Introducción a Infraestructura como Código (IaC)

Miguel Barajas

Version 1.0, 2021

Contenido

Dedicación

A Valeria, Roberto y Carlota. Mi mundo

Prefacio

Por qué escribí este libro

Una gran parte de mi carrera profesional, la he dedicado a la automatización y orquestación. Es un tema que me apasiona, sin embargo, la automatización "tradicional" cada vez se ha convertido en un *break and fix*. Es muy estática y requiere mucho esfuerzo mantener los flujos de trabajo, integraciones y *scripts* para que la automatización se siga dando en un mundo tan dinamico como el que vivimos en las tecnologías de la información. Desde el primer momento que descubrí los manejadores de configuración, quedé enamorado. ***INCOMPLETO***

Por qué debería de leer este libro

Convenciones usadas en este libro

Recursos en linea

Como contactarme

COPIA ESTE LIBRO

Sí, copia este libro, usalo para cualquier fin, auméntalo, regalalo. Este libro está bajo la licencia CC

Capítulo 1. Introducción.

Para entender la infraestructura como código (IaC por sus siglas en inglés) primero debemos entender qué es infraestructura y qué es código. Infraestructura en el contexto de informática, es el hardware, físico, virtual o lógico que soporta una aplicación o plataforma. Del mismo modo, el código son las instrucciones que permiten crear una aplicación o software. Luego entonces, la infraestructura como código, es la descripción del estado deseado de la configuración de esta infraestructura en un documento, normalmente escrito en texto plano. Al estar escrito en texto plano, esta descripción puede tratarse de la misma manera que se trata un código de software, donde podemos versionarlo, probarlo, compartirlo, etc.

Para hacer esto realidad, debemos contar con un intermediario o intérprete que comprenda esta descripción y la convierta en una configuración que sea entendida por la infraestructura. Esto nos permite automatizar la infraestructura de tal manera que esa configuración la tratemos como software, esto nos da varias ventajas que estaremos discutiendo durante la presente obra, algunas de ellas son:

- Automatización declarativa en vez de imperativa
- Versionamiento de la configuración
- Replicación sencilla
- **ESCRIBIR MAS**

Automatización de Infraestructura

La automatización de la infraestructura no es un tema nuevo, desde hace décadas, está práctica existe, dado a la necesidad de hacer más eficiente el despliegue de esta infraestructura, esto se hace mediante ***scripts*** o desarrollos que permiten que esta infraestructura se despliegue automáticamente, el problema con este acercamiento, es que no solo le tenemos que planear qué se va a desplegar, si no tambien cómo se desplegará, por lo que es una automatización **imperativa**, esto nos acarrea un problema dado que muchas veces la infraestructura no puede ser cambiada una vez desplegada, puesto que en la automatización debemos tener en cuenta todas las excepciones que pudieran ocurrir durante el ciclo de vida de la infraestructura. Por ende, la automatización imperativa funciona muy bien solo para el despliegue de nueva infraestructura, pero no para mantener el ciclo de vida completa de ella. Es decir, despliegue, cambios y decomización.

Más allá de la automatización, existe el concepto de ***orquestación*** en el que normalmente existe un software llamado **orquestador** (***orchestrator*** en inglés) el cual cuenta con los conectores necesarios para automatizar el despliegue de las diferentes capas de la infraestructura: Almacenamiento, Red, Computo, Sistema Operativo, etc).

Esto permite "orquestrar" el despliegue de la pila completa de la infraestructura. Pero de la misma manera que se hace al automatizar de manera imperativa, capa por capa, la orquestación imperativa no permite manejar el ciclo de vida completo de la infraestructura.

Manejadores de configuración

Los manejadores de configuración, han existido, de igual manera, desde hace décadas, son piezas de software que nos permiten mantener la configuración de un sistema operativo o aplicación de manera declarativa. Es decir, en vez de indicar el qué y el cómo, se describe el qué, es decir se declara el estado deseado de la configuración del Sistema Operativo o aplicación y es el manejador de configuración quien hace la conciliación entre el estado declarado y el estado actual. Es decir, que solo modifica las partes de la configuración que no concuerdan con el estado deseado. Como se manifestó, los Manejadores de configuración (***Configuration Manager*** en inglés) fueron concebidos para mantener la configuración del Sistema Operativo y/o la aplicación. Con el avance de las tecnologías de virtualización y del concepto de ***Software Defined***, el despliegue y configuración de la infraestructura, también puede ser declarada y manejada como si de un sistema operativo o una aplicación se tratara.

