



# // GITOPS MIT K8S IN DER PRAXIS – EIN ERFAHRUNGSBERICHT

Gerd Huber, ITZBund

Johannes Schnatterer, Cloudogu GmbH

 @jschnatterer

Version: 202011032223-c18862f

# Agenda

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

# Was ist GitOps?

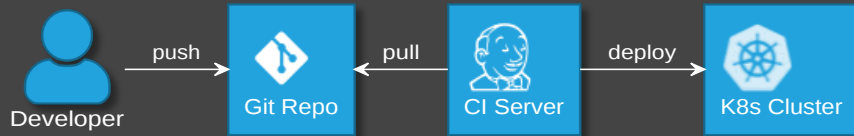
- Begriff (August 2017):

use developer tooling to drive operations

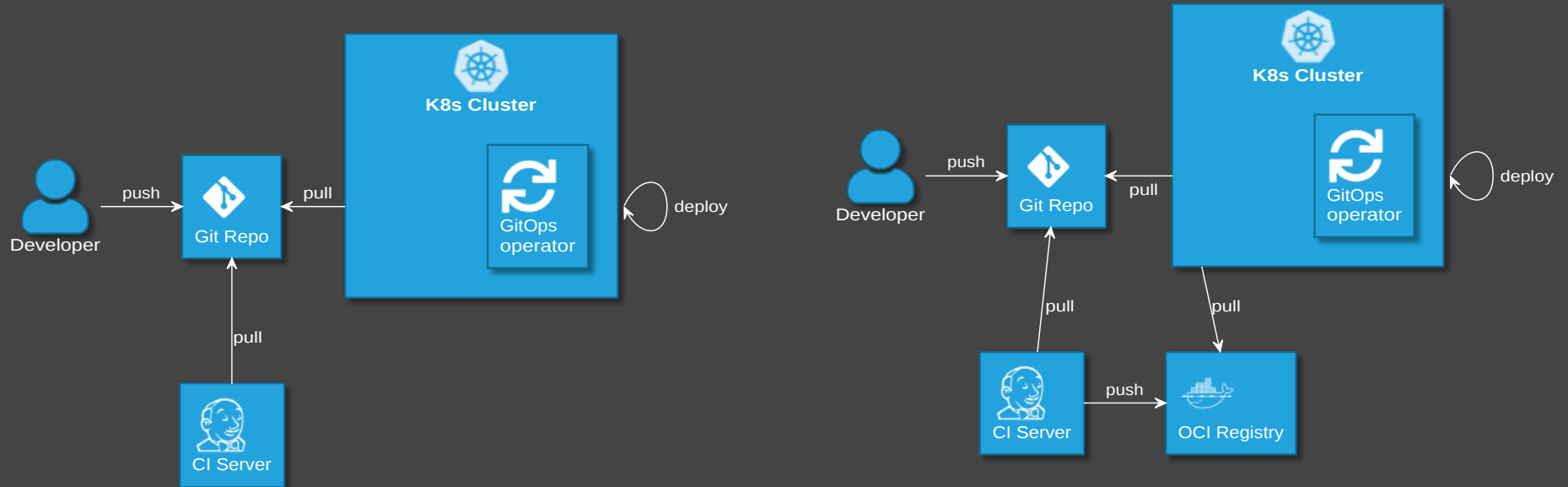
 [weave.works/blog/gitops-operations-by-pull-request](https://weave.works/blog/gitops-operations-by-pull-request)

