Engineer

Terraform

Herramienta IaC de HashiCorp que planifica y aplica cambios

Qué es

Terraform es la herramienta de infraestructura como código de HashiCorp. Los recursos de la nube y de las instalaciones propias se describen en archivos de texto sin formato que se mantienen bajo control de versiones, y Terraform determina qué objetos reales debe crear, modificar o destruir para que la infraestructura en funcionamiento coincida con esos archivos. El programa se invoca como `terraform`, en minúsculas, que es como aparece su nombre en todos los comandos y scripts.

El lenguaje es declarativo. La documentación de HashiCorp lo describe como una forma de «describir un objetivo previsto en lugar de los pasos para alcanzarlo». Una configuración indica que existe una máquina de cierto tamaño en una red determinada; no enumera las llamadas a la API que permitirían crearla. Los archivos llevan la extensión `.tf`, y también existe una variante basada en JSON con la extensión `.tf.json`. La sintaxis nativa se define a partir de una sintaxis llamada HCL y tiene dos elementos principales: el bloque, que contiene otro contenido, y el argumento, que asigna un valor a un nombre.

El resto de la herramienta se apoya en tres ideas.

Los proveedores son complementos. La configuración indica a Terraform qué complementos debe instalar, y `terraform init` los busca, descarga e instala. HashiCorp afirma que los proveedores permiten a Terraform trabajar con prácticamente cualquier plataforma o servicio que tenga una API accesible, y los disponibles públicamente figuran en Terraform Registry. Un proveedor define los tipos de recurso de su plataforma, de modo que el mismo programa central puede gestionar una zona DNS, una máquina virtual y una regla de monitorización de tres proveedores comerciales distintos.

Los módulos son directorios. Un módulo es un conjunto de archivos `.tf` y `.tf.json` que se guardan juntos en un directorio, y el módulo raíz es el directorio de trabajo desde el que se invoca Terraform.

El estado es la memoria. Terraform lleva un registro de la infraestructura real en un archivo de estado que su introducción denomina la fuente de verdad del entorno. El estado relaciona cada recurso de la configuración con el objeto real que gestiona y sus metadatos. Por defecto es un archivo local llamado `terraform.tfstate`, y la versión anterior se conserva a su lado como `terraform.tfstate.backup`.

La licencia ha cambiado. El 10 de agosto de 2023, HashiCorp anunció que sustituiría Mozilla Public License 2.0 por Business Source License 1.1 como licencia de su código fuente para todas las versiones futuras de sus productos, mientras que sus API, SDK y casi todas las demás bibliotecas seguirían bajo MPL 2.0. OpenTofu se describe como una bifurcación de Terraform iniciada por Gruntwork, Spacelift, Harness, Env0, Scalr y otros, y desarrollada bajo la tutela de Linux Foundation. Su repositorio se publica bajo MPL 2.0, y su sitio lo presenta como un sustituto de Terraform que conserva los flujos de trabajo y las configuraciones existentes.

Utilice el concepto en una decisión

Aplique «¿Qué es Terraform?» 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

HashiCorp describe el flujo de trabajo central en tres pasos. Escribir: se definen los recursos, que pueden abarcar varios proveedores y servicios. Planificar: Terraform determina qué crearía, modificaría o destruiría y lo presenta como un plan de ejecución. Aplicar: una vez aprobado el plan, ejecuta esos cambios en el orden que imponen las dependencias entre los recursos, que deduce de un grafo de recursos.

El paso intermedio concentra la mayor parte del valor. Un plan es una vista previa que alguien que no escribió el cambio puede leer antes de que ocurra nada. Quien lo revisa ve más que las diferencias entre archivos: ve las consecuencias, dos recursos creados, uno actualizado y, de vez en cuando, uno destruido que nadie pretendía tocar. Detectar esa última línea en una solicitud de cambios cuesta poco; descubrirla después puede costar mucho.

