Engineer

GitOps

operar infraestructura desde un estado deseado en Git

Qué es

GitOps es una forma de operar sistemas de software en la que el estado deseado de un sistema se escribe como declaraciones en Git y unos agentes de software mantienen el sistema en ejecución conforme a ellas. Nadie cambia el entorno de producción a mano ni con un script desde un portátil. Un cambio es un commit: un agente dentro del entorno detecta el nuevo commit y hace que el sistema coincida con él. El proyecto OpenGitOps expresa la idea en cuatro principios, cuya versión 1.0.0 publicó el 8 de octubre de 2021 el GitOps Working Group, que supervisa el proyecto.

Los principios dicen que el estado deseado debe ser declarativo, versionado e inmutable, recuperado automáticamente y conciliado continuamente. Declarativo significa que los archivos describen cómo debe ser el sistema, no las órdenes para llegar a él. Versionado e inmutable significa que una versión guardada nunca se edita, solo se sustituye por otra, y se conserva todo el historial. Recuperado automáticamente significa que los agentes obtienen por sí mismos las declaraciones desde la fuente. Conciliado continuamente significa que comparan repetidamente el sistema en ejecución con su declaración e intentan hacerlos coincidir; el glosario del proyecto añade que continuo significa que el trabajo sigue ocurriendo, no que sea instantáneo.

Es fácil pasar por alto dos límites de ese glosario. Git, del que viene el nombre, es el almacén de estado canónico, pero sirve cualquiera que conserve versiones inmutables y ofrezca control de acceso y auditoría de los cambios al estado deseado. Además, ese estado es la configuración necesaria para recrear el sistema, lo que normalmente excluye datos persistentes como el contenido de una base de datos. GitOps puede reconstruir las máquinas donde se ejecuta una base, pero no las filas que contiene.

Utilice el concepto en una decisión

Aplique «¿Qué es GitOps?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cambios probados, publicados y reversibles, para que lleguen a sus clientes un martes por la tarde en vez de un viernes por la noche.

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. Despliegue GIS SaaS con GitHub Actions — despliegue, datos e informes automatizados

Despliegue GIS SaaS con GitHub Actions Póngase en contacto →

Por qué importa

La diferencia respecto a una cadena de despliegue normal es qué inicia el trabajo. El glosario de OpenGitOps contrapone trabajos CI/CD iniciados por desencadenantes previamente configurados y GitOps, donde la conciliación se activa siempre que el estado real se aparta del declarado. Esa divergencia puede venir de cualquiera de los lados: un commit nuevo cambia lo que se quiere o el sistema en ejecución se aleja de lo declarado. Una cadena que ejecutó `kubectl apply` una vez no ve lo que ocurrió después. La propia documentación de Kubernetes señala que la aplicación declarativa conserva las escrituras realizadas directamente en objetos activos sin incorporarlas de nuevo a los archivos de configuración, de modo que archivos y clúster pueden discrepar sin que se note. Un agente que concilia detecta ambos tipos de divergencia.

La segunda ventaja es dónde reside el acceso. La documentación de Argo CD señala que, con sincronización automática, una cadena de despliegue puede trabajar sin credenciales para el servidor de la API de Argo CD: crea un commit y lo envía. La cadena necesita acceso de escritura a un repositorio en vez de al clúster, y en el repositorio ya se revisan los cambios.

La tercera es el historial. Como el estado deseado está versionado y es inmutable, el registro del repositorio recoge cada cambio previsto: qué cambió, quién y cuándo. Volver atrás significa elegir una versión anterior, y Argo CD incluye entre sus funciones la reversión a cualquier configuración de aplicación confirmada en Git. Tras un incidente, ese historial responde a la primera pregunta, qué cambió, sin que nadie lo reconstruya a partir del historial del shell y de la memoria.

