Engineer

GitOps

⑂ at drive infrastruktur ud fra en ønsket tilstand i Git

Hvad det er

GitOps er en måde at drive softwaresystemer på, hvor den ønskede tilstand af et system er skrevet ned som erklæringer i Git, og hvor softwareagenter holder det kørende system i overensstemmelse med dem. Betegnelsen bruges uoversat på dansk. Ingen ændrer produktionen i hånden eller fra et script på en bærbar. En ændring er et commit, og en agent inde i miljøet opdager det nye commit og får systemet til at svare til det. Projektet OpenGitOps formulerer idéen som fire principper, og version 1.0.0 af principperne blev udgivet den 8. oktober 2021 af GitOps Working Group, der fører tilsyn med projektet.

Principperne siger, at den ønskede tilstand skal være deklarativ, versioneret og uforanderlig, hentet automatisk og løbende afstemt. Deklarativ betyder, at filerne beskriver, hvordan systemet skal se ud, ikke hvilke kommandoer der fører derhen. Versioneret og uforanderlig betyder, at en gemt version aldrig redigeres, kun afløses, og at hele historikken bevares. Hentet automatisk betyder, at softwareagenterne selv henter erklæringerne fra kilden. Løbende afstemt betyder, at agenterne hele tiden observerer den faktiske tilstand og forsøger at anvende den ønskede; projektets ordliste tilføjer, at løbende betyder, at arbejdet bliver ved med at ske, ikke at det sker øjeblikkeligt.

To afgrænsninger i ordlisten er lette at overse. Git, som navnet kommer af, er det kanoniske eksempel på et lager for tilstanden, men ethvert lager kan bruges, hvis det gemmer uforanderlige versioner og giver adgangskontrol og et revisionsspor for ændringer i den ønskede tilstand. Og den ønskede tilstand er den konfiguration, der skal til for at genskabe systemet, hvilket som regel udelukker vedvarende data som indholdet af en database. GitOps kan genopbygge de maskiner, en database kører på, men ikke rækkerne inde i den.

Brug begrebet i en beslutning

Brug »Hvad er GitOps?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Testet, udgivet og til at rulle tilbage, så ændringer når kunderne en tirsdag eftermiddag i stedet for sent en fredag aften.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har automatiseret GIS-SaaS-udrulning med CI/CD — udrulning, data og rapportering, alt sammen uden manuelt arbejde

Har automatiseret GIS-SaaS-udrulning med CI/CD Kontakt os →

Hvorfor det betyder noget

Forskellen fra en almindelig pipeline til udrulning er, hvad der sætter arbejdet i gang. Ordlisten fra OpenGitOps stiller traditionel CI/CD, hvor automatiseringen som regel drives af forudindstillede udløsere, op over for GitOps, hvor afstemningen sættes i gang, hver gang den faktiske tilstand afviger fra den erklærede. Afvigelsen kan komme fra begge sider: Et nyt commit ændrer, hvad der ønskes, eller det kørende system glider væk fra det erklærede. En pipeline, der kørte `kubectl apply` én gang, ved intet om, hvad der skete bagefter. Dokumentationen til Kubernetes bemærker, at deklarativ apply bevarer ændringer, der er skrevet direkte til levende objekter, uden at flette dem tilbage i konfigurationsfilerne, så filerne og clusteret i det stille kan være uenige. En agent, der afstemmer, opdager begge slags afvigelser.

Den anden gevinst er, hvor adgangen ligger. Dokumentationen til Argo CD påpeger, at pipelines med automatisk synkronisering ikke længere behøver direkte adgang til API-serveren i Argo CD for at udrulle: Pipelinen laver et commit og pusher det. Det, pipelinen har brug for, er skriveadgang til et repository i stedet for til clusteret, og repositoryet er alligevel det sted, hvor ændringer bliver gennemgået.

Den tredje er historikken. Fordi den ønskede tilstand er versioneret og uforanderlig, er repositoryets log en optegnelse over hver tilsigtet ændring: hvad der blev ændret, hvem der ændrede det, og hvornår. At gå tilbage betyder at vælge en tidligere version, og Argo CD nævner blandt sine funktioner rollback til enhver applikationskonfiguration, der er committet i Git. Efter en driftshændelse besvarer historikken det første spørgsmål, nemlig hvad der blev ændret, uden at nogen skal stykke det sammen fra shell-historik og hukommelse.

