Engineer

GitOps

⑂ att driva infrastruktur utifrån ett önskat tillstånd i Git

Vad det är

GitOps är ett sätt att driva mjukvarusystem där det önskade tillståndet för ett system är nedskrivet som deklarationer i Git, och där mjukvaruagenter håller det körande systemet i överensstämmelse med dem. Namnet används oöversatt på svenska. Ingen ändrar produktionen för hand eller från ett skript på en bärbar dator. En ändring är en commit, och en agent inne i miljön upptäcker den nya commiten och får systemet att stämma överens med den. Projektet OpenGitOps formulerar idén som fyra principer, och version 1.0.0 av principerna publicerades den 8 oktober 2021 av GitOps Working Group, som har tillsyn över projektet.

Principerna säger att det önskade tillståndet ska vara deklarativt, versionerat och oföränderligt, hämtas automatiskt och stämmas av kontinuerligt. Deklarativt betyder att filerna beskriver hur systemet ska se ut, inte kommandona som leder dit. Versionerat och oföränderligt betyder att en sparad version aldrig redigeras, bara ersätts, och att hela historiken bevaras. Hämtas automatiskt betyder att mjukvaruagenterna själva hämtar deklarationerna från källan. Stäms av kontinuerligt betyder att agenterna hela tiden observerar det faktiska tillståndet och försöker tillämpa det önskade; projektets ordlista tillägger att kontinuerligt betyder att arbetet pågår hela tiden, inte att det sker omedelbart.

Två avgränsningar i ordlistan är lätta att missa. Git, som namnet kommer från, är det kanoniska stället att förvara tillståndet på, men vilket system som helst duger om det bevarar oföränderliga versioner och ger åtkomstkontroll och granskning av ändringar i det önskade tillståndet. Och det önskade tillståndet är den konfiguration som behövs för att återskapa systemet, vilket i regel utesluter beständiga data som innehållet i en databas. GitOps kan återskapa maskinerna som en databas körs på, men inte raderna i den.

Använd begreppet i ett beslut

Använd ”Vad är GitOps?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Testade, släppta och möjliga att backa, så att ändringar når kunderna en tisdagseftermiddag i stället för sent en fredagskväll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Automatiserade GIS-driftsättning i GitHub Actions — driftsättning, data och rapportering, helt utan manuellt arbete

Automatiserade GIS-driftsättning i GitHub Actions Hör av er →

Varför det spelar roll

Skillnaden mot en vanlig pipeline för driftsättning är vad som sätter igång arbetet. Ordlistan från OpenGitOps ställer traditionell CI/CD, där automatiseringen i regel drivs av förinställda utlösare, mot GitOps, där avstämningen sätts igång så snart det faktiska tillståndet avviker från det deklarerade. Avvikelsen kan komma från båda håll: en ny commit ändrar vad som önskas, eller så glider det körande systemet bort från det som deklarerats. En pipeline som körde `kubectl apply` en gång vet ingenting om vad som hände sedan. Kubernetes dokumentation påpekar att deklarativ apply behåller ändringar som skrivits direkt till levande objekt utan att föra in dem i konfigurationsfilerna igen, så att filerna och klustret i tysthet kan vara oense. En agent som stämmer av upptäcker båda typerna av avvikelser.

Den andra vinsten är var åtkomsten finns. Dokumentationen för Argo CD påpekar att CI/CD-pipelines med automatisk synkronisering inte längre behöver direkt åtkomst till API-servern i Argo CD för att driftsätta: pipelinen gör en commit och pushar den. Det pipelinen behöver är skrivrättighet till ett repo i stället för till klustret, och repot är ändå där ändringar granskas.

Den tredje är historiken. Eftersom det önskade tillståndet är versionerat och oföränderligt, är loggen i repot en förteckning över varje avsedd ändring: vad som ändrades, vem som ändrade det och när. Att gå tillbaka betyder att välja en tidigare version, och Argo CD listar bland sina funktioner återställning till vilken applikationskonfiguration som helst som har committats i Git. Efter en incident besvarar historiken den första frågan, alltså vad som ändrades, utan att någon behöver pussla ihop det ur skalhistorik och minne.

