo de estado cuando se admite el bloqueo.
- -lock-timeout=0s - Duración para volver a intentar un bloqueo de estado.
- -module=path - La ruta del módulo donde existe el recurso corrupto. Por defecto es la ruta raíz.
- -no-color - Desactiva la salida con color
- -state=path - Ruta de acceso para leer y escribir el archivo de estado. El valor predeterminado es "terraform.tfstate".
- -state-out=path - Ruta para escribir el archivo de estado actualizado. De forma -state , se -state ruta -state .

CAPÍTULO 6

Ficheros de Configuración

El lenguaje de los ficheros de configuración de Terraform se llama HashiCorp Configuration Language (HCL). Los ficheros se deberán crear con la extensión ".tf".

Ejemplo de fichero Terraform, en el cual nos conectamos a la base de datos Mysql y creamos una base de datos llamada terraformbbdd.

```
# nano /home/usuario/terraform_example/mysql_create.tf

# Configuracion the MySQL server
provider "mysql" {
  endpoint = "localhost:3306"
  username = "root"
  password = "root"
}

# Crear base de datos
resource "mysql_database" "app" {
  name = "terraformbbdd"
}
```

Opciones básicas:

- Los comentarios de una sola línea comienzan con #
- Los comentarios de varias líneas se envuelven con /* y */
- Los valores se asignan con la sintaxis key = value (el espacio en blanco no importa). El valor puede ser cadena, número, booleano, una lista o un diccionario.
- Las cadenas están en comillas dobles.
- Las cadenas pueden tomar otros valores usando la sintaxis envuelta en ${} , como ${var.foo}.

- Las cadenas de multilínea pueden usar la sintaxis de “aquí doc”, con la cadena comenzando con un marcador como <<EOF y luego la cadena que termina con EOF en una línea propia. Las líneas de la cadena y el marcador final no deben estar sangrados.
- Se supone que los números son la base 10. Si se prefiere un número con 0x , se trata como un número hexadecimal.
- Valores booleanos: true , false .
- Las listas de tipos primitivos se pueden hacer con corchetes ([]). Ejemplo: [“foo”, “bar”, “baz”] .
- Los diccionarios se pueden hacer con llaves ({}) y dos puntos (:): { “foo”: “bar”, “bar”: “baz” } . Las citas pueden omitirse en las teclas, a menos que la clave comience con un número, en cuyo caso se requieren comillas. Se requieren comas entre pares clave / valor para diccionarios de una sola línea.

Terraform también admite los archivos de configuración con formato json. Aquí tenemos el mismo ejemplo que utilizamos arriba. Los ficheros Terraform con este formato se crearan con la extensión “.tf.json”.

```
# nano /home/usuario/terraform_example/mysql_create.tf.json

{

  "provider" :   {
    "mysql" :   {
      "endpoint" : "localhost:3306" ,
      "username" : "root" ,
      "password" : "root"
  }
 },

   "resource" :   {
     "mysql_database" :   {
       "app" :  {
         "name" : "terraformbbdd"
    }
   }
  }
}
```

CAPÍTULO 7

Proveedores de Terraform

Los proveedores de Terraform se utilizan para crear, administrar y manipular recursos de la infraestructura. Los recursos de los proveedores son utilizados en máquinas físicas, máquinas virtuales, conmutadores de red, contenedores, etc.

La lista completa de proveedores es la siguiente:

Aicloud	Archive	Arukas	AWS	Bitbucket	CenturyLinkCloud
Chef	Circonus	Cloudflare	Cobbler	CloudStack	Consul
Datadog	DNS	DigitalOcean	DNSMadeEasy	DNSimple	Docker
Dyn	Exteral	Fastly	Github	Google Cloud	Grafana
Heroku	HTTP	Icinga2	Ignition	influxDB	Kubernetes
Librato	Local	Logentries	Mailgun	New Relic	Nomad
NS1	Microsoft Azure	MySQL	Microsoft Azure(LegacyASM)	1&1	Oracle Public Cloud
Openstack	OpsGenie	Packet	PagerDuty	PostgresSQL	PowerDNS
ProfitBrcks	RabbitMQ	Rancher	Random	Rundeck	Scaleway
Softlayer	StatusCake	Spotinst	Template	Terraform	Terraform Enterprise
TLS	Triton	UltraDNS	Vault	VMwareCloud	VMware vSphere

CAPÍTULO 8

Comenzando con Terraform

Creando nuestro primer fichero de configuración en el cual vamos a crear una base de datos y un usuario en mysql, en una maquina virtual con sistema operativo debian 8 jessie.

Creando un fichero Terraform

Con el un editor de texto creamos el fichero de configuración.

```
# nano mysql_crearbd_crearuser.tf

# Configuracion the MySQL server
provider "mysql" {
  endpoint = "localhost:3306"
  username = "root"
  password = "root"
}

# Crear base de datos
resource "mysql_database" "proyecto" {
  name = "proyecto"
}

# Crear usuario
resource "mysql_user" "josemaria" {
  user     = "josemaria"
  host     = "localhost"
  password = "josemaria"
}
```

Creando un plan de ejecución

Ahora procedemos a ejecutar el comando terraform apply para crear nuestra base de datos y el usuario en en el servidor mysql.

```
root@debian:/home/usuario/terraform_example# terraform apply
mysql_database.proyecto: Creating...
  default_character_set: "" => "utf8"
  default_collation:     "" => "utf8_general_ci"
  name:                  "" => "proyecto"
mysql_user.josemaria: Creating...
  host:     "" => "localhost"
  password: "<sensitive>" => "<sensitive>"
  user:     "" => "josemaria"
mysql_user.josemaria: Creation complete (ID: josemaria@localhost)
mysql_database.proyecto: Creation complete (ID: proyecto)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:
```

Ahora entramos en la consola de mysql y comprobamos que se han creado la base de datos y el usuario.

```
root@debian:/home/usuario/terraform_example# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 39
Server version: 5.5.55-0+deb8u1 (Debian)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| proyecto           |
| terraformbbdd      |
+--------------------+
5 rows in set (0.03 sec)
```

```
mysql> select user from mysql.user;
+------------------+
| user             |
+------------------+
| root             |
| root             |
| root             |
| debian-sys-maint |
| josemaria        |
| root             |
+------------------+
6 rows in set (0.00 sec)
```

Actualizando el plan de ejecución

Modificando el plan de ejecución, en nuestro caso cambiaremos el estado del fichero creado anteriormente, en el que modificamos el usuario y la clave del usuario mysql.

```
root@debian:/home/usuario/terraform_example# nano mysql_crearbd_crearuser.tf
root@debian:/home/usuario/terraform_example# terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

mysql_database.proyecto: Refreshing state... (ID: proyecto)
mysql_user.josemaria: Refreshing state... (ID: josemaria@localhost)
The Terraform execution plan has been generated and is shown below.
Resources are shown in alphabetical order for quick scanning. Green resources
will be created (or destroyed and then created if an existing resource
exists), yellow resources are being changed in-place, and red resources
will be destroyed. Cyan entries are data sources to be read.

Note: You didn't specify an "-out" parameter to save this plan, so when
"apply" is called, Terraform can't guarantee this is what will execute.

-/+ mysql_user.josemaria
    host:     "localhost" => "localhost"
    password: "<sensitive>" => "<sensitive>" (attribute changed)
    user:     "josemaria" => "jose" (forces new resource)


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

En la imagen comprobamos que al ejecutar terraform plan detecta el cambio en el estado del fichero y nos informa que para llevar acabo el cambio tenemos que ejecutar terraform apply.

