



// GITOPS MIT K8S IN DER PRAXIS – EIN ERFAHRUNGSBERICHT

Gerd Huber, ITZBund

Johannes Schnatterer, Cloudogu GmbH

 @jschnatterer

Version: 202011030842-ec40a00

Agenda

- Was ist GitOps?
- Anwendungsbeispiele
 - Neueinführung von GitOps (OnPrem)
 - Migration CI/CD ➡ GitOps (Public Cloud)
- Herausforderungen in der Praxis
- Fazit und Empfehlung

Was ist GitOps

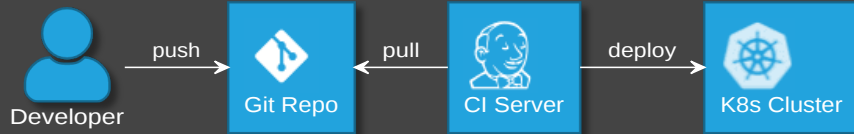
- Begriff (August 2017):

use developer tooling to drive operations

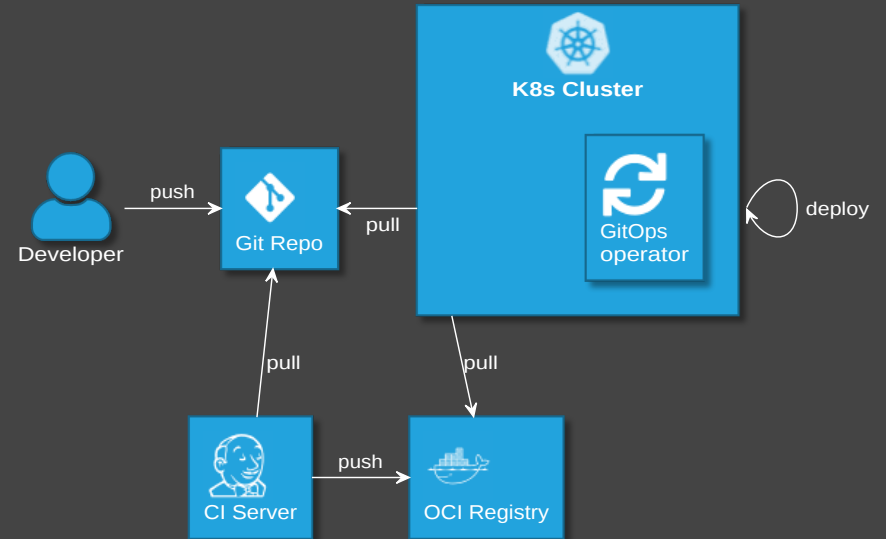
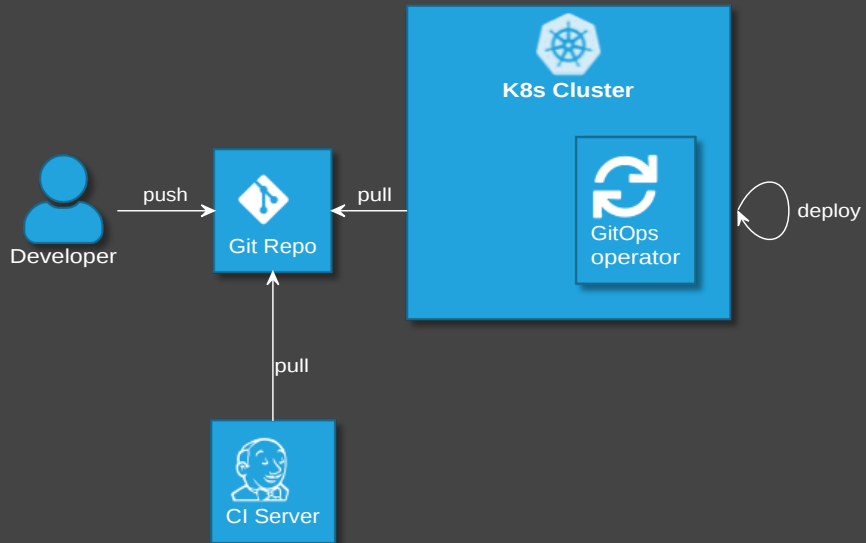
 <https://www.weave.works/blog/gitops-operations-by-pull-request>

