Certified Tech Developer
The Ultimate Degree

Infraestructura II

Terraform: Detrás de la escena

Como hemos visto, la función principal de Terraform es la de crear, modificar y destruir recursos de infraestructura.

Pero, ¿cómo trabaja este componente realmente? ¿Cómo se comunica con nuestro proveedor de Cloud? ¿Cómo se configura?

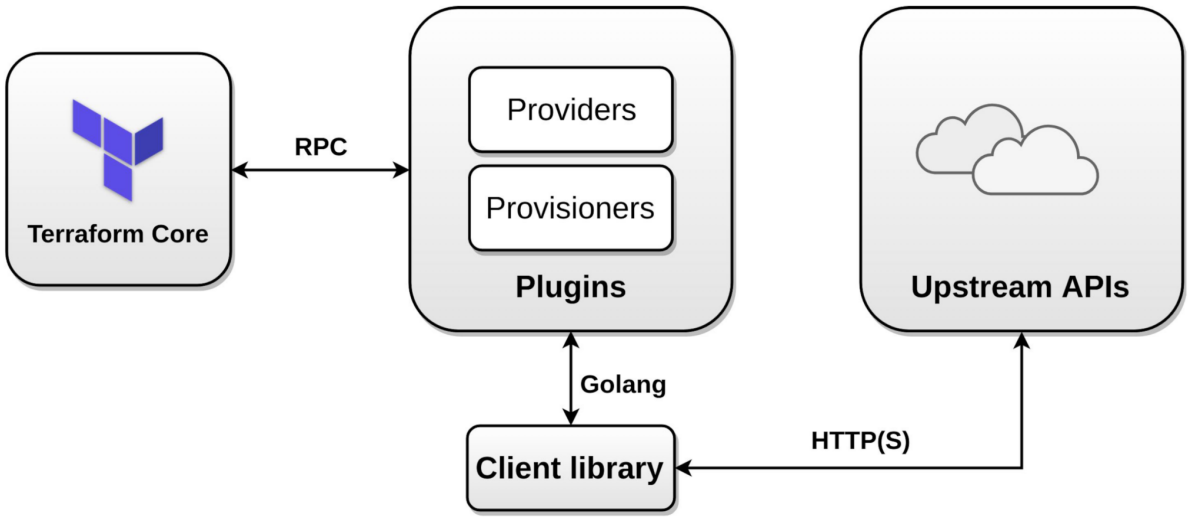
¡Descubrámoslo juntos!

Arquitectura

El núcleo de TF está compuesto por varias partes móviles que:

1. Nos proporcionan una capa de abstracción por encima de la API subyacente.
2. Son responsables de interpretar las interacciones de la API y exponer recursos.
3. Soportan múltiples instancias de proveedores cloud.

Si pudiéramos hacer una radiografía de este componente y cómo maneja su flujo de datos, podríamos ver algo similar a lo siguiente:



Vamos a darle un vistazo a cada uno de estos elementos y, luego, a tratar de comprender cómo se comunica con nuestro proveedor de cloud.

Plugins

Es una aplicación complementaria, generalmente pequeña, que sirve para agregar una funcionalidad extra o adicional (muy específica) a algo ya existente. Los plugins que se utilizan están divididos en: Providers y Provisioners.

Providers

Un provider es un plugin “específico” permitirá que nuestro proveedor cloud pueda comprender el idioma en el cual le vamos a hablar. Por ejemplo, para decirle que queremos disponer de un servidor nuevo.
El uso del término “específico” refiere a que existen varios providers, por ejemplo: un provider para AWS, otro para GCP, para Azure, Kubernetes, etc. Aquí podemos ver la lista completa.

¿Cómo se comunica con la Nube?

Del lado de la nube, existe una API especialmente diseñada para saber interpretar los comandos provenientes desde nuestra computadora. Es decir, está a la escucha de nuestras peticiones.
Si el "provider" no existiera, entonces no habría comunicación entre ambas partes.
Al ejecutar, por ejemplo, el comando "terraform plan", este binario va a buscar al "Provider" que le hemos definido en nuestro módulo de terraform:

```
# ================================================================
# Declaramos el Cloud Provider con el que queremos trabajar
terraform {
# Le decimos que queremos:
# a. la versión del binario de terraform mayor o igual a 0.12
      required_version = ">=0.12"
      required_providers {
      aws = {
# Especificamos desde donde queremos descargar el binario:
            source = "hashicorp/aws"
# Le decimos que solo permitirá:
# b. la versión del binario del provider 3.20.0 (con cierta restricción)
     version = "~> 3.20.0"
   }
 }
}
# ================================================================
```

Aquí podemos ver que la sentencia "required_provisioners" está seteada a "aws", es decir, no me interesa trabajar con Google o con Microsoft, sino, con AWS específicamente.
Luego, mediante la sentencia "source = "hashicorp/aws"", le decimos desde dónde vamos a descargar (algo que se produce de forma automática) este provisioner.
Siguiendo con nuestra ilustración, el término "Upstream APIs" se refiere al método que usa el protocolo HTTP para "subir" o "bajar" datos hacia o desde la fuente de origen.

¿Por qué hace uso de la terminología HTTP?

La API que AWS nos ofrece para contactarse con nuestros plugins utiliza las operaciones básicas CRUD (create, read, update, delete). Este modelo es tomado por las operaciones HTTP REST.

Provisioner

Un provisioner es un método que se escribe en el código mismo de HCL de Terraform y sirve para saltar cualquier brecha o gap que no pueda ser cubierta por los métodos estándar que Terraform ofrece. Por ejemplo: ejecutar comandos remotos en un servidor.
Nota: de igual manera, Hashicorp, empresa dueña del producto Terraform, recomienda el uso de provisioners solo en casos extremos.
Para esta tarea, existen herramientas de "Configuration Management", como por ejemplo Ansible o Puppet. Si por alguna razón no se pudiese hacer uso de estas herramientas, Terraform nos ofrece la posibilidad de utilizar este método en su código programable.
Un ejemplo del uso de provisioner sería la siguiente pieza de código HCL:

```
resource "aws_instance" "web" {
# ...
   provisioner "remote-exec" {
       inline = [
           "puppet apply",
           "consul join ${aws_instance.web.private_ip}",
               ]
   }
}
```

En esta pieza o "snippet", podemos ver al método "remote-exec" usado para ejecutar comandos remotos.

Concluyendo

Cuando nos referimos a ejecutar “terraform", generalmente hablamos de aprovisionamiento para afectar a los objetos de infraestructura reales. Recordemos de clases previas que el binario de Terraform tiene otros subcomandos para una amplia variedad de acciones administrativas: plan, apply, destroy, etc. Pero detrás de todos estos comandos, la arquitectura es la misma.