



Tag 3: Docker, GitOps, Deployment-Strategien

10.07.2024, Daniel Krämer

© Copyright 2024 anderScore GmbH

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

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

Grundlagen von **GitOps**



*“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**.”*

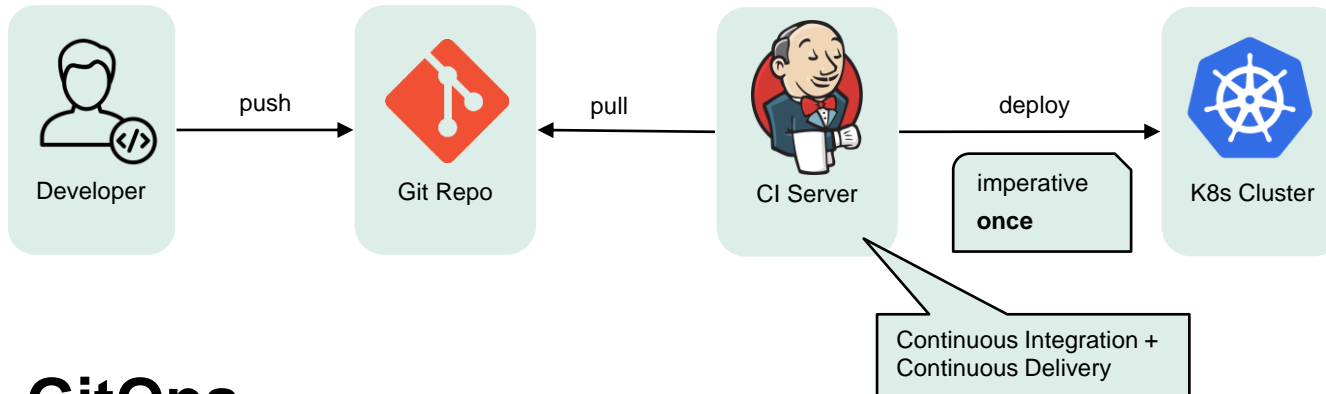
- Someone from GitLab



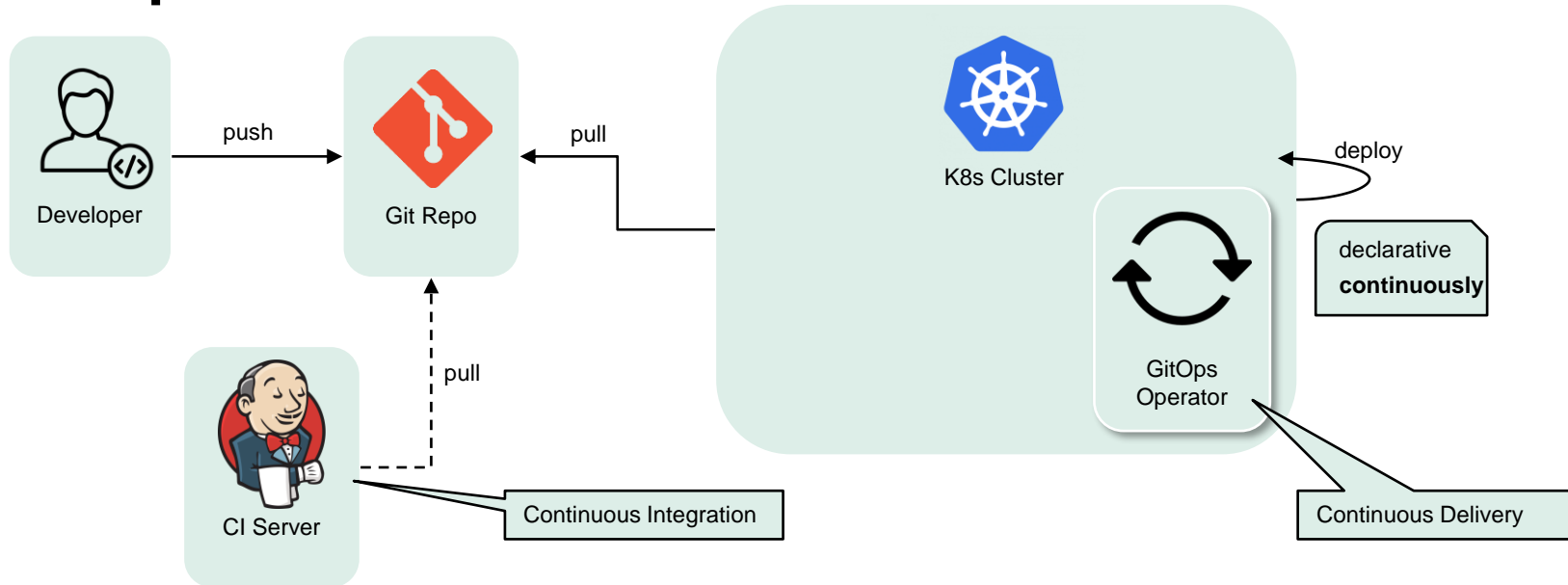
Grundprinzipien der Konfiguration

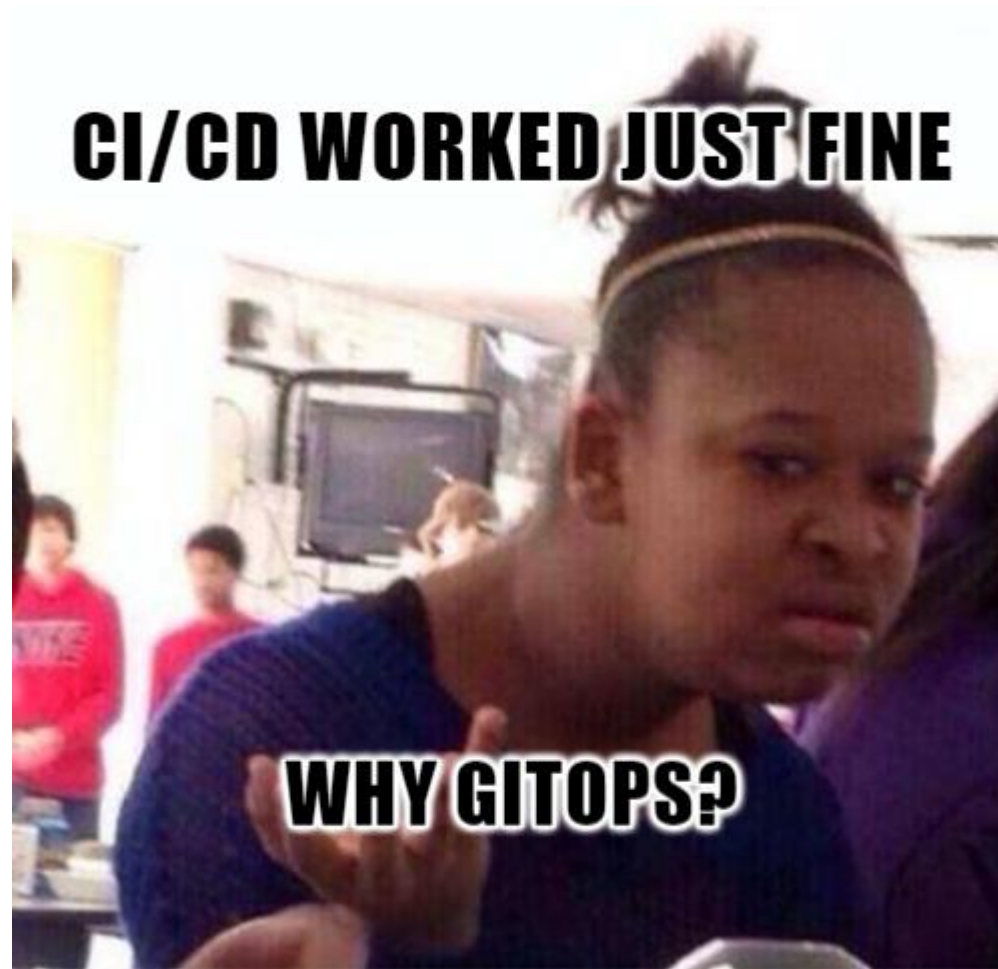
1. Deklarativ (statt programmatisch)
2. Versioniert und unveränderlich
3. Automatische Pulls
4. Kontinuierliche Anpassung

„Klassisches“ Continuous Delivery („CIOps“)