Så ser det ut i praktiken

Flux och Argo CD är två agenter som omsätter detta i praktiken på Kubernetes. Argo CD beskriver sig som ett deklarativt GitOps-verktyg för continuous delivery på Kubernetes, och Flux kallar sin samling av controllers och API:er för GitOps Toolkit. Ta ett team som har sina produktionsmanifest i ett repo som heter fleet. Två Flux-objekt kopplar klustret till det: det första talar om för source controller i Flux var repot finns och hur ofta det ska hämtas, och det andra talar om för kustomize controller vilken katalog som ska tillämpas och hur ofta klustret ska kontrolleras mot den.

```
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
```

Följ nu en ändring. En ingenjör mergar en pull request som flyttar en tjänst till en ny image-tagg. Source controller hämtar grenen var femte minut och ser en ny revision vid nästa hämtning. En ny revision från källan hanteras genast i stället för vid nästa intervall, så kustomize controller bygger manifesten under `./production` och tillämpar dem. Flux dokumentation beskriver resten för just det här paret av objekt. Var tionde minut, som intervallet anger, använder `Kustomization` en server-side apply som dry-run för att upptäcka avvikelser och rättar dem därefter, så en kollega som ändrar samma objekt med `kubectl edit` ser ändringen återställas. Eftersom `prune` är påslaget, raderas ett objekt

vars manifest har tagits bort från katalogen ur klustret vid nästa revision. Att rulla tillbaka är en `git revert` och en merge, som går samma väg som alla andra ändringar.

Argo CD gör samma jobb genom en resurs av typen `Application`, men med andra standardvärden. En sida i dokumentationen säger att Argo CD pollar repon var tredje minut; en annan, om automatisk synkronisering, anger intervallet till 120 sekunder plus upp till 60 sekunders jitter, högst tre minuter sammanlagt. När det levande tillståndet avviker från måltillståndet i Git, markeras applikationen som OutOfSync, och med automatisk synkronisering påslagen synkroniserar Argo CD den. Som standard raderar automatisk synkronisering ändå inget som har försvunnit ur Git, och den reagerar inte på ändringar som gjorts direkt i klustret. Rensning och självläkning är separata reglage, `prune` och `selfHeal`, och båda är avslagna från början.

Samma mönster fungerar även utanför Kubernetes. En DNS-zon med 20 poster, avstämd mot leverantörens API, utförd i augusti 2026, gjorde poster som låg i en leverantörs webbkonsol till en deklaration i repot. En ingång stämmer av deklarationen mot leverantören; en annan, som måste slås på uttryckligen, gör bara en granskning och rapporterar skillnaden utan att ändra något. Sidan anger granskningen som den del som används mest, eftersom det oftare är skillnaden man vill känna till än att täppa till den. Mätt mot de fyra principerna är det de två första: zonen är deklarerad och versionerad. Sidan beskriver ingen agent som själv hämtar deklarationen och tillämpar den, så de två sista görs inget anspråk på. Det är också ett bra argument för att börja där Argo CD börjar som standard: visa avvikelsen innan något får rätta den.

Var det går fel

Hemligheter i klartext i Git. Ett repo som rymmer hela det önskade tillståndet frestar folk att lägga lösenorden där också, och historiken behåller dem efter att de raderats. Flux dokumentation kallar det osäkert att lägga Kubernetes-hemligheter i Git som klartext eller base64, oavsett hur synligt repot är och vilka åtkomstbegränsningar det har. Svaret är att kryptera dem innan de committas. Flux dekrypterar med SOPS, med age, OpenPGP eller en nyckelhanteringstjänst i molnet, konfigurerad genom `.spec.decryption`; kryptera bara fälten `data` och `stringData`, eftersom `metadata`, `kind` och `apiVersion` måste förbli läsbara. Dokumentationen för Argo CD rekommenderar starkt en annan väg: fyll i hemligheterna på målklustret med ett verktyg som Sealed Secrets, External

Secrets Operator eller Secrets Store CSI Driver i stället för att föra in dem medan Argo CD genererar manifest. Det senare avråder den från bland annat för att Argo CD sparar genererade manifest okrypterade i sin Redis-cache.