Sådan ser det ud i praksis

Flux og Argo CD er to agenter, der omsætter dette til praksis på Kubernetes. Argo CD beskriver sig selv som et deklarativt GitOps-værktøj til continuous delivery på Kubernetes, og Flux kalder sit sæt af controllere og API'er for GitOps Toolkit. Tag et team, der har sine produktionsmanifester i et repository ved navn fleet. To Flux-objekter forbinder clusteret med det: Det første fortæller source controlleren i Flux, hvor repositoryet ligger, og hvor ofte det skal hentes, og det andet fortæller kustomize controlleren, hvilken mappe der skal anvendes, og hvor ofte clusteret skal tjekkes mod 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 nu én ændring. En ingeniør merger en pull request, der flytter en tjeneste til et nyt image-tag. Source controlleren henter branchen hvert femte minut og ser ved den næste hentning en ny revision. En ny revision fra kilden håndteres med det samme i stedet for ved næste interval, så kustomize controlleren bygger manifesterne under `./production` og anvender dem. Dokumentationen til Flux beskriver resten for netop dette par af objekter. Hvert tiende minut, som intervallet bestemmer, bruger `Kustomization` en server-side apply som dry-run til at opdage afvigelser og retter dem derefter, så en kollega, der ændrer det samme objekt med `kubectl edit`, ser ændringen forsvinde igen. Fordi `prune` er

slået til, bliver et objekt, hvis manifest er fjernet fra mappen, slettet fra clusteret ved næste revision. At rulle tilbage er en `git revert` og en merge, som går samme vej som enhver anden ændring.

Argo CD gør det samme arbejde gennem en ressource af typen `Application`, men med andre standardværdier. Én side i dokumentationen siger, at Argo CD poller repositories hvert tredje minut; en anden, om automatisk synkronisering, angiver intervallet til 120 sekunder plus op til 60 sekunders jitter, højst tre minutter i alt. Når den levende tilstand afviger fra måltilstanden i Git, bliver applikationen markeret som OutOfSync, og med automatisk synkronisering slået til synkroniserer Argo CD den. Som standard sletter automatisk synkronisering dog intet, der er forsvundet fra Git, og reagerer ikke på ændringer, der er lavet direkte i clusteret. Oprydning og selvheling er separate kontakter, `prune` og `selfHeal`, og begge starter slået fra.

Det samme mønster virker uden for Kubernetes. En DNS-zone med 20 poster, afstemt mod udbyderens API, udført i august 2026, gjorde poster, der lå i en udbyders webkonsol, til en erklæring i repositoryet. Én indgang afstemmer erklæringen mod udbyderen; en anden, bag et eksplicit tilvalg, foretager kun en gennemgang og rapporterer forskellen uden at ændre noget. Siden nævner gennemgangen som den del, der bliver brugt mest, fordi det oftere er forskellen, man vil kende, end at få den lukket. Målt mod de fire principper er det de to første: Zonen er erklæret og versioneret. Siden beskriver ikke en agent, der selv henter erklæringen og anvender den, så der gøres ikke krav på de to sidste. Det er også et godt argument for at begynde der, hvor Argo CD begynder som standard: Vis afvigelsen, før noget får lov at rette den.

Hvor det går galt

Hemmeligheder i klartekst i Git. Et repository, der rummer hele den ønskede tilstand, frister folk til også at lægge adgangskoderne der, og historikken beholder dem, efter at de er slettet. Dokumentationen til Flux kalder det usikkert at gemme hemmeligheder i Git som klartekst eller base64, uanset repositoryets synlighed eller adgangsbegrænsninger. Svaret er kryptering, før noget bliver committet. Flux dekrypterer med SOPS, med age, OpenPGP eller en nøglehåndteringstjeneste i en cloud, konfigureret gennem `.spec.decryption`; kryptér kun felterne `data` og `stringData`, fordi `metadata`, `kind` og `apiVersion` skal forblive læsbare. Dokumentationen til Argo CD anbefaler kraftigt en anden vej: Læg hemmelighederne ind på destinationsclusteret med et værktøj som Sealed Secrets, External Secrets Operator eller Secrets Store CSI

Driver i stedet for at indsætte dem, mens Argo CD genererer manifester. Det sidste fraråder den blandt andet, fordi den gemmer genererede manifester i klartekst i sin cache i Redis.

