



Documentación de Terraform



Arquitectura Web

Profesor Antonio Cardeña

García Ramos, Bernardo

Primavera 2021

ÍNDICE

INTRODUCCIÓN	3
DESARROLLO	4
Teoría	4
¿Qué es Terraform?	4
CORE	4
Proveedores y Servicios	5
Ventajas de usar Terraform	5
Comandos	6
Práctica	7
Instalación de Terraform	7
Ejemplo de implementación con AWS	8
Ejemplo de implementación con Google Cloud	16
CONCLUSIONES	20
BIBLIOGRAFÍA/REFERENCIAS	21

INTRODUCCIÓN

El objetivo de este reto y documentación es que quién la lea pueda entender la importancia de herramientas como Terraform y que lo pueda implementar sin mayor problema en los futuros proyectos que lo requieran.

Es por lo anterior que en esta documentación se encuentra todo lo necesario para entender todo lo relacionado con infraestructura en código, al igual que de ciertos servicios como Google Firebase o Amazon Web Services para poder trabajar con ellos. Se va a ir detalle a detalle de todos los conceptos necesarios para entender lo que es Terraform como tal. De igual manera, se va a ilustrar de por qué esta herramienta nos sirve mucho como desarrolladores de software y/o web.

DESARROLLO

Teoría

¿Qué es Terraform?

Terraform es una herramienta que permite automatizar y manejar la infraestructura de tu plataforma (incluyendo todo tipo de servicios). Lo que permite Terraform es poder modelar tu infraestructura DESDE CERO en la nube a través de código. Al igual que Docker, este es un tipo de servicio del cual tiene que estar constantemente actualizado para que tenga un buen funcionamiento y no perjudique tus proyectos, sin embargo, si requieres hacer algo que Terraform no pueda hacer, fácilmente puedes usar algún servicio por a parte junto con Terraform.

Para que se pueda usar y entender Terraform de la forma más sencilla posible, es necesario entender dos conceptos principales: el núcleo (funcionamiento) y los proveedores (servicios).

CORE

Terraform funciona a partir de un archivo con la extensión *.tf*. Este archivo es básicamente contiene todo lo que esperas que tenga tu infraestructura. Para crearlo va a ser necesario usar la documentación oficial de Terraform y elegir tus servicios para incluirlos en el archivo. De igual forma hay algo que se le conoce como el Estado, el cual es una referencia para que Terraform pueda hacer los cambios necesarios e igualar lo que tienes en tu archivo de configuración (lo deseado) con lo que tienes como infraestructura actualmente (lo anterior). Para esto tú ya no te tienes que



preocupar, ya que Terraform es el que se preocupa de igualar ambos para que puedas contar con la infraestructura que desees.

Proveedores y Servicios

Con servicios nos referimos a aquellos que proporcionan plataformas como Amazon Web Services, Google Cloud, Azure, entre otros. Recordemos que este tipo de plataformas sirven para proporcionar servicios en la nube tales como bases de datos, almacenamiento, servidores, entre otros. Al final esta es la que compone la infraestructura de tu proyecto, por lo que saber definirlos y encontrar el indicado no es tan trivial. Lo importante aquí a destacar es que, aunque Terraform cuenta con soporte para la mayoría de estas plataformas, no es ilimitado, por lo que revisar a documentación oficial va a ser indispensable para saber si Terraform conviene usarlo en tu proyecto a desarrollar.

Ventajas de usar Terraform

La ventaja principal de Terraform es que puedes tener tu infraestructura en código. Claramente, mientras tu aplicación dependa de más servicios, más útil va a ser Terraform para ti, de tal forma que no tengas que estar creando ni modificando uno por uno. De igual manera, tener tu infraestructura en código te va a permitir poder subirla a tu repositorio y tener un registro de actualizaciones y cambios de tu infraestructura. Esto va a permitir que el despliegue continuo, ya que toda tu infraestructura estará automatizada, y cualquier cambio que sea necesario implementar lo podrás hacer desde una línea de código.

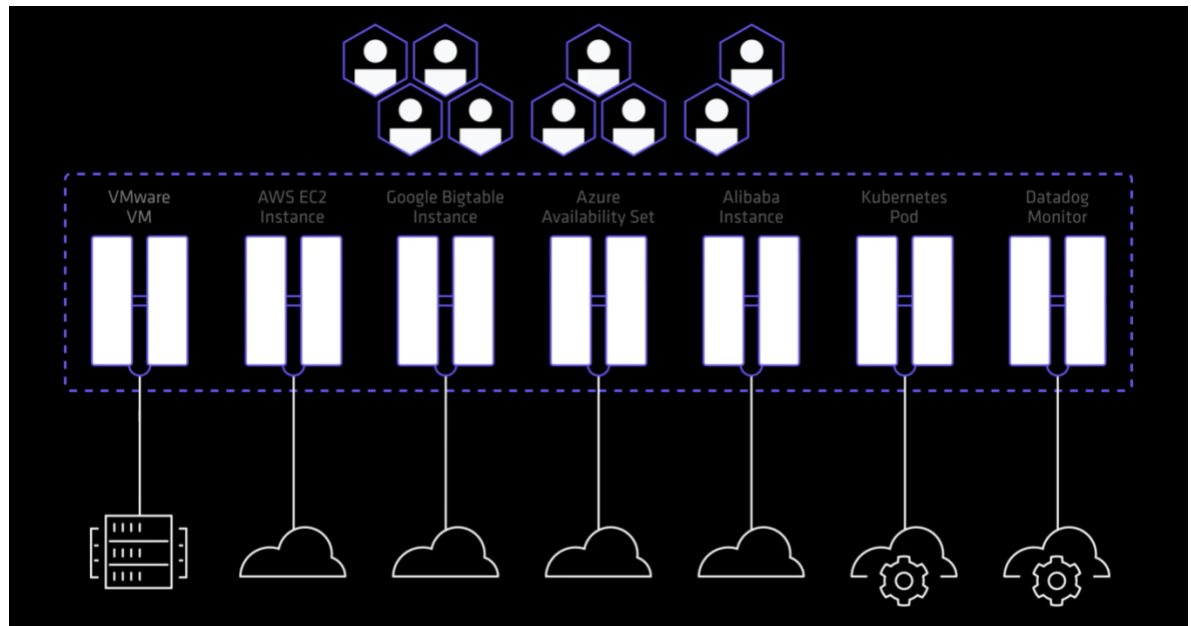


FIGURA 1: Representación de dependencia a varios servicios

Comandos

Aunque existen varios comandos dentro de Terraform, realmente solo vamos a utilizar unos pocos:

- refresh : Permite saber el estado actual de tu infraestructura
- init: Inicializa tu archivo de configuración para buscar errores y demás
- plan : Crea un plan a ejecutar dependiendo de tu archivo de configuración
- apply : Aplica el plan creado con el comando plan
- destroy : Destruye toda tu infraestructura en un solo comando

Como podemos observar, el utilizar estos comandos es muy trivial, sin embargo, es importante considerar la importancia y responsabilidad que tiene esto. Puedes destruir toda tu infraestructura de un solo comando, por lo que es necesario tener cuidado con las acciones que tomes con Terraform.

Práctica

Instalación de Terraform

Lo primero que vamos a necesitar es instalar Terraform. Existen diferentes opciones de descarga dependiendo el sistema operativo que manejes, por lo que recomendamos visitar la siguiente página para descargarlo sin más problemas:

<https://learn.hashicorp.com/tutorials/terraform/install-cli>

Una vez descargado, en tu terminal escribir el comando *terraform*. Si la instalación fue correcta, te desplegará los comandos y configuraciones que puedes hacer. De igual forma, si escribes *terraform versión* y le das enter se espera que te aparezca la captura de pantalla 1:

```
(base) MacBook-Pro-de-Mac:Terraform bernardo$ terraform version
Terraform v0.14.0
+ provider registry.terraform.io/hashicorp/google v3.62.0
+ provider registry.terraform.io/hashicorp/google-beta v3.62.0

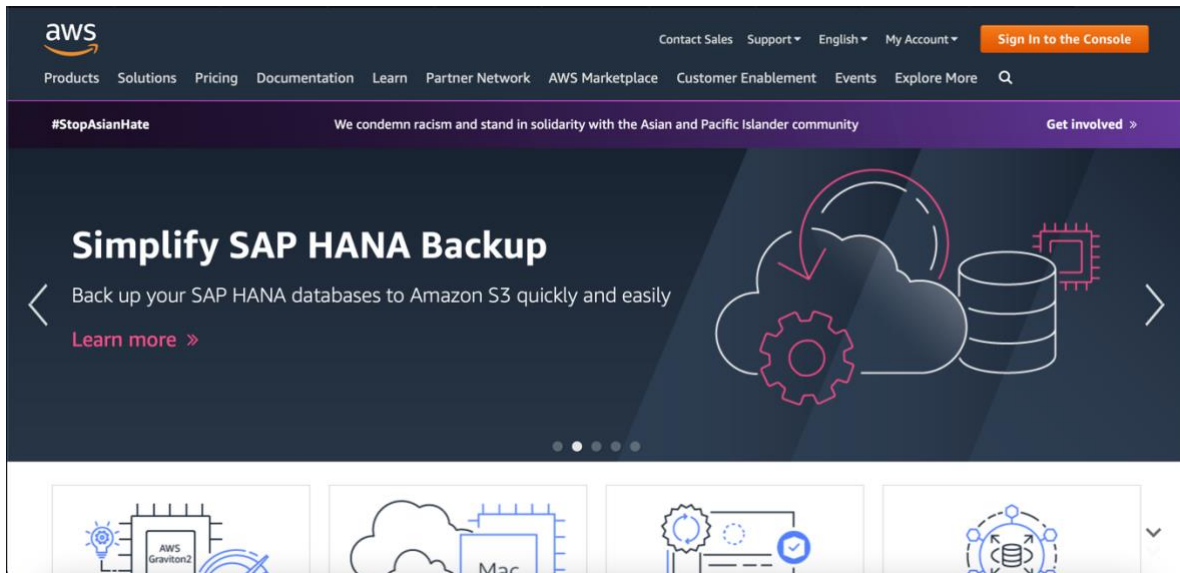
Your version of Terraform is out of date! The latest version
is 0.14.9. You can update by downloading from https://www.terraform.io/downloads.html
```

CAPTURA DE PANTALLA: Resultado de *terraform versión*

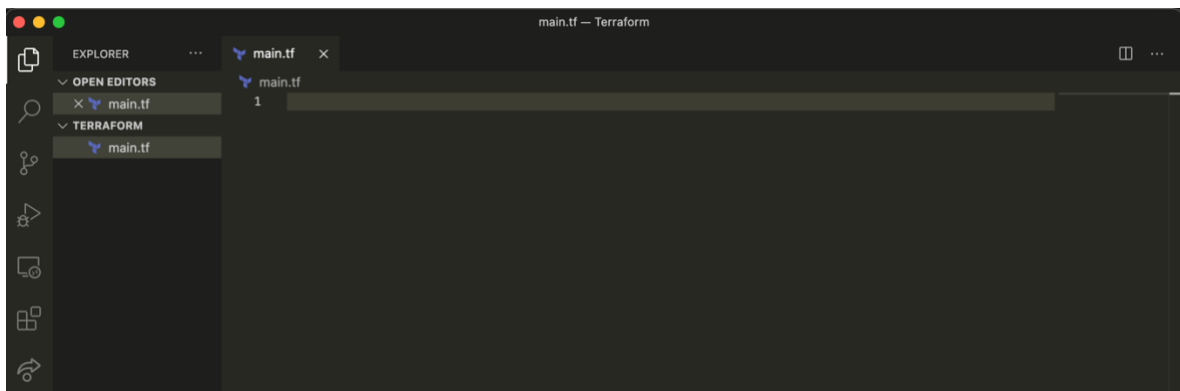
Ejemplo de implementación con AWS

Lo primero que se va a necesitar en este ejemplo es crear una cuenta de Amazon Web Services: <https://aws.amazon.com>

Para esto va a ser necesario que ingreses los datos de alguna forma de pago, pero para el ejemplo no se necesita pagar ningún tipo de servicio, ya que también los vamos a eliminar acabando la prueba.