- Funktioniert gut mit k8s ist aber nicht darauf beschränkt

Continuous Delivery

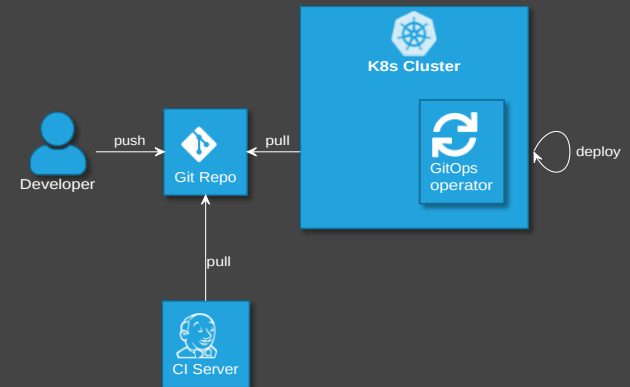


GitOps



Vorteile von GitOps

- Weniger schreibende Zugriff auf Cluster nötig
- Keine Credentials im CI Server
- Config As Code: Auditierung, Reproduzierbarkeit, Cluster und Git automatisch synchronisiert
- Zugriff auf Git oft organisatorisch einfacher als auf API-Server. Stichwort: Firewall-Freischaltung



Anwendungsfall: Neueinführung von GitOps (OnPrem)

ITZBund – IT-Dienstleister für Bundesverwaltungen

Dienstleistungen (u.a.)

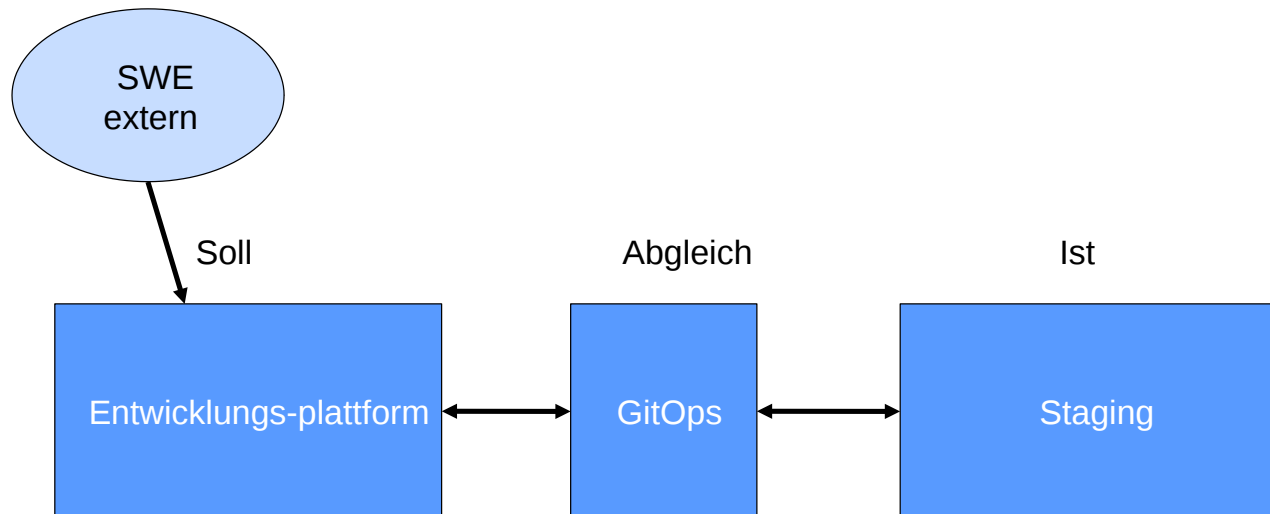
- bietet IT-Infrastruktur (z.B. Einwahlplattformen, Client-Virtualisierung, Cloud-Lösungen)
- Hosting von Anwendungen