GitOps





Beobachtungen

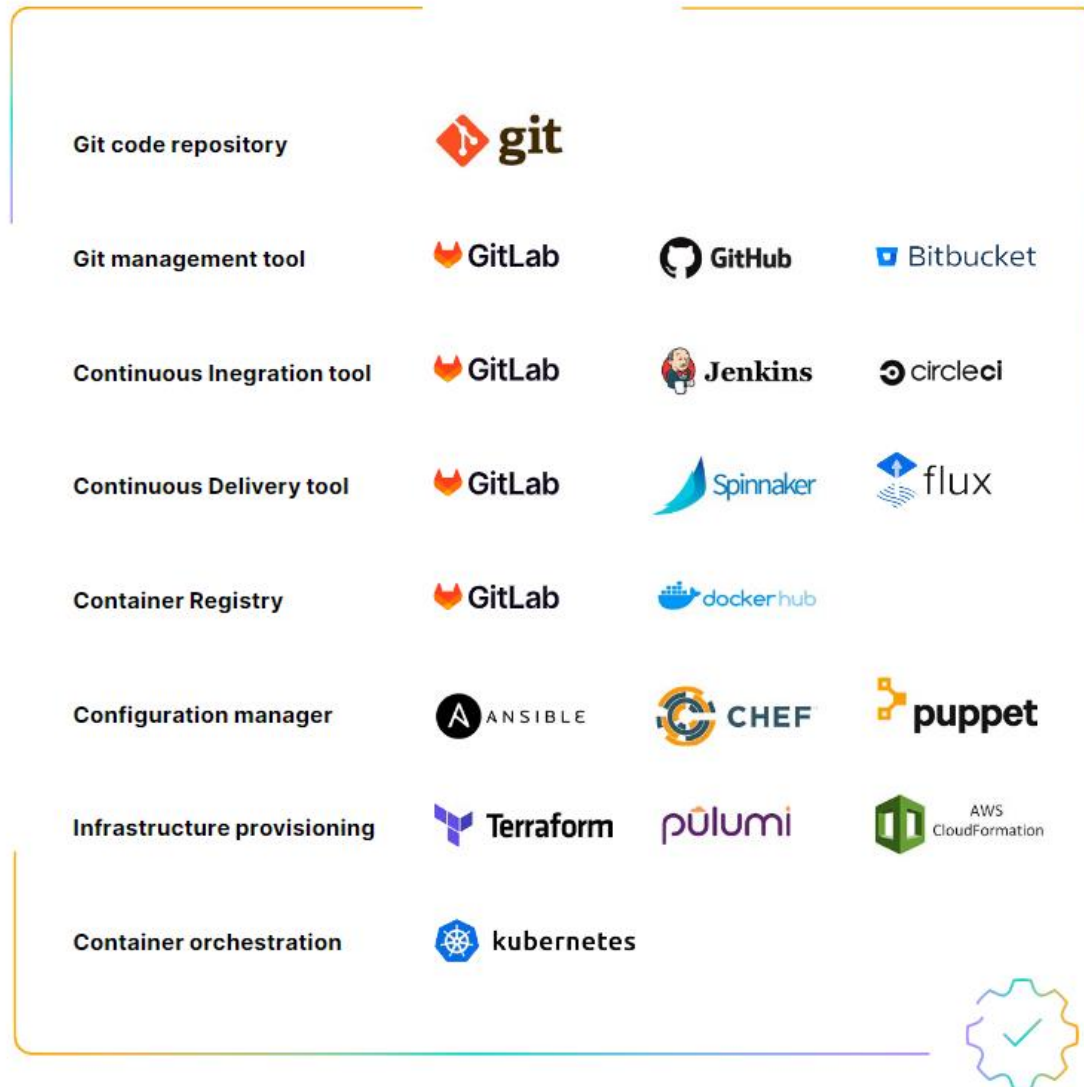
- Softwareentwicklungs-Lebenszyklus bereits automatisiert
 - Infrastruktur meist weitgehend manuell → benötigt spezialisierte Teams
- Cloud Native Apps
 - Zunehmender Fokus auf Geschwindigkeit und Skalierbarkeit
 - Trend: Infrastruktur in die Cloud
- Ziel: Infrastruktur-Provisionierung automatisieren
 - Verlagerung des Know Hows in die Pipelines
 - → Konsistente Infrastruktur; analog Deployables

Was genau ist das jetzt?

- Workflow zur Anwendungsbereitstellung
- Infrastruktur als Code (**laC**)
- Git Repositories als Single Source of Truth
- Umgebungen passen sich automatisch der aktuellen Konfiguration an

Was genau ist das jetzt?

- Vorgehen
 - CI-Prozess prüft eingeecheckten Code
 - CD-Prozess prüft Anforderungen und wendet diese an (IST vs. SOLL)
- Freie Toolwahl, um ein GitOps Framework aufzubauen
 - Git Repositories
 - CI/CD-Tools (wie Jenkins, Spinnaker, circleci, flux, Argo CD, ...)
 - Kubernetes, Virtuelle Maschinen
 - Konfigurationsmanagement (Ansible, Chef, puppet, Helm)



Quelle: gitlab-ebook

Warum GitOps?

- Framework für 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 Workflows um Operations
 - Deployment
 - Application Life Cycle Management
 - Infrastructure Configuration
- Änderungen im Git Repository nachverfolgbar
 - → Audit möglich

Warum GitOps?