Et hotfix, der ikke bliver, eller en afvigelse, der aldrig går væk. De to værktøjer opfører sig forskelligt her som standard. Under Flux bliver en ændring lavet med `kubectl edit`, `patch` eller `delete` straks rullet tilbage, hvilket er rigtigt, indtil nogen lægger en hasterettelse ind i hånden under et nedbrud og ser den forsvinde. Under Argo CD udløser en ændring, der er lavet direkte i clusteret, ikke en automatisk synkronisering, medmindre selvheling er slået til, så afvigelsen kan blive liggende på ubestemt tid, markeret som OutOfSync. Løsningen er den samme for begge: Lav ændringen i Git, eller sæt afstemningen synligt på pause. Med `.spec.suspend` stopper Flux med at anvende nye revisioner og sætter rettelsen af afvigelser på pause, indtil feltet sættes tilbage til false. For et felt, som en anden controller ejer, for eksempel antallet af replikaer i en `Deployment`, som en autoscaler styrer, tager `.spec.ignore` i Flux imod JSON Pointer-stier som `/spec/replicas`, så de to controllere holder op med at overskrive hinanden. Rul også tilbage gennem Git: Argo CD afviser en rollback på en applikation med automatisk synkronisering slået til.

Slettet i Git, men stadig kørende. At fjerne et manifest fjerner erklæringen, ikke objektet. Den automatiske synkronisering i Argo CD sletter ikke ressourcer, der ikke længere er defineret i Git, medmindre oprydning er slået til, og Flux sletter dem kun, når `.spec.prune`, et felt som enhver `Kustomization` skal angive, er true. Lader man det være slået fra, fyldes clusteret langsomt med objekter, som ingen erklærer. Slå det til, og markér de få objekter, der skal overleve deres manifest, med labelen eller annotationen `kustomize.toolkit.fluxcd.io/prune: disabled`. Argo CD har et værn, der er værd at beholde: Automatisk synkronisering med oprydning efterlader ikke en applikation helt uden ressourcer, medmindre `allowEmpty` er sat, og det forhindrer, at et fejlagtigt commit, der tømmer en mappe, også tømmer clusteret.

En merge, der venter på næste poll. En agent, der henter efter et ur, bytter hastighed for uafhængighed: Argo CD angiver tre minutter som sin polling-periode, og Flux ser et commit ved kildens næste hentning. Begge tager imod en webhook fra Git-serveren for at forkorte ventetiden. Argo CD modtager den på sit endpoint `/api/webhook`, mens Flux bruger en ressource af typen `Receiver`, der gør en push-hændelse til en afstemning af de kilder, den lister. Den delte webhook-hemmelighed i Argo CD er valgfri, fordi en payload kun udløser en opdatering, men dokumentationen anbefaler en, når Argo CD kan nås offentligt,

for at forhindre distribuerede denial-of-service-angreb. Flux genererer en unik webhook-sti ud fra receiverens navn, namespace og et hemmeligt token og verificerer for GitHub payloadens HMAC-signatur.

Git bliver en afhængighed i produktionen. Hvis agenten henter fra Git-serveren, stopper et nedbrud hos Git enhver ændring, rettelser inklusive. Den OCI-baserede variant af Flux, som projektet kalder Gitless GitOps, beholder Git som den grænseflade, folk arbejder i, men lader clusteret hente versionerede artefakter fra et container-registry, så Git-serveren ifølge Flux ikke længere er en afhængighed i produktionen. Uanset hvilken kilde agenten læser fra, bør den have den tilgængelighed, du forventer af produktionen, for under et nedbrud er den den eneste vej, en ændring kan tage.

Læs mere

- OpenGitOps — de fire GitOps-principper i version 1.0.0 fra GitOps Working Group.
- GitOps Glossary v1.0.0 — definitionerne bag principperne: ønsket tilstand, afvigelse, afstemning og lageret for tilstanden.
- Automated Sync Policy — automatisk synkronisering i Argo CD med kontakterne for oprydning, selvheling og tomme applikationer, og hvornår hver synkronisering kører.
- Kustomization — API'et i Flux til at anvende en mappe med manifester: interval, prune, suspend, ignore og dekryptering med SOPS.
- Secret Management — begrundelsen i Argo CD for at lægge hemmeligheder ind på destinationsclusteret frem for under genereringen af manifester.
- Declarative Management of Kubernetes Objects Using Configuration Files — hvordan `kubectl apply` og `kubectl diff` virker på en mappe med konfigurationsfiler, og hvad apply gør med ændringer i levende objekter.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er GitOps?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/gitops/