Anforderungen

- Staging von SW-Entwicklungen im Haus (mittels standardisierter Entwicklungsumgebung)
- Staging für SW-Entwicklungen außerhalb des Hauses
- Continuous Delivery/Staging
 - Forderung, fertige SWE-Produkte schnell zu stagen
 - Abstimmung der Konfiguration → Infrastructure as Code

Motivation für GitOps

- automatisiertes Stagen
- Berücksichtigung der Umgebungskonfiguration
- pull-Operationen von einem höheren Security-Level
- kein Veröffentlichen von credentials der Staging-Umgebungen an Dev



Anwendungsfall:

Migration CI/CD ➡ GitOps (Public Cloud)

Ausgangslage

mycloudogu

- Kleines, junges Unternehmen
- Ziel: Quick Time to Market, geringe Aufwände
- Seit 2017 Continuous Delivery nach K8s in public Cloud


Motivation für GitOps

Continuous Delivery funktioniert gut. Aber:

- Viele 3rd Party Anwendungen ohne CD Pipeline, mit manuellem Deployment
 - ➔ Gefahr: commit/push vergessen
- Schreibender Zugriff auf Cluster notwendig (Devs & CI)
 - ➔ Security?
 - ➔ Zusätzliche Gefahr: "ausversehen etwas deployt"
- Erneuter Build für jede Stage
 - ➔ langsam
- Helm: Chart URL und Version in CD Pipeline festlegen? 🤔
 - ➔ Helm Operator


Herausforderungen in der Praxis

Mehr Infrastruktur - GitOps Operator / CI/CD

- Flux (ehemals weaveworks, jetzt CNCF Sandbox)
- Argo (CNCF Incubator)
- JenkinsX (CDF)
- Spinnaker (CDF)
- viele weitere:
 <https://github.com/weaveworks/awesome-gitops>



Entscheidung und erste Erfahrungen


- Viele Lösungen sind vollständige CI/CD Lösungen
- Flux: Reiner GitOps-Operator
 - ➔ Integriert gut mit bestehender CI/CD Lösung - 
- Einfach deployt und konfiguriert
- Technischer Durchstich schnell erreicht

Offene Fragen bei Flux

Aber: Es warten viele Detailfragen

- Git-sync via Polling?
- Wie Helm/Kustomize deployen?
- Ressourcen löschen?
- Umgang mit Fehlern?
- Wie Staging implementieren?
- Infrastruktur im Applikations-Repo oder im GitOps-Repo?
- Lokale Entwicklung?
- Zukunft: Flux v2 / GitOps Toolkit / GitOps Engine?
- ...

Mehr Infrastruktur 2 - webhook receiver

- Flux pollt Git alle 5 Minuten ➡ langsames Deployment
- Alternativen
 - Mehr Infra:  fluxcd/flux-recv
 - Manuell anstoßen:


```
fluxctl sync --k8s-fwd-ns kube-system
```


Mehr Infrastruktur 3 - Helm/Kustomize Operators

Je nach verwendeten Tools, mehr Operators notwendig

- Helm Operator
- Kustomize Operator
- was tun bei anderen Templating Tools?

Löschen von Ressourcen

- "garbage collection" kann in Flux aktiviert werden
- 
- ... oder doch lieber manuell löschen

Fehlerbehandlung

- Push, Build und Deployment entkoppelt
- Fehlermeldung asynchron ➡ Fehler werden später bemerkt
- Ursachen im Flux und Helm Operator Log schwer zu finden
- Abhilfe:
 - Fail early mit CI Server - wenn Pipeline vorhanden
 - Monitoring und Alerting - schwer wartbar

Herausforderungen Flux Monitoring und Alerting

```
delta(flux_daemon_sync_duration_seconds_count{success='true'}[6m]) < 1
```

- Monitoring-Queries in Doku nicht intuitiv
- Erzeugt viele Alerts
- Betroffene Anwendung und Ursache muss im Log gesucht werden
- Alerts und Neustarts schwierig zu differenzieren von "echten"

Deployment-Fehlern. Beispiele:

- Alerts während Wartungsfenster von Git Server
- Operator Pod Neustarts
- Operator Pod OOM Kills

