dataport

anderScore
trust in competence

Tag 3: GitOps, Docker in der Entwicklung und Deployment-Strategien

19.06.2024, Daniel Krämer & Malte Fischer

Agenda

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

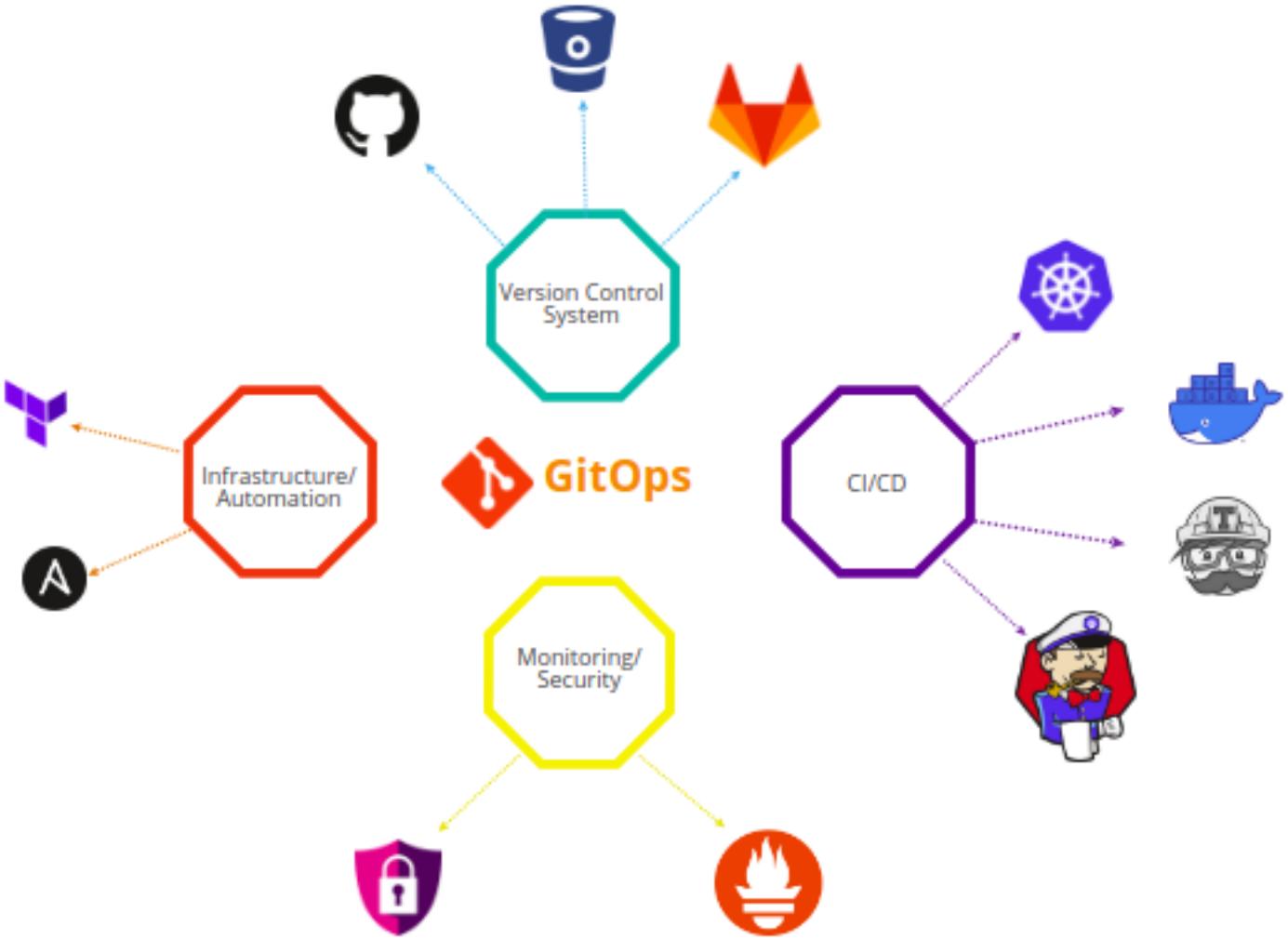
Agenda

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Grundlagen von

GitOps

GitOps


Version Control System
Infrastructure/ Automation
GitOps
CI/CD
Monitoring/ Security

GitOps

anderScore trust in competence

“GitOps is an operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD, and applies them to infrastructure automation.”

