Engineer

GitOps

Betrieb von Infrastruktur nach einem Sollzustand in Git

Was es ist

GitOps ist eine Betriebsweise für Softwaresysteme: Der gewünschte Zustand eines Systems wird als Deklaration in Git festgehalten. Softwareagenten halten das laufende System damit in Einklang. Niemand ändert die Produktion von Hand oder über ein Skript auf einem Laptop. Eine Änderung ist ein Commit. Ein Agent innerhalb der Umgebung bemerkt den neuen Commit und bringt das System in den dazu passenden Zustand. Das Projekt OpenGitOps formuliert die Idee in vier Prinzipien. Deren Version 1.0.0 wurde am 8. Oktober 2021 von der GitOps Working Group veröffentlicht, die das Projekt betreut.

Nach diesen Prinzipien muss der Sollzustand deklarativ, versioniert und unveränderlich sein, automatisch abgerufen und fortlaufend abgeglichen werden. Deklarativ bedeutet: Die Dateien beschreiben, wie das System aussehen soll, nicht die Befehle, die es dorthin bringen. Versioniert und unveränderlich bedeutet: Eine gespeicherte Version wird nie bearbeitet, nur abgelöst; die vollständige Historie bleibt erhalten. Automatisch abgerufen bedeutet: Softwareagenten holen die Deklarationen selbst von der Quelle. Fortlaufend abgeglichen bedeutet: Agenten vergleichen das laufende System wiederholt mit seiner Deklaration und arbeiten daran, beide in Einklang zu bringen. Das Projektglossar ergänzt: Fortlaufend heißt, dass die Arbeit immer wieder stattfindet, nicht dass sie augenblicklich geschieht.

Zwei Abgrenzungen in diesem Glossar lassen sich leicht übersehen. Git, von dem der Name stammt, ist der kanonische Zustandsspeicher. Es eignet sich aber jeder Speicher, der unveränderliche Versionen hält sowie Zugriffskontrolle und die Nachvollziehbarkeit von Änderungen am Sollzustand bietet. Und der Sollzustand umfasst die Konfiguration, die zum Neuerstellen des Systems nötig ist. Dauerhaft gespeicherte Daten wie Datenbankinhalte sind im Allgemeinen ausgenommen. GitOps kann die Rechner einer Datenbank neu aufbauen, aber nicht die darin gespeicherten Zeilen.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist GitOps?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Geprüft, veröffentlicht und rückgängig zu machen: Änderungen erreichen Ihre Kunden am Dienstagnachmittag statt spät am Freitagabend.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. GIS-SaaS-Deployments mit GitHub automatisiert — Bereitstellung, Daten und Berichte ohne Handarbeit

GIS-SaaS-Deployments mit GitHub automatisiert Kontakt aufnehmen →

Warum es wichtig ist

Der Unterschied zu einer gewöhnlichen Bereitstellungspipeline liegt darin, was die Arbeit auslöst. Das OpenGitOps-Glossar stellt CI/CD-Jobs mit zuvor konfigurierten Auslösern GitOps gegenüber, bei dem der Abgleich ausgelöst wird, sobald der Istzustand vom deklarierten abweicht. Die Abweichung kann von beiden Seiten kommen: Ein neuer Commit ändert das Gewünschte, oder das laufende System entfernt sich vom deklarierten Zustand. Eine Pipeline, die einmal `kubectl apply` ausgeführt hat, sieht nicht, was danach geschieht. Die eigene Kubernetes-Dokumentation hält fest, dass deklaratives Apply direkt an laufenden Objekten vorgenommene Änderungen beibehält, ohne sie in die Konfigurationsdateien zurückzuführen. Dateien und Cluster können deshalb unbemerkt auseinanderlaufen. Ein abgleichender Agent erkennt beide Arten von Abweichungen.