- Dev(Ops)-Teams bestimmen eigene Geschwindigkeit
 - → Keine/kaum Wartezeit auf Ressourcen
 - Operations-Teams müssen keine Ressourcen zuweisen/genehmigen
- Änderungen sind transparent
 - Probleme schnell nachvollzieh- und reproduzierbar
 - → Sicherheit insgesamt verbessert!
- Up-to-date Audit Trail
 - Ungewünschte Änderungen schnell korrigierbar
- Codeänderungen von Entwicklung bis Produktion
 - Mehr Agilität bei Geschäfts- und Wettbewerbsveränderung

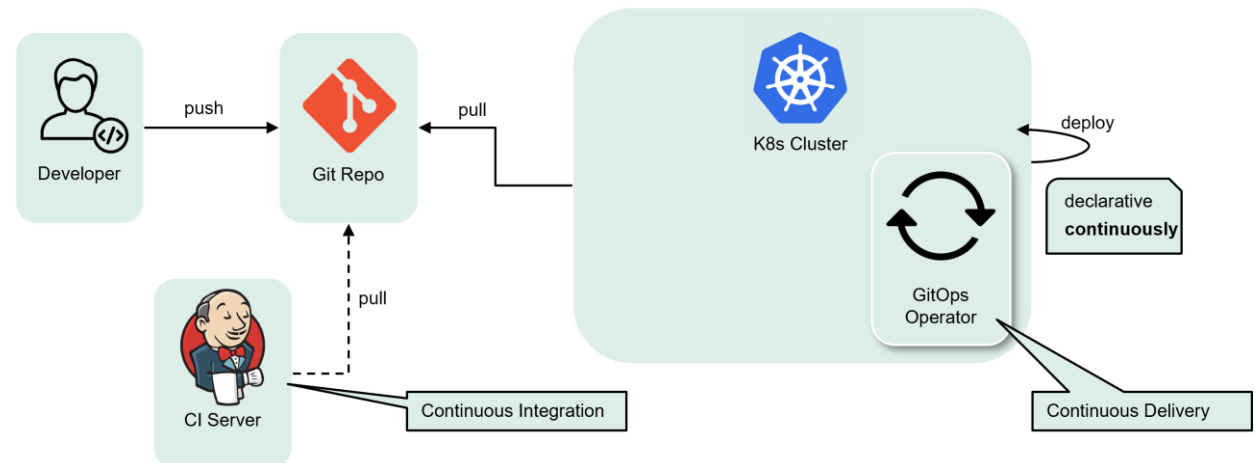
- Infrastruktur deklarativ managen
 - → Kubernetes und Cloud-native Softwareentwicklung
 - GitOps = Enabler für Continuous Deployment mit Kubernetes!
- Kubernetes ist **kein Muss!**
 - Andere Infrastruktur- und Deployment-Pipelines möglich
- Also... Mit GitOps lassen sich:
 - Development Pipelines erstellen
 - Anwendungen entwickeln
 - Konfigurationen verwalten
 - Kubernetes-Cluster bereitstellen und
 - Deployments auf Kubernetes oder Container-Registries vornehmen

GitOps-Workflow...?

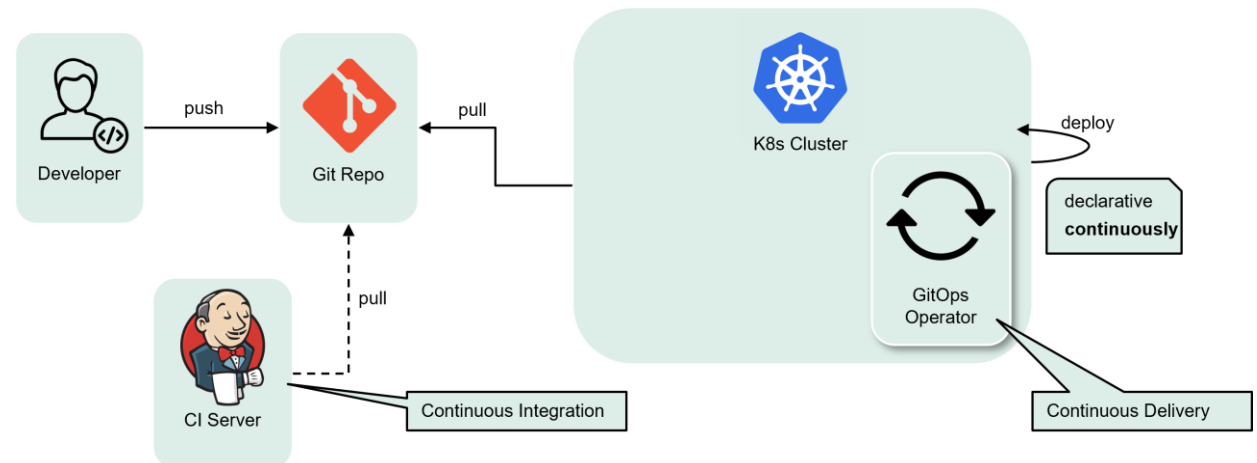
- Git als Versionskontrollsystem für IaC
- Teams nutzen weiterhin bekannte CI/CD Praktiken
- CI/CD-Pipelines durch externes Event ausgelöst
- Änderungen nur über Pull Requests (PR) oder Merge Requests (MR)
- Neues Deployment?! PR in git!
 - Ändert den deklarierten Zustand des Clusters
 - GitOps-Operator zwischen der GitOps-Pipeline und Orchestrierung (Kubernetes)