El crecimiento de la popularidad de IaC

Sin duda, la adopción del ***Cloud Computing*** ha permitido que la Infraestructura como código se haya popularizado, la disponibilización de la interfaces de programabilidad (APIs), así como el cobro por tiempo utilizado de la infraestructura, ha hecho que IaC sea muy popular para este tipo de modelos de consumo. Sin embargo, también e influenciado los fabricantes de infraestructura física, y estos han empezado a disponibilizar *APIs* Así como también han separado el plano de control del plano de datos, que es el principio del *hardware* definido por *software*. Empresas como ***Hashicorp*** han creado soluciones de Infraestructura como código que estaremos usando en esta obra, que lleva por nombre ***Terraform***. Terraform es una herramienta de código abierto para infraestructura como código que provee un flujo de trabajo por linea de comandos consistente para manejar cientos de servicios de nube. Terraform codifica las *APIs* de las nubes en archivos de configuración declarativa. [1]

[1] https://www.terraform.io/

Capítulo 2. Caso de estudio.

Através de este libro, estaremos creando los bloques necesarios para mantener la operación de una aplicación mantenida en la nube, la cual cuenta con varios elementos de infraestructura que definiremos como código. La idea de este caso de estudio, será que podamos desplegar esta aplicación, modificarla, escalarla, decomisarla, cambiarla de una región a otra, etc.

El casó de estudio será de la empresa imaginaría de repostería "Galletería Carlota" la cual mantiene su sistema de tienda en linea en la ***Amazon Web Service*** o ***AWS*** como le llamaremos de aquí en adelante. La aplicación tienen varias capas o niveles como se muestra en la siguiente figura, esta será la arquitectura propuesta para la aplicación. El cual cuenta con lo siguiente:

- Tres (3) Balanceador de Carga nativo de *AWS* para la capas de presentación, aplicación y base de datos.
- Una capa de presentación *web* que permite ser escalada verticalmente
- Las imágenes de los productos a vender serán almacenadas en un *S3 Bucket*
- Una capa de Aplicación que será donde se procese la logíca de negocio la cual tambien debe ser verticalmente escalable
- Una capa de Base de datos, la cual es nativa a la nube y permite ser escalada verticalmente, en este caso usaremos *CoackroachDB*