Der zweite Vorteil betrifft den Ort der Zugriffsrechte. Argo CDs Dokumentation weist darauf hin, dass eine Bereitstellungspipeline bei automatischer Synchronisierung ohne Zugangsdaten für Argo CDs API-Server arbeiten kann: Die Pipeline erstellt einen Commit und pusht ihn. Sie braucht Schreibzugriff auf ein Repository statt auf den Cluster. Im Repository werden Änderungen ohnehin geprüft.

Der dritte Vorteil ist die Historie. Weil der Sollzustand versioniert und unveränderlich ist, hält das Repository-Log jede beabsichtigte Änderung fest: was sich geändert hat, wer es geändert hat und wann. Zurückzugehen bedeutet, eine frühere Version auszuwählen. Argo CD nennt den Rollback auf jede in Git committete Anwendungskonfiguration als Funktion. Nach einem Vorfall beantwortet diese Historie die erste Frage: Was hat sich geändert? Niemand muss es aus Shell-Verläufen und Erinnerungen rekonstruieren.

Wie es in der Praxis aussieht

Flux und Argo CD sind zwei Agenten, die dies auf Kubernetes umsetzen. Argo CD beschreibt sich als deklaratives GitOps-Werkzeug für Continuous Delivery auf Kubernetes. Flux nennt seine Controller und APIs zusammen GitOps Toolkit. Nehmen wir ein Team, das seine Produktionsmanifeste in einem Repository namens fleet hält. Zwei Flux-Objekte verbinden den Cluster damit: Das erste sagt dem Source-Controller von Flux, wo das Repository liegt und wie oft er es abrufen soll. Das zweite sagt dem Kustomize-Controller, welches Verzeichnis er anwenden und wie oft er den Cluster dagegen prüfen soll.

```
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
```

Verfolgen wir nun eine Änderung. Ein Entwickler führt einen Pull Request zusammen, der einen Dienst auf ein neues Image-Tag umstellt. Der Source-Controller ruft den Branch alle fünf Minuten ab und sieht beim nächsten Abruf eine neue Revision. Eine neue Quellrevision wird sofort behandelt, statt bis zum nächsten Intervall zu warten. Der Kustomize-Controller baut daher die Manifeste unter `./production` und wendet sie an. Die Flux-Dokumentation beschreibt den Rest genau für dieses Objektpaar. Alle zehn Minuten führt die `Kustomization` entsprechend dem Intervall einen Probelauf von Server-Side Apply aus, um Abweichungen zu finden, und gleicht sie dann ab. Ein Kollege, der dasselbe

Objekt mit `kubectl edit` ändert, sieht seine Änderung daher rückgängig gemacht. Weil `prune` aktiv ist, wird ein Objekt, dessen Manifest aus dem Verzeichnis entfernt wurde, mit der nächsten Revision aus dem Cluster gelöscht. Ein Rollback ist ein `git revert` mit anschließendem Merge und nimmt denselben Weg wie jede andere Änderung.

Argo CD erledigt dieselbe Aufgabe über eine Ressource vom Typ `Application`, mit anderen Vorgaben. Eine Dokumentationsseite sagt, dass es Repositorys alle drei Minuten abfragt. Eine andere, zur automatischen Synchronisierung, nennt 120 Sekunden plus bis zu 60 Sekunden zufällige Verzögerung, insgesamt höchstens drei Minuten. Weicht der laufende Zustand vom Zielzustand in Git ab, wird die Anwendung als OutOfSync markiert. Ist automatische Synchronisierung aktiv, synchronisiert Argo CD sie. Standardmäßig löscht diese Synchronisierung aber nichts, was aus Git verschwunden ist, und reagiert nicht auf direkte Änderungen im Cluster. Bereinigung und Selbstheilung haben eigene Schalter, `prune` und `selfHeal`. Beide sind anfangs abgeschaltet.