Una vez teniendo una cuenta de AWS, podemos empezar a crear nuestro archivo de configuración de Terraform. Para eso, crea alguna carpeta dentro de tu computadora y crea un archivo con cualquier nombre y la extensión `.tf`

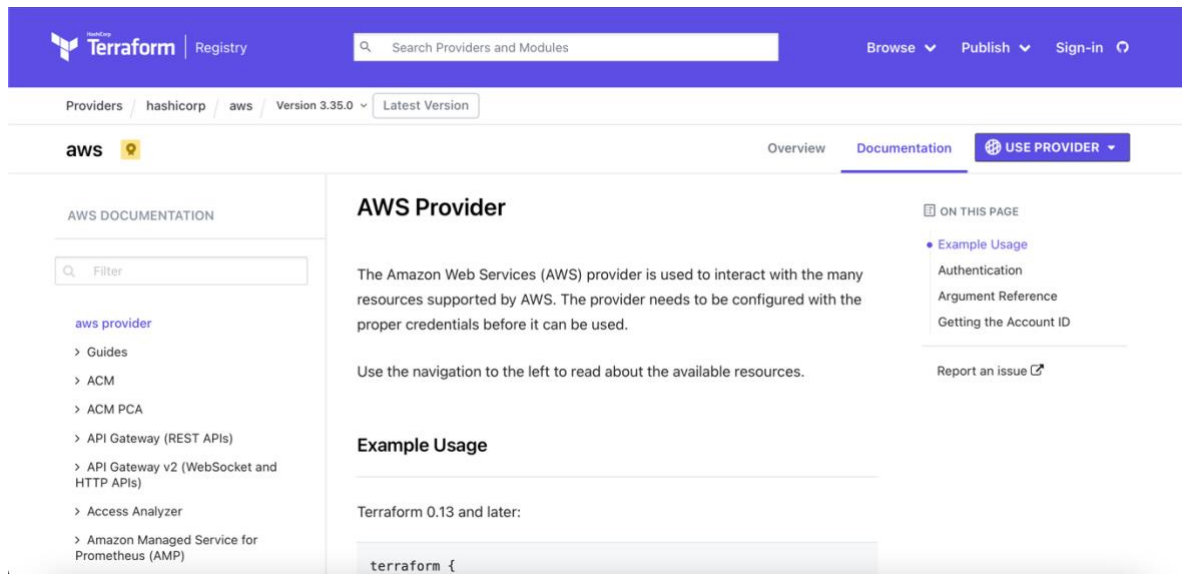


CAPTURA DE PANTALLA: Archivo main.tf

Lo que sigue es empezar a llenar nuestro archivo. ¿Pero cómo? Para eso va a ser necesario utilizar la documentación oficial de Terraform. En este caso buscamos como

proveedor a Amazon Web Services, por lo que la siguiente liga nos va a ser útil:

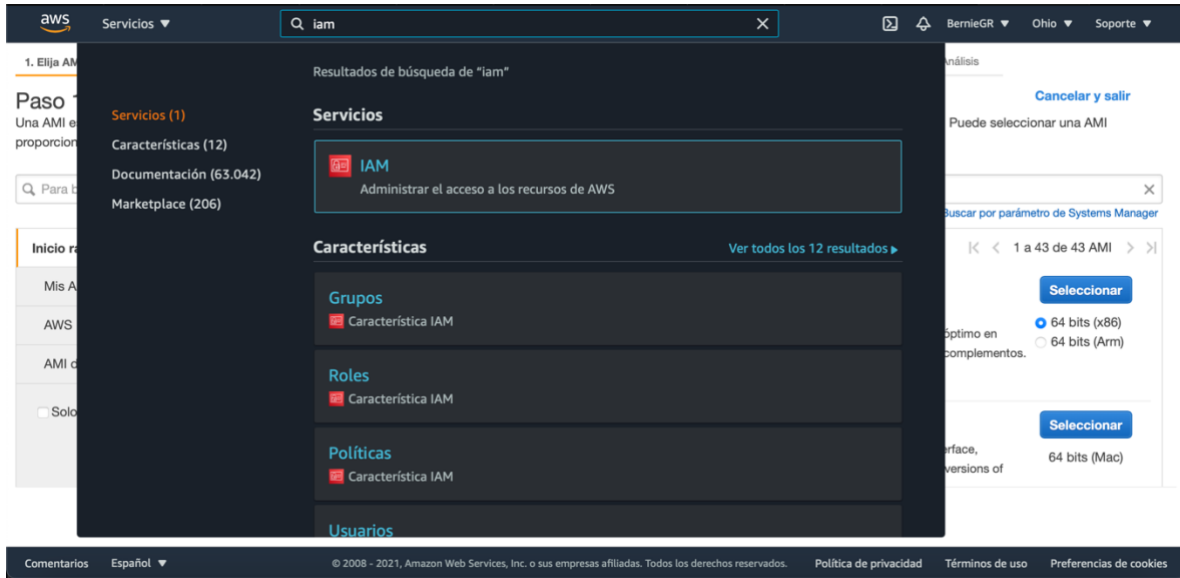
<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>



CAPTURA DE PANTALLA: Terraform y AWS

Para simplificar el ejemplo, vamos a usar las credenciales estáticas que menciona la documentación. Es recomendable leer bien la documentación e implementar credenciales de alguna otra manera para no perjudicar la seguridad del proyecto.