- Someone from GitLab

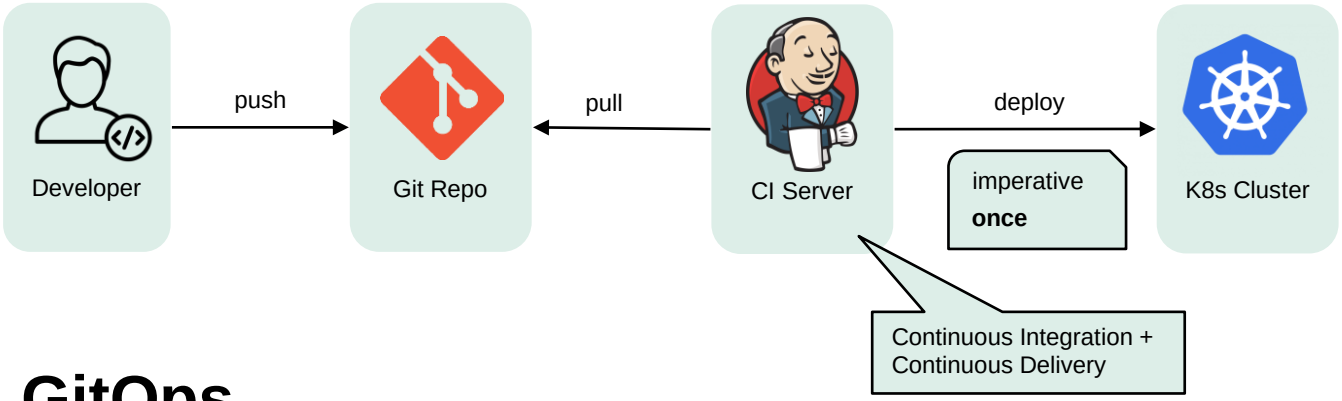
GitOps

Grundprinzipien

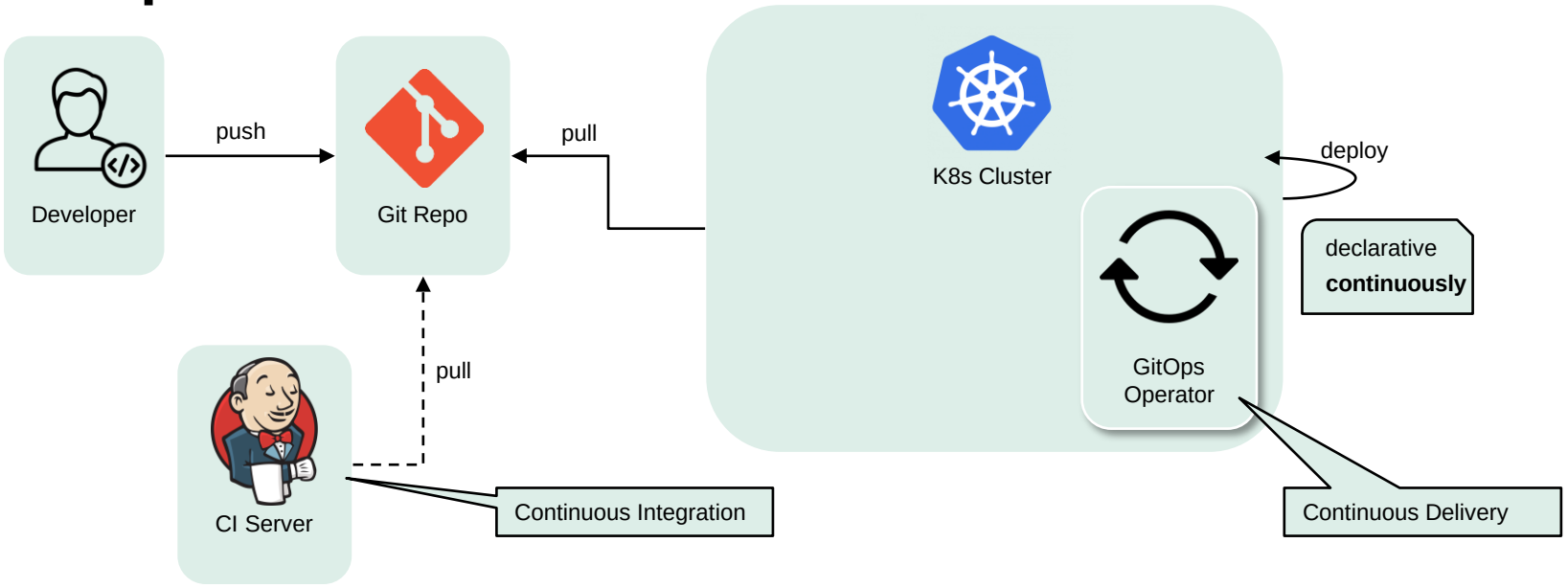
1. **Declarative**
   Der gewünschte Zustand (state) eines von GitOps verwalteten Softwaresystems muss deklarativ beschrieben (declarative description) werden.
2. **Versioned and Immutable**
   Der gewünschte State (desired state) wird so gespeichert (stored), dass Unveränderlichkeit und Versionierung gewährleistet sind und ein vollständiger Versionsverlauf erhalten bleibt.
3. **Pulled Automatically**
   Software agents (= GitOps Operator) pullen automatisch den gewünschten state aus der Quelle (= Git Repo).
4. **Continuously Reconciled**
   Software agents überwachen kontinuierlich (continuously) den tatsächlichen Systemstatus (actual state) und versuchen, den desired state anzuwenden.
   → Reconcilation

*** Unterstrichene Wörter: https://github.com/open-gitops/documents/blob/v1.0.0/GLOSSARY.md**

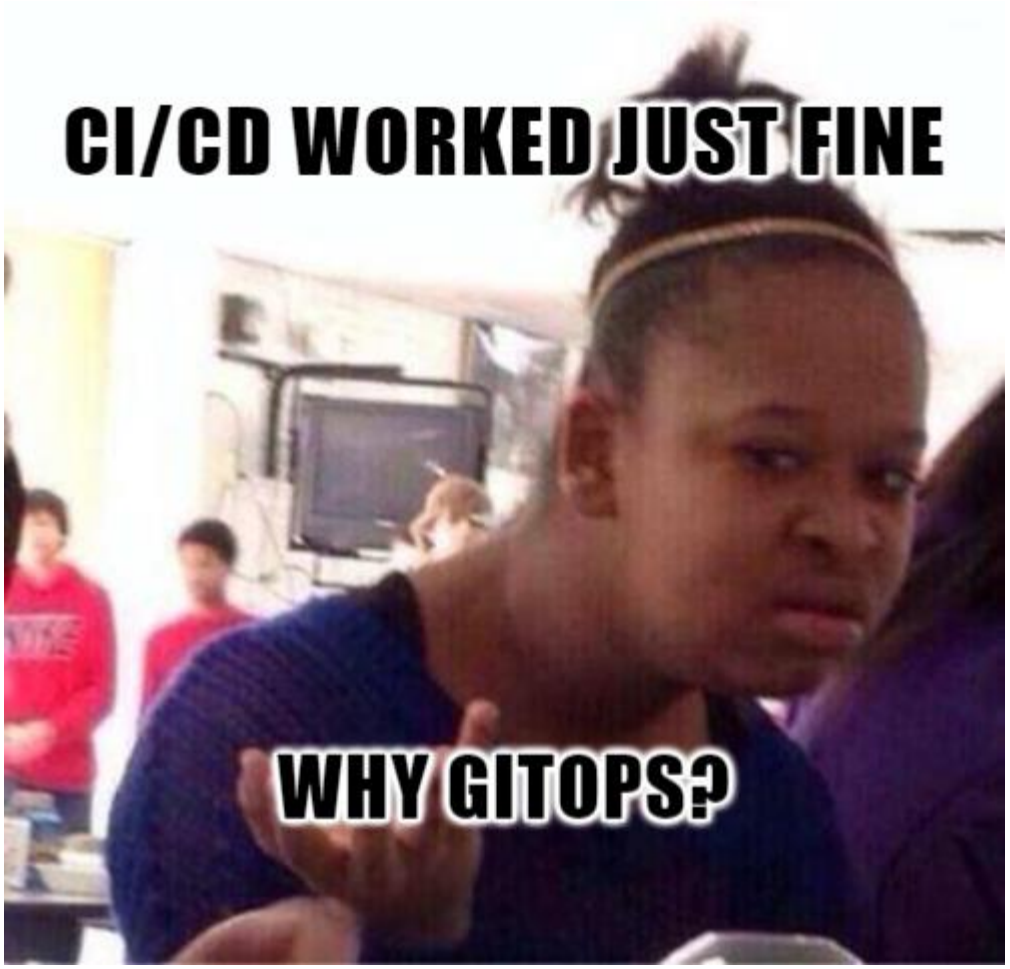
„Klassisches“ Continuous Delivery („CIOps“)



