Engineer

Infraestructura como código

Gestionar infraestructura definida en archivos versionados

Qué es

La infraestructura como código consiste en describir aquello sobre lo que funciona un sistema —máquinas, redes, reglas de cortafuegos, equilibradores de carga y registros DNS— en archivos de texto, mantenerlos bajo control de versiones y dejar que una herramienta construya el entorno real desde ellos. Los archivos son la copia de referencia. Un cambio de infraestructura empieza en un archivo, pasa la misma revisión que el código de la aplicación y solo llega al sistema activo cuando lo aplica la herramienta. El término inglés también se escribe con guiones, infrastructure-as-code, y se abrevia IaC.

La publicación especial 800-204C del NIST, guía de DevSecOps para aplicaciones de microservicios, describe así la infraestructura como código: código que aprovisiona y configura los recursos donde se despliega una aplicación, «de forma repetible y coherente». Indica también que esas definiciones se mantienen bajo control de versiones con las mismas herramientas que el código fuente de la aplicación. La repetibilidad es el objetivo: un entorno producido desde archivos puede volver a producirse, junto al primero, después de un desastre o en una cuenta de pruebas un martes por la tarde.

Las definiciones tienen dos estilos principales. Una declarativa expresa el resultado final, por ejemplo un servidor de cierto tamaño en cierta red con un puerto abierto; deja que la herramienta decida qué crear, cambiar o no tocar. Una imperativa enumera pasos: crear esto, conectar aquello y abrir el puerto. SP 800-204C clasifica la infraestructura como código dentro del código declarativo: el equipo escribe sus requisitos y una herramienta los convierte en una instancia activa. HashiCorp describe el lenguaje de Terraform del mismo modo: un objetivo previsto, no los pasos para alcanzarlo. La diferencia aparece en la segunda ejecución. Los pasos registran cómo se hizo algo una vez; una declaración, qué debe existir. Eso necesita saber la herramienta cuando parte ya existe.

No hay un lenguaje único. SP 800-204C señala que, según la herramienta, el código usa un lenguaje de scripts general como Python, JavaScript o TypeScript, o uno de configuración propio, como HCL de HashiCorp, que puede ser compatible o no con un formato estándar como JSON. El lenguaje de Terraform es del segundo tipo y usa archivos terminados en `.tf`. Azure acepta plantillas Resource Manager en JSON o en el lenguaje más breve Bicep. Ansible escribe playbooks en YAML y accede a las máquinas por SSH con las credenciales existentes de sus sistemas operativos, sin instalar agentes.

Conviene separar dos tareas incluidas en el término. El aprovisionamiento crea las piezas: máquina virtual, disco, red y cortafuegos. La configuración determina lo que funciona dentro de una máquina existente: paquetes, cuentas y archivos de servicio. Algunas herramientas se orientan más a una tarea y otras a la otra; el límite es difuso. Ambas son infraestructura como código si la definición está en un repositorio y la herramienta ajusta el sistema real a ella.

También difieren en cómo recuerdan lo construido. Terraform conserva un archivo de estado que relaciona los recursos de la configuración con los objetos reales creados; su documentación describe ese estado como fuente de verdad del entorno. Bicep no conserva tal archivo porque Azure registra lo desplegado. Dónde se guarda el registro determina qué ocurre si dos personas ejecutan la herramienta a la vez o si se pierde.

Utilice el concepto en una decisión

Aplique «¿Qué es la infraestructura como código?» 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

Antes de la infraestructura como código, cada entorno se preparaba y parcheaba a mano. Preproducción recibía una corrección que no llegaba a producción, un puerto abierto para una prueba nunca se cerraba y una de tres máquinas recibía un paquete más reciente. SP 800-204C llama a esto deriva: la configuración existente ya no produce el estado operativo previsto. Nombra también su

entrada habitual: un cambio en una consola web o en la línea de comandos que evita el código. La deriva es una causa habitual de que una versión funcione en preproducción y falle en producción.