En snabbfix som inte stannar kvar, eller en avvikelse som aldrig försvinner. De två verktygen beter sig olika här som standard. Under Flux återställs en ändring som gjorts med `kubectl edit` , `patch` eller `delete` snabbt, vilket är rätt ända tills någon lägger in en akut rättning för hand mitt under ett avbrott och ser den försvinna. Under Argo CD utlöser en ändring som gjorts direkt i klustret inte en automatisk synkronisering om inte självläkning är påslagen, så avvikelsen kan ligga kvar på obestämd tid, markerad som OutOfSync. Lösningen är densamma för båda: gör ändringen i Git, eller pausa avstämningen öppet. Med `.spec.suspend` slutar Flux att tillämpa nya revisioner och pausar rättningen av avvikelser tills fältet sätts tillbaka till false. För ett fält som en annan controller äger, till exempel antalet repliker i en `Deployment` som en autoskalare styr, tar `.spec.ignore` i Flux emot JSON Pointer-sökvägar som `/spec/replicas` , så att de två controllerna slutar skriva över varandra. Rulla tillbaka via Git också: Argo CD avvisar en återställning av en applikation som har automatisk synkronisering påslagen.

Raderad i Git, men körs fortfarande. Att ta bort ett manifest tar bort deklarationen, inte objektet. Den automatiska synkroniseringen i Argo CD raderar inte resurser som inte längre är definierade i Git om inte rensning är påslagen, och Flux raderar dem bara när `.spec.prune` , ett fält som varje `Kustomization` måste ange, är true. Låter man det vara avslaget, fylls klustret sakta med objekt som ingen deklarerar. Slå på det, och märk de få objekt som ska överleva sitt eget manifest med etiketten eller annoteringen `kustomize.toolkit.fluxcd.io/prune: disabled` . Argo CD har ett skydd som är värt att behålla: automatisk synkronisering med rensning lämnar inte en applikation helt utan resurser om inte `allowEmpty` är satt, vilket hindrar att en felaktig commit som tömmer en katalog också tömmer klustret.

En merge som väntar på nästa pollning. En agent som hämtar enligt en klocka byter snabbhet mot oberoende: Argo CD anger tre minuter som pollningsintervall, och Flux ser en commit vid källans nästa hämtning. Båda tar emot en webhook från Git-servern för att korta väntan. Argo CD tar emot den på slutpunkten `/api/webhook` , medan Flux använder en resurs av typen `Receiver` som gör om en push-händelse till en avstämning av källorna den listar. Den delade webhook-hemligheten i Argo CD är valfri, eftersom en payload bara utlöser en uppdatering, men dokumentationen rekommenderar en när Argo CD

kan nås publikt, för att förhindra distribuerade överbelastningsattacker. Flux genererar en unik webhook-sökväg utifrån resursens namn och namnrymd och en hemlig token, och för GitHub verifierar den payloadens HMAC-signatur.

Git blir ett beroende i produktionen. Hämtar agenten från Git-servern, stoppar varje avbrott i Git alla ändringar, rättningar inräknade. Den OCI-baserade varianten av Flux, som projektet kallar Gitless GitOps, behåller Git som gränssnittet folk arbetar i, men låter klustret hämta versionerade artefakter från ett containerregister, så att Git-servern enligt Flux inte längre är ett beroende i produktionen. Oavsett vilken källa agenten läser från, bör den ha den tillgänglighet du förväntar dig av produktionen, för under ett avbrott är den den enda väg en ändring kan ta.

Läs mer

- OpenGitOps — de fyra GitOps-principerna i version 1.0.0 från GitOps Working Group.
- GitOps Glossary v1.0.0 — definitionerna bakom principerna: önskat tillstånd, avvikelse, avstämning och platsen där tillståndet förvaras.
- Automated Sync Policy — automatisk synkronisering i Argo CD med reglagen för rensning, självläkning och tomma applikationer, och när varje synkronisering körs.
- Kustomization — API:et i Flux för att tillämpa en katalog med manifest: intervall, prune, suspend, ignore och dekryptering med SOPS.
- Secret Management — Argo CD:s motivering för att fylla i hemligheter på målklustret i stället för under genereringen av manifest.
- Declarative Management of Kubernetes Objects Using Configuration Files — hur `kubectl apply` och `kubectl diff` fungerar på en katalog med konfigurationsfiler, och vad apply gör med ändringar i levande objekt.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är GitOps?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/gitops/