GitOps



GitOps



Die Idee dahinter

- Softwareentwicklungs-Lebenszyklus größtenteils automatisiert
  - Infrastruktur weitestgehend ein manueller Prozess
    - → Benötigt (oft) spezialisierte Teams
- Moderne und Cloud-native Anwendungen werden mit Blick auf Geschwindigkeit und Skalierbarkeit entwickelt
  - Trend: Infrastruktur in die Cloud
- Ziel: Prozess der Bereitstellung von Infrastruktur automatisieren
  - Analog zur Softwareentwicklung verwenden Operation-Teams Konfigurationsdateien als Code
  - Diese Dateien garantieren konsistente Infrastruktur bei jeder Bereitstellung, ähnlich wie Softwarecode konsistente Binärdateien erzeugt

GitOps

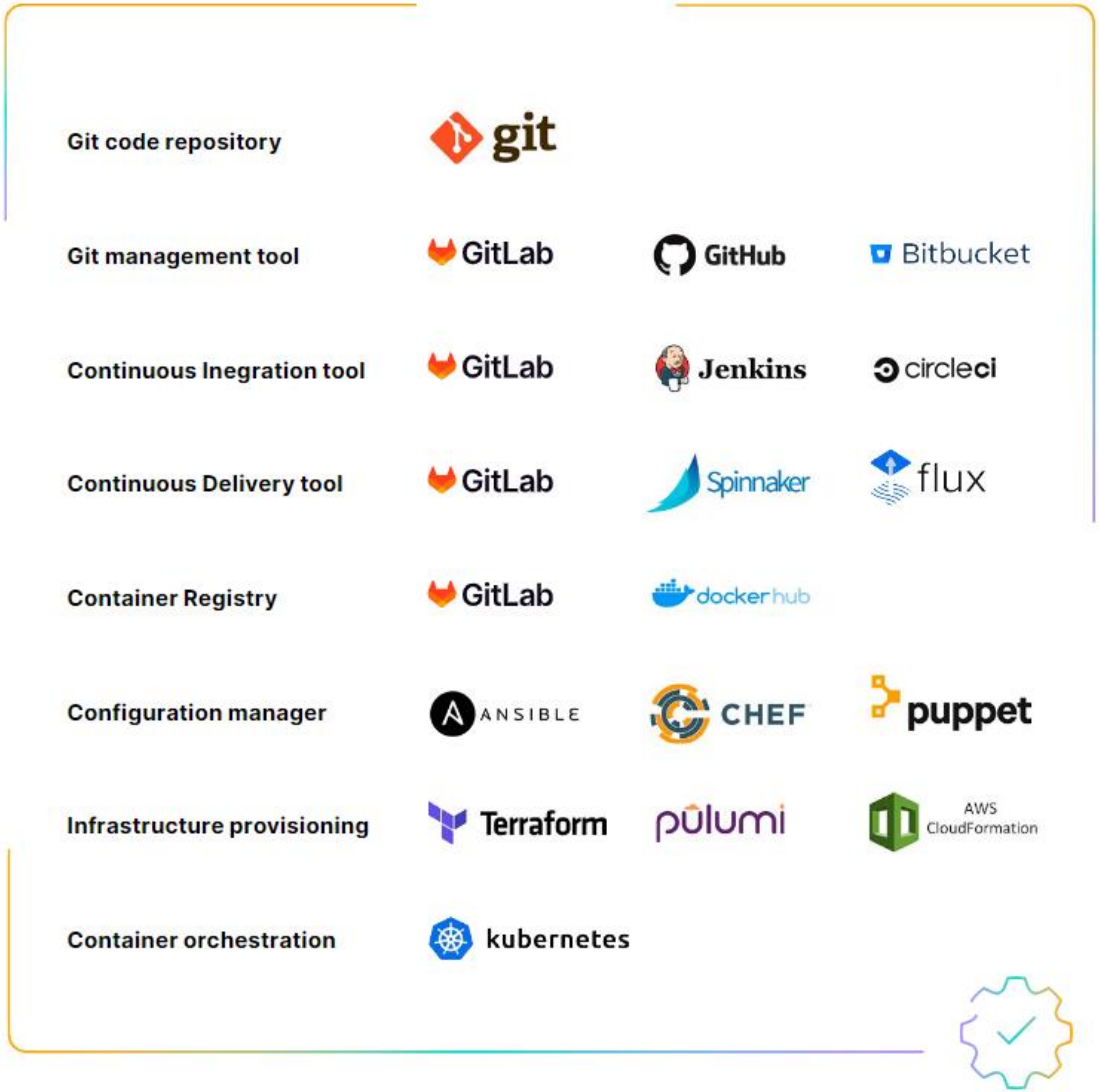
Was genau ist das jetzt?

- Git Repositories als Single Source of Truth
- Bereitstellung von Infrastruktur als Code (IaC)
- Vorgehen:
  - CI-Prozess prüft eingecheckten Code
  - CD-Prozess prüft Anforderungen für Sicherheit, IaC oder andere Richtlinien und wendet die Anforderungen an (IST vs. SOLL)
- Codeänderungen sind nachvollziehbar (git)
  - → Updates vereinfacht
  - → Rollbacks möglich

Was bietet mir das?

- Workflow für Anwendungsentwicklung
- Transparenz (git)
- Konsistenz (für Cluster, Clouds und On-Premise)
- Freie Toolwahl um ein GitOps-Framework aufzubauen
  - Git-Repositories
  - CI/CD-Tools (wie Jenkins, Spinnaker, cicleci, flux, Argo CD, …)
  - Kubernetes
  - Konfiugrationsmanagementtools (Ansible, Chef, puppet)

GitOps



Quelle: gitlab-ebook

Warum GitOps?

- Als Framework für eine DevOps Umsetzung/Evolution
- State of DevOps Report
  - https://services.google.com/fh/files/misc/2023_final_report_sodr.pdf
  - Innovationsrate sowie Stabilität verbessert
    - Für Anwendungen und Code!
- Git-basierte Workflows verwendbar ➔ Entwickler = ☺
- GitOps erweitert diese Workflows um Operations, konkret:
  - Deployment
  - Application life cycle management
  - Infrastructure configuration
- Jede Änderung ist im Git Repository nachverfolgbar
  - → Auditierung/Audit möglich

Warum GitOps?