La infraestructura como código saca la copia de referencia del sistema activo. El equipo cambia la definición y el sistema la sigue. Nadie inicia sesión para cambiar una regla: cambia el archivo que describe el cortafuegos y la herramienta lo aplica. De ello se derivan tres cosas.

La primera es revisión e historial. Un cambio de red pasa a ser una diferencia en una pull request que otra persona lee antes de que ocurra nada. Meses después, quién abrió un puerto y por qué tiene respuesta en el historial de commits, no en la memoria de alguien. Deshacer un cambio consiste en revertirlo y ejecutar de nuevo la herramienta.

La segunda es la repetibilidad. Una definición bien escrita es idempotente: no cambia nada en un entorno que ya coincide y corrige otro que se ha desviado. Hay dos caminos: ajustar lo existente o desecharlo y construirlo de nuevo desde la definición. Adoptar el segundo como regla es infraestructura inmutable.

El tercer resultado son los entornos de bajo coste. SP 800-204C describe herramientas que automatizan toda la vida de la infraestructura, desde aprovisionar hasta retirar recursos; parte de ella dura tan poco que ni siquiera compensa mantenerla. Un equipo puede crear varios entornos de prueba a demanda y eliminarlos después. Puede comprobarse el propio código antes de que toque nada: la guía pide analizar cada fragmento en busca de vulnerabilidades antes de fusionarlo. Una rama puede tener su copia efímera de la disposición de producción; quien revisa puede ver una vista previa del cambio antes de aprobarlo.

Los costes merecen expresarse con claridad. Cambiar una regla de cortafuegos mediante pull request es más lento que en una consola web y, durante una interrupción, pesa. Hay que aprender el lenguaje y modelo de estado de cada herramienta. Un archivo capaz de construir un entorno también puede borrarlo: la palanca llega igual de lejos en ambos sentidos. Por eso la vista previa importa tanto como la ejecución.

Cómo se aplica en la práctica

Este cambio pasa de la edición al sistema activo en una configuración donde una tarea de CI ejecuta la herramienta y publica la simulación en la pull request:

```
Día 1   09:10  Editar la definición de red: permitir TCP 443 desde la oficina
        09:12  Abrir la pull request (PR); CI simula la ejecución (el plan)
        09:14  Plan publicado en la PR: una regla añadida; ninguna eliminada
        10:30  Un revisor lee el plan y la diferencia de código, y lo aprueba
        10:31  Fusión; CI aplica el plan revisado
        10:33  Segunda ejecución de la misma definición: ningún cambio
Día 40         Alguien pregunta por qué está abierto 443; el historial de
               commits dice quién y por qué
```

El plan es la parte que justifica su utilidad. La diferencia de código muestra qué pretendía cambiar quien lo escribió; el plan, qué hará la herramienta, incluido lo que quien lo escribió no esperaba, como un recurso que prevé destruir. Terraform puede guardar un plan en un archivo con `-out` ; `terraform apply` , al recibirlo, realiza sus operaciones. Así, lo ejecutado a las 10:31 es lo aprobado a las 10:30. La segunda ejecución sin cambios a las 10:33 comprueba que definición y sistema ya coinciden.

En este sitio, la página del servicio de infraestructura como código describe el aprovisionamiento repetible de servidores a partir de una configuración versionada, en lugar de depender de la memoria de alguien.

Llegar con seguridad es, sobre todo, cuestión de orden, como muestra una entrada de septiembre de 2026. Añade un segundo servidor alojado en otro proveedor. El anterior se preparó a mano y se protegió después: durante un tiempo, un host recién creado con política de contraseñas predeterminada quedó abierto a Internet. Esta vez se creó desde el repositorio y la protección existía antes que la máquina. El play del cortafuegos se ejecuta antes que el del servidor y, en el segundo proveedor, las etiquetas se asignan antes que el cortafuegos cloud: las reglas dirigidas a etiquetas no tienen dónde asociarse hasta que estas existan. Reejecutar no causa daño porque el rol de creación adopta una máquina existente. Es el único rol que cuesta dinero: su documentación registra región, tipo de máquina, características y precio mensual en euros. La entrada también explica qué omite una simulación. Ambos cortafuegos se configuran llamando a las API REST de los proveedores desde

una tarea HTTP genérica que no admite el modo de comprobación: la simulación la omite. El ensayo real ejecuta solo el play del cortafuegos, sin coste y de forma reversible.

