Infraestructura como Código

Celia Barral Juárez
Sofía Yiyu Qiu
04/03/2022

Introducción

La infraestructura de código surge por la necesidad de automatizar, cuando ya se había convertido este en un tema principal se llegó a la conclusión de que la automatización sin prácticas de infraestructura como código solo estimulaba el crecimiento de la caótica expansión de TI.

Han sido principalmente las infraestructuras como servicio, es decir, la nube y la virtualización y la mentalidad de DevOps los que han forzado de alguna manera la automatización como código. Concretamente, en 2006, el lanzamiento de la Nube de Computación Elástica de Amazon Web Services provocó problemas de escalamiento generalizados para muchas empresas, ya que el crecimiento de la infraestructura estaba limitado por el ciclo de compra de hardware. Por eso se empezó a trabajar con máquinas virtuales, y aunque gracias a la clonación y a las plantillas de imagen podíamos poner en marcha nuevas máquinas virtuales de manera muy rápida, el problema pasó a ser la necesidad de mantener actualizados un número de servidores que se mantenía en constante cambio y crecimiento.

Por ello, CFengine, Puppet y Chef establecieron una nueva herramienta de automatización de infraestructura que fue gratamente adoptada por las organizaciones ágiles, que en aquel momento aprovechaban al máximo la nube para desarrollar sus infraestructuras. La idea de modelar la infraestructura con código, y luego tener la capacidad de diseñar, implementar y desplegar la infraestructura de las aplicaciones con las mejores prácticas de software conocidas atrajo a los desarrolladores de software y a los administradores de la infraestructura de TI.

En resumen, la esencia de la infraestructura como código es tratar la configuración de los sistemas de la misma manera que se trata el código fuente del software. Los sistemas de gestión de código fuente, el desarrollo dirigido por pruebas (TDD), la integración continua (CI), la refactorización y otras prácticas de XP son especialmente útiles para garantizar que los cambios en la infraestructura se prueben exhaustivamente, sean idénticos, repetibles y permanezcan consistentes a lo largo del tiempo, ya que todos están configurados con el mismo código declarativo repetibles y transparentes.

Valores que aportan

- La **eficiencia:** gracias a la automatización de la infraestructura se evitan pérdidas de mucho tiempo y errores debidos a una configuración manual.
- La **capacidad de repetición:** se obtiene al usar los mismos scripts con la misma configuración.
- Los scripts pueden servir como **documentación** para la infraestructura de aplicaciones.

Un ejemplo que usa terraform:

```
resource "digitalocean_droplet" "web" {
        image   = "ubuntu-18-04-x64"
        name    = "domain.com"
        region  = "nyc1"
        size    = "s-1vcpu-1gb"
}
```

"Se está definiendo la creación de un servidor virtual en Digital Ocean, con 1 GB de RAM, usando ubuntu 18.04 como sistema operativo base, en el datacenter de Nueva York." (Perez, 2019)

- Como todo se inicializa desde los scripts, se obtiene un **lugar consistente** para comenzar todos los despliegues.
- La **independencia de la infraestructura.** Los scripts bien hechos se pueden ejecutar en cualquier nube y en múltiples nubes permitiendo a las organizaciones:
  - Evitar el bloqueo de la nube (Cloud Lock-In)
  - Aumentar la redundancia de datos.
  - Beneficiar de la tolerancia a fallos.
- Ayuda a evitar el **problema de que "Funciona en una máquina, pero en la otra, no",** incluso con las mismas configuraciones. Y en caso de querer comunicar algún error se puede tomar instantáneas de código e infraestructura completos, en lugar de enviar capturas de pantalla y descripciones textuales de errores.

Tipos de enfoque: declarativo vs imperativo

En general, hay tres enfoques de la IaC: declarativo (funcional) vs. imperativo (de procedimiento) vs. inteligente (consciente del entorno). La diferencia entre el enfoque declarativo, el imperativo y el inteligente es esencialmente "qué" vs. "cómo" vs. "por qué".

El enfoque **declarativo** define el estado deseado del sistema, incluidas las propiedades que debe tener y los recursos que el cliente necesita, mientras que la herramienta de IaC se encarga de configurarlo por el cliente. Asimismo, detalla en una lista el estado actual de los objetos de su sistema, lo que facilita el desmontaje de la infraestructura. Es decir, define el estado deseado y el sistema ejecuta lo que necesita suceder para alcanzar ese estado deseado.

En cambio, el enfoque **imperativo** define los comandos específicos para lograr la configuración deseada, los cuales se deben ejecutar en el orden correcto.

Por otro lado, el **inteligente** determina el estado deseado correcto antes de que el sistema ejecute lo que debe suceder para alcanzar un estado deseado que no afecte a las aplicaciones codependientes.