Para comenzar, va a ser necesario registrar usuarios dentro de AWS, esto lo puedes hacer buscando *Identity and Access Managment (IAM)*



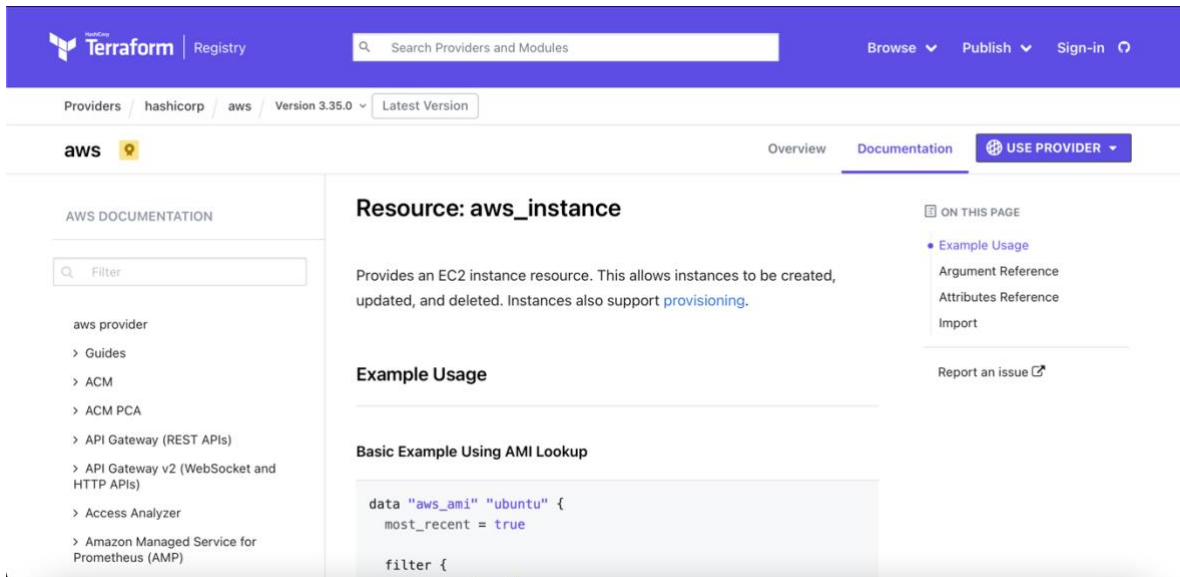
CAPTURA DE PANTALLA: Buscar IAM

Una vez ahí puedes crear un usuario, darle acceso programático y dejar las configuraciones default. Al crear el usuario te van a aparecer las credenciales que se necesitan ingresar en el archivo de configuración, para que quede de la siguiente forma:



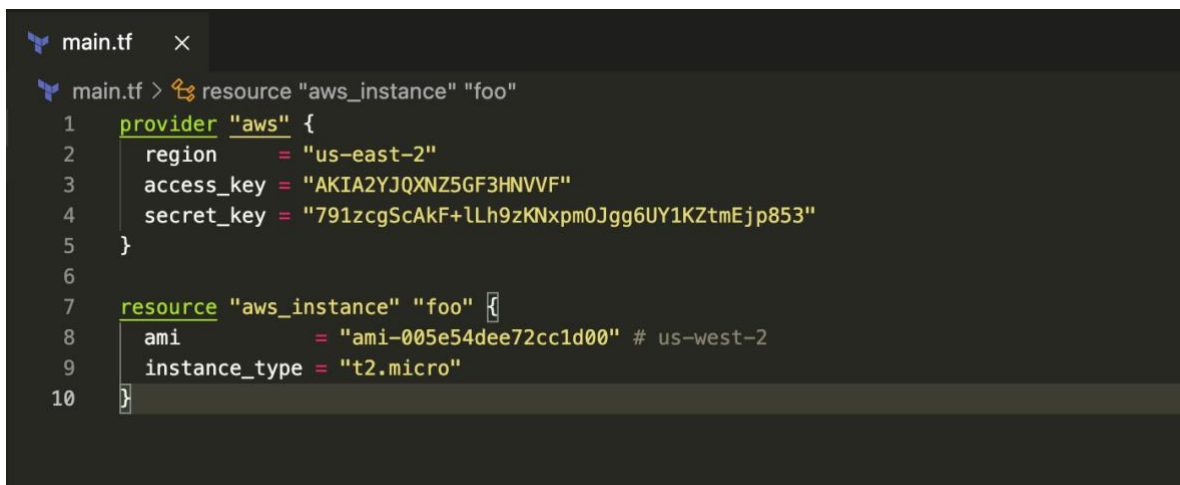
CAPTURA DE PANTALLA: Archivo main.tf con proveedor

Una vez teniendo a nuestro proveedor asignado, tenemos que seleccionar los recursos o servicios que requerimos. Igual para eso buscamos el servicio de AWS junto con Terraform. En este caso buscamos EC2 (servidores virtuales en la nube).



CAPTURA DE PANTALLA: Terraform y EC2

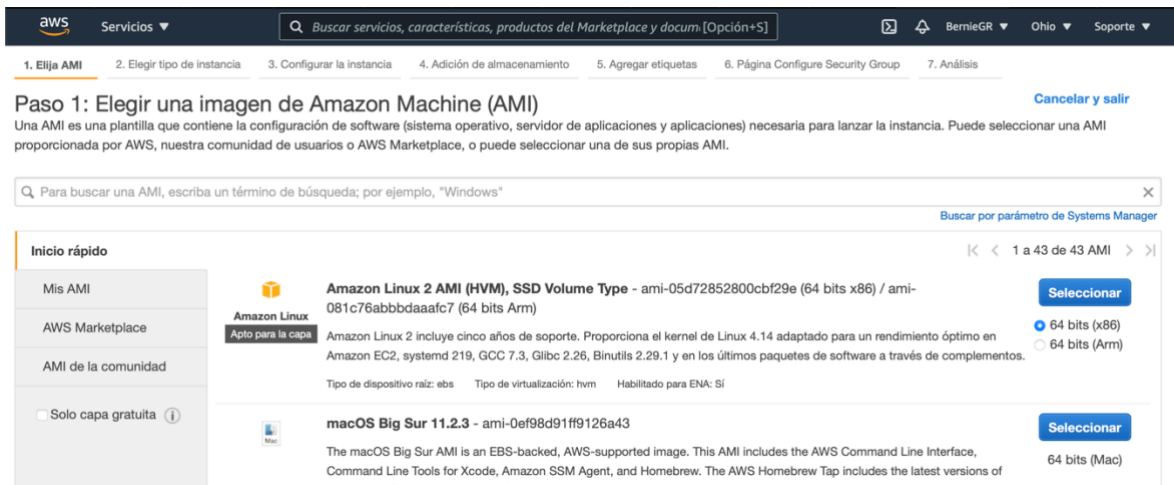
Una vez leyendo un poco de la documentación podemos ingresar esta parte de código a nuestro archivo.