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
- Skalierbarkeit
- Self-healing

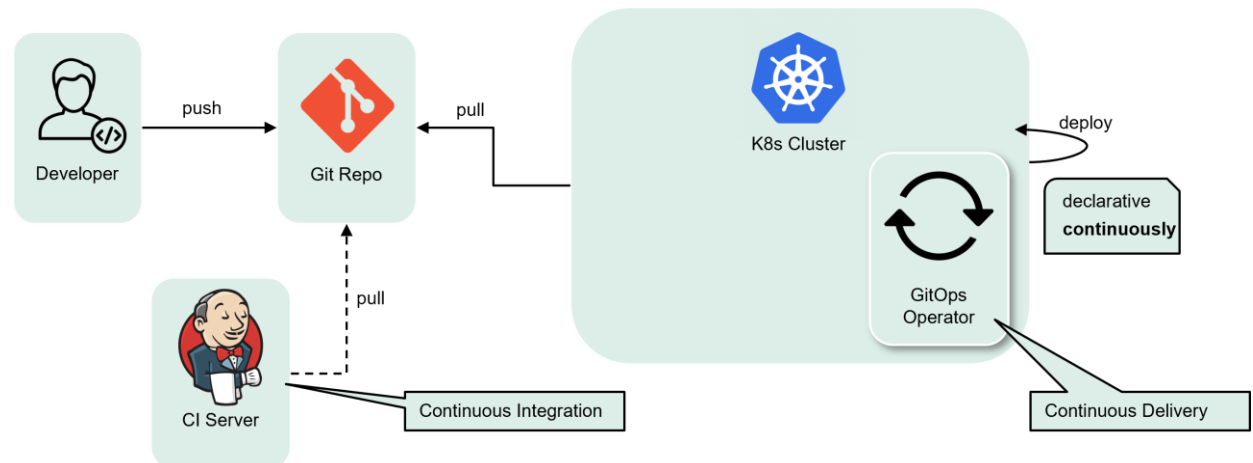


- Codeänderungen sind nachvollziehbar (git)