- Dev(Ops)-Teams geben eigene Entwicklungsgeschwindigkeit vor
  - → Keine/kaum Wartezeit auf Ressourcen
    - Operation-Teams müssen weder Ressourcen zuweisen noch genehmigen
- Änderungen sind transparent
  - Probleme schnell nachvollzieh- und reproduzierbar
  - → Sicherheit insgesamt verbessert!
- Up-to-date Audit Trail
  - Ungewünschte Änderungen können schnell korrigiert werden
- Codeänderungen von Entwicklung bis Produktion
  - Mehr Agilität bei der Reaktion auf Geschäfts- und Wettbewerbsveränderung

GitOps

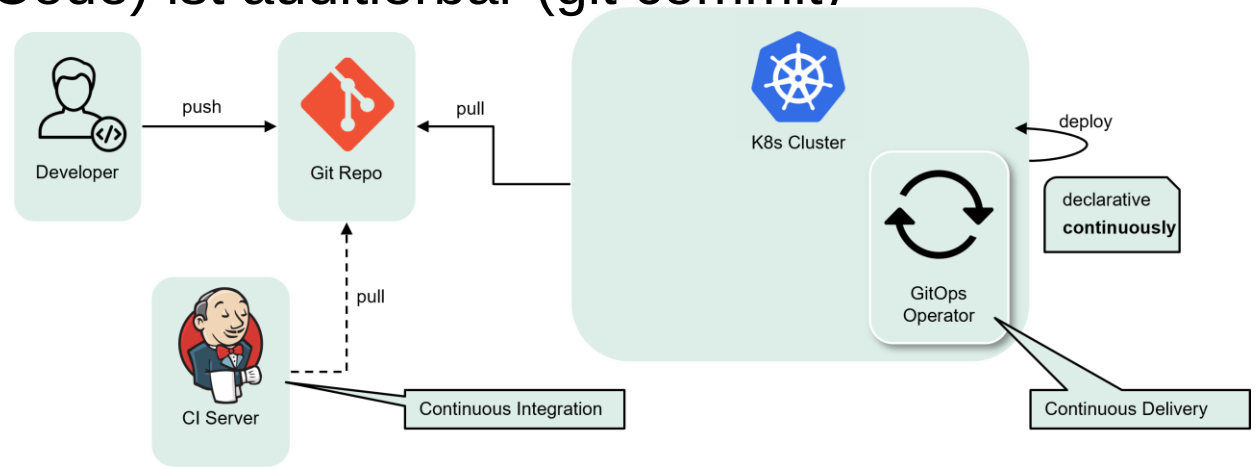
- Infrastruktur muss deklarativ gemangt werden können
  - → Kubernetes und Cloud-native Softwareentwicklung
  - GitOps = Enabler für Continuous Deployment mit Kubernetes!
- Kubernetes nicht zwingend erforderlich
  - Auch andere Infrastruktur- und Deployment-Pipelines möglich
- Also… Mit GitOps lassen sich:
  - Development Pipelines erstellen,
  - Anwendungen entwickeln,
  - Konfigurationen verwalten,
  - Kubernetes-Cluster bereitstellen und
  - Deployments auf Kubernetes oder Container-Registries vornehmen

GitOps-Workflow…?

- Git als Versionskontrollsystem für Infrastrukturkonfigurationen (IaC)
- CI/CD-Pipelines normalerweise durch externes Event ausgelöst
- Bei GitOps: Änderungen über Pull-Requests (PR) oder Merge-Requests (MR)
- Neues Release?! PR in git!
  - Ändert den deklarierten Zustand des Clusters
  - GitOps-Operator sitzt zwischen der GitOps-Pipeline und der Orchestrierung (Kubernetes), picked den Commit und pulled den neuen deklarativen Zustand aus git
- Sobald Änderungen im PR approved und merged sind, wird die Infrastruktur aktualisiert
  - Dev-Teams können weiterhin Ihre CI/CD-Praktiken verwenden

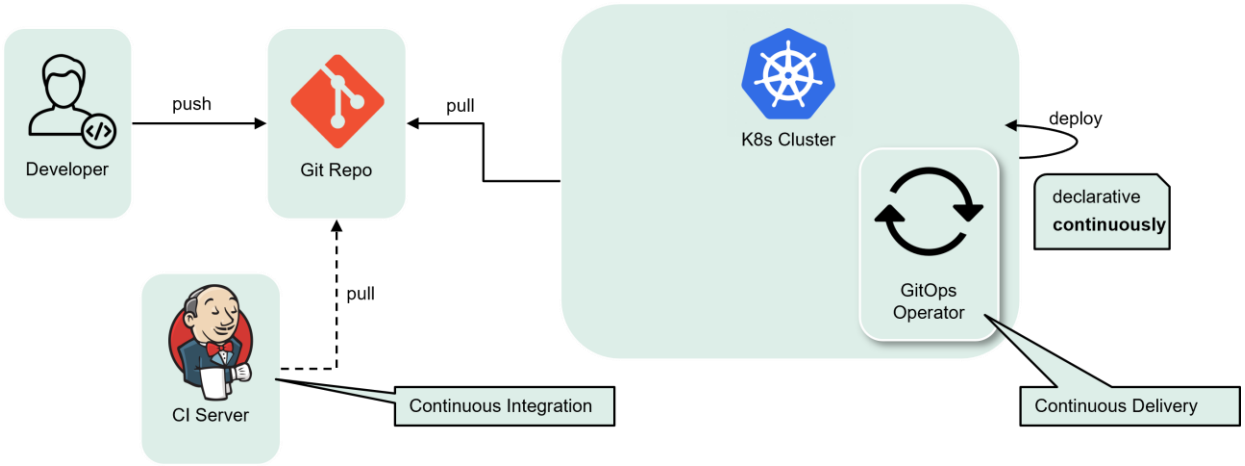
Vorteile

- Erhöhte Sicherheit
  - Kein Zugriff von Außen auf das Cluster
  - Keine Credentials auf dem CI Server
    - Zielscheibe des CI Servers wird kleiner ☺
- Erzwingt deklarative Beschreibung
  - Keine Änderungen am CI Server selbst
- Infrastruktur (als Code) ist auditierbar (git commit)
- Skalierbarkeit
- Self-healing



Vorteile

- Erhöhte Produktivität
  - Schnelle Veröffentlichung von Änderungen
  - Reproduzierbarkeit der Infrastruktur
  - Schnellere Rollbacks
  - Vereinfachte Berechtigungsstrukturen