Dasselbe Muster funktioniert außerhalb von Kubernetes. Eine Zone mit 20 DNS-Einträgen, die gegen die API des Anbieters abgeglichen wird, umgesetzt im August 2026, überführte Einträge aus der Webkonsole eines Anbieters in eine Deklaration im Repository. Ein Einstiegspunkt gleicht die Deklaration mit dem Anbieter ab. Ein anderer, der eine ausdrückliche Aktivierung erfordert, prüft nur und meldet Unterschiede, ohne etwas zu ändern. Die Seite nennt die Prüfung als den am häufigsten genutzten Teil: Meist geht es eher darum, den Unterschied zu kennen, als ihn zu beseitigen. An den vier Prinzipien gemessen erfüllt das die ersten beiden: Die Zone ist deklariert und versioniert. Die Seite beschreibt keinen Agenten, der die Deklaration selbstständig abruft und anwendet. Die letzten beiden werden daher nicht beansprucht. Es ist auch ein nachvollziehbares Argument, dort anzufangen, wo Argo CD standardmäßig anfängt: erst die Abweichung zeigen, bevor irgendetwas sie korrigieren darf.

Wo es schiefgeht

Geheimnisse im Klartext in Git. Ein Repository mit dem gesamten Sollzustand verführt dazu, auch Passwörter dort abzulegen. Seine Historie behält sie nach dem Löschen. Laut Flux-Dokumentation ist das Speichern von Kubernetes-Secrets als Klartext oder base64 in Git unsicher, selbst in einem privaten Repository mit eingeschränktem Zugriff. Verschlüsseln Sie sie vor dem Commit. Flux entschlüsselt mit SOPS über age, OpenPGP oder einen Schlüsselverwaltungsdienst eines Cloudanbieters, konfiguriert durch `.spec.decryption`. Verschlüsseln Sie nur `data` und `stringData`, denn `metadata`,

`kind` und `apiVersion` müssen lesbar bleiben. Argo CDs Dokumentation empfiehlt nachdrücklich einen anderen Weg: Geheimnisse im Zielcluster mit einem Werkzeug wie Sealed Secrets, External Secrets Operator oder Secrets Store CSI Driver auflösen, statt sie während Argo CDs Manifestgenerierung einzuspeisen. Davor warnt sie unter anderem, weil der Redis-Cache eine unverschlüsselte Kopie der erzeugten Manifeste hält.

Ein Hotfix, der nicht bleibt, oder eine Abweichung, die nicht verschwindet. Die beiden Werkzeuge verhalten sich standardmäßig unterschiedlich. Unter Flux wird eine mit `kubectl edit`, `patch` oder `delete` vorgenommene Änderung umgehend zurückgenommen. Das ist richtig, bis jemand während eines Ausfalls von Hand eine dringende Korrektur vornimmt und sie verschwinden sieht. Bei Argo CD löst eine direkte Änderung im Cluster ohne aktive Selbstheilung keine automatische Synchronisierung aus. Eine Abweichung kann daher unbegrenzt als OutOfSync markiert bestehen bleiben. Die Lösung ist bei beiden dieselbe: Nehmen Sie die Änderung in Git vor oder setzen Sie den Abgleich ausdrücklich aus. Flux' `.spec.suspend` stoppt das Anwenden neuer Revisionen und pausiert die Korrektur von Abweichungen, bis es wieder auf false gesetzt wird. Für ein von einem anderen Controller verwaltetes Feld, etwa die Anzahl der Replikate eines vom Autoscaler betreuten `Deployment`, nimmt Flux' `.spec.ignore` JSON-Pointer-Pfade wie `/spec/replicas` entgegen. So überschreiben sich die beiden Controller nicht mehr gegenseitig. Nehmen Sie auch einen Rollback über Git vor: Argo CD verweigert ihn bei einer Anwendung mit aktiver automatischer Synchronisierung.