```
root@debian:/home/usuario/terraform_example# terraform apply
mysql_user.josemaria: Refreshing state... (ID: josemaria@localhost)
mysql_database.proyecto: Refreshing state... (ID: proyecto)
mysql_user.josemaria: Destroying... (ID: josemaria@localhost)
mysql_user.josemaria: Destruction complete
mysql_user.josemaria: Creating...
  host:     "" => "localhost"
  password: "<sensitive>" => "<sensitive>"
  user:     "" => "jose"
mysql_user.josemaria: Creation complete (ID: jose@localhost)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:
```

Volvemos abrir la consola de mysql y consultamos los usuarios de la base de datos de mysql. Como comprobamos se ha eliminado el usuario creado en el primer plan de ejecución y se ha generado el nuevo usuario definido en el fichero de configuración de terraform.

```
root@debian:/home/usuario/terraform_example# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 48
Server version: 5.5.55-0+deb8u1 (Debian)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select user from mysql.user;
+------------------+
| user             |
+------------------+
| root             |
| root             |
| root             |
| debian-sys-maint |
| jose             |
| root             |
+------------------+
6 rows in set (0.00 sec)
```

Representando el plan de ejecución gráficamente

Representando nuestra infraestructura de terraform, representaremos la infraestructura creada anteriormente. En nuestro caso generamos una imagen con terraform graph y GraphViz.

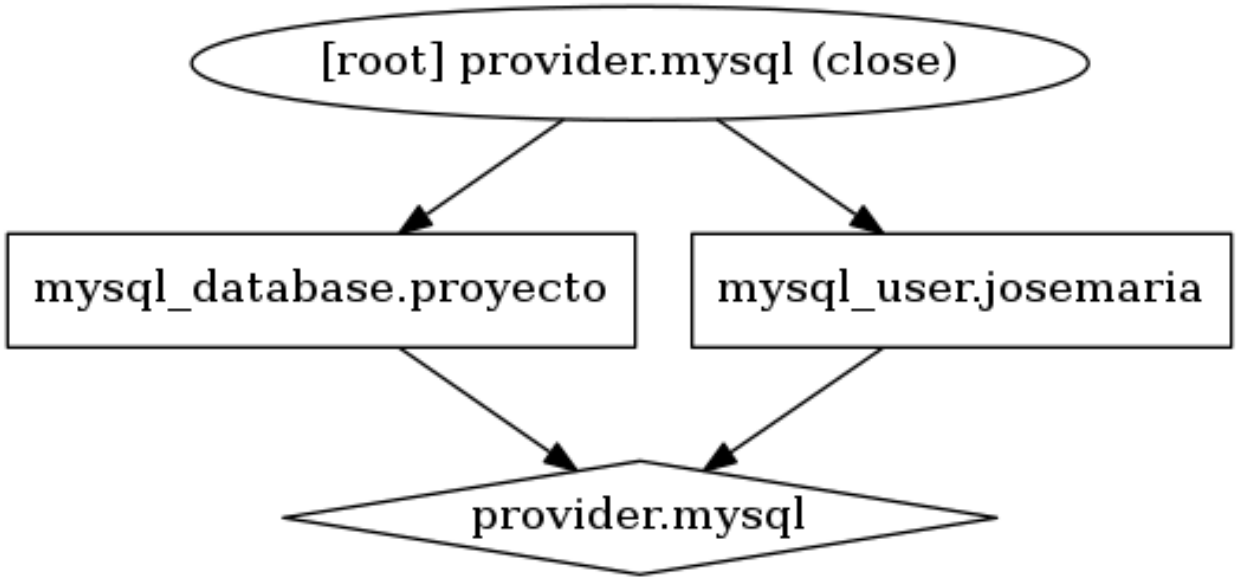
Lo primero instalamos el paquete gráfico GraphViz:

```
# apt install GraphViz
```

Una vez instalado el paquete ejecutamos el siguiente comando para generar una imagen en formato png.

```
# terraform graph | dot -Tpng > mysql.png
```

Nos dirigimos al directorio donde se ha generado la imagen y la abrimos con el visor de imágenes.



Para visualizar el plan de ejecución ejecutaremos el comando terraform show.

```
root@debian:/home/usuario/terraform_example# terraform show
mysql_database.proyecto:
  id = proyecto
  default_character_set = utf8
  default_collation = utf8_general_ci
  name = proyecto
mysql_user.josemaria:
  id = jose@localhost
  host = localhost
  password = jose
  user = jose
```

CAPÍTULO 9

Administración de openstack con Terraform

OpenStack es un proyecto de computación en la nube para proporcionar una infraestructura como servicio. Este proveedor se utiliza para interactuar con los muchos recursos soportados por OpenStack.

Conexión a Openstack

Creamos un directorio donde generaremos un entorno virtual.

```
root@castillo:/home/jose# mkdir openstack
```

Instalamos el paquete virtualenv del repositorio.

```
root@castillo:/home/jose# apt-get install python-virtualenv
```

Nos situamos en la carpeta openstack y ejecutamos la siguiente sentencia.

```
root@castillo:/home/jose# cd openstack/
root@castillo:/home/jose/openstack# virtualenv mi_proyecto
```

Ahora nos movemos al directorio mi_proyecto y activamos el entorno virtual

```
root@castillo:/home/jose/openstack# cd mi_proyecto/
root@castillo:/home/jose/openstack/mi_proyecto# source bin/activate
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto#
```

Instalamos el cliente openstack en el entorno virtual.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto# pip install python-openstackclient
```

Exportamos la variable de entorno de terraform.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto# export PATH=/home/jose/terraform:$PATH
```

Ahora ya es posible establecer una conexión a openstack mediante un fichero terraform como el siguiente:

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}
```

Comenzamos el fichero especificando las credenciales de conexión de nuestro usuario en openstack.

- user_name: Nombre de usuario.
- tenant_name: Nombre de proyecto.
- domain_name: Nombre de dominio.
- password: Contraseña de usuario.
- auth_url: Dirección de autenticación.

Otros parámetros que se pueden especificar en la conexión:

- user_id: Id del usuario.
- tenant_id: Id del proyecto.
- token: En el caso de no especificar ningún usuario se genera un medio de acceso temporal.
- domain_id: Id del dominio.
- insecure: Acepta la conexión a servidores con certificados menos seguros.
- cacert_file: Certificado CA del servidor.
- cert : Certificado del cliente.
- key : Clave privada del cliente.

Subiendo par de claves

Creamos un fichero.tf el cual llamaremos keypair.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso “keypair”, un nombre y la clave publica.