Muchas herramientas de IaC utilizan un enfoque declarativo y prepararán la infraestructura deseada de manera automática. Si realiza cambios en ese estado, la herramienta declarativa los aplicará automáticamente, mientras que la imperativa requerirá que el usuario resuelva cómo se deben aplicar.

Retos que afrontar

- Las **aplicaciones de varios niveles** hacen que el scripting sea exponencialmente más complejo. Más aún si se añaden **redes y seguridad** o los de las **máquinas virtuales**.
- Muchas herramientas aún requieren especialización. Por lo tanto, se necesita mucho esfuerzo para aprender su funcionamiento, la sintaxis de su lenguaje, etc. Y no todas las organizaciones pueden permitirse gastar tiempo y dinero en ello.
- Cuando todas las aplicaciones son PaaS o una combinación de PaaS e infraestructura, ya sea en la nube o en las instalaciones, se vuelve más difícil hacer todos los scripts que lo abarcan. Además, hay problema adicional a la hora de combinar la ejecución de ambos.

Relación con DevOps

Si no estáis familiarizados con el término infraestructura de código, pero si estáis interesados en DevOps, es algo que debéis saber. Para ello, primero, vamos a definir **DevOps** como un modo de abordar la cultura, la automatización y el diseño de las plataformas para generar mayor valor empresarial y capacidad de respuesta, mediante la prestación ágil de servicios de alta calidad. DevOps implica vincular las aplicaciones heredadas con las aplicaciones creadas en la nube y las infraestructuras más nuevas.

La IaC es una parte fundamental de la implementación de las prácticas de DevOps y de la integración y distribución continuas (CI/CD). Ya que, la automatización en general tiene como objetivo tomar el aspecto de confusión y propensión a errores de los procesos manuales y hacerlo más eficiente y productivo. Permitiendo que se creen mejores programas y aplicaciones con flexibilidad, menos tiempo de inactividad y una manera rentable para la empresa. IaC tiene como objetivo reducir la complejidad que mata la eficiencia de la configuración manual, ya que dejarían de ser los desarrolladores los que realizaran la mayor parte del trabajo de preparación, ya que solo deberían ejecutar un script y la infraestructura estaría lista para funcionar.

El entorno debe ser uniforme para poder automatizarse. La automatización de las implementaciones no funciona si el equipo de desarrollo implementa las aplicaciones o configura los entornos de una manera, y los equipos de operaciones lo hacen de otra. Con la IaC conseguimos un mismo enfoque DevOps ya que genera el mismo entorno cada vez que se utiliza, garantiza que el entorno de producción sea uniforme.

Herramientas

“En términos generales, cualquier marco o herramienta que realice cambios o configure la infraestructura de forma declarativa o imperativa basada en un enfoque programático puede considerarse IaC. “ (Wikipedia, 2022)

<u>Pautas para elegir herramientas:</u>

- Tiene que ofrecer la posibilidad de **externalizar** las definiciones que se usan para la creación y la actualización de configuraciones del sistema y tiene que estar en un formato que se pueda almacenar en **sistemas de control de versiones estándar** como Git. De esta manera, **se evita** el **encierro** en el conjunto de **herramientas de un solo proveedor** permitiendo el uso de una amplia variedad de herramientas para tratar con el código fuente.

- Debería poder validar definiciones en varios niveles de granularidad reduciendo así el acoplamiento. De esta forma, hace posible la aplicación de una variación de la pirámide de prueba como se puede ver en la ilustración 1. Debido a ello, se obtienen beneficios como la **retroalimentación rápida** y la **corrección de cambios**, y es el sustento para la **integración continua** y la **canalización de entrega continua**.

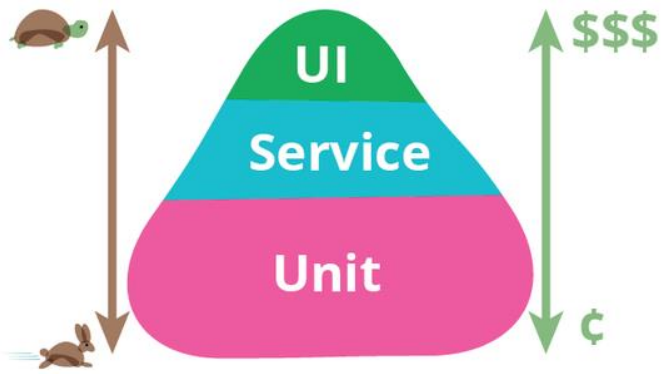

Ilustración 1. Pirámide de prueba

Referencias

Infrastructure as Code - Automation is not enough - kief.com

Conozca la infraestructura como código - DevOps.com

Infraestructura como código - Wikipedia, la enciclopedia libre

Infraestructura como código (codigofacilito.com)

TestPyramid (martinfowler.com)