In Git gelöscht, weiterhin in Betrieb. Das Entfernen eines Manifests entfernt die Deklaration, nicht das Objekt. Ressourcen, die nicht mehr in Git definiert sind, löscht Argo CDs automatische Synchronisierung nur bei aktiver Bereinigung. Flux löscht sie nur, wenn `.spec.prune` auf true steht, ein Feld, das jede `Kustomization` setzen muss. Bleibt es abgeschaltet, füllt sich der Cluster langsam mit von niemandem deklarierten Objekten. Schalten Sie es ein und markieren Sie die wenigen Objekte, die ihr Manifest überleben müssen, mit dem Label oder der Annotation `kustomize.toolkit.fluxcd.io/prune: disabled`. Argo CD ergänzt eine bewahrenswerte Schutzmaßnahme: Automatische Synchronisierung mit Bereinigung lässt eine Anwendung nur dann ohne jegliche Ressourcen zurück, wenn `allowEmpty` gesetzt ist. Ein fehlerhafter Commit, der ein Verzeichnis leert, leert dadurch nicht den Cluster.

Ein Merge wartet auf die nächste Abfrage. Ein Agent, der nach Zeitplan abruft, tauscht Geschwindigkeit gegen Unabhängigkeit. Argo CD nennt drei Minuten als Abfrageintervall, und Flux sieht einen Commit beim nächsten Abruf der Quelle. Beide nehmen einen Webhook des Git-Servers an, um die Wartezeit zu verkürzen.

Argo CD empfängt ihn am Endpunkt `/api/webhook`. Flux nutzt eine Ressource vom Typ `Receiver`, die ein Push-Ereignis in den Abgleich der darin aufgelisteten Quellen umsetzt. Argo CDs gemeinsames Webhook-Geheimnis ist optional, weil die Nutzdaten nur eine Aktualisierung auslösen. Seine Dokumentation empfiehlt aber eines, wenn Argo CD öffentlich erreichbar ist, um verteilte Denial-of-Service-Angriffe zu verhindern. Flux erzeugt einen eindeutigen Webhook-Pfad aus Name, Namensraum und geheimem Token des Receivers. Bei GitHub prüft es die HMAC-Signatur der Nutzdaten.

Git wird zur Produktionsabhängigkeit. Holt der Agent vom Git-Server ab, stoppt ein Git-Ausfall jede Änderung, auch Korrekturen. Die OCI-basierte Flux-Variante namens Gitless GitOps behält Git als Schnittstelle für Menschen bei, lässt den Cluster aber versionierte Artefakte aus einer Container-Registry abrufen. Der Cluster kann seine Produktionsartefakte dadurch ohne Kontakt zum Git-Server beziehen. Welche Quelle der Agent auch liest: Geben Sie ihr die Verfügbarkeit, die Sie von der Produktion erwarten. Während eines Ausfalls ist sie der einzige Weg für eine Änderung.

Weiterführende Lektüre

- OpenGitOps — die vier GitOps-Prinzipien, Version 1.0.0, von der GitOps Working Group.
- GitOps-Glossar v1.0.0 — die Definitionen hinter den Prinzipien: Sollzustand, Drift, Abgleich und Zustandsspeicher.
- Richtlinie für automatische Synchronisierung — Argo CDs automatische Synchronisierung mit Schaltern für Bereinigung, Selbstheilung und leere Anwendungen sowie den Zeitpunkten jedes Abgleichs.
- Kustomization — Flux' API zum Anwenden eines Verzeichnisses mit Manifesten: Intervall, Bereinigung, Pausieren, Ignorieren und SOPS-Entschlüsselung.
- Verwaltung von Geheimnissen — Argo CDs Argumente für das Auflösen von Geheimnissen im Zielcluster statt während der Manifestgenerierung.
- Deklarative Verwaltung von Kubernetes-Objekten mit Konfigurationsdateien — wie `kubectl apply` und `kubectl diff` auf einem Verzeichnis mit Konfigurationsdateien arbeiten und was Apply mit Änderungen an laufenden Objekten macht.



Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist GitOps?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/gitops/