GitOps Operators / Tools

- ArgoCD - Declarative continuous deployment for Kubernetes
- Flux - Open and extensible continuous delivery solution for Kubernetes. Powered by GitOps Toolkit
- Flagger - Progressive delivery Kubernetes operator (Canary, A/B testing and Blue/Green deployments automation)
- Jenkins X - a CI/CD platform for Kubernetes that provides pipeline automation, built-in GitOps and preview environments
- Werf - GitOps tool with advanced features to build images and deploy them to Kubernetes (integrates with any existing CI system)
- PipeCD - Continuous Delivery for Declarative Kubernetes, Serverless and Infrastructure Applications
- GitLab K8s Agent - Connecting a Kubernetes cluster with GitLab

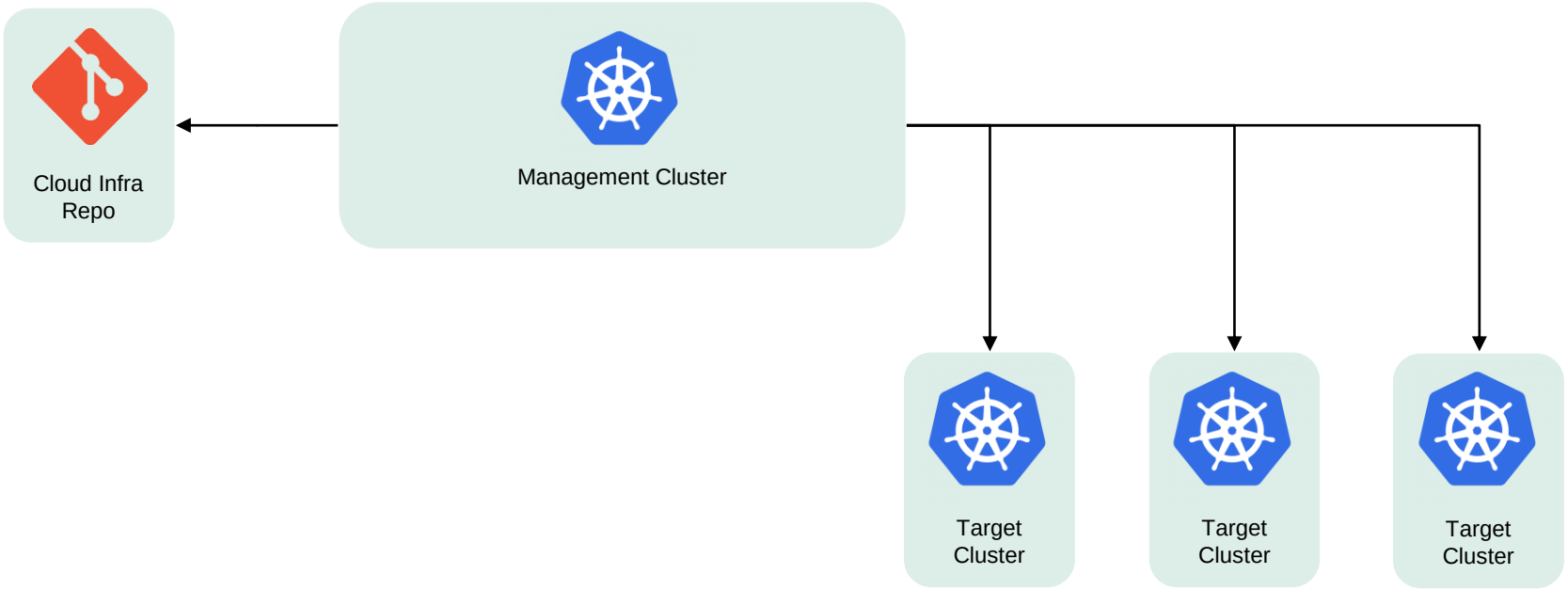
GitOps

Secrets

- Bei CIOps oft im CI Server hinterlegt…
- Secrets im Repository speichern
  - encrypted/sealed!
- Secret im Key Management System (KMS) speichern
  - Verschiedene KMS
    - Proprietär – idR. Cloud-Anbieter (AWS, Azure, Google, …)
    - Hashicrop Vault
  - Kubernetes Integration
    - Operator, Container Storage Interface (CSI) Driver, Sidecar (Injector), Helm/Kustomize Plugin, GitOps Operator: nativer Support oder Plugin