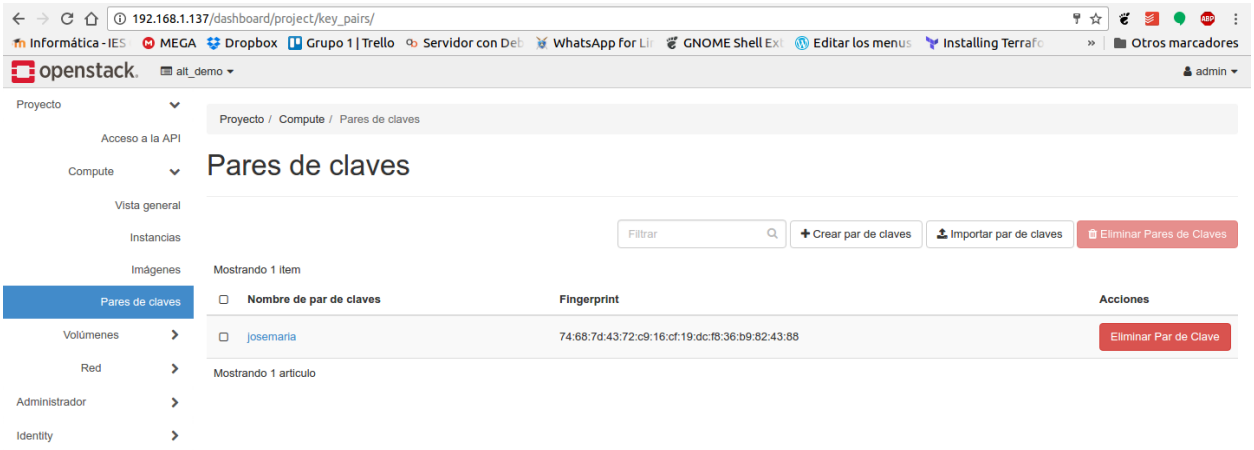
```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}

resource "openstack_compute_keypair_v2" "keypair" {
  name       = "josemaria"
```

```
  public_key = "ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQDC4uo4aXvL+8f+y8pidtU5XGrJx46WKaIt5OW09VjcwnVuRx2tRWuAYGyvVrNHp8o91rk
↪uXFnI+FLY4GFhWA4VFY0OT00vLxRR4SPMOFVFYNfSLIr6IsQyD8ghc++j3h/
↪qIZIKUGf8sR3LniyChDO6QQYkZw2iTJiTyAkFHH5/
↪8ol9598lJ8rMSV7Fyh34t0b08avwD4alCz4ij9JVvcsT0aRbeEtDE0HCBY88r5bEJlcKPaRL3Z5UizYU26n+ZfQRm4A3A9IdpI
↪jose@castillo"
}
```

```
root@castillo:/home/jose/openstack/mi_proyecto/keypair# te
openstack_compute_keypair_v2.keypair: Creating...
  name:       "" => "josemaria"
  public_key: "" => "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABA
46WKaIt5OW09VjcwnVuRx2tRWuAYGyvVrNHp8o91rklUUaKDcnSD+op4kE
XFnI+FLY4GFhWA4VFY0OT00vLxRR4SPMOFVFYNfSLIr6IsQyD8ghc++j3h
2iTJiTyAkFHH5/8ol9598lJ8rMSV7Fyh34t0b08avwD4alCz4ij9JVvcsT
3Z5UizYU26n+ZfQRm4A3A9IdpI0BZwXLlQV5kwvQp7AXkUnQ2qR2+91u7
openstack_compute_keypair_v2.keypair: Creation complete (I

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

The state of your infrastructure has been saved to the pat
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete s
use the `terraform show` command.

State path:
```

Ahora ejecutamos el plan para subir la clave a openstack.

Accedemos a openstack para comprobar que nuestra clave publica se encuentra disponible.



Creando un volumen

Creamos un fichero.tf el cual llamaremos volumen.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso “vol”, un nombre, descripción, tamaño en gigabytes, tipo de volumen y zona .

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
```

```
  auth_url     = "http://192.168.1.137/identity/v3"
}


resource "openstack_blockstorage_volume_v2" "vol" {
  name         = "vol_1"
  description = "volumen 1Gb"
  size         = 1
  volume_type = "lvmdriver-1"
  availability_zone = "nova"
}
```

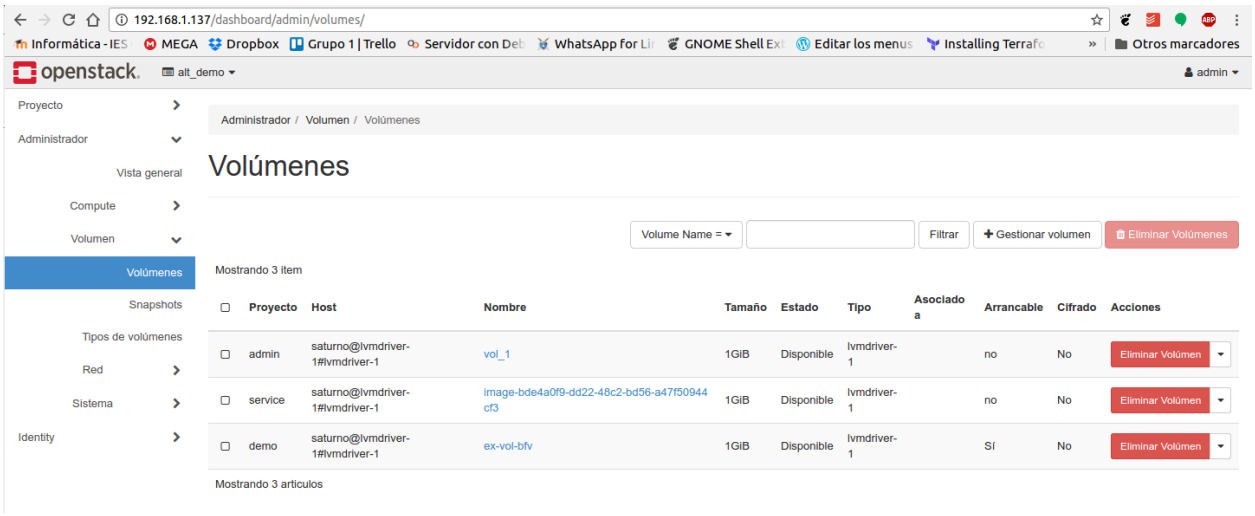
Otros parámetros que se pueden especificar en la creación del volumen:

- region: Región en la que se crea el volumen.
- consistency_group_id: Id del grupo consistencia.
- image_id:Id de la imagen desde la cual creamos el volumen.
- metadatos: Metadatos clave / valor pares para asociar con el volumen.
- snapshot_id: Id de instantánea desde la cual creamos el volumen.
- source_replica: Id de volumen con el que se replicará.
- source_vol_id: Id de volumen desde el cual creamos el volumen.

Ejecutamos el plan para crear el volumen en openstack.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto# terraform apply volumen/
openstack_blockstorage_volume_v2.vol1: Refreshing state... (ID: 50452fbf-...68c9ea6b)
openstack_blockstorage_volume_v2.vol1: Destroying... (ID: 50452fbf-...68c9ea6b)
openstack_blockstorage_volume_v2.vol: Creating...
  attachment.#:       "" => "<computed>"
  availability_zone: "" => "nova"
  description:        "" => "volumen 1Gb"
  metadata.%:         "" => "<computed>"
  name:               "" => "vol_1"
  size:               "" => "1"
  volume_type:        "" => "lvmdriver-1"
openstack_blockstorage_volume_v2.vol1: Still destroying... (ID: 50452fbf-...68c9ea6b, 10s elapsed)
openstack_blockstorage_volume_v2.vol: Still creating... (10s elapsed)
openstack_blockstorage_volume_v2.vol1: Destruction complete
openstack_blockstorage_volume_v2.vol: Creation complete (ID: ea00b0ed-...55086978)

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

Accedemos a openstack y comprobamos que el volumen se ha creado correctamente.



Subiendo imágenes

Creamos un fichero.tf el cual llamaremos imagen.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso imagen, un nombre, la ruta donde esta la imagen, formato de contenedor, y tipo de imagen.

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}

resource "openstack_images_image_v2" "imagen" {
  name   = "Debian 8"
  local_file_path = "/home/jose/Descargas/debian-8-openstack-amd64.qcow2"
  container_format = "bare"
  disk_format = "qcow2"
}
```