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

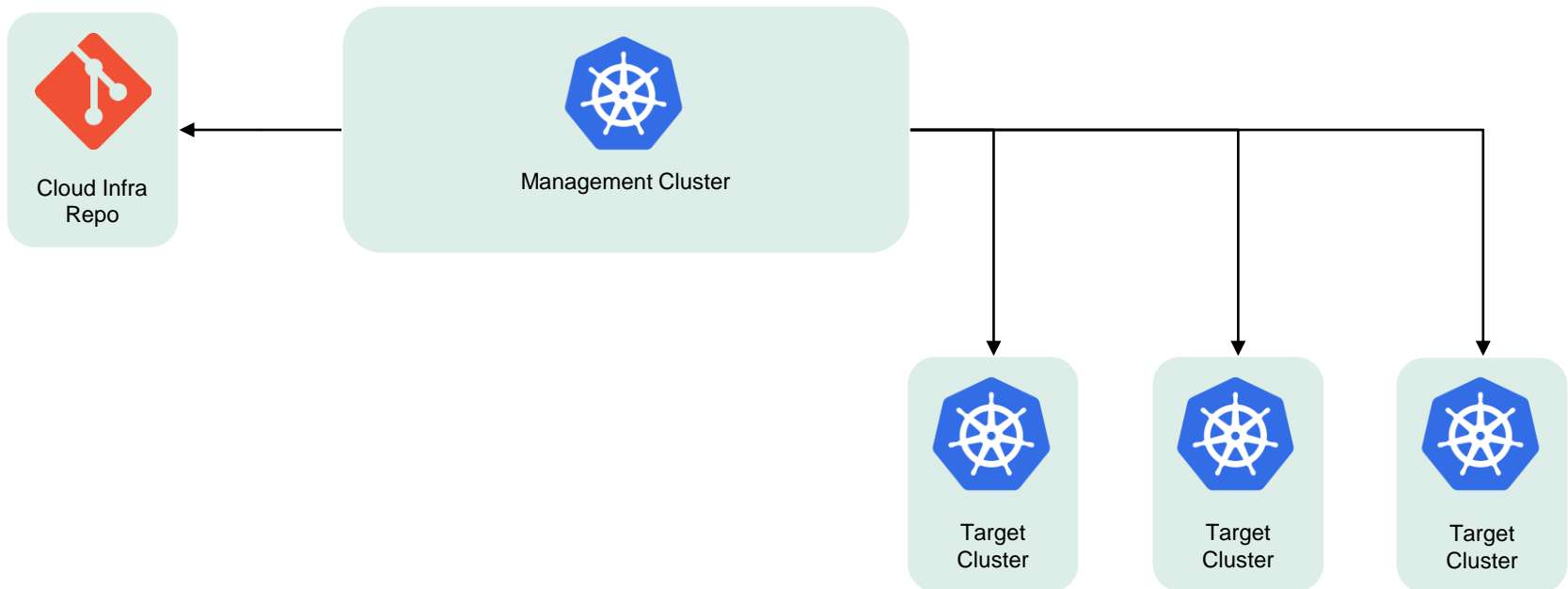


Secrets

- Bei CI/CD oft im CI Server hinterlegt...
- Besser: Secrets im Repository speichern
 - encrypted/sealed!
- Noch besser: Secrets → Key Management System (KMS)
 - Möglichkeiten
 - Cloud-Anbieter (AWS, Azure, Google, ...)
 - HashiCorp Vault
 - Kubernetes Integration
 - Operator, Container Storage Interface (CSI) Driver, Sidecar (Injector), Helm/Kustomize Plugin
 - GitOps Operator: nativer Support oder Plugin