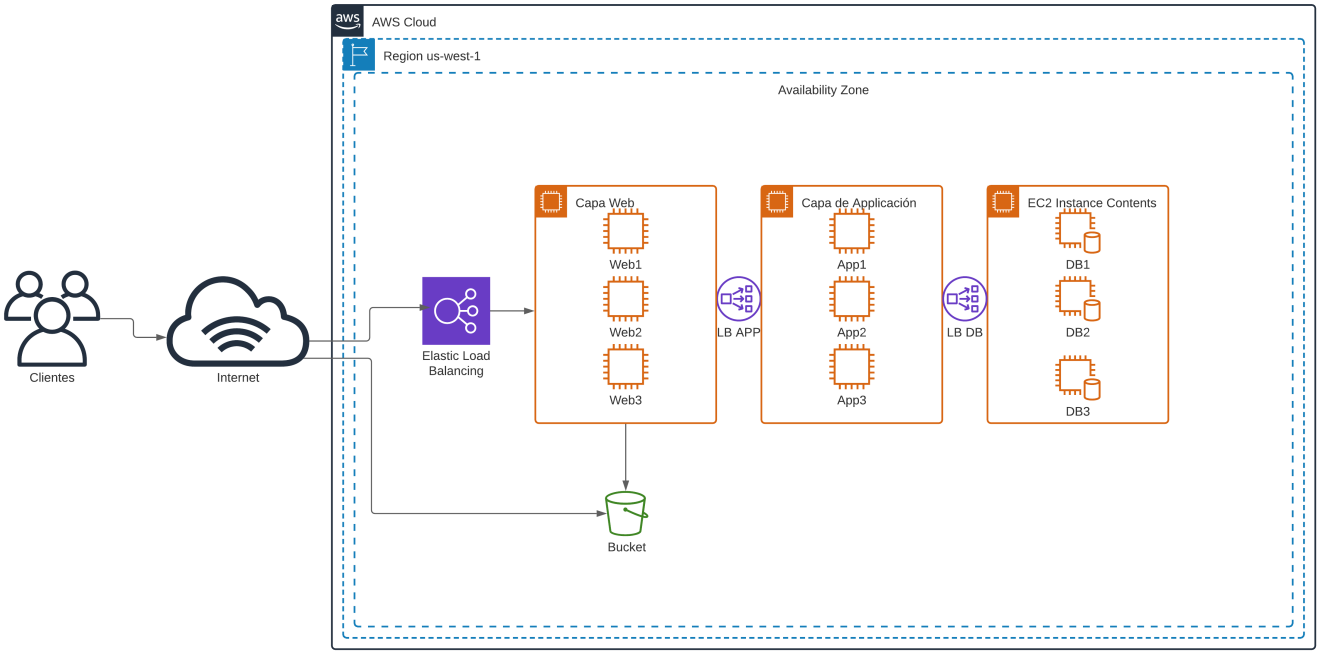

Imagen 1. Arquitectura de la Aplicación en AWS

Capítulo 3. Introducción a Terraform

Como ya se platicó en el capítulo uno, durante el desarrollo de esta obra estaremos utilizando la versión abierta y gratuita de ***Hashicorp Terraform***, la cual ha tomado mucha popularidad entre intenieros que hacen ***DEVOPS*** y ***CI/CD***, temas que trataremos más adelante. Y no es casualidad, *Terraform* ofrece un entorno bien documentado y con cientos de integraciones que permiten a las organizaciones tratar su infraestructura como código al escribir archivos con la declaración del estado deseado de las configuraciones. *Terraform* usa su propio lenguage llamado ***Hashicorp Configuration Language (HLC)*** qué permite una descripción consisa de los recursos usando bloques de código, argumentos y expresiones.

De la misma manera, *Terraform* permite correr las verificaciones necesarias antes de aplicar las configuraciones deseadas para asegurarse que los cambios a realizar son factibles y el operador tiene oportunidad de entender qué cambios sucederán y como afectará el ambiente desplegado. Una vez hecho esto el operador puede proceder a aplicar los cambios, para poder llegar al estado deseado solo modificando el delta entre el estado actual y el deseado.

Código Ejemplo escrito en HCL para AWS

```
variable "ami_id" {
  type        = string
  description = "AMI ID to use"
  default     = "ami-09d95fab7fff3776c"
}

variable "instance_type" {
  type        = string
  description = "Instance type to use"
  default     = "t3.micro"
}

variable "availability_zone" {
  type        = string
  description = "Availability Zone to use"
  default     = "us-east-1a"
}
```

Características

Algunas de las características más importantes de *Terraform* son:

Algunos

- **Archivos de Configuración Declarativa:** Definición de Infraestructura como código para manejar el ciclo de vida completo, creación de nuevos recursos, manejo de los existentes así como la decomisación cuando estos ya no son necesarios.
- **Modulos instalables:** Intalación automática de modulos de la comunidad o terceros desde el

registro de *Hashicorp* por medio de `terraform init`.

- **Planeación y predicción de cambios:** *Terraform* permite a los operadores hacer cambios a la infraestructura de manera segura y predectible.
- *Graficación de dependencias
- **Manejo del estado:** Mapeo de la configuración de los recursos del mundo real, mantenimiento de los *metadatos* y mejoramiento del *performance* en infraestructuras grandes
- **Registro de más de 500 proveedores:** El operador puede escoger de una serie de proveedores para las diferentes plataformas de nube disponibles en el mercado.