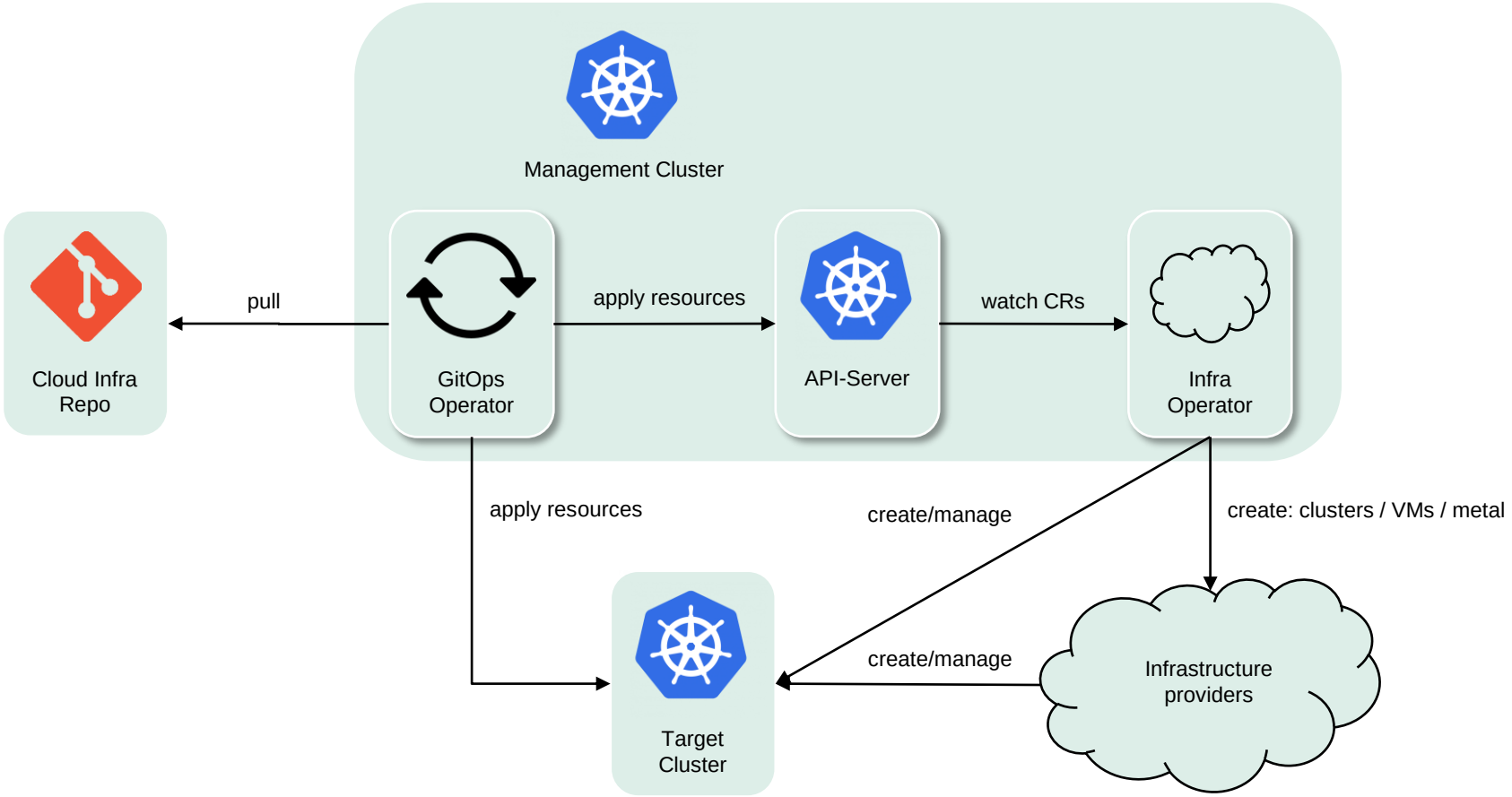
Einsatzbereiche

- Gesamte Cloud-Infrastruktur mit GitOps betreiben!
- Kubernetes Cluster mit… sich selbst betreiben?



Im Management Cluster ➔ GitOps umsetzen

… im Management Cluster



Best Practises

- Lokale Entwicklung
- Staging
- Rolle des CI Servers
- Anzahl der Repositories
- Erweiterte Rolle des CI Servers

Lokale Entwicklung

Verschiedene Möglichkeiten

1. GitOps Operator und Git Server auf lokalem Cluster deployen
   - Möglicherweise komplex

2. Einfach ohne GitOps weiterentwickeln
   - Kann funktionieren, wenn App und Infra Code im gleichen Repo liegen

Staging Branches?

- Dev-Branch nach Staging-Branch
  - … und unser Main-Branch geht in die Produktion
- Merging wird schnell sehr kompliziert ☹
  - …und wird pro Stage komplexer ☹
- ➔ Wird generell von abgeraten!

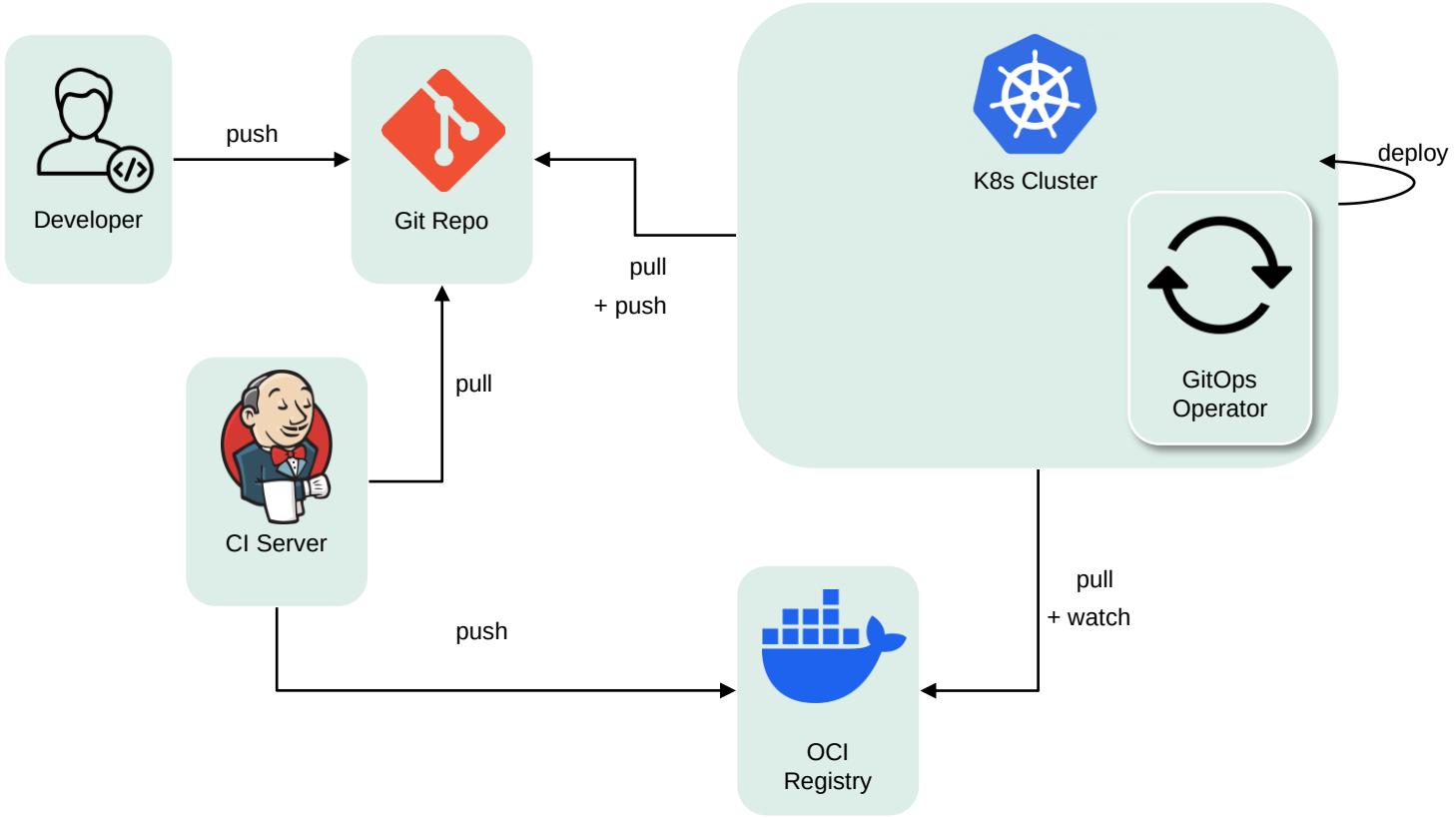
Staging Folders!