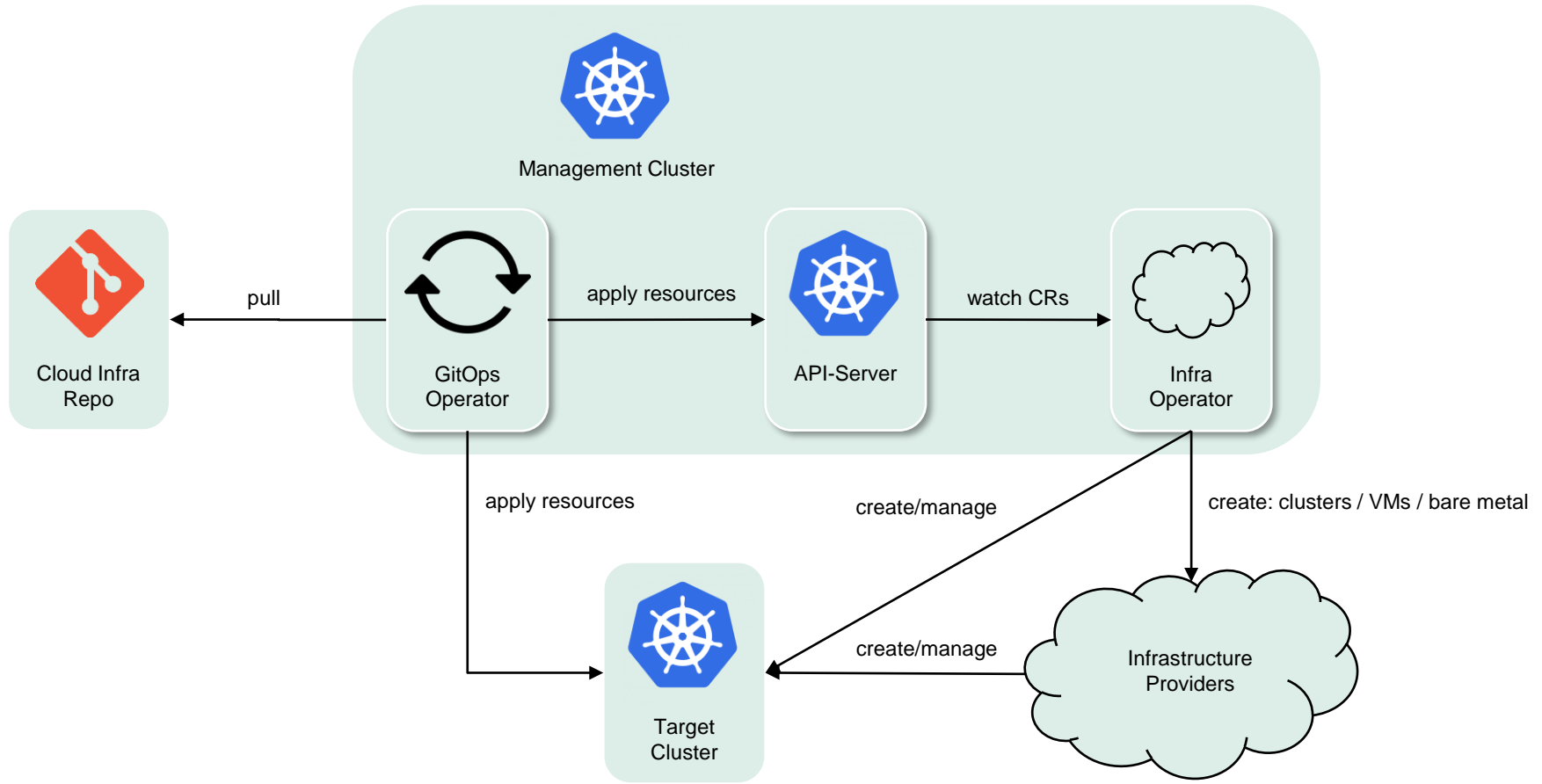
Erweiterte Einsatzbereiche

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



Im Management Cluster → GitOps umsetzen

Management Cluster



Best Practices

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

Lokale Entwicklung

- a) GitOps Operator und Git Server lokal deployen
 - Möglicherweise komplex