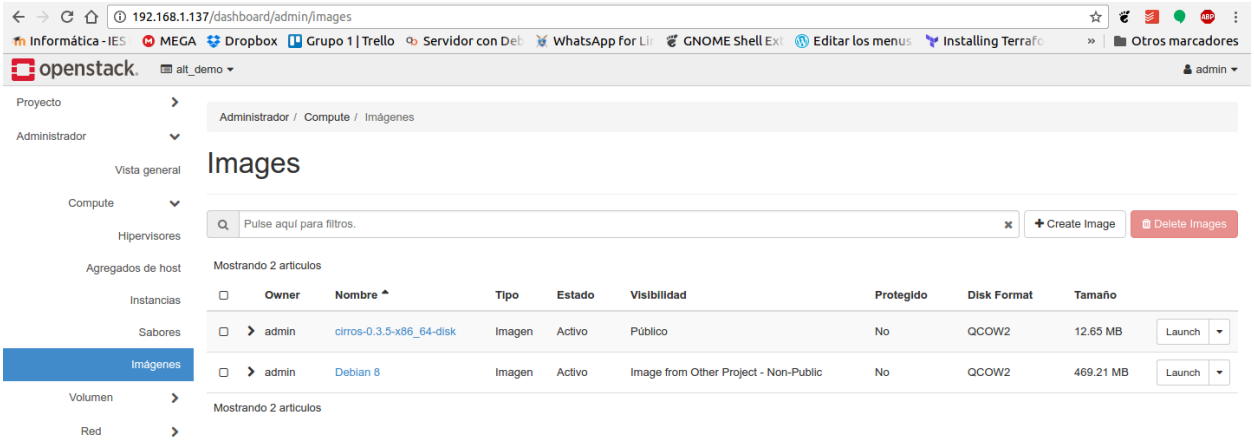
Otros parámetros que se pueden especificar en la subida de la imagen:

- container_format: Formato del contenedor (“ami”, “ari”, “aki”, “bare”, “ovf”).
- disk_format: Formato de la imagen (“ami”, “ari”, “aki”, “vhd”, “vmdk”, “raw”, “qcow2”, “vdi”, “iso”).
- image_cache_path: Es el directorio donde se descargarán las imágenes. El valor predeterminado es “$ HOME / .terraform / image_cache”
- image_source_url: Url de descarga de la imagen.
- min_disk_gb: Espacio de disco en GB necesario para iniciar la imagen. El valor predeterminado es 0.
- min_ram_mb: Memoria Ram en MB necesario para iniciar la imagen. El valor predeterminado es 0.
- protected: Especifica con valor true/false si la imagen puede ser eliminada.
- tags: Etiqueta de la imagen.
- Visibility: Visibilidad de la imagen.visibility (“public”, “private”,”community”, “shared”).

Ejecutamos el plan para subir nuestra imagen a openstack.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/imagen# terraform apply ../imagen/
openstack_images_image_v2.imagen: Refreshing state... (ID: 222a9754-...e07eb1db)
openstack_images_image_v2.imagen: Destroying... (ID: 222a9754-...e07eb1db)
openstack_images_image_v2.imagen: Destruction complete
openstack_images_image_v2.imagen: Creating...
  checksum:         "" => "<computed>"
  container_format: "" => "bare"
  created_at:       "" => "<computed>"
  disk_format:      "" => "qcow2"
  file:             "" => "<computed>"
  image_cache_path: "" => "/root/.terraform/image_cache"
  local_file_path:  "" => "/home/jose/Descargas/debian-8-openstack-amd64.qcow2"
  metadata.%:       "" => "<computed>"
  min_disk_gb:      "" => "0"
  min_ram_mb:       "" => "0"
  name:             "" => "Debian 8"
  owner:            "" => "<computed>"
  protected:        "" => "false"
  schema:           "" => "<computed>"
  size_bytes:       "" => "<computed>"
  status:           "" => "<computed>"
  update_at:        "" => "<computed>"
  visibility:       "" => "private"
openstack_images_image_v2.imagen: Still creating... (10s elapsed)
openstack_images_image_v2.imagen: Still creating... (20s elapsed)
openstack_images_image_v2.imagen: Still creating... (30s elapsed)
openstack_images_image_v2.imagen: Creation complete (ID: bde3c862-...768e7d3a)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:
```

Comprobamos en openstack que se encuentra disponible la imagen subida.



Asignando ip flotantes al proyecto

Creamos un fichero.tf el cual llamaremos imagen.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso ip, especificamos el pool.

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
```

```
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}

resource "openstack_compute_floatingip_v2" "ip" {
  pool = "public"
}
```

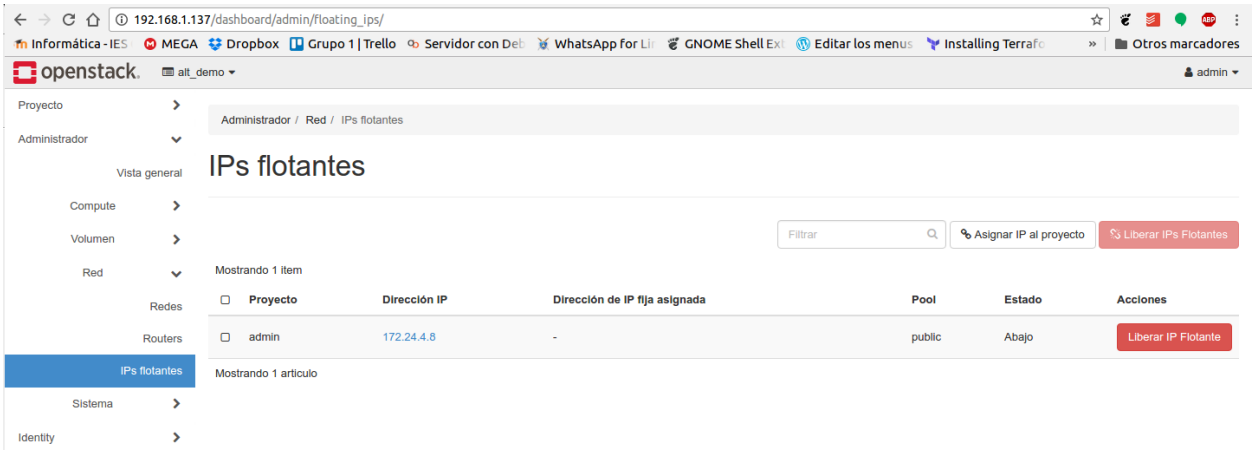
Otros parámetros que se pueden especificar asignando ip flotantes:

- region: Región en la que se crea el volumen.
- pool: El nombre desde el cual obtener la IP flotante.