- Funktioniert gut mit k8s ist aber nicht darauf beschränkt

# Continuous Delivery

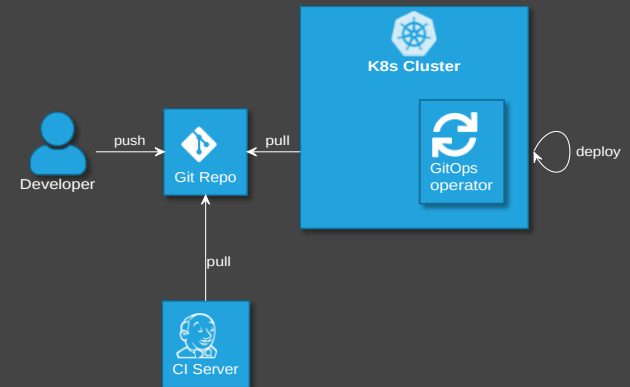


# GitOps



# 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.)**

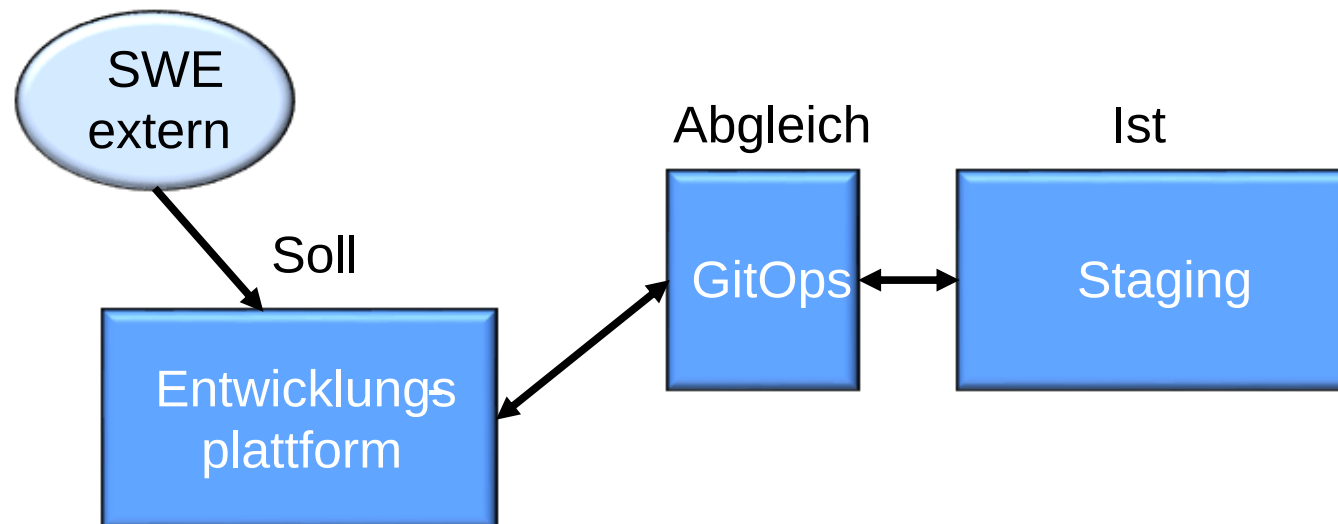
- bietet IT-Infrastruktur (z.B. Einwahlplattformen, Client-Virtualisierung, Cloud-Lösungen)
- Hosting von Anwendungen

### **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 Umgebungsconfiguration
- 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
- Prio: Quick Time to Market
- 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 und Erkenntnisse aus der Praxis**


# Mehr Infrastruktur - GitOps Operator / CI/CD

- Flux (ehemals weaveworks, jetzt CNCF Sandbox)
- Argo (CNCF Incubator)
- JenkinsX (CDF)
- Spinnaker (CDF)
- viele weitere:

 [weaveworks/awesome-gitops](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

# Offene Fragen bei Flux

- Technischer Durchstich schnell erreicht
- Direkt danach 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 von Flux?
  - ...

# Mehr Infrastruktur 2 - webhook receiver

- Flux pollt Git alle 5 Minuten ➡ langsames Deployment
- Alternativen
  - Mehr Infra:  [fluxcd/flux-recv](https://github.com/fluxcd/flux-recv)
  - Manuell anstoßen

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

- Warten 🕒




# 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
- Abhilfe:
  - Fail early mit CI Server - wenn Pipeline vorhanden
  - Monitoring und Alerting - mit Flux ineffizient

# Herausforderungen Monitoring und Alerting mit Flux (1)

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

 [docs.fluxcd.io/en/1.21.0/references/monitoring](https://docs.fluxcd.io/en/1.21.0/references/monitoring)

## Herausforderungen Monitoring und Alerting mit Flux (2)

- Erzeugt viele Alerts
- Betroffene Anwendung und Ursache muss im Log gesucht werden
- Ursachen im Flux und Helm Operator Log schwer zu finden
- Alerts und Neustarts schwierig zu differenzieren von "echten"

Deployment-Fehlern. Beispiele:

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

# 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

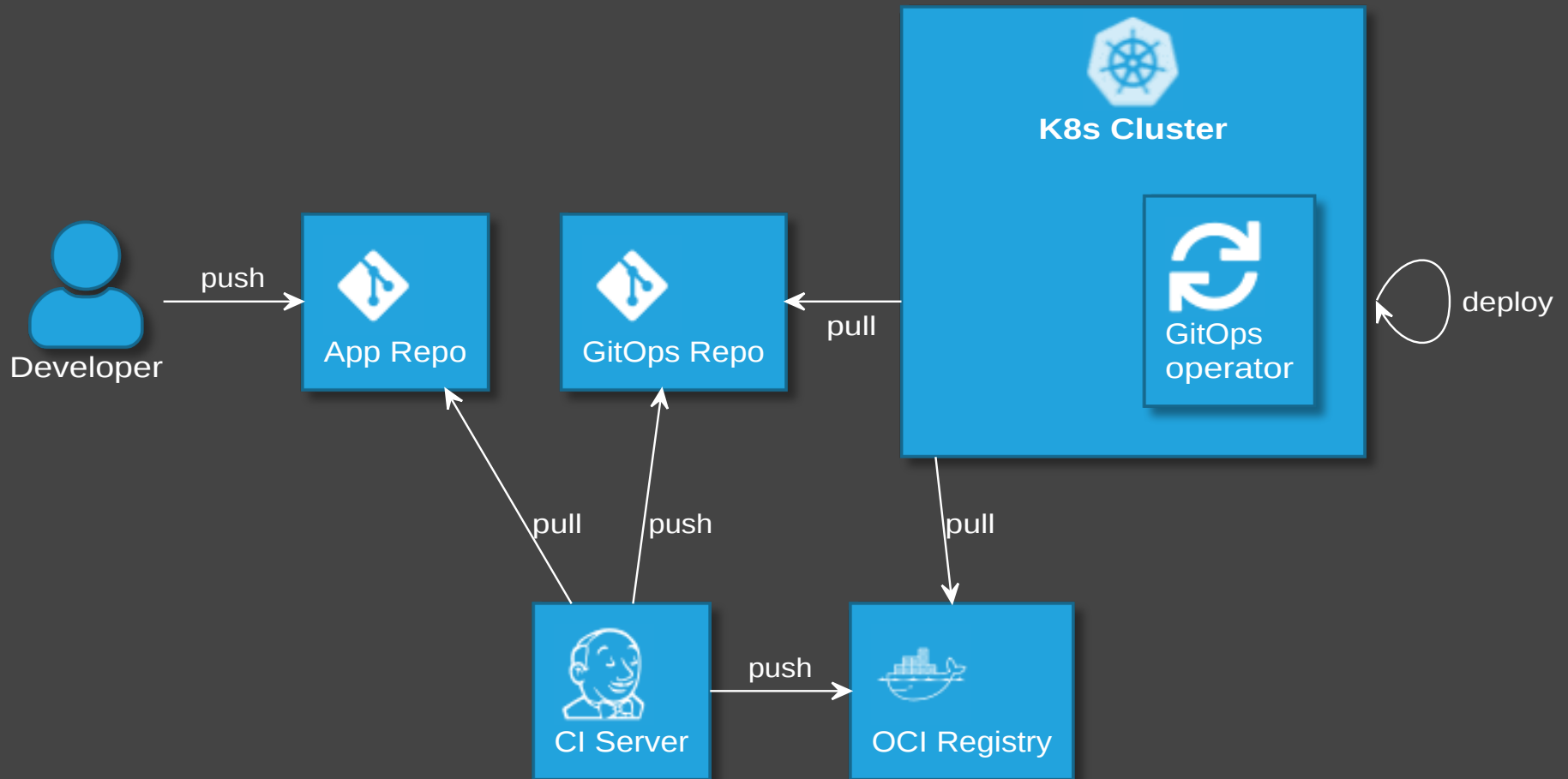
- Bisher: Infrastruktur direkt neben Code im App Repo
- Jetzt: Infrastruktur getrennt vom Code im GitOps Repo ?!