La segunda razón es el alcance. Como el conocimiento de cada plataforma reside en su proveedor, un lenguaje, un flujo de trabajo y una misma práctica de revisión cubren todos los proveedores que utiliza un equipo. Una empresa que utiliza dos nubes y aloja el DNS por separado puede describir los tres servicios en un repositorio y revisarlos de la misma manera. Bicep, de Microsoft, hace una elección distinta en su flujo de trabajo con Azure Resource Manager: no mantiene un archivo de estado propio, porque Azure guarda el estado del despliegue. Eso no convierte Bicep en un lenguaje exclusivo de Azure: sus extensiones también admiten recursos ajenos a Resource Manager, como Kubernetes y Microsoft Graph.

La tercera razón es el estado. Como Terraform registra qué objeto real corresponde a cada bloque, puede distinguir un objeto que gestiona de otro que nunca ha visto, y comparar el registro con lo que está en funcionamiento. Ese registro también es el precio de la herramienta. Debe estar en un lugar accesible para todo el equipo, necesita un bloqueo que impida que dos ejecuciones escriban a la vez y puede contener secretos, por lo que requiere un control de acceso tan estricto como el de estos.

También hay costes menores. HCL es otro lenguaje que aprender, y cada proveedor tiene sus propios tipos de recurso y peculiaridades. Los equipos que construyen un producto sobre Terraform también tienen una licencia que leer: las preguntas frecuentes de HashiCorp sobre su licencia dedican una pregunta a qué se considera una oferta competidora. OpenTofu ofrece el mismo flujo de trabajo bajo MPL 2.0.

Cómo se aplica en la práctica

La página de HashiCorp sobre la sintaxis de los recursos utiliza este bloque como ejemplo:

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

`resource` es el tipo de bloque y `aws_instance` es el tipo de recurso, definido por el proveedor de AWS. `"web"` es el nombre local: Terraform utiliza esa etiqueta para seguir el recurso en su archivo de estado, y no influye en la configuración de la máquina real. Las dos líneas interiores del bloque son argumentos.

Tres comandos llevan ese archivo desde el texto hasta una máquina en funcionamiento:

```
terraform init
terraform plan -out=tfplan
terraform apply tfplan
```

`terraform init` prepara el directorio de trabajo: instala los complementos de proveedor que solicita la configuración y configura dónde se conservará el estado. Se puede ejecutar varias veces sin riesgo, y su documentación indica que el archivo de bloqueo de dependencias debe incorporarse al control de versiones.

`terraform plan -out=tfplan` calcula el plan y lo escribe en un archivo de formato opaco. La planificación no aplica los cambios de infraestructura propuestos. Después, `terraform apply tfplan` realiza las operaciones de ese archivo sin pedir confirmación, porque HashiCorp considera que proporcionar un plan guardado constituye la aprobación. Por la misma razón, la opción `-auto-approve` se ignora cuando se proporciona un archivo de plan.

En un equipo, la separación resulta natural. Una tarea de CI genera el plan para la solicitud de cambios y conserva el archivo, una persona lee tanto el plan como las diferencias y, después de la integración, una segunda tarea aplica ese mismo archivo, de modo que se ejecuta lo que se leyó. La documentación de apply añade una condición fácil de olvidar: hay que asegurarse de que nadie pueda cambiar la infraestructura fuera del flujo de Terraform, o el plan describirá un mundo que ya ha cambiado.

En este sitio, la página de tecnologías sitúa Terraform junto a Ansible: ambos dejan por escrito cómo deben ser las máquinas, para que un entorno pueda leerse en lugar de reconstruirse. Un caso del portfolio aplica la idea con otra herramienta: se aprovisionó infraestructura de Azure como código en Bicep. Los entornos de pruebas, preproducción y producto se construyen a partir del mismo conjunto de plantillas y, como estas aceptan parámetros, añadir o cambiar un entorno se resuelve con una solicitud de cambios. Junto a este artículo, el caso muestra la elección descrita antes. Bicep no tiene un archivo de estado propio porque Azure mantiene el registro, mientras que Terraform conserva su propio estado y, a cambio, trabaja con más de una nube.

Dónde surgen los problemas

La mayoría de los accidentes con Terraform siguen el mismo patrón: la configuración es válida, la ejecución termina correctamente y algo real ha desaparecido o ha cambiado.