Ejecutamos el plan para reservar una ip flotante al proyecto.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/ipflotante# nano ipflotante.tf
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/ipflotante# terraform apply ../ipflotante/
openstack_compute_floatingip_v2.ip: Creating...
  address:     "" => "<computed>"
  fixed_ip:    "" => "<computed>"
  instance_id: "" => "<computed>"
  pool:        "" => "public"
openstack_compute_floatingip_v2.ip: Creation complete (ID: 0a58d66f-...8900fced)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.
```

Accedemos a openstack para comprobar que se ha reservado una ip flotante en el proyecto admin.



Creando un nuevo grupo de seguridad

Creamos un fichero.tf el cual llamaremos seguridad.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso gruposeguridad, especificamos el nombre del grupo de seguridad y la descripción.

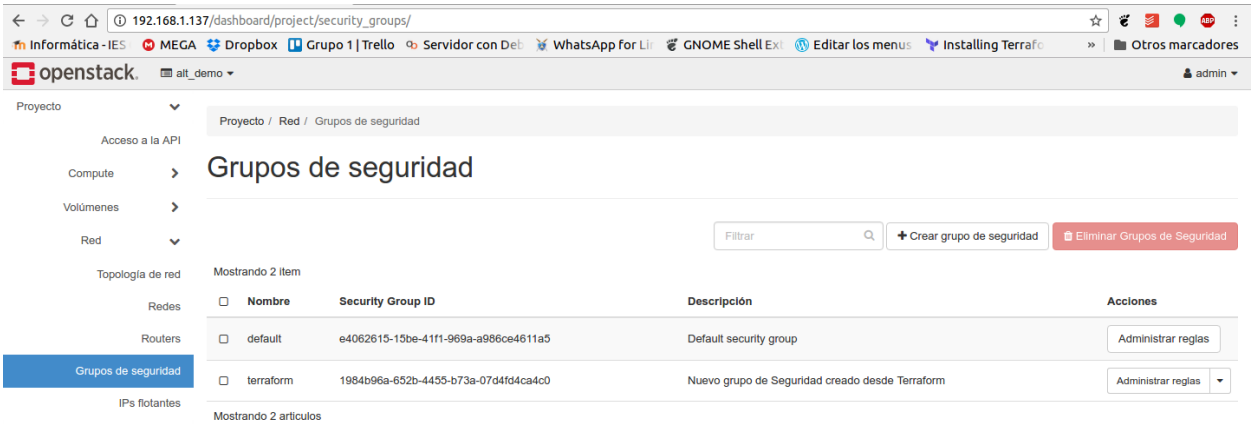
```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
```

```
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}

resource "openstack_networking_secgroup_v2" "gruposeguridad" {
  name        = "terraform"
  description = "Nuevo grupo de Seguridad creado desde Terraform"
  tenant_id   = "039182bb2d1c4c4cb806e380c9e2413c"
}
```

Ejecutamos el plan para crear el nuevo grupo de seguridad llamado terraform.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/seguridad# nano seguridad.tf
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/seguridad# terraform apply ../seguridad/
openstack_networking_secgroup_v2.gruposeguridad: Creating...
  description: "" => "Nuevo grupo de Seguridad creado desde Terraform"
  name:        "" => "terraform"
  tenant_id:   "" => "039182bb2d1c4c4cb806e380c9e2413c"
openstack_networking_secgroup_v2.gruposeguridad: Creation complete (ID: 1984b96a-...fd4ca4c0)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/seguridad#
```

Ahora accedemos a openstack y comprobamos que se ha generado e nuevo grupo de seguridad.



Agregando regla de seguridad

Creamos un fichero.tf el cual llamaremos regla.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso reglassh, especificamos el id del grupo de seguridad, la dirección de la regla si es de entrada o salida, tipo de red IPv4 o IPv6, protocolo de red y rango de puertos.

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}
```

```
resource "openstack_networking_secgroup_rule_v2" "reglassh" {
  direction         = "ingress"
  ethertype         = "IPv4"
  protocol          = "tcp"
  port_range_min    = 22
  port_range_max    = 22
  remote_ip_prefix  = "0.0.0.0/0"
  security_group_id = "aeebe670-38b5-4fff-a9c4-95f18510a9f3"
}
```

Ejecutamos el plan para añadir la nueva regla al grupo de seguridad terraform.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/reglas# nano regla.tf
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/reglas# terraform apply ../reglas/
openstack_networking_secgroup_rule_v2.reglassh: Creating...
  direction:         "" => "ingress"
  ethertype:         "" => "IPv4"
  port_range_max:    "" => "22"
  port_range_min:    "" => "22"
  protocol:          "" => "tcp"
  remote_group_id:   "" => "<computed>"
  remote_ip_prefix:  "" => "0.0.0.0/0"
  security_group_id: "" => "aeebe670-38b5-4fff-a9c4-95f18510a9f3"
  tenant_id:         "" => "<computed>"
openstack_networking_secgroup_rule_v2.reglassh: Creation complete (ID: 90c04e06-...4ad0f1b5)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/reglas#
```

Comprobamos que se ha añadido la regla ssh en el grupo de seguridad de terraform en openstack.

Creando una red

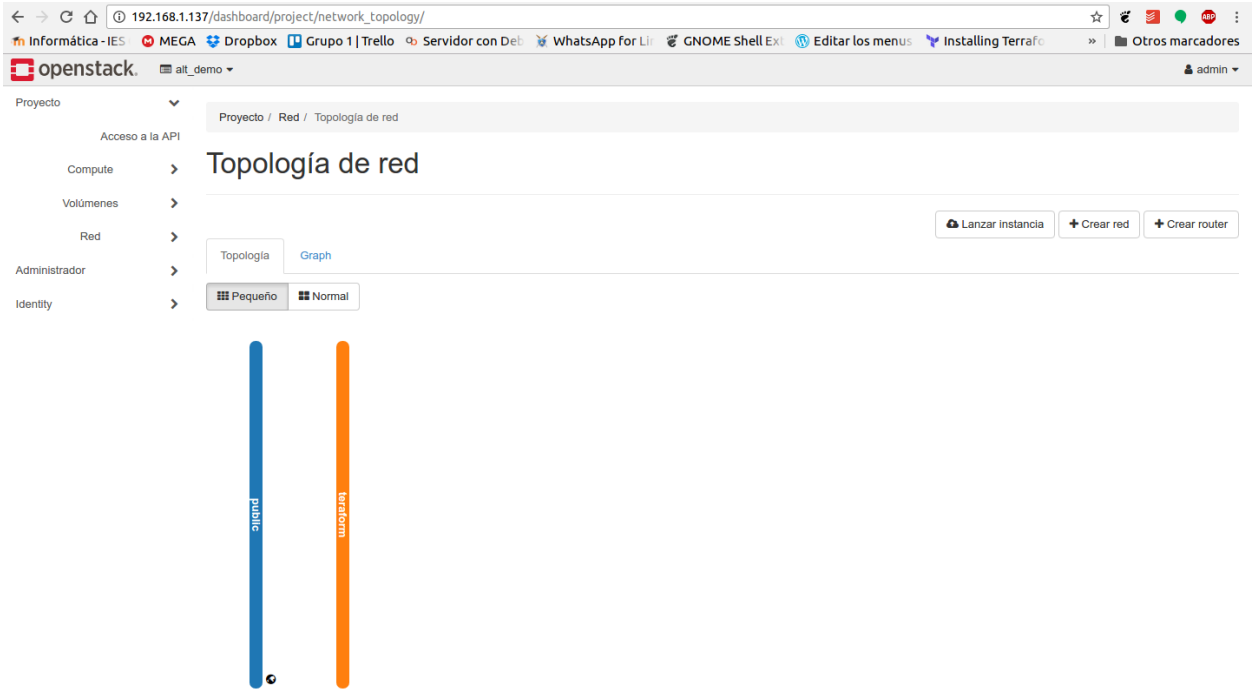
Creamos un fichero.tf el cual llamaremos red.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso red, especificamos un nombre, el Id del proyecto y el estado administración con valor true o false.