## ➡ 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

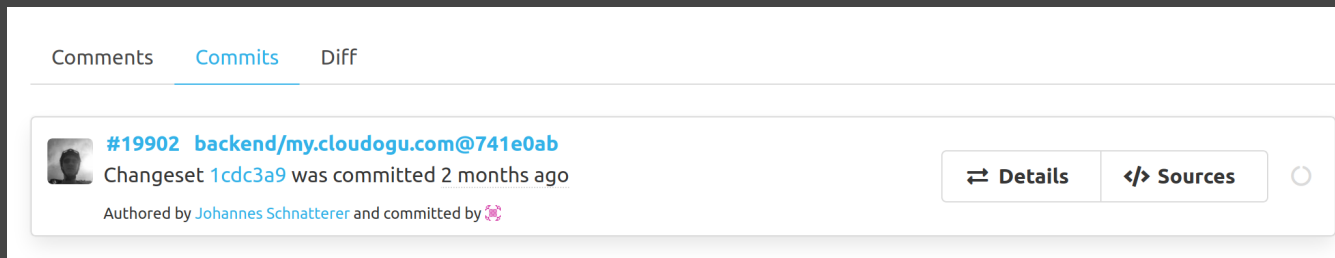
 [twitter.com/ibuildthecloud/status/1311474999148961798](https://twitter.com/ibuildthecloud/status/1311474999148961798)

## Nachteile

- Komplexität
- Entwicklungsaufwand für Logik der CI-Pipeline
- Viele Fehlerfälle. Beispiele:
  - Git Conflicts durch Concurrency
  - Dadurch Gefahr von Inkonsistenz

# Vorteile



- Fail early: statische YAML-Analyse durch CI-Server (yamlint, kubeval)
- 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



# 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 von Flux

- August 2019: Flux + Argo GitOps Engine = Flux v2
- Juli 2020: Flux v2  ~~GitOps Engine~~ GitOps Toolkit
- Egal wie: Breaking Changes
- Dafür viele neue Features:
  - Alerting
  - Webhook Receiver eingebaut
  - Helm und Kustomize Operators
  - Mehrere Git Repos
  - Mandanten
  - Flux aktualisiert sich selbst mit GitOps
  - ...  [toolkit.fluxcd.io](https://toolkit.fluxcd.io)

# Das Flux-Dilemma (Stand 11/2020)

⚠ This also means that Flux v1 is in maintenance mode.

 [github.com/fluxcd/flux/blob/738d47/README.md](https://github.com/fluxcd/flux/blob/738d47/README.md)

Once we have reached feature-parity [..], we will continue to support Flux v1 and Helm Operator v1 for 6 more months

 [github.com/fluxcd/flux/issues/3320](https://github.com/fluxcd/flux/issues/3320)

- Flux v2 hat aber noch nicht alle Features von Flux:

 [toolkit.fluxcd.io/roadmap](https://toolkit.fluxcd.io/roadmap)

- und ist nicht stable - Version 0 . x

 [github.com/fluxcd/flux2/releases](https://github.com/fluxcd/flux2/releases)

# Fazit

- ITZBund
  - Vereinfacht / beschleunigt Prozesse
  - Konfiguration liegt an definierter, zentraler Stelle vor
  - GitOps-Prozess als Konvention ➡ einheitlich über viele Projekte
  - Vorteile bei Security
- Cloudogu
  - CI/CD-Prozess "runder"
    - schnelleres Deployment in Produktion
    - Git und Cluster immer in Sync
  - Aber: Security-Vorteile tragen erst nach vollständiger Migration



# Destillierte GitOps-Erfahrung

- + Funktionierendes GitOps hat viele Vorteile
- Der Weg dorthin kann aufwendig sein!

# Empfehlung für Flux?

Dilemma: Flux v1 in maintenance, Flux v2 noch nicht stable.

- Bei bestehendem CI/CD-Prozess:  
Nur mit gutem Grund migrieren bevor Flux v2 stable ist
- Bei Neueinführung von Kubernetes:  
Flux v2 ausprobieren

Gerd Huber, ITZBund

Johannes Schnatterer, Cludogu GmbH

 [cloudogu.com/gitops](https://cloudogu.com/gitops)

 [cloudogu.com/schulungen](https://cloudogu.com/schulungen)

 In Arbeit

- GitOps-Jenkins Library

 [github.com/cloudogu/k8s-gitops-playground](https://github.com/cloudogu/k8s-gitops-playground)

- GitOps-Artikel

 [cloudogu.com/blog](https://cloudogu.com/blog)

 @cloudogu

 @jschnatterer

