Engineer

GitOps

⑂ å drifte infrastruktur fra en ønsket tilstand lagret i Git

Hva det er

GitOps er en måte å drifte programvaresystemer på der den ønskede tilstanden til et system er skrevet ned som deklarasjoner i Git, og der programvareagenter holder det kjørende systemet i samsvar med dem. Navnet brukes uoversatt på norsk. Ingen endrer produksjon for hånd eller fra et skript på en bærbar maskin. En endring er en commit, og en agent inne i miljøet oppdager den nye commiten og får systemet til å samsvare med den. Prosjektet OpenGitOps formulerer idéen som fire prinsipper, og versjon 1.0.0 av prinsippene ble publisert 8. oktober 2021 av GitOps Working Group, som fører tilsyn med prosjektet.

Prinsippene sier at den ønskede tilstanden skal være deklarativ, versjonert og uforanderlig, hentet automatisk og kontinuerlig avstemt. Deklarativ betyr at filene beskriver hvordan systemet skal se ut, ikke kommandoene som fører dit. Versjonert og uforanderlig betyr at en lagret versjon aldri redigeres, bare erstattes, og at hele historikken tas vare på. Hentet automatisk betyr at programvareagentene selv henter deklarasjonene fra kilden. Kontinuerlig avstemt betyr at agentene hele tiden observerer den faktiske tilstanden og prøver å anvende den ønskede; prosjektets ordliste legger til at kontinuerlig betyr at arbeidet fortsetter, ikke at det skjer øyeblikkelig.

To avgrensninger i ordlisten er lette å overse. Git, som navnet kommer fra, er det kanoniske eksempelet på et lager for tilstanden, men ethvert lager kan brukes hvis det tar vare på uforanderlige versjoner og gir tilgangskontroll og revisjon av endringer i den ønskede tilstanden. Og den ønskede tilstanden er konfigurasjonen som trengs for å gjenskape systemet, noe som som regel utelukker varige data som innholdet i en database. GitOps kan bygge opp igjen maskinene en database kjører på, men ikke radene inne i den.

Bruk begrepet i en beslutning

Bruk «Hva er GitOps?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Testet, utgitt og mulig å reversere, slik at endringer når kundene en tirsdag ettermiddag i stedet for sent en fredagskveld.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Automatiserte GIS-SaaS-utrulling i GitHub Actions — produksjonssetting, data og rapportering, alt uten manuelle grep

Automatiserte GIS-SaaS-utrulling i GitHub Actions Ta kontakt →

Hvorfor det er viktig

Forskjellen fra en vanlig pipeline for produksjonssetting er hva som setter arbeidet i gang. Ordlisten fra OpenGitOps stiller tradisjonell CI/CD, der automatiseringen som regel drives av forhåndsinnstilte utløsere, opp mot GitOps, der avstemmingen settes i gang hver gang den faktiske tilstanden avviker fra den deklarerte. Avviket kan komme fra begge sider: en ny commit endrer hva som ønskes, eller det kjørende systemet glir bort fra det som er deklarert. En pipeline som kjørte `kubectl apply` én gang, vet ingenting om hva som skjedde etterpå. Dokumentasjonen til Kubernetes påpeker at deklarativ apply beholder endringer som er skrevet direkte til levende objekter, uten å flette dem tilbake i konfigurasjonsfilene, så filene og klyngen i det stille kan være uenige. En agent som avstemmer, oppdager begge slags avvik.

Den andre gevinsten er hvor tilgangen ligger. Dokumentasjonen til Argo CD påpeker at pipelines med automatisk synkronisering ikke lenger trenger direkte tilgang til API-serveren i Argo CD for å rulle ut: pipelinen lager en commit og pusher den. Det pipelinen trenger, er skrivetilgang til et repo i stedet for til klyngen, og repoet er uansett der endringer blir gjennomgått.

Den tredje er historikken. Fordi den ønskede tilstanden er versjonert og uforanderlig, er loggen i repoet en oversikt over hver tilsiktet endring: hva som ble endret, hvem som endret det, og når. Å gå tilbake betyr å velge en tidligere versjon, og Argo CD regner tilbakerulling til enhver applikasjonskonfigurasjon som er committet i Git, blant funksjonene sine. Etter en driftshendelse svarer historikken på det første spørsmålet, nemlig hva som ble endret, uten at noen må stykke det sammen fra skallhistorikk og hukommelse.