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}
resource "openstack_networking_network_v2" "red" {
  name           = "teraform"
  admin_state_up = "true"
  tenant_id      = "039182bb2d1c4c4cb806e380c9e2413c"
}
```

Ejecutamos el plan para crear la red terraform.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/redes# nano red.tf
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/redes# terraform apply ../redes/
openstack_networking_network_v2.red: Creating...
  admin_state_up: "" => "true"
  name:           "" => "teraform"
  shared:         "" => "<computed>"
  tenant_id:      "" => "039182bb2d1c4c4cb806e380c9e2413c"
openstack_networking_network_v2.red: Creation complete (ID: 5a2ea55d-...ccefb9be)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/redes#
```

Ahora desde openstack accedemos a la topología de red para comprobar que se creado la nueva red.



Creando subred

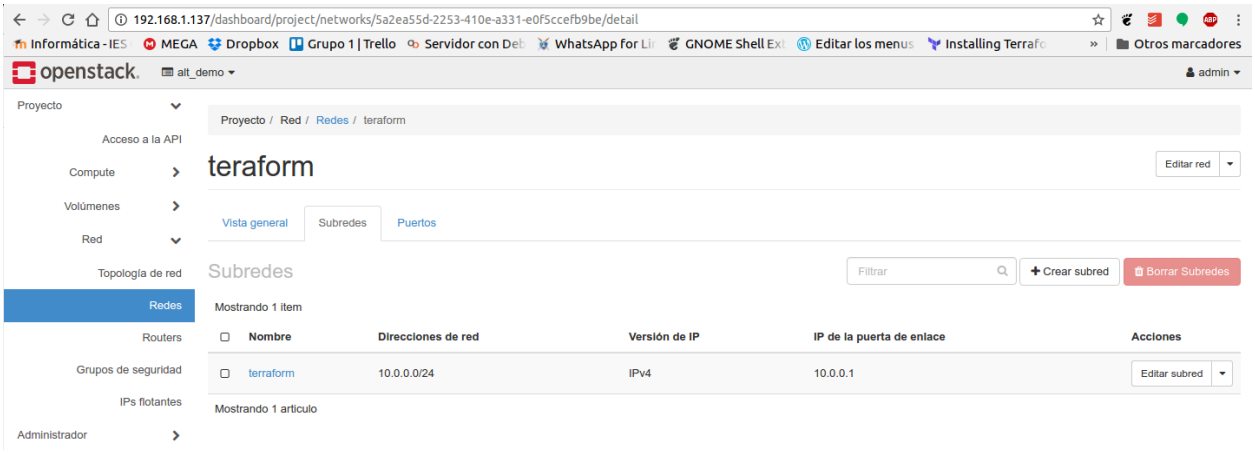
Creamos un fichero.tf el cual llamaremos red.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso subred, especificamos Id de la red, el rango de ip, el nombre y el Id del proyecto.

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}


resource "openstack_networking_subnet_v2" "subred" {
  network_id = "5a2ea55d-2253-410e-a331-e0f5ccefb9be"
  cidr       = "10.0.0.0/24"
  name       = "terraform"
  tenant_id  = "039182bb2d1c4c4cb806e380c9e2413c"
}
```

Ejecutamos el plan para crear la subred en openstack.

```
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/subredes# nano subred.tf
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/subredes# terraform apply ../subredes/
openstack_networking_subnet_v2.subred: Creating...
  allocation_pools.#: "" => "<computed>"
  cidr:               "" => "10.0.0.0/24"
  enable_dhcp:        "" => "true"
  gateway_ip:         "" => "<computed>"
  ip_version:         "" => "4"
  name:               "" => "terraform"
  network_id:         "" => "5a2ea55d-2253-410e-a331-e0f5ccefb9be"
  tenant_id:          "" => "039182bb2d1c4c4cb806e380c9e2413c"
openstack_networking_subnet_v2.subred: Creation complete (ID: d3d77b5a-...68470e9a)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:
(mi_proyecto) root@castillo:/home/jose/openstack/mi_proyecto/subredes#
```

Comprobamos desde el apartado redes e openstack que se ha creado la subred.



Creando un router en la red pública

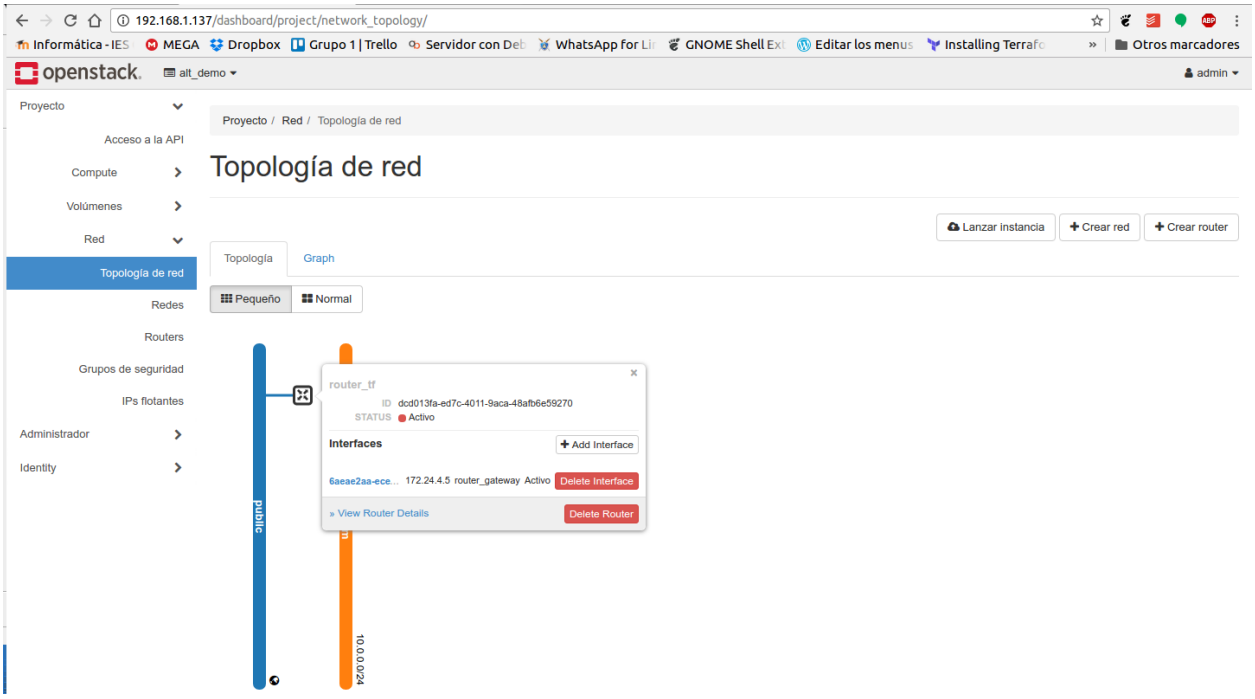
Creamos un fichero.tf el cual llamaremos router.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso router_tf, especificamos un nombre, el Id del proyecto y el Id de la red pública.

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}

resource "openstack_networking_router_v2" "router_tf" {
  name             = "router_tf"
  external_gateway = "c000a18b-689a-48ad-be82-c3671f3bb00c"
  tenant_id        = "039182bb2d1c4c4cb806e380c9e2413c"
}
```

Ejecutamos el plan para crear el router conectado a la red publica de openstack.

```
root@castillo:/home/jose/openstack/mi_proyecto/router# nano router.tf
root@castillo:/home/jose/openstack/mi_proyecto/router# terraform apply ../router
/
openstack_networking_router_v2.router_tf: Creating...
  admin_state_up:    "" => "<computed>"
  distributed:       "" => "<computed>"
  external_gateway:  "" => "c000a18b-689a-48ad-be82-c3671f3bb00c"
  name:              "" => "router_tf"
  tenant_id:         "" => "039182bb2d1c4c4cb806e380c9e2413c"
openstack_networking_router_v2.router_tf: Creation complete (ID: dcd013fa-...b6e
59270)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:
root@castillo:/home/jose/openstack/mi_proyecto/router#
```

Comprobamos en openstack en el apartado topología de red que el router se ha generado.