- b) Ohne GitOps Operator entwickeln
 - Möglich bei App und Infra Code im gleichen Repo
 - Validierung der Konfiguration offline

Staging Branches?

- dev-Branch nach Staging-Branch
 - ... main-Branch in Produktion
- Merging schnell kompliziert ☹️
 - ...pro Stage komplexer ☹️
- ➔ Generell abzuraten!

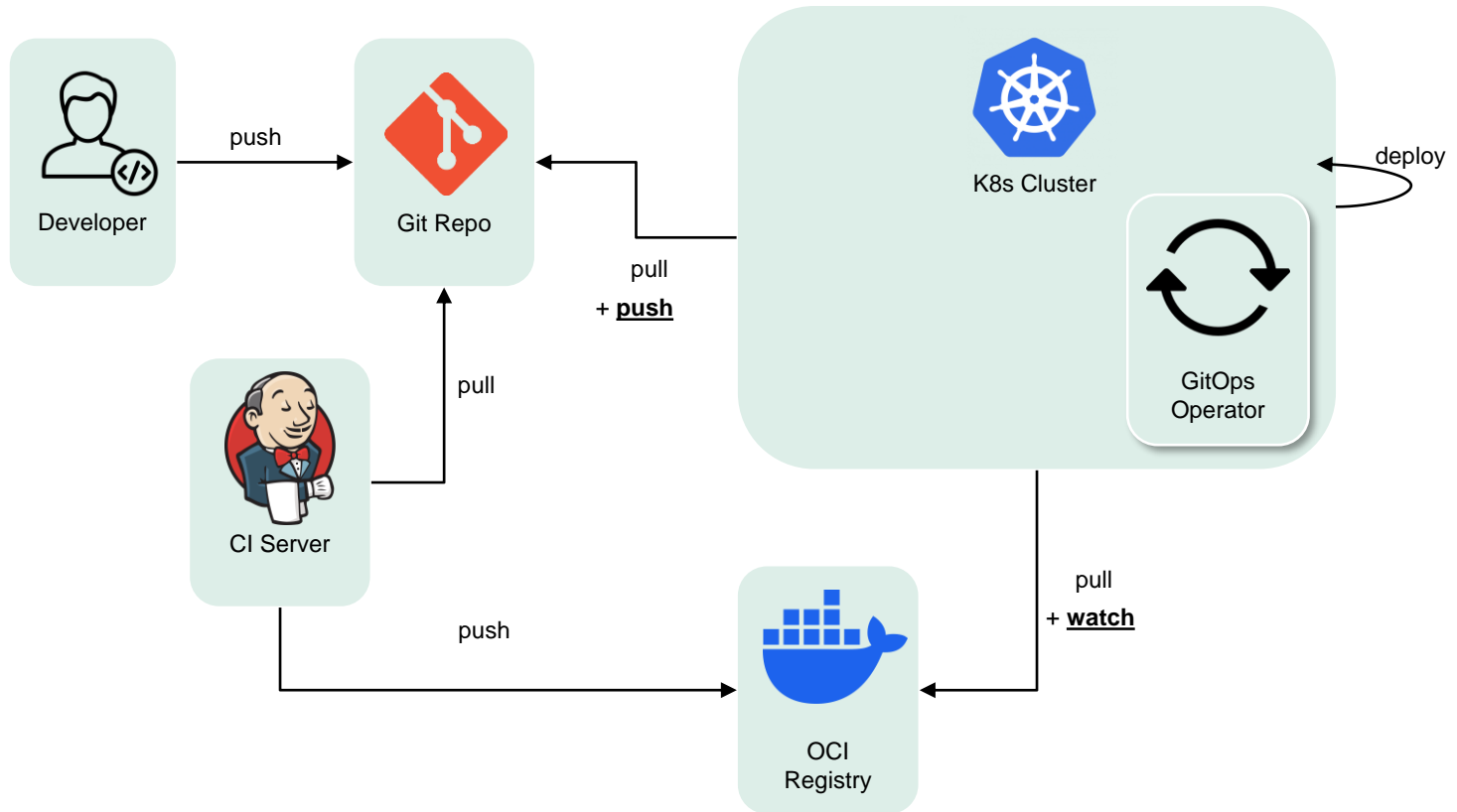
Staging Folders!