Cómo se aplica en la práctica

Flux y Argo CD son dos agentes que llevan esto a la práctica en Kubernetes. Argo CD se describe como una herramienta declarativa de entrega continua GitOps para Kubernetes; Flux llama GitOps Toolkit a su conjunto de controladores y API. Pensemos en un equipo que guarda sus manifiestos de producción en un repositorio llamado fleet. Dos objetos de Flux conectan el clúster a él: el primero indica al controlador de fuentes dónde está el repositorio y cada cuánto recuperarlo; el segundo indica al controlador kustomize qué directorio aplicar y cada cuánto comprobar el clúster frente a él.

```
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: fleet
  namespace: flux-system
spec:
  interval: 5m
  url: https://git.example.com/platform/fleet.git
  ref:
    branch: main
---
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: production
  namespace: flux-system
spec:
  interval: 10m
  sourceRef:
    kind: GitRepository
    name: fleet
  path: "./production"
  prune: true
  timeout: 1m
```

Sigamos un cambio. Un ingeniero fusiona un pull request que pasa un servicio a una nueva etiqueta de imagen. El controlador de fuentes recupera la rama cada cinco minutos y en la siguiente recuperación ve una revisión nueva. Una revisión de fuente nueva se procesa de inmediato, sin esperar al siguiente intervalo, por lo que el controlador kustomize construye los manifiestos de `./production` y los aplica. La documentación de Flux describe el resto para ese par exacto de objetos. Cada diez minutos, según el intervalo, `Kustomization` utiliza una simulación de la aplicación de cambios del lado del servidor para descubrir divergencias y después las concilia, de modo que un colega que cambie el mismo

objeto con `kubectl edit` verá deshecho su cambio. Como `prune` está activado, un objeto cuyo manifiesto se haya quitado del directorio se elimina del clúster con la siguiente revisión. Revertir es hacer un `git revert` y una fusión, por la misma ruta que cualquier otro cambio.

Argo CD hace el mismo trabajo mediante un recurso `Application`, con valores predeterminados distintos. Una página de su documentación dice que comprueba los repositorios cada tres minutos; otra, sobre sincronización automática, sitúa el intervalo en 120 segundos más hasta 60 de variación aleatoria, con un máximo de tres minutos. Cuando el estado activo difiere del objetivo en Git, la aplicación queda marcada como OutOfSync y, con la sincronización automática activada, Argo CD la sincroniza. De forma predeterminada, sin embargo, esa sincronización no elimina lo que haya desaparecido de Git ni reacciona a cambios hechos directamente en el clúster. La eliminación de sobrantes y la autocorrección son interruptores separados, `prune` y `selfHeal`, ambos desactivados al principio.

El mismo patrón funciona fuera de Kubernetes. La conciliación de una zona DNS de 20 registros mediante la API del proveedor, realizada en agosto de 2026, convirtió los registros que vivían en la consola web de un proveedor en una declaración en el repositorio. Un punto de entrada concilia la declaración con el proveedor; otro, tras una activación explícita, solo audita y comunica la diferencia sin cambiar nada. La página señala que la auditoría es la parte más usada, porque suele interesar más conocer la diferencia que eliminarla. Al contrastarlo con los cuatro principios, eso cubre los dos primeros: la zona está declarada y versionada. La página no describe un agente que recupere la declaración y la aplique por sí solo, de modo que no se atribuyen los dos últimos. También es un argumento razonable para empezar donde Argo CD empieza de forma predeterminada: mostrar la desviación antes de permitir que algo la corrija.

Dónde surgen los problemas

Secretos en texto claro en Git. Un repositorio que contiene todo el estado deseado tienta a guardar también las contraseñas, y su historial las conserva después de borrarlas. La documentación de Flux califica de inseguro guardar secretos de Kubernetes en Git como texto claro o base64, incluso en un repositorio privado con acceso restringido. Cífrelos antes de confirmarlos. Flux descifra con SOPS mediante age, OpenPGP o un servicio de gestión de claves en la nube, configurado mediante `.spec.decryption`; cifre solo los campos `data` y `stringData`, porque `metadata`, `kind` y `apiVersion` deben seguir legibles. La documentación de Argo CD recomienda con firmeza otra vía: resolver los

secretos en el clúster de destino con una herramienta como Sealed Secrets, External Secrets Operator o Secrets Store CSI Driver, en vez de inyectarlos mientras Argo CD genera manifiestos. Desaconseja esto último, en parte porque su caché Redis guarda una copia sin cifrar de los manifiestos resultantes.