Creando interfaz de red en un router

Creamos un fichero.tf el cual llamaremos interfaz.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso interfaz, especificamos el Id del router y el Id de la subred a la cual vamos a conectar.

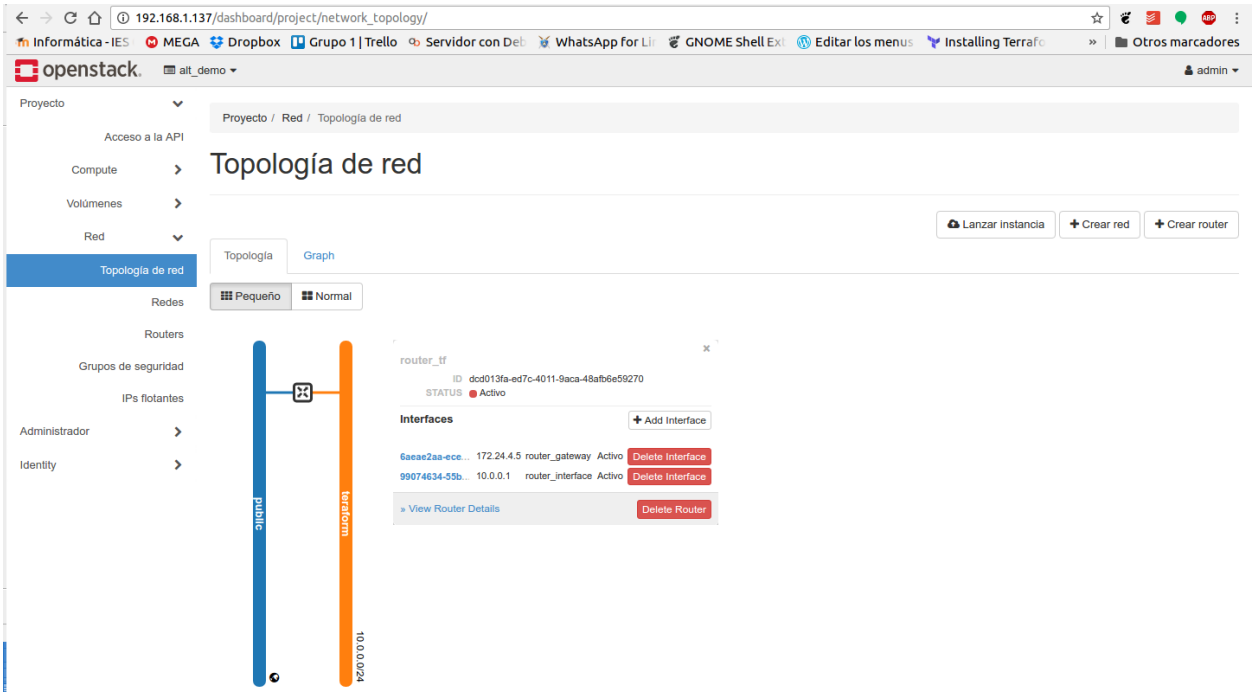
```
provider "openstack" {
  user_name   = "admin"
```

```
  tenant_name = "admin"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}

resource "openstack_networking_router_interface_v2" "interfaz" {
  router_id = "dcd013fa-ed7c-4011-9aca-48afb6e59270"
  subnet_id = "d3d77b5a-380d-4eab-a50e-423868470e9a"
}
```

Ejecutamos el plan para crear la interfaz en el router creado en el punto anterior en openstack.

```
root@castillo:/home/jose/openstack/mi_proyecto/interfaz# nano interfaz.tf
root@castillo:/home/jose/openstack/mi_proyecto/interfaz# terraform apply ../interfaz/
openstack_networking_router_interface_v2.interfaz: Creating...
  router_id: "" => "dcd013fa-ed7c-4011-9aca-48afb6e59270"
  subnet_id: "" => "d3d77b5a-380d-4eab-a50e-423868470e9a"
openstack_networking_router_interface_v2.interfaz: Creation complete (ID: 99074634-...
80ed1cf2)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.
```

Comprobamos en openstack en el apartado topología de red que el router ahora tiene una segunda interfaz conectada a la red terraform.



Creando instancia

Creamos un fichero.tf el cual llamaremos instancia.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso terraform, especificamos el nombre, Id de la imagen, Id del sabor, nombre par de claves, grupo de seguridad, red a la que va a pertenecer y ip flotante.

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "alt_demo"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}


resource "openstack_compute_instance_v2" "terraform" {
  name            = "terraform"
  image_id        = "bde4a0f9-dd22-48c2-bd56-a47f50944cf3"
  flavor_id       = "c1"
  key_pair        = "josemaria"
  security_groups = ["terraform"]

  metadata {
    this = "terraform"
  }

  network {
    name = "teraform"
    floating_ip = "172.24.4.4"
  }
}
```

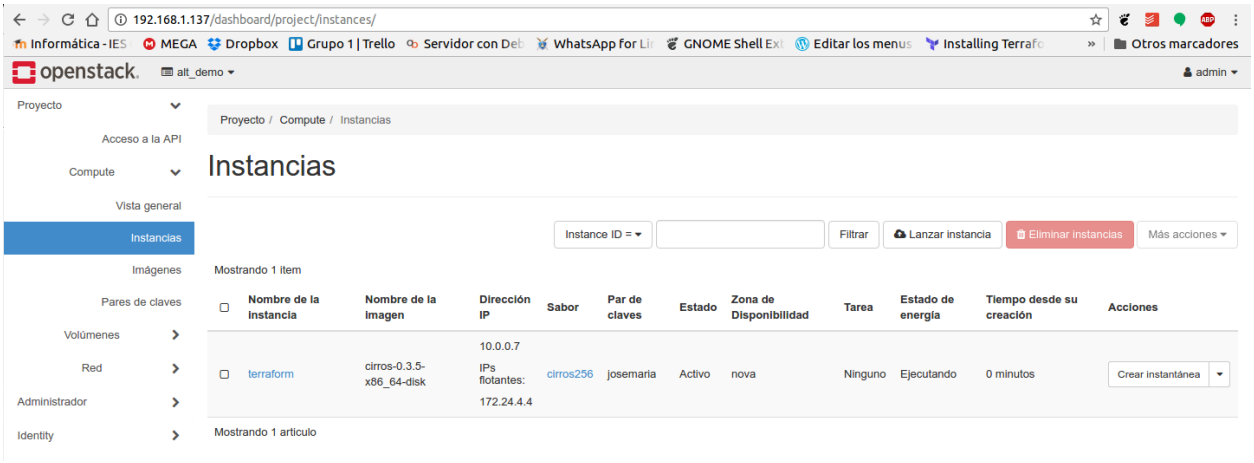
Otros parámetros que se pueden especificar creando instancias:

- region: Región en la que se va a crear la instancia del servidor.
- image_name: El nombre de la imagen deseada para el servidor.
- flavor_name: El nombre del sabor deseado para el servidor.
- user_data: Datos de usuario que se deben proporcionar al iniciar la instancia.
- availability_zone: Zona de disponibilidad en la que se crea el servidor.
- network: Una matriz de una o más redes para adjuntar a la instancia.
- config_drive: Si desea utilizar la función config_drive para configurar la instancia.
- admin_pass: La contraseña administrativa para asignar al servidor. Al cambiar esto, se cambia la contraseña de root en el servidor existente.
- block_device: Configuración de los dispositivos de bloque.
- volume: Asosia un volumen existente a la instancia.
- scheduler_hints: Proporciona al planificador de Nova con sugerencias el cómo se debe iniciar la instancia.
- personality: Personaliza la personality de una instancia definiendo uno o más archivos y su contenido.
- stop_before_destroy: Intenta detener la instancia antes de destruirla, dando así la oportunidad de que los demonios SO invitados se detengan correctamente.

Ejecutamos el plan de para crear la instancia en openstack.