CAPTURA DE PANTALLA: Archivo main.tf con proveedor y recurso



Notemos que la imagen de la máquina Amazon (ami) está destinada para ser usada en us-west-2. En nuestro caso buscamos la de us-east-2 porque es la más cercana para nosotros, así que la buscamos.



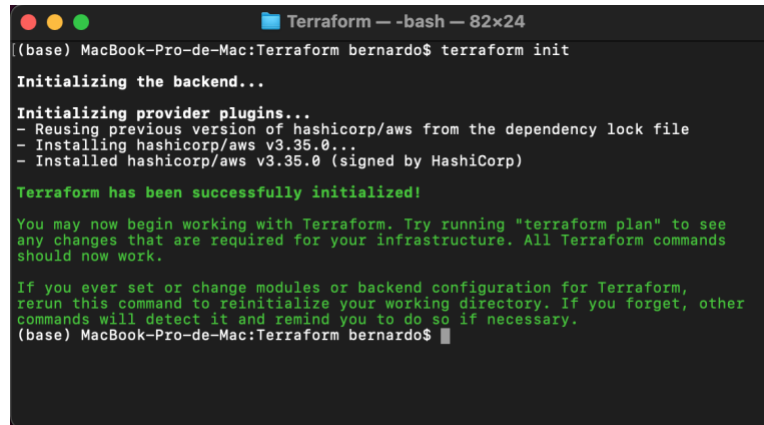
CAPTURA DE PANTALLA: Buscar imagen de Amazon Machine

Ahí podemos ver los identificadores de las ami que queremos utilizar. Elegimos uno y lo agregamos al código.



CAPTURA DE PANTALLA: Archivo main.tf final

Teniendo esto, ya podemos empezar a utilizar los comando de Terraform para tener nuestra infraestructura corriendo. En tu terminal, vas a ingresar los comandos init, plan y apply (en ese orden).



```
((base) MacBook-Pro-de-Mac:Terraform bernardo$ terraform init

Initializing the backend...

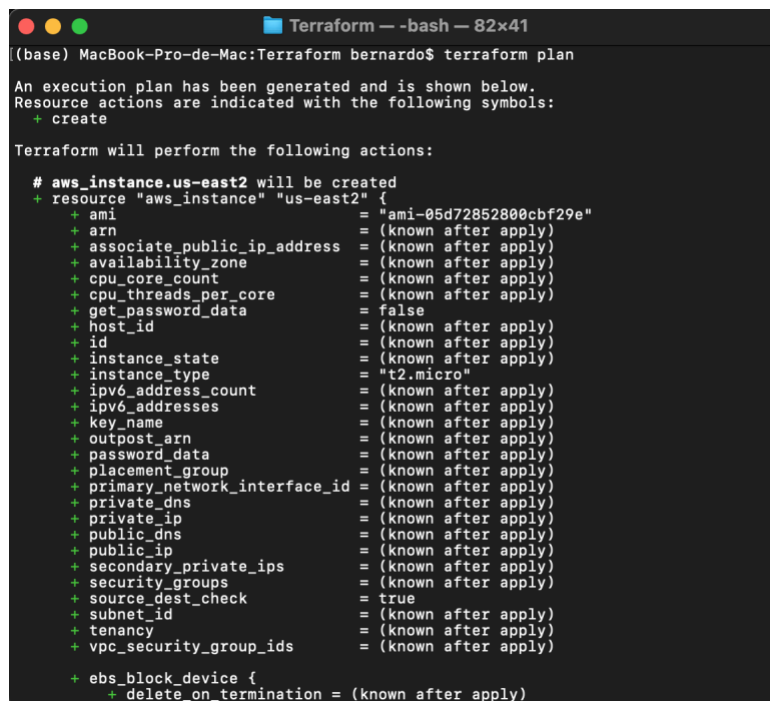
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v3.35.0...
- Installed hashicorp/aws v3.35.0 (signed by HashiCorp)

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.
(base) MacBook-Pro-de-Mac:Terraform bernardo$
```

CAPTURA DE PANTALLA: Comando init



```
((base) MacBook-Pro-de-Mac:Terraform bernardo$ terraform plan

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_instance.us-east2 will be created
+ resource "aws_instance" "us-east2" {
  + ami                    = "ami-05d72852800cbf29e"
  + arn                    = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone       = (known after apply)
  + cpu_core_count          = (known after apply)
  + cpu_threads_per_core    = (known after apply)
  + get_password_data       = false
  + host_id                 = (known after apply)
  + id                     = (known after apply)
  + instance_state          = (known after apply)
  + instance_type           = "t2.micro"
  + ipv6_address_count      = (known after apply)
  + ipv6_addresses          = (known after apply)
  + key_name                = (known after apply)
  + outpost_arn             = (known after apply)
  + password_data           = (known after apply)
  + placement_group         = (known after apply)
  + primary_network_interface_id = (known after apply)
  + private_dns             = (known after apply)
  + private_ip              = (known after apply)
  + public_dns              = (known after apply)
  + public_ip               = (known after apply)
  + secondary_private_ips   = (known after apply)
  + security_groups         = (known after apply)
  + source_dest_check        = true
  + subnet_id               = (known after apply)
  + tenancy                 = (known after apply)
  + vpc_security_group_ids  = (known after apply)

  + ebs_block_device {
    + delete_on_termination = (known after apply)
  }
}
```