Una corrección urgente que no permanece o una desviación que nunca se va. Las dos herramientas se comportan de forma distinta con la configuración predeterminada. Con Flux, un cambio hecho con `kubectl edit`, `patch` o `delete` se revierte enseguida: lo correcto hasta que alguien aplica a mano una corrección urgente durante una caída y ve que desaparece. Con Argo CD, un cambio directo en el clúster no activa una sincronización automática salvo que esté activada la autocorrección, de modo que la desviación puede quedarse indefinidamente, marcada como OutOfSync. La solución es la misma para ambos: hacer el cambio en Git o pausar abiertamente la conciliación. `.spec.suspend` de Flux impide aplicar revisiones nuevas y pausa la corrección de desviaciones hasta que su valor vuelva a false. Para un campo que pertenece a otro controlador, como el número de réplicas de un `Deployment` gestionado por un escalador automático, `.spec.ignore` de Flux acepta rutas JSON Pointer como `/spec/replicas`, para que los controladores dejen de sobrescribirse. Revierta también a través de Git: Argo CD rechaza una reversión en una aplicación con sincronización automática activada.

Borrado en Git, aún en ejecución. Quitar un manifiesto elimina la declaración, no el objeto. La sincronización automática de Argo CD no elimina recursos que ya no se definen en Git salvo que se active la eliminación de sobrantes, y Flux solo los elimina cuando `.spec.prune`, un campo que todo `Kustomization` debe establecer, es true. Déjelo desactivado y el clúster se llenará poco a poco de objetos que nadie declara. Actívelo y marque los pocos objetos que deban sobrevivir a su manifiesto con la etiqueta o anotación `kustomize.toolkit.fluxcd.io/prune: disabled`. Argo CD añade una protección que conviene conservar: la sincronización automática con eliminación de sobrantes no deja una aplicación sin recurso alguno salvo que se establezca `allowEmpty`, lo que impide que un commit defectuoso que vacíe un directorio vacíe el clúster.

Una fusión que espera al siguiente sondeo. Un agente que recupera según un temporizador cambia rapidez por independencia: Argo CD da tres minutos como periodo de sondeo, y Flux ve un commit en la siguiente recuperación de la fuente. Ambos aceptan un webhook del servidor Git para acortar la espera. Argo CD lo recibe en `/api/webhook`; Flux utiliza un recurso `Receiver` que convierte un evento push en una conciliación de las fuentes que enumera. El secreto compartido del webhook de Argo CD es opcional, porque una carga útil solo

activa una actualización, pero la documentación lo recomienda si Argo CD es accesible públicamente para evitar ataques distribuidos de denegación de servicio. Flux genera una ruta única de webhook con el nombre del receptor, su espacio de nombres y un token secreto y, para GitHub, verifica la firma HMAC de la carga útil.

Git pasa a ser una dependencia de producción. Si el agente recupera del servidor Git, una caída de Git detiene todos los cambios, incluidas las correcciones. La variante de Flux basada en OCI, denominada Gitless GitOps, mantiene Git como interfaz para las personas, pero hace que el clúster recupere artefactos versionados de un registro de contenedores, permitiendo obtener los artefactos de producción sin contactar con el servidor Git. Sea cual sea la fuente que lea el agente, dele la disponibilidad que espera de producción, porque durante una caída es la única ruta por la que puede pasar un cambio.

Lecturas adicionales

- OpenGitOps — Los cuatro principios de GitOps, versión 1.0.0, del GitOps Working Group.
- Glosario de GitOps v1.0.0 — Las definiciones tras los principios: estado deseado, desviación, conciliación y almacén de estado.
- Política de sincronización automática — La sincronización automática de Argo CD, con los interruptores de eliminación de sobrantes, autocorrección y aplicación vacía, y cuándo se ejecuta cada sincronización.
- Kustomization — La API de Flux para aplicar un directorio de manifiestos: intervalo, eliminación de sobrantes, suspensión, exclusiones y descifrado SOPS.
- Gestión de secretos — Los argumentos de Argo CD para resolver secretos en el clúster de destino en lugar de durante la generación de manifiestos.
- Gestión declarativa de objetos de Kubernetes mediante archivos de configuración — Cómo funcionan `kubectl apply` y `kubectl diff` sobre un directorio de archivos de configuración y qué hace apply con los cambios en objetos activos.



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 GitOps?» 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/gitops/