Slik ser det ut i praksis

Flux og Argo CD er to agenter som setter dette ut i praksis på Kubernetes. Argo CD beskriver seg selv som et deklarativt GitOps-verktøy for continuous delivery på Kubernetes, og Flux kaller samlingen sin av kontrollere og API-er for GitOps Toolkit. Ta et team som har produksjonsmanifestene sine i et repo som heter fleet. To Flux-objekter kobler klyngen til det: det første forteller source controlleren i Flux hvor repoet ligger og hvor ofte det skal hentes, og det andre forteller kustomize controlleren hvilken katalog som skal anvendes, og hvor ofte klyngen skal sjekkes 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
```

Følg nå én endring. En ingeniør merger en pull request som flytter en tjeneste til en ny image-tag. Source controlleren henter grenen hvert femte minutt og ser en ny revisjon ved neste henting. En ny revisjon fra kilden håndteres med en gang i stedet for ved neste intervall, så kustomize controlleren bygger manifestene under `./production` og anvender dem. Dokumentasjonen til Flux beskriver resten for nettopp dette paret av objekter. Hvert tiende minutt, slik intervallet bestemmer, bruker `Kustomization` en server-side apply som dry-run til å oppdage avvik og retter dem deretter, så en kollega som endrer det samme objektet med `kubectl edit`, ser endringen bli rullet tilbake. Fordi `prune` er slått på, blir et

objekt hvis manifest er fjernet fra katalogen, slettet fra klyngen ved neste revisjon. Å rulle tilbake er en `git revert` og en merge, som går samme vei som enhver annen endring.

Argo CD gjør den samme jobben gjennom en ressurs av typen `Application`, men med andre standardverdier. Én side i dokumentasjonen sier at Argo CD poller repoer hvert tredje minutt; en annen, om automatisk synkronisering, oppgir intervallet til 120 sekunder pluss opptil 60 sekunders jitter, høyst tre minutter til sammen. Når den levende tilstanden avviker fra måltilstanden i Git, blir applikasjonen markert som OutOfSync, og med automatisk synkronisering slått på synkroniserer Argo CD den. Som standard sletter automatisk synkronisering likevel ingenting som har forsvunnet fra Git, og reagerer ikke på endringer som er gjort direkte i klyngen. Opprydding og selvreparasjon er egne brytere, `prune` og `selfHeal`, og begge starter slått av.

Det samme mønsteret virker utenfor Kubernetes. En DNS-sone med 20 poster, avstemt mot leverandørens API, utført i august 2026, gjorde poster som lå i en leverandørs nettkonsoll, til en deklarasjon i repoet. Ett inngangspunkt avstemmer deklarasjonen mot leverandøren; et annet, som må slås på eksplisitt, gjør bare en revisjon og rapporterer forskjellen uten å endre noe. Siden nevner revisjonen som den delen som brukes mest, fordi det oftere er forskjellen man vil vite, enn at den skal lukkes. Målt mot de fire prinsippene er det de to første: sonen er deklarert og versjonert. Siden beskriver ingen agent som selv henter deklarasjonen og anvender den, så de to siste gjøres det ikke krav på. Det er også et godt argument for å begynne der Argo CD begynner som standard: vis avviket før noe får lov til å rette det.

Hvor det går galt

Hemmeligheter i klartekst i Git. Et repo som rommer hele den ønskede tilstanden, frister folk til å legge passordene der også, og historikken beholder dem etter at de er slettet. Dokumentasjonen til Flux kaller det utrygt å legge Kubernetes-hemmeligheter i Git som klartekst eller base64, uansett hvor synlig repoet er og hvilke tilgangsbegrensninger det har. Svaret er å kryptere dem før de blir committet. Flux dekrypterer med SOPS, med age, OpenPGP eller en nøkkelhåndteringstjeneste i skyen, konfigurert gjennom `.spec.decryption`; krypter bare feltene `data` og `stringData`, fordi `metadata`, `kind` og `apiVersion` må forbli lesbare. Dokumentasjonen til Argo CD anbefaler sterkt en annen vei: fyll inn hemmelighetene på målklyngen med et verktøy som Sealed

Secrets, External Secrets Operator eller Secrets Store CSI Driver i stedet for å sette dem inn mens Argo CD genererer manifester. Det siste fraråder den blant annet fordi Argo CD lagrer genererte manifester ukryptert i Redis.