Dónde surgen los problemas

El código de infraestructura suele fallar con apariencia de éxito. La herramienta termina sin errores, CI aparece en verde y los archivos discrepan del sistema real sin avisar.

La siguiente ejecución deshace una corrección manual hecha durante un incidente. Alguien abre un puerto en la consola a las dos de la mañana, acaba la interrupción y, una semana después, la ejecución rutinaria lo cierra: los archivos no sabían del cambio. También ocurre lo contrario: nadie vuelve a ejecutar la herramienta y los archivos dejan de describir la realidad. Incorpore la corrección de emergencia al código ese mismo día y programe una comprobación entre archivos y sistema activo. SP 800-204C pide compararlos justo después de cada despliegue y a intervalos durante la ejecución. En Terraform, el código de salida detallado de plan permite una tarea nocturna breve:

```
terraform plan -detailed-exitcode
# exit 0: no changes, exit 1: error, exit 2: changes present
```

En una rama ya aplicada, el código de salida 2 significa que el sistema real se ha apartado del código; alguien debería averiguar por qué.

Eliminar un recurso del archivo no siempre lo elimina. Las herramientas difieren al interpretar una ausencia. Para plantillas Azure Resource Manager, la documentación what-if de Microsoft indica que un recurso existente pero ausente de la plantilla solo se elimina en modo completo. En cualquier otro modo sigue funcionando y costando dinero. Aprenda la regla de eliminación antes de confiar en ella y busque en la vista previa tanto las eliminaciones previstas como las temidas.

La simulación no ve todo. Una vista previa modela el cambio, no lo garantiza. La documentación what-if enumera propiedades que se notifican como eliminadas cuando Azure solo asigna un valor predeterminado durante el despliegue y lo llama ruido; toda propiedad establecida mediante `reference` aparece como cambiante en cada ejecución. La entrada anterior muestra otro hueco: una tarea

sin compatibilidad con el modo de comprobación se omite, por lo que la vista previa no dice nada de ella. Infórmese sobre las fuentes de ruido conocidas de su herramienta y considere sin probar todo lo que no pueda ver hasta ensayarlo en algo barato y reversible.

Los secretos acaban en el repositorio o en el estado. Una contraseña escrita en una definición permanece en el historial de cada clon. Menos evidente es la advertencia de Terraform: el estado puede contener secretos y, de forma predeterminada, es un archivo local llamado `terraform.tfstate`. Guarde los secretos en un almacén y refiérase a ellos por nombre; mantenga el estado en almacenamiento remoto con bloqueo y control de acceso, como recomienda HashiCorp. Nunca edite el archivo de estado a mano: para los cambios necesarios existe `terraform state`.

Un script se disfraza de definición. Un script shell alojado en el repositorio que crea un servidor está versionado, pero no es idempotente: ejecutarlo dos veces da dos servidores o un error a mitad. Use recursos declarativos o pasos que comprueben antes de actuar, como el rol anterior que adopta una máquina existente. La documentación de Ansible establece la norma con claridad: si un sistema ya está en el estado descrito por un playbook, ejecutarlo de nuevo no cambia nada.

Lecturas adicionales

- Implementación de DevSecOps para una aplicación de microservicios con malla de servicios (NIST SP 800-204C) — Infraestructura como código declarativo y versionado, deriva y análisis de vulnerabilidades.
- Documentación del lenguaje Terraform — Lenguaje declarativo, recursos y bloques.
- Comando plan — Simulación, planes guardados y códigos de salida detallados.
- Estado — Qué registra Terraform, dónde lo guarda y por qué es sensible.
- Introducción a Ansible — Automatización sin agentes por SSH, playbooks y estado deseado.
- What-if de despliegues de plantillas — Vista previa de despliegues Azure, tipos de cambios y ruido conocido.

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 como código?» 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/infrastructure-as-code/