CAPTURA DE PANTALLA: Comando plan

```
Terraform — -bash — 77x35

+ delete_on_termination = (known after apply)
+ device_index          = (known after apply)
+ network_interface_id  = (known after apply)
}

+ root_block_device {
+ delete_on_termination = (known after apply)
+ device_name           = (known after apply)
+ encrypted              = (known after apply)
+ iops                   = (known after apply)
+ kms_key_id             = (known after apply)
+ tags                   = (known after apply)
+ throughput             = (known after apply)
+ volume_id              = (known after apply)
+ volume_size            = (known after apply)
+ volume_type            = (known after apply)
}

}

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

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

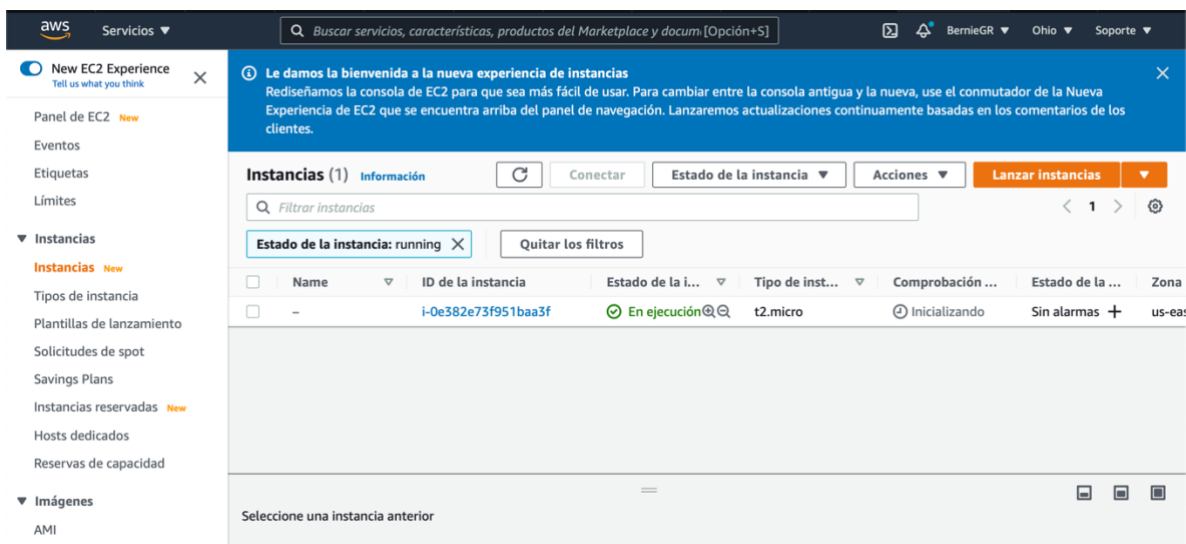
Enter a value: yes

aws_instance.us-east2: Creating...
aws_instance.us-east2: Still creating... [10s elapsed]
aws_instance.us-east2: Still creating... [20s elapsed]
aws_instance.us-east2: Still creating... [30s elapsed]
aws_instance.us-east2: Creation complete after 33s [id=i-0e382e73f951baa3f]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
(base) MacBook-Pro-de-Mac:Terraform bernardo$
```

CAPTURA DE PANTALLA. Comando apply

Como podemos ver, la instancia se completó exitosamente y en Amazon Web Services deberíamos de tener un servidor virtual activo.



CAPTURA DE PANTALLA: Instancia de AWS

Como podemos ver, la instancia se completó correctamente. Esto se puede hacer para todo tipo de instancias y servicios que proporciona AWS. Ahora solo falta destruirlo ya que no lo vamos a estar utilizando.

```
- http_tokens          = "optional" -> null
}

- root_block_device {
  - delete_on_termination = true -> null
  - device_name           = "/dev/xvda" -> null
  - encrypted             = false -> null
  - iops                  = 100 -> null
  - tags                  = {} -> null
  - throughput            = 0 -> null
  - volume_id             = "vol-0ee2b3d0a8f08afb3" -> null
  - volume_size           = 8 -> null
  - volume_type           = "gp2" -> null
}

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

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

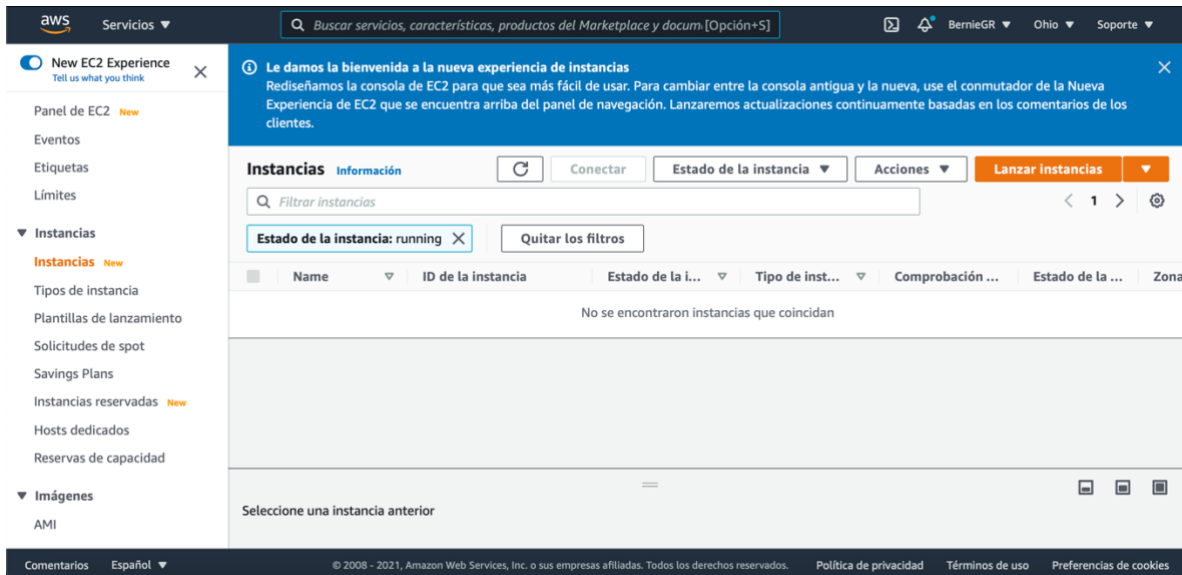
Enter a value: yes

aws_instance.us-east2: Destroying... [id=i-0e382e73f951baa3f]
aws_instance.us-east2: Still destroying... [id=i-0e382e73f951baa3f, 10s elapsed]
aws_instance.us-east2: Still destroying... [id=i-0e382e73f951baa3f, 20s elapsed]
aws_instance.us-east2: Still destroying... [id=i-0e382e73f951baa3f, 30s elapsed]
aws_instance.us-east2: Destruction complete after 30s

Destroy complete! Resources: 1 destroyed.
(base) MacBook-Pro-de-Mac:Terraform bernardo$
```