```
root@castillo:/home/jose/openstack/mi_proyecto/instancia# nano instancia.tf
root@castillo:/home/jose/openstack/mi_proyecto/instancia# terraform apply ../instancia/
openstack_compute_instance_v2.terraform: Creating...
  access_ip_v4:                "" => "<computed>"
  access_ip_v6:                "" => "<computed>"
  all_metadata.%:              "" => "<computed>"
  availability_zone:           "" => "<computed>"
  flavor_id:                   "" => "c1"
  flavor_name:                 "" => "<computed>"
  force_delete:                "" => "false"
  image_id:                    "" => "bde4a0f9-dd22-48c2-bd56-a47f50944cf3"
  image_name:                  "" => "<computed>"
  key_pair:                    "" => "josemaria"
  metadata.%:                  "" => "1"
  metadata.this:               "" => "terraform"
  name:                        "" => "terraform"
  network.#:                   "" => "1"
  network.0.access_network:    "" => "false"
  network.0.fixed_ip_v4:       "" => "<computed>"
  network.0.fixed_ip_v6:       "" => "<computed>"
  network.0.floating_ip:       "" => "172.24.4.4"
  network.0.mac:               "" => "<computed>"
  network.0.name:              "" => "teraform"
  network.0.port:              "" => "<computed>"
  network.0.uuid:              "" => "<computed>"
  security_groups.#:           "" => "1"
  security_groups.969816526:   "" => "terraform"
  stop_before_destroy:         "" => "false"
openstack_compute_instance_v2.terraform: Still creating... (10s elapsed)
openstack_compute_instance_v2.terraform: Still creating... (20s elapsed)
openstack_compute_instance_v2.terraform: Creation complete (ID: 767ae92c-...d53ae893)

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

Comprobamos en openstack en el apartado Instancias que la maquina se ha creado.



Accedemos a la maquina para comprobar que se ha creado.

```
jose@castillo:~/openstack/mi_proyecto$ ssh cirros@172.24.4.4
The authenticity of host '172.24.4.4 (172.24.4.4)' can't be established.
RSA key fingerprint is SHA256:tRRrH0FOABApeIx06PLDckEKJoP504t+Ze8w1fwXNDs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.24.4.4' (RSA) to the list of known hosts.
$ hostname
terraform
$
```

Asociando un volumen a la máquina

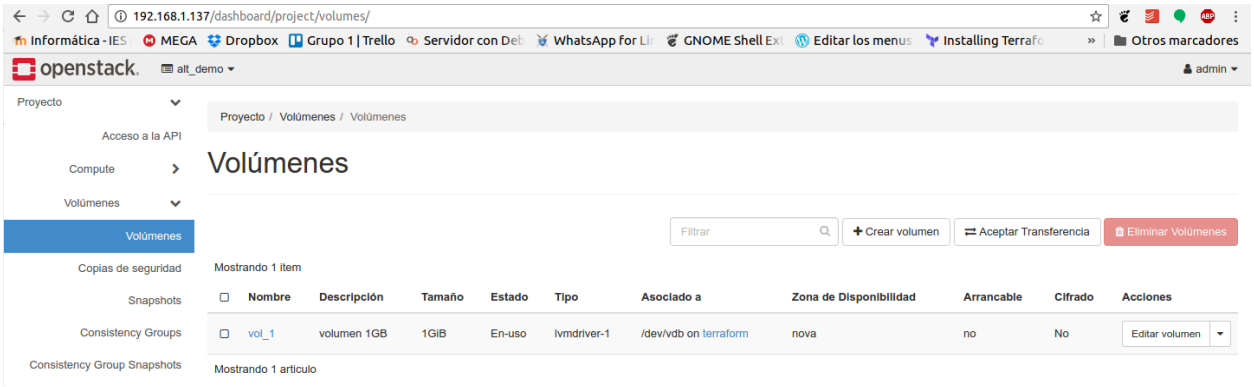
Creamos un fichero.tf el cual llamaremos attach.tf, en el especificamos la conexión con el servidor openstack, en el resource le indicamos el id del plan de ejecución, en nuestro caso attached, especificamos el Id de la instancia y el Id del volumen.

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "alt_demo"
  domain_name = "Default"
  password    = "admin"
  auth_url    = "http://192.168.1.137/identity/v3"
}


resource "openstack_compute_volume_attach_v2" "attached" {
  instance_id = "767ae92c-0602-4c89-bc24-76b4d53ae893"
  volume_id   = "49c97096-c3fe-412d-b598-24fbda878dd7"
}
```

Ejecutamos el plan de para asociar el volumen a la instancia creada en el apartado anterior.

```
root@castillo:/home/jose/openstack/mi_proyecto/attach# nano attach.tf
root@castillo:/home/jose/openstack/mi_proyecto/attach# terraform apply ../attach/
openstack_compute_volume_attach_v2.attached: Creating...
  device:      "" => "<computed>"
  instance_id: "" => "767ae92c-0602-4c89-bc24-76b4d53ae893"
  volume_id:   "" => "49c97096-c3fe-412d-b598-24fbda878dd7"
openstack_compute_volume_attach_v2.attached: Still creating... (10s elapsed)
openstack_compute_volume_attach_v2.attached: Still creating... (20s elapsed)
openstack_compute_volume_attach_v2.attached: Still creating... (30s elapsed)
openstack_compute_volume_attach_v2.attached: Creation complete (ID: 767ae92c-...da878dd7)

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.
```

Comprobamos en openstack en el apartado Volúmenes que se ha a asociado a la maquina terraform.



Accedemos a la máquina y comprobamos si tenemos el volumen nuevo.

```
jose@castillo:~/openstack/mi_proyecto$ ssh cirros@172.24.4.4
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
vda    253:0    0  39.2M  0 disk
`-vda1 253:1    0  31.4M  0 part /
vdb    253:16   0     1G  0 disk
$
```

CAPÍTULO 10

Bibliografía

https://www.terraform.io/intro/index.html https://www.terraform.io/docs/providers/openstack/index.html https://docs.openstack.org/developer/devstack/ https://www.paradigmadigital.com/dev/terraform-la-navaja-suiza-dominar-todos-los-iaas/ http://superuser.openstack.org/articles/how-to-use-terraform-to-deploy-openstack-workloads/ https://www.youtube.com/watch?v=TFLQcgZr0no https://www.hashicorp.com/blog/Terraform-announcement/ https://docs.joyent.com/public-cloud/api-access/hashicorp https://es.slideshare.net/AmazonWebServices/using-hashicorps-terraform-to-build-your-infrastructure-on-aws-popup-loft-tel-aviv https://platform9.com/blog/how-to-use-terraform-with-openstack/ https://www.matt-j.co.uk/2015/03/27/openstack-infrastructure-automation-with-terraform-part-2/