Casos de uso

Algunos de los casos de uso que se pueden cubrir con terraform son: [2]

- Despliegue de aplicaciones en *Heruku*.
- Aplicaciones Multi Capa. [3]
- *Clusteres* de Autoservicio.
- Demostraciones de *Software*.
- Ambientes deshechables.
- Despliegues multi nube.

Instalación de *Terraform*

En esta sección estaremos discutiendo la instalación del binario de *Terraform* en nuestro equipo personal, donde crearemos un ambiente de desarrollo para escribir, probar y desplegar nuestra infraestructura. Existen varios métodos para la instalación de *Terraform* a continuación discutiremos los más importantes.

Instalación Manual

Para la instalación manual de *Terraform*, necesitamos encontrar el paquete apropiado [4] para nuestro sistema operativo y bajarlo, este será un archivo *zip*.

Una vez que se haya bajado *Terraform*, procederemos a expander el archivo *zip*. Encontraremos que es un solo binario llamado `terraform`. Todos los demás archivos que pudiera contener el archivo *zip* pueden ser borrados de manera segura sin que esto afecte el funcionamiento de *Terraform*.

Finalmente, nos aseguraremos de que `terraform` se encuentre disponible en nuestro `PATH`. Esto se realiza de manera diferente, dependiendo el sistema operativo.

Mac o Linux

Obtenemos la lista de rutas que están disponibles en la variable de entorno `PATH`

```
echo $PATH
```

Movemos el binario te *Terraform* a uno de las rutas listadas. Este comando asume que el binario se encuentra en el archivo de descargas y que `PATH` contiene `/usr/local/bin`, personaliza en caso de las rutas en tu sistema operativo sean diferentes.

```
mv ~/Downloads/terraform /usr/local/bin/
```

Windows

En la siguiente dirección de internet, podemos encontrar las instrucciones exactas para modificar el `PATH` en *Windows* através de la interfaz gráfica: https://stackoverflow.com/questions/1618280/where-can-i-set-path-to-make-exe-on-windows

Instalación con *Homebrew* en *MacOS*

Homebrew es un manejador de paquetes de fuente abierta para el sistema operativo *MacOS*. Instala la *formula* oficial de *Terraform* desde la terminal.

Primero, installamos el *tap* de *HashiCorp*, un repositorio para todos los paquetes de *Homebrew* de la compañia:

```
brew tap hashicorp/tap
```

Ahora, Instalamos *Terraform* con `hashicorp/tap/terraform`

```
brew install hashicorp/tap/terraform
```

Para actualizar a la última versión, ejecutamos:

```
brew upgrade hashicorp/tap/terraform
```

Instalación con *Chocolatey* en *Windows*

Chocolatey es un manejador de paquetes de codigo abierto para *Windows*. Installamos el paquete de *Terraform* desde la linea de comandos.

```
choco install terraform
```

Instalación en Linux

Ubuntu/Debian

Agregamos la llave *GPG* de *HashiCorp*.

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
```

Agregamos los repositorios oficiales de *HashiCorp* para *Linux*.

```
sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com
$(lsb_release -cs) main"
```

Actualización e instalación.

```
sudo apt-get update && sudo apt-get install terraform
```

CentOS/RHEL

Instalamos `yum-config-manager` para manejar repositorios.

```
sudo yum install -y yum-utils
```

Usamos `yum-config-manager` para agregar el repositorio oficial de *HachiCorp* para *Linux*

```
sudo yum-config-manager --add-repo
https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo
```

Instalamos

```
sudo yum -y install terraform
```

Fedora

Instalamos `dnf config-manager` para manejar repositorios.

```
sudo dnf install -y dnf-plugins-core
```

Usamos `dnf config-manager` para agregar el repositorio oficial de *HachiCorp* para *Linux*

```
sudo dnf config-manager --add-repo
https://rpm.releases.hashicorp.com/fedora/hashicorp.repo
```

Instalamos

```
sudo dnf -y install terraform
```

Verificación de la instalación.

Para verificar la instalación abrimos una nueva terminal y ejecutamos `terraform -help`

```
terraform -help
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.
...
```

Cualquier sub comando despues de `terraform -help` permite aprender más sobre el mismo.