CAPTURA DE PANTALLA: Comando destroy

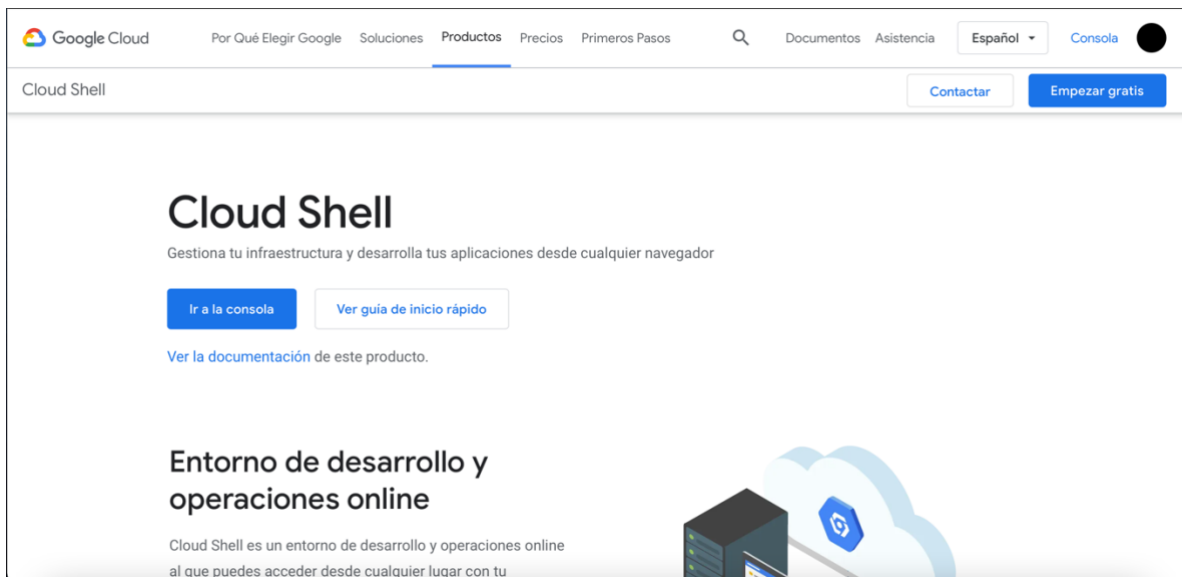
Con el comando destroy pudimos destruir y eliminar por completo la infraestructura que estaba asignada a nuestro archivo de configuración. Para comprobarlo no queda más que regresar a AWS y ver que no hay instancias en ejecución.



CAPTURA DE PANTALLA: Instancia eliminada de AWS

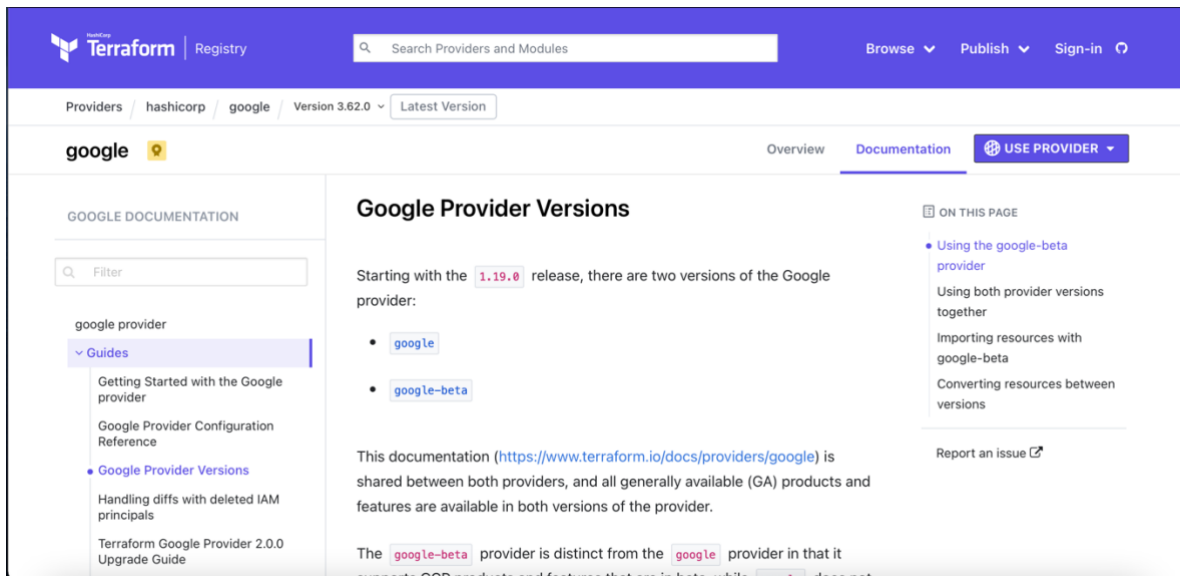
Ejemplo de implementación con Google Cloud

Para comenzar con esto, va a ser necesario conocer que existe una Cloud Shell (terminal) de parte de Google, la cual nos va a facilitar el uso de Terraform junto con Google Cloud.