- Ein Ordner pro Stage!

```
├── production
│   └── application
│         ├── deployment.yaml
│         └── ...
└── staging
    └── application
          ├── deployment.yaml
          └── ...
```

  - Commits nur im jeweiligen Staging-Ordner
  - Kurzlebige Pull Requests um die Änderungen zu aktivieren
- Duplikate pro Stage ☹
- Branching ist einfacher ☺
- Unterstützt beliebige Anzahl von Stages ☺

Rolle des CI Servers

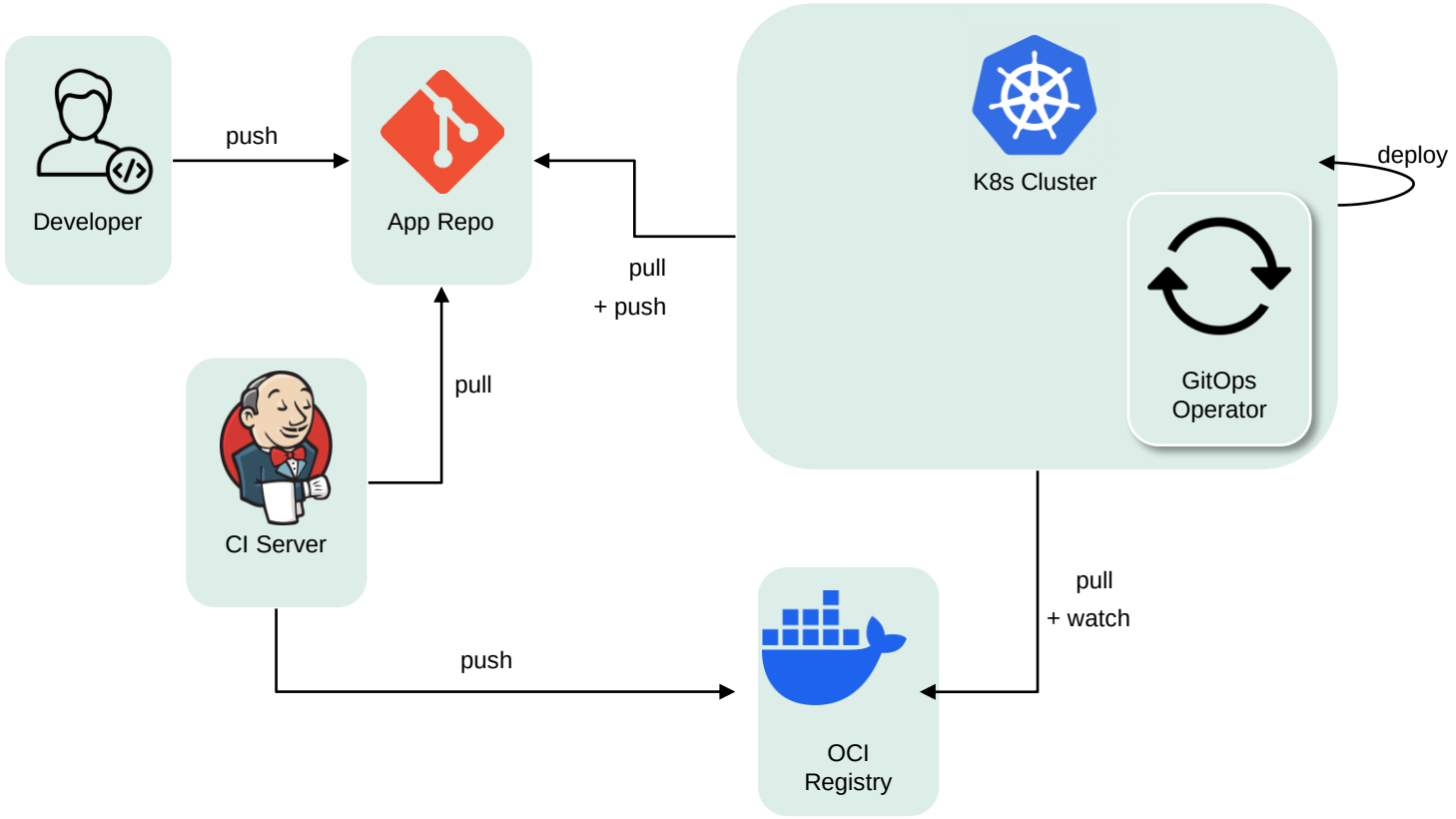


Optional: GitOps Operator aktualisiert image version in Git

- https://github.com/argoproj-labs/argocd-image-updater
- https://fluxcd.io/flux/guides/image-update/

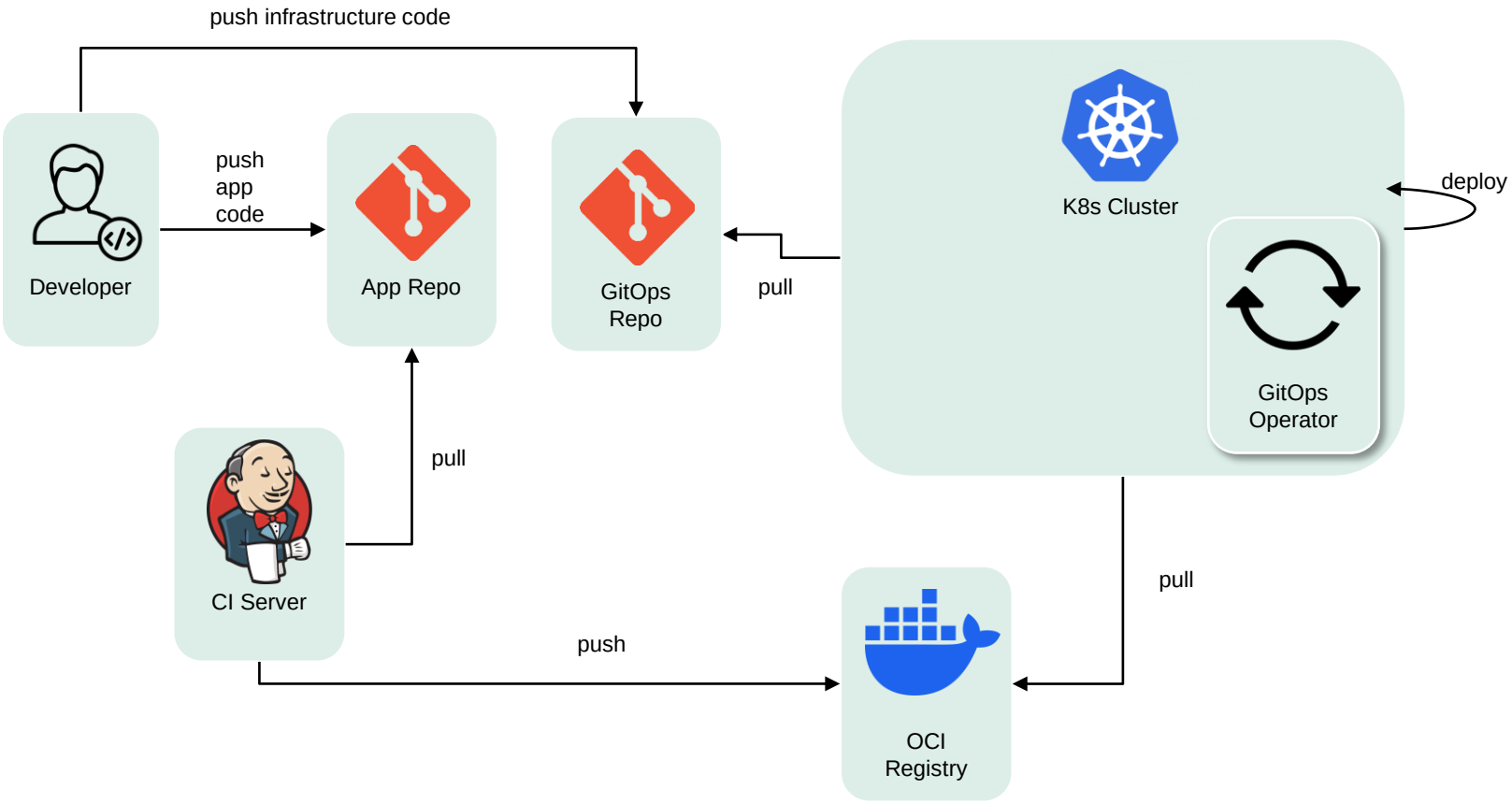
Anzahl der Repositories: Anwendung vs. GitOps Repo