- ➔ Betriebsaufwände der Operator nicht vernachlässigbar
- ➔ Umgewöhnung bei Entwicklern notwendig

Implementierung von Stages

Idee 1: Staging Branches

- Develop ➡ Staging
- Main ➡ Production
- Flux kann nur mit einem Git Repo umgehen
 - ➡ Ein Flux pro Stage (Cluster/Namespace)



- Branching-Logik aufwendig und fehleranfällig
- Betrieb aufwendig (mehrere Flux-Instanzen notwendig)

Idee 2: Staging Ordner

- Ein Ordner pro Stage
- Alle auf demselben Branch
- Wenn nötig: Staging Namespace in Ressourcen nennen
- Prozess: Staging einfach committen; für Prod PR erstellen
- Manuell zwar umständlich, aber gut für Automatisierung



- Branching-Logik simpler
- Betrieb weniger aufwendig

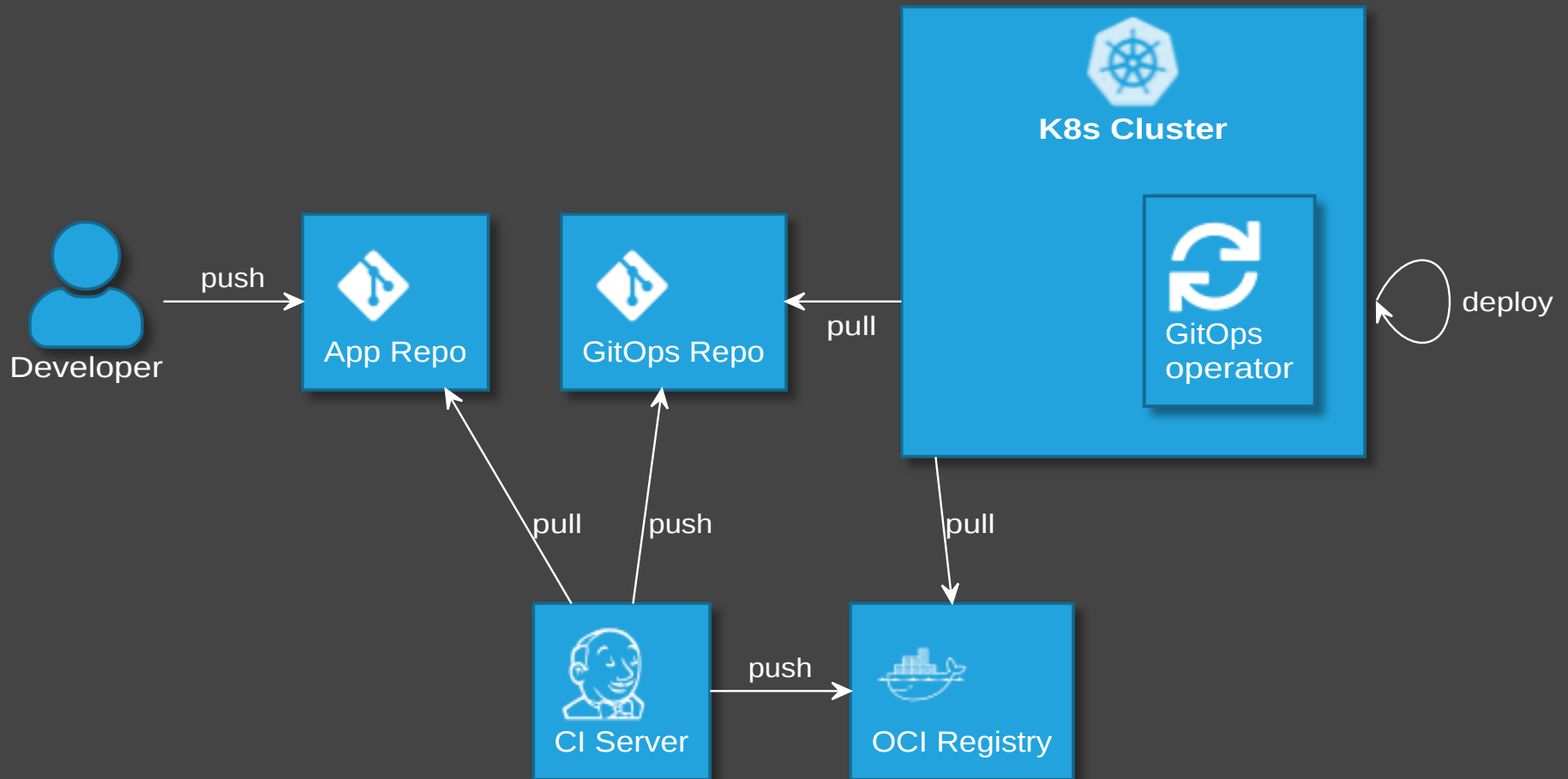
Applikations-Repo vs im GitOps-Repo

- Bisher: Infrastruktur direkt neben Code im App Repo
- Jetzt: Infrastruktur getrennt vom Code im GitOps Repo ?!

➡ Nachteile:

- Getrennte Pflege
- Getrennte Versionierung
- Aufwendigeres Review
- Aufwendigere lokaler Entwicklung

Lösung: CI-Server



Resultat

My gitops workflow might be turing complete

– Darren Shepherd, CTO Rancher Labs

 <https://twitter.com/ibuildthecloud/status/1311474999148961798>

Nachteile

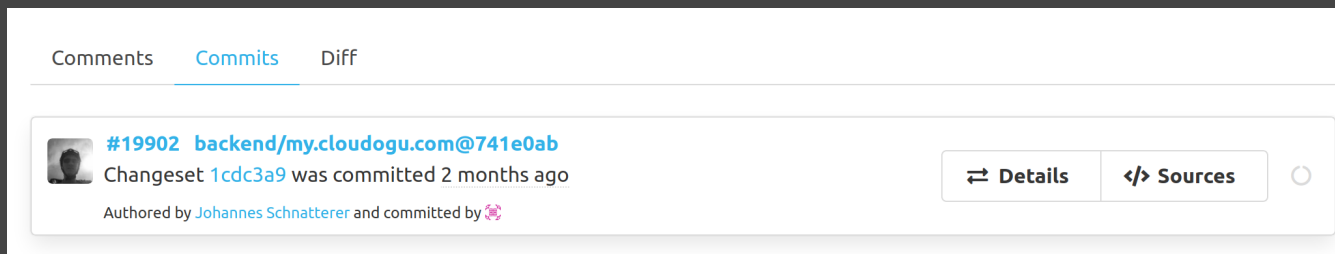