En hasteretting som ikke blir værende, eller et avvik som aldri går bort. De to verktøyene oppfører seg forskjellig her som standard. Under Flux blir en endring gjort med `kubectl edit`, `patch` eller `delete` raskt rullet tilbake, noe som er riktig helt til noen legger inn en hasteretting for hånd midt i et avbrudd og ser den forsvinne. Under Argo CD utløser en endring som er gjort direkte i klyngen, ikke en automatisk synkronisering med mindre selvreparasjon er slått på, så avviket kan bli liggende på ubestemt tid, markert som OutOfSync. Løsningen er den samme for begge: gjør endringen i Git, eller sett avstemmingen synlig på pause. Med `.spec.suspend` slutter Flux å anvende nye revisjoner og setter rettingen av avvik på pause til feltet settes tilbake til false. For et felt som en annen kontroller eier, for eksempel antallet replikaer i en `Deployment` som en autoskalerer styrer, tar `.spec.ignore` i Flux imot JSON Pointer-stier som `/spec/replicas`, så de to kontrollerne slutter å overskrive hverandre. Rull også tilbake gjennom Git: Argo CD avviser en tilbakerulling på en applikasjon med automatisk synkronisering slått på.

Slettet i Git, men kjører fortsatt. Å fjerne et manifest fjerner deklarasjonen, ikke objektet. Den automatiske synkroniseringen i Argo CD sletter ikke ressurser som ikke lenger er definert i Git, med mindre opprydding er slått på, og Flux sletter dem bare når `.spec.prune`, et felt som enhver `Kustomization` må angi, er true. Lar man det være av, fylles klyngen sakte med objekter som ingen deklarerer. Slå det på, og merk de få objektene som skal overleve sitt eget manifest, med etiketten eller annotasjonen `kustomize.toolkit.fluxcd.io/prune: disabled`. Argo CD har en sperre det er verdt å beholde: automatisk synkronisering med opprydding etterlater ikke en applikasjon helt uten ressurser med mindre `allowEmpty` er satt, og det hindrer at en feilaktig commit som tømmer en katalog, også tømmer klyngen.

En merge som venter på neste polling. En agent som henter etter en klokke, bytter fart mot uavhengighet: Argo CD oppgir tre minutter som pollingperiode, og Flux ser en commit ved kildens neste henting. Begge tar imot en webhook fra Git-serveren for å korte ned ventetiden. Argo CD tar imot den på endepunktet `/api/webhook`, mens Flux bruker en ressurs av typen `Receiver` som gjør en push-hendelse om til en avstemming av kildene den lister opp. Den delte webhook-hemmeligheten i Argo CD er valgfri, fordi en payload bare utløser en oppdatering, men dokumentasjonen anbefaler en når Argo CD kan nås offentlig,

for å hindre distribuerte tjenestenektangrep. Flux genererer en unik webhook-sti ut fra navnet og navnerommet til ressursen og et hemmelig token, og for GitHub verifiserer den HMAC-signaturen til payloaden.

Git blir en avhengighet i produksjonen. Henter agenten fra Git-serveren, stopper ethvert avbrudd i Git alle endringer, rettelser inkludert. Den OCI-baserte varianten av Flux, som prosjektet kaller Gitless GitOps, beholder Git som grensesnittet folk jobber i, men lar klyngen hente versjonerte artefakter fra et container-register, slik at Git-serveren ifølge Flux ikke lenger er en avhengighet i produksjonen. Uansett hvilken kilde agenten leser fra, bør den ha den tilgjengeligheten du forventer av produksjonen, for under et avbrudd er den den eneste veien en endring kan ta.

Les mer

- OpenGitOps — de fire GitOps-prinsippene i versjon 1.0.0 fra GitOps Working Group.
- GitOps Glossary v1.0.0 — definisjonene bak prinsippene: ønsket tilstand, avvik, avstemming og lageret for tilstanden.
- Automated Sync Policy — automatisk synkronisering i Argo CD med bryterne for opprydding, selvreparasjon og tomme applikasjoner, og når hver synkronisering kjører.
- Kustomization — API-et i Flux for å anvende en katalog med manifester: intervall, prune, suspend, ignore og dekryptering med SOPS.
- Secret Management — begrunnelsen i Argo CD for å fylle inn hemmeligheter på målklyngen fremfor under genereringen av manifester.
- Declarative Management of Kubernetes Objects Using Configuration Files — hvordan `kubectl apply` og `kubectl diff` virker på en katalog med konfigurasjonsfiler, og hva apply gjør med endringer i levende objekter.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er GitOps?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/gitops/