Engineer

Infraestructura inmutable

❄ Sustituir servidores en vez de modificarlos

Qué es

La infraestructura inmutable es una forma de operar servidores en la que una máquina no vuelve a modificarse una vez puesta en servicio. El Well-Architected Framework de HashiCorp la describe como infraestructura que, «una vez desplegada, nunca se modifica: solo se sustituye». Una nueva versión de la aplicación, una biblioteca actualizada, un parche de seguridad y un cambio de configuración llegan de la misma manera: una máquina o un contenedor nuevos, construidos desde una imagen nueva, sustituyen a los anteriores. Después, los antiguos se apagan y eliminan.

Lo contrario es la infraestructura mutable, donde se actualiza un servidor de larga vida sin sustituirlo. En junio de 2013, Chad Fowler describió en qué se convierte tras años de actualizaciones: parcheado una y otra vez, con cambios manuales de configuración hechos durante interrupciones y tareas cron que solo una persona conoce, hasta que el equipo teme tanto cambiarlo como sustituirlo. Su remedio, al que llamó despliegues inmutables, consistía en construir cada sistema mediante automatización y no tocarlo después. Una actualización o nueva versión implica entonces crear un servidor o imagen nuevos y desechar los antiguos.

Dos tipos de imagen hacen viable el enfoque. Una imagen de máquina, según la documentación de Packer, es una unidad estática que contiene un sistema operativo preconfigurado y el software instalado. Packer, una herramienta de la comunidad, las construye: desde una configuración fuente produce imágenes equivalentes para varias plataformas. El segundo tipo son las imágenes de contenedor. La documentación de Docker es clara: una imagen creada no se puede modificar; las únicas opciones son crear otra o añadir cambios encima como capas nuevas.

El término describe cómo se opera un sistema, no un producto que se instala. También tiene un límite: los servidores de aplicaciones pueden sustituirse en cada versión, pero los datos con los que trabajan deben vivir en algún lugar que sobreviva a cada sustitución. Ese límite concentra la mayor parte del trabajo práctico de diseño.

Utilice el concepto en una decisión

Aplique «¿Qué es la infraestructura inmutable?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Toda la configuración escrita en archivos para poder recrearla con exactitud y evitar que exista solo en la memoria de una persona.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 1.000 reglas RBAC para servidores Linux — 1,000 reglas de acceso escritas para cerrar una brecha de seguridad

1.000 reglas RBAC para servidores Linux Póngase en contacto →

Por qué importa

La primera razón es la deriva. El marco de HashiCorp la nombra directamente: actualizar manualmente y modificar los servidores sin sustituirlos hace que se aparten de su configuración original con el tiempo. Dos servidores que partieron de la misma configuración con un año de diferencia no tienen por qué acabar iguales: cada uno recibió una secuencia distinta de correcciones, reintentos y cambios manuales. Si cada servidor es una copia nueva de una imagen, esta lo describe por completo y no queda nada que pueda desviarse.

La segunda razón es la reversión. Si una versión se comporta mal, el marco propone desplegar otra vez el artefacto anterior en lugar de reparar los servidores en ejecución. La imagen anterior es un estado conocido que ya funcionó en producción: bajo presión, el equipo no necesita reconstruir de memoria y con el historial del intérprete de comandos cómo era ayer el servidor.

La tercera es reducir la intervención manual en producción. El marco señala que conectarse a servidores para actualizarlos a mano introduce errores humanos, como erratas y pasos olvidados, y que, con infraestructura inmutable, nadie tiene que iniciar sesión manualmente en un servidor. Menos personas necesitan acceso a una máquina activa, y quienes lo tienen rara vez necesitan usarlo.

La cuarta es el rastro de auditoría. La misma página recuerda que los auditores quieren pruebas de que producción no se modificó fuera de procesos aprobados, mientras que los cambios manuales dejan registros incompletos. Si cada cambio

es un artefacto nuevo, el control de versiones y los registros de CI/CD muestran qué cambió, cuándo y quién lo aprobó. Un parche de seguridad sigue el mismo camino que una funcionalidad nueva: se convierte en un artefacto que sustituye al que está en ejecución y se construye, prueba y registra del mismo modo.