- Komplexität
- Entwicklungsaufwand für Logik der CI-Pipeline
- Viele Fehlerfälle. Beispiele:
 - Git Conflicts durch Concurrency
 - Dadurch Gefahr von Inkonsistenz
 - Ohne reproducible build: Jeder Build erstellt GitOps PR

➡ Abhilfe: Wiederverwendung

Unser Beispiel:  github.com/cloudogu/k8s-gitops-playground

Vorteile

- Fail early: statische YAML-Analyse durch CI-Server (kubeval, yamlint)
- Automatische PR-Erstellung
- Arbeit auf echten Dateien ➡ CI-Server erzeugt inline YAML
- Test-Deployment von Feature Branch möglich
- Lokale Entwicklung ohne GitOps weiterhin möglich
- Erleichterung von Reviews durch Anreicherung Commit Message:
Autor, original Commit, Issue-ID und Build-Nummer



Lokale Entwicklung

- Option 1: Flux und Git Repo in lokalen Cluster deployen
 - ➔ Umständlich
- Option 2: Keine Änderung. Möglich, wenn Infrastruktur im Applikations-Repo verbleibt.

Zukunft: Flux v2 / GitOps Toolkit / GitOps Engine?

TODO

Gerd Huber, ITZBund

Johannes Schnatterer, Cloudogu GmbH

 cloudogu.com/schulungen
 cloudogu.com/gitops

 GitOps-Jenkins Library (WIP)

 github.com/cloudogu/k8s-gitops-playground

 GitOps-Artikel Java aktuell 2/21

 cloudogu.com/blog

 @cloudogu

 @jschnatterer