- 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
- Eventuell Duplikate pro Stage 😞
- Branching ist einfacher 😊
- Unterstützt beliebige Anzahl von Stages 😊

Übliche Rolle des CI Servers

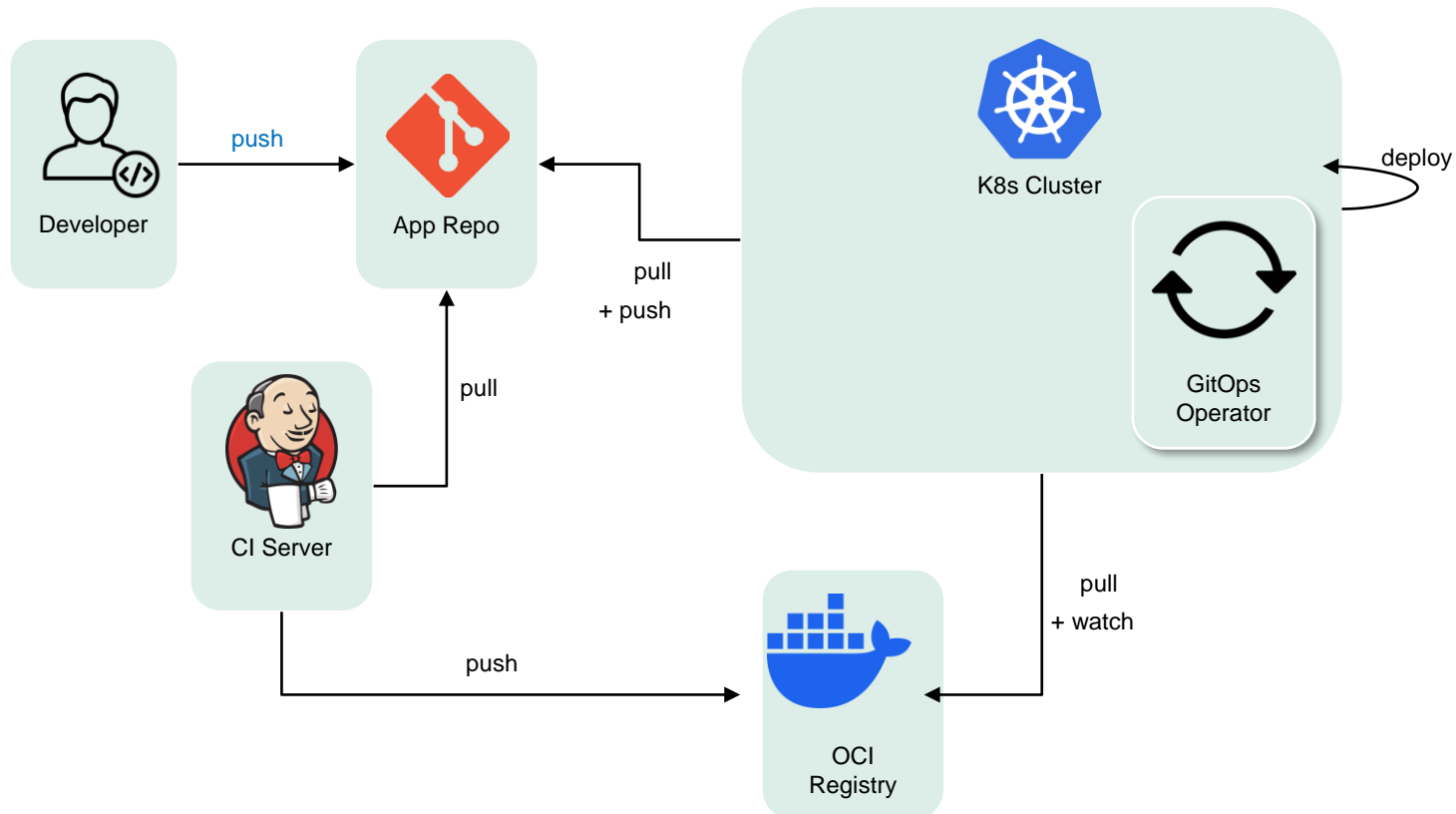


Optional: GitOps Operator aktualisiert Image Version in Git

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