Nada de esto es gratuito. El marco señala que arrancar una máquina virtual tarda más y consume más recursos que arrancar un contenedor, mientras que los contenedores necesitan un orquestador. Cada cambio, incluso un ajuste de una línea, exige ahora una construcción completa y una sustitución: el proceso debe ser automático y lo bastante rápido para que nadie quiera saltárselo. El estado debe trasladarse fuera de las máquinas sustituibles antes de la primera sustitución, no después.

Cómo se aplica en la práctica

Una plantilla de Packer describe cómo construir una imagen de máquina. Esta plantilla se basa en el ejemplo de la documentación del constructor Amazon EBS de Packer y añade un bloque `build` con la forma mostrada en la referencia de ese bloque:

```
source "amazon-ebs" "web" {
  access_key    = var.aws_access_key
  secret_key    = var.aws_secret_key
  region        = "us-east-1"
  source_ami    = "ami-fce3c696"
  instance_type = "t2.micro"
  ssh_username  = "ubuntu"
  ami_name      = "web {{timestamp}}"
}

build {
  sources = ["source.amazon-ebs.web"]

  provisioner "shell" {
    scripts = fileset(".", "scripts/{install,secure}.sh")
  }
}
```

El bloque `source` nombra el constructor y la imagen base, `source_ami`, cuyo volumen raíz se copia y aprovisiona en una instancia en ejecución. El bloque `build` indica qué fuentes construir y cómo aprovisionarlas. El aprovisionador shell ejecuta los dos scripts dentro de esa máquina antes de convertirla en un

artefacto estático: una imagen nueva. Aquí se omiten las declaraciones de las dos variables. Un comando ejecuta todas las construcciones de la plantilla y produce los artefactos:

```
packer build web.pkr.hcl
```

El despliegue es la segunda mitad. El marco de HashiCorp coloca ambas herramientas juntas: Packer construye las imágenes, Terraform las despliega y se sustituyen por completo cuando necesitan actualizarse. Si no se puede realizar un cambio sobre el recurso existente, el comportamiento predeterminado de Terraform, según su referencia de ciclo de vida, es destruir el objeto y crear después su sustituto. Las actualizaciones de este proceso no requieren iniciar sesión en un servidor de producción en funcionamiento. Un cambio en los scripts produce una imagen nueva; esta produce máquinas nuevas y las antiguas se retiran.

La misma disciplina sirve para un solo host. La entrada sobre ejecutar contenedores con Podman y Quadlet separa aprovisionamiento y despliegue. El usuario root prepara la ruta de despliegue una sola vez; después, las versiones las publica un usuario normal, sin privilegios especiales, que solo puede ejecutar un script concreto y nada más. Una versión consiste en una imagen nueva seguida del reinicio del servicio, sin intérprete de comandos de root ni comandos arbitrarios. La entrada sobre CI/CD con GitHub Actions va más lejos con las imágenes: el frontend de producción usa una base distroless sin intérprete de comandos ni gestor de paquetes, y los flujos de trabajo, no una persona, promueven una construcción entre entornos. Una imagen así deja muy poco en lo que iniciar sesión y ninguna herramienta con la que instalar nada: construir su sucesora es la única manera sensata de cambiarla.

La entrada sobre servidores que puede reconstruir desde cero es un ejemplo cuidadoso del otro modelo. Ansible lleva un host activo al estado escrito en el repositorio, cada play es idempotente y el repositorio proporciona la configuración para reconstruir el entorno declarado. La entrada explica abiertamente el coste de modificar un host sin sustituirlo: cualquier cambio tarda más que una edición directa en el servidor, y una ejecución detenida a medias deja el host peor que otra que se niega a arrancar. Por eso se ejecuta antes un play de comprobaciones previas. Un host modificado a medias se acerca al servidor frágil descrito por Fowler; sustituir la máquina en lugar de modificarla es la otra forma de evitar ese riesgo.

Dónde surgen los problemas

El modelo es fácil de expresar y de erosionar. La mayoría de los fallos proceden de una excepción que parecía inocua.