Cambiar el nombre de un recurso lo destruye. Si se cambia la etiqueta `"web"` por `"frontend"`, el recurso pasa a tener una dirección nueva. La guía de refactorización de HashiCorp explica que, por defecto, Terraform interpreta un cambio de dirección como una instrucción para destruir el objeto existente y crear otro. En una máquina desechable cuesta unos minutos; en una base de datos cuesta los datos. La solución es un bloque `moved`, que actualiza la dirección sin destruir nada:

```
moved {
  from = aws_instance.web
  to   = aws_instance.frontend
}
```

Conserve el bloque después de que haya cumplido su función. HashiCorp recomienda encarecidamente mantener los bloques `moved` históricos y señala que eliminar uno es un cambio incompatible.

Dos personas aplican cambios con dos copias del estado local. De entrada, el estado de cada espacio de trabajo es un archivo local. Eso sirve para una persona con un portátil. En cuanto otra persona ejecuta Terraform desde otra copia, los dos archivos de estado discrepan sobre lo que existe y cada ejecución actúa según su propia versión. La solución es conservar el estado en HCP Terraform o

en almacenamiento remoto con bloqueo y control de acceso, como recomienda HashiCorp. Terraform bloquea entonces el estado automáticamente para toda operación que pueda escribir en él y no continúa si no puede obtener el bloqueo. No todas las opciones de almacenamiento admiten bloqueos, así que compruébelo antes de elegir y no utilice `-lock=false` : la opción existe, pero la documentación desaconseja su uso.

Se fuerza la liberación de un bloqueo mientras alguien sigue trabajando. Cuando falla la liberación automática, el bloqueo permanece, y el comando `force-unlock` lo libera con el identificador de bloqueo que mostró Terraform. La documentación sobre bloqueos pide mucha cautela: forzar la liberación de un bloqueo que mantiene otra persona puede dejar dos procesos escribiendo en el mismo estado, y el comando está pensado únicamente para liberar el bloqueo propio después de que haya fallado su liberación automática. Antes de forzar nada, hay que averiguar de quién es el bloqueo y si esa ejecución sigue activa.

El archivo de estado filtra secretos. HashiCorp advierte de que los secretos pueden acabar en el estado y de que guardarlo en un sistema sin un control de acceso seguro puede exponerlos. Un archivo de estado en un bucket de lectura pública, o copiado a un artefacto de construcción, supone una filtración de credenciales. Manténgalo fuera del repositorio, aplíquele el mismo control de acceso que a los secretos que contiene y nunca lo edite a mano; el comando `terraform state` existe para realizar las correcciones necesarias.

La aprobación se convierte en un trámite. La opción `-auto-approve` omite la aprobación interactiva del plan, y proporcionar un archivo de plan guardado también la omite, porque el archivo cuenta como aprobación. La documentación de apply advierte de que esto incluye cambios destructivos, entre ellos la eliminación de recursos. Una tarea de CI que aplica en cada integración sin un plan revisado ha eliminado el único momento en que una persona podría haber previsto la destrucción. La solución es el flujo con un plan guardado descrito antes, con revisión humana obligatoria entre la tarea que genera el plan y la que lo aplica. Cuando se utiliza `-auto-approve` en automatizaciones, el consejo de HashiCorp es generar primero un plan y revisarlo.

Lecturas adicionales

- ¿Qué es Terraform? — introducción de HashiCorp: escribir, planificar, aplicar, proveedores y estado.

- Comando terraform apply — planes guardados, `-auto-approve` y ejecución del comando en automatizaciones.
- Bloqueo del estado — bloqueos automáticos, almacenamiento no compatible y advertencia sobre `force-unlock` .
- Refactorizar módulos — por qué un cambio de dirección significa destruir y crear, y el bloque `moved` .
- HashiCorp adopta Business Source License — anuncio de agosto de 2023 del cambio de MPL 2.0 a BSL 1.1.
- OpenTofu — bifurcación de Terraform mantenida bajo Linux Foundation.

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 Terraform?» 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/terraform/