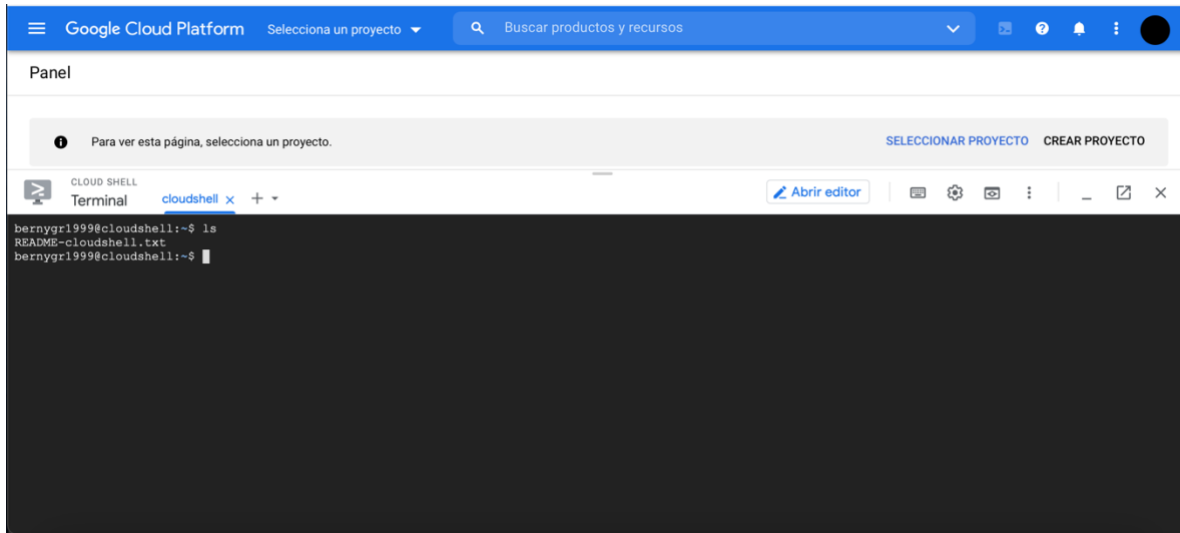
CAPTURA DE PANTALLA: Cloud Shell

Cloud Shell ya cuenta con Terraform, por lo que no va a ser necesario crearlo. El siguiente paso va a ser empezar a buscar la documentación de Terraform con Google Cloud / Firebase.



CAPTURA DE PANTALLA: Google como proveedor

Según la documentación, no es necesario establecer el rubro de *provider*, ya que lo asigna en automático, sin embargo, en nuestro caso lo vamos a asignar en nuestro archivo. Para eso vamos a crear el archivo con ayuda de Google Cloud Plataform.



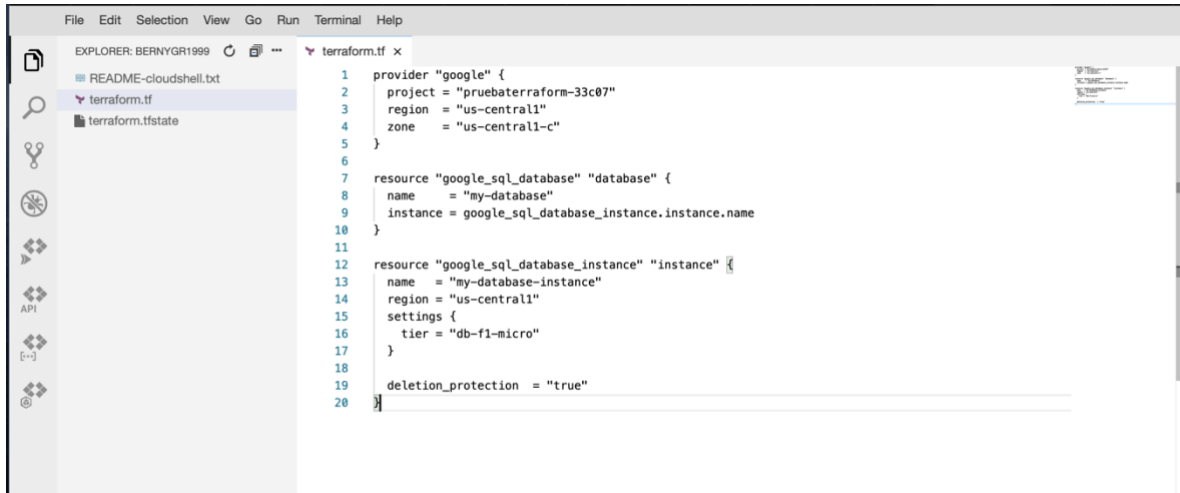
CAPTURA DE PANTALLA: Dar click en Abrir editor

Una vez adentro podemos crear (o no) una carpeta y un archivo con la extensión .tf para poder ingresar el proveedor como código.



CAPTURA DE PANTALLA: Editor de google

Ahora simplemente vamos a agregar una base de datos como almacenamiento. Para eso buscamos la información e ingresamos el código en el mismo editor.



```
1 provider "google" {
2   project = "pruebaterraform-33c07"
3   region  = "us-central1"
4   zone    = "us-central1-c"
5 }
6
7 resource "google_sql_database" "database" {
8   name      = "my-database"
9   instance = google_sql_database_instance.instance.name
10 }
11
12 resource "google_sql_database_instance" "instance" {
13   name      = "my-database-instance"
14   region    = "us-central1"
15   settings {
16     tier = "db-f1-micro"
17   }
18
19   deletion_protection = "true"
20 }
```

CAPTURA DE PANTALLA: Archivo terraform.tf final

Una vez corriendo los respectivos comandos (init, plan y apply), ya tendras generada tu base de datos dentro de tu proyecto. De igual manera recomendamos eliminar todo lo relacionado a las pruebas con el comando detroy.

CONCLUSIONES

Las ventajas de Terraform son bastantes, y más cuando tienes muchos servicios de los cuales dependes. El utilizarlo en ningún punto puede llegar a afectar la infraestructura de tu aplicación (a menos que lo codifiques mal), por lo que parece ser que utilizarlo es una herramienta que todo desarrollador web/aplicaciones tiene que conocer.

De hecho, esta aplicación hace aún más fácil la implementación de Docker, ya que se podría hacer algo similar con ese tipo de servicios. Aunque nuevamente, sirve más cuando tienes varios servicios a utilizar. Si solo vamos a usar Docker, usar Terraform no hace mucho sentido. Es por eso que es necesario juzgar la situación correcta y aplicarlo correctamente para que la infraestructura de nuestras aplicaciones esté actualizada y registrada, ayudándonos así a tener un despliegue continuo de nuestra aplicación.

BIBLIOGRAFÍA/REFERENCIAS

<https://www.terraform.io>

https://www.youtube.com/watch?v=k09fTeW3_Qs&t=920s

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance>

https://www.youtube.com/watch?v=l5k1ai_GBDE