- Anwendungs Repo:



Anzahl der Repositories: Application vs. GitOps Repo

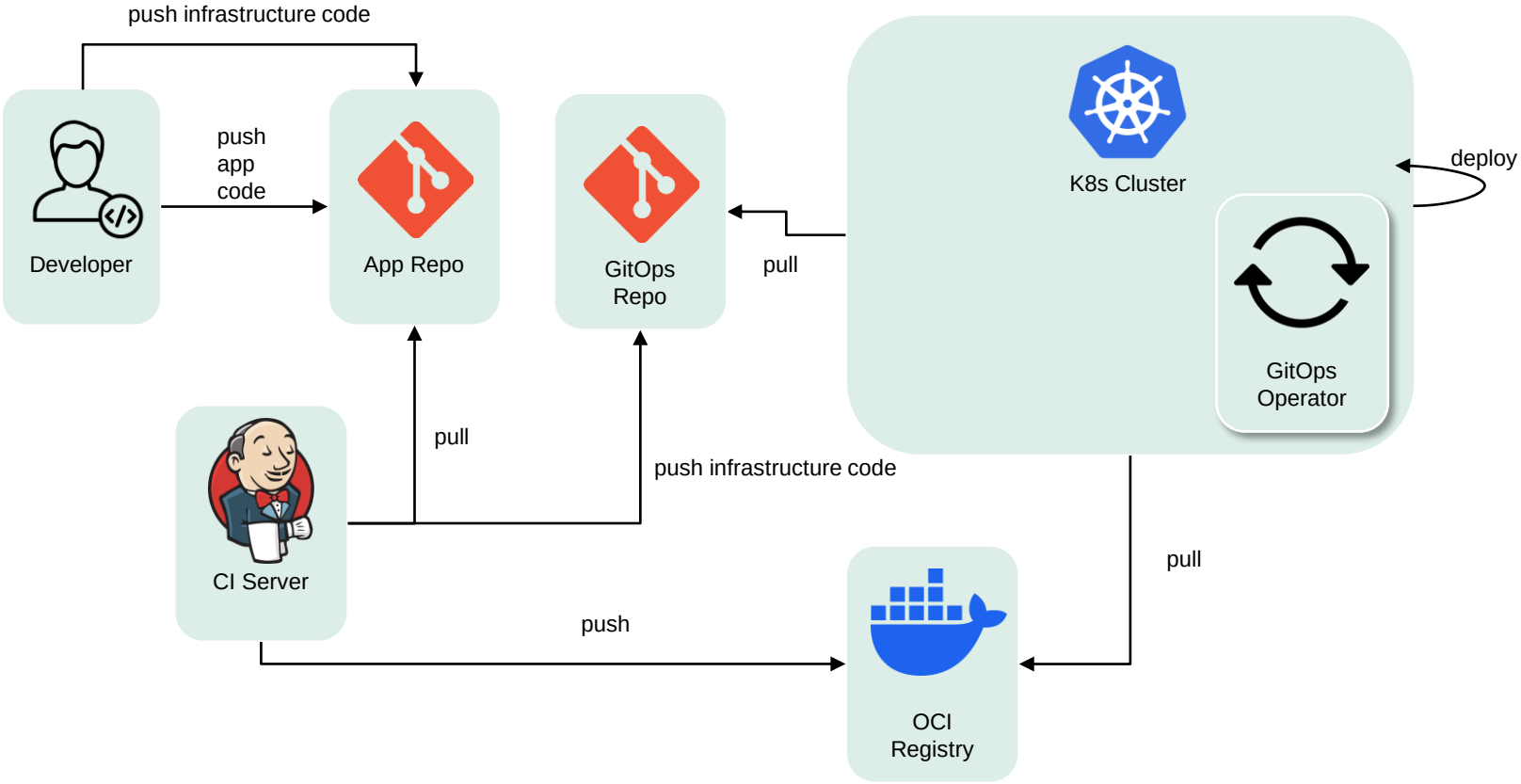
- GitOps Repo:



Herausforderungen beim GitOps Repo

- Mehrere Repos müssen gewartet werden
- Refactorings und tags sind schwerer
- Lokale Entwicklung wird komplizierter
- Shift-Left-Ansatz nur beim Anwendungscode (CI Server)
  - Tests, Linting, statische Codeanalyse, …

Erweiterte Rolle des CI Servers



Erweiterte Rolle des CI Servers

- Vorteile
  - Einzelnes Repo für die Entwicklung (→ Höhere Effizienz)
  - Automatisiertes Staging möglich
  - Shift-Left-Ansatz möglich
    - https://github.com/adrienverge/yamllint
    - https://github.com/instrumenta/kubeval
    - https://github.com/helm/chart-testing
- Nachteile
  - Komplexität steckt im Detail
    - … oder eben in den CI Pipelines

Abschließende Herausforderungen

- GitOps Operator: 1-n (custom) Controllers
- Helm, Kustomize Controllers
- Operator für zusätzliche Tools
  - Beispielsweise Secrets
- Operators konsumieren Ressourcen
- Steile Lernkurve
- Error handling
  - Operators failen teilweise spät und „silently“
  - Monitoring und Alerting wichtig!

GitOps