```
terraform -help plan
```

Con esto finalizamos la instalación de *Terraform* y estamos listos para empezar a construir infraestructura como código.

Contrucción de Infraestructura

Con *Terraform* installado, estamos listos para crear nuestra primera infraestructura.

Vamos a aprovisionar una ***Amazon Machine Image (AMI)*** en ***AWS*** esto lo haremos de esta manera, dado de las *AMIs* son muy populares.

Pre requisitos

Para continuar, necesitamos:

- Una cuenta de *AWS*
- La intefaz de Linea de comando de *AWS*
- Las credenciales de *AWS* configuradas localmente

Esto está descrito en el Apéndice Creación y configuración de una cuenta de Amazon Web Services

Escribir configuraciones

El set de archivos que es usado para describir la infraestructura en *Terraform* es conocido como *Terraform Configuration*. Vamos a escribir nuestra primera configuración ahora para lanzar una instancia de *EC2 de AWS*.

Cada configuración debe estar en su propio directorio. Crearemos un directorio para esta nueva

configuración.

```
mkdir iac-libro-aws-instancia
```

Nos cambiamos al directorio recién creado

```
cd iac-libro-aws-instancia
```

Pegamos la configuración que está a continuación dentro de un archivo que tenga por nombre `ejemplo.tf` y lo guardamos. *Terraform* carga todos los archivos en el directorio actual que tengan la extención `.tf`

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 2.70"
    }
  }
}

provider "aws" {
  profile = "default"
  region  = "us-west-2"
}

resource "aws_instance" "ejemplo" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"
}
```

Esta es una configuración completa de *Terraform* y está lista para ser aplicada. En las siguientes secciones, veremos como funciona cada uno de estos bloques en más detalle.

Bloques de *Terraform*

El bloque `terraform {}` es requerido para que *Terraform* conosca qué proveedor tiene que descargar desde el registro de *Terraform*. En la condiguración anterior, el proveedor `aws` está definido como `hashicorp/aws` que es una abreviación de `redistry.terraform.io/hashicorp/aws`.

También podemos asignar una versión definida en el bloque `required_providers`. El argumento `version` es opcional, pero recomendado.

Proveedores

El bloque `provider` configura el nombre del proveedor, en nuestro caso `aws`, que es responsable de crear y manejar los recursos. Un proveedor es un *plugin* que *Terraform* usa para traducir las

interacciones con las *API* del servicio a administrar. Un proveedor es el responsable por entender tales interacciones y exponer los recursos. Por el hecho de que *Terraform* puede interactuar con cualquier *API*, podemos representar casi cualquier tipo de infraestructura como recursos en *Terraform*.

El atributo `profile` en nuestro bloque de proveedor se refiere, en este caso, a las credenciales de *AWS* almacenadas en la el archivo de configuración de *AWS*, que se creo cuando se hizo la configuración del *CLI* de *AWS*. La mejor práctica recomendada, es que no se use credenciales directamente en los archivos `.tf`.

Pueden existir multiples bloques de proveedor en caso de ser necesario. Podemos, incluso, usar multiples proveedores juntos. Por ejemplo podemos pasar el identificador de una intancia de *AWS* para monitorear tal recurso con *DataDog*.

Recursos

El bloque de `resources` define una pieza de la infraestructura. Un recurso puede er un componente físico o virtual como una instancia de *EC2* o puede ser un recurso lógico como una *IP elástica*.

El bloque de recursos tiene dos entradas antes del bloque como tal: el tipo de recurso y el nombre del recurso. en este ejemplo, el tipo de recurso es `aws_instance` y el nombre es `ejemplo`. El prefijo en el tipo de recurso se mapea al proveedor. En nuestro caso `aws_instance` automaticamente le dice a *Terraform* que este serña manejado por el proveedor *aws*

[2] https://www.terraform.io/intro/use-cases.html

[3] Este es el caso de uso específico que estaremos cubriendo en esta obra

[4] https://www.terraform.io/downloads.html

Apéndice A: Creación y configuración de una cuenta de *Amazon Web Services*

TODO