Una corrección urgente por SSH devuelve la deriva. Bajo presión, alguien entra en un servidor activo y edita un archivo. Funciona hasta que el siguiente despliegue sustituye la máquina: desaparece la corrección y cualquier registro de por qué hacía falta. Hasta entonces, el servidor ya no coincide con su imagen, precisamente lo que se quería evitar. Pase todos los cambios, también los urgentes, por la construcción de la imagen y hágala lo bastante rápida para ser la vía más veloz. Las recomendaciones de HashiCorp descartan por completo conectarse a un servidor activo para editarlo o actualizarlo manualmente.

Los datos escritos en la máquina se eliminan con ella. La documentación de Docker dice que, al eliminar un contenedor, también se eliminan los cambios de archivos hechos dentro. Los archivos subidos o generados y las bases de datos locales de una máquina sustituible desaparecen en la siguiente versión. Mantenga el estado fuera de la parte sustituida. Los volúmenes de Docker conservan datos más allá del ciclo de vida de un contenedor: uno nuevo conectado al mismo volumen encuentra los archivos. En máquinas, cumple ese papel un disco separado o un servicio de base de datos ajeno a la imagen.

La sustitución deja el servicio fuera de línea. Terraform destruye de forma predeterminada el objeto existente antes de crear el sustituto, dejando un intervalo sin nada en ejecución. El ajuste `create_before_destroy` del bloque `lifecycle` de un recurso invierte el orden:

```
resource "aws_instance" "web" {
  # ...
  lifecycle {
    create_before_destroy = true
  }
}
```

HashiCorp exige activarlo expresamente porque dos copias de un objeto no siempre pueden coexistir: muchos tipos exigen un nombre único o están sujetos a otras limitaciones. Su referencia pide conocer las limitaciones de cada tipo de recurso antes de confiar en el ajuste.

Una reconstrucción parte sin avisar de una base más reciente. En vez de un `source_ami` fijo, el constructor Amazon EBS puede obtenerlo desde `source_ami_filter`. Con `most_recent` en true, selecciona la imagen coincidente más reciente, opción que la documentación considera especialmente útil para una construcción diaria de una distribución. Mantenerse al día resulta cómodo, pero dos construcciones de la misma plantilla en días distintos parten de bases diferentes: reconstruir una versión antigua ya no la reproduce. Para imágenes de versiones, fije `source_ami` a una imagen concreta y actualice deliberadamente esa referencia mediante un commit revisado.

La imagen incorpora secretos. El marco de HashiCorp insiste en que los artefactos nunca deben contener secretos, como contraseñas de bases de datos, claves API o claves de cifrado. Quien pueda leer el artefacto puede extraer esos secretos; cambiar una credencial obliga entonces a reconstruir y volver a desplegar todos los artefactos. Obtenga los secretos al arrancar la máquina o el contenedor. El patrón de HashiCorp usa Vault: la máquina se autentica al arrancar y recibe credenciales temporales que caducan; las credenciales se rotan en Vault sin reconstruir nada. La plantilla anterior sigue la misma regla con sus propias claves, recibidas como variables y no como texto en el archivo.

El artefacto anterior ha desaparecido cuando hace falta. Revertir consiste en volver a desplegar el anterior, lo que presupone que aún existe. Una limpieza que borre imágenes antiguas, o reutilizar el mismo nombre en todas las construcciones, convierte la reversión en reconstruir desde código antiguo sobre una base que quizá cambió. Conserve las imágenes de versiones recientes que funcionaron en producción, dé a cada una un nombre que nunca se reutilice y ensaye la reversión a una de ellas antes de necesitarla.

Lecturas adicionales

- Crear infraestructura inmutable — Well-Architected Framework sobre deriva, reversión, tipos de imágenes y secretos.
- Deseche sus servidores y queme su código: infraestructura inmutable y componentes desechables — Artículo de Chad Fowler de junio de 2013, consultado en una copia de Internet Archive.
- Introducción a Packer — Qué construye Packer y qué contiene una imagen de máquina.
- Constructor Amazon EBS — Constructor del ejemplo, con `source_ami` y `source_ami_filter`.
- Referencia del metaargumento lifecycle — Orden predeterminado de Terraform, destruir y crear, y `create_before_destroy`.

- Conservar datos de contenedores — Por qué los archivos escritos dentro desaparecen con el contenedor y cómo los conservan los volúmenes.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es la infraestructura inmutable?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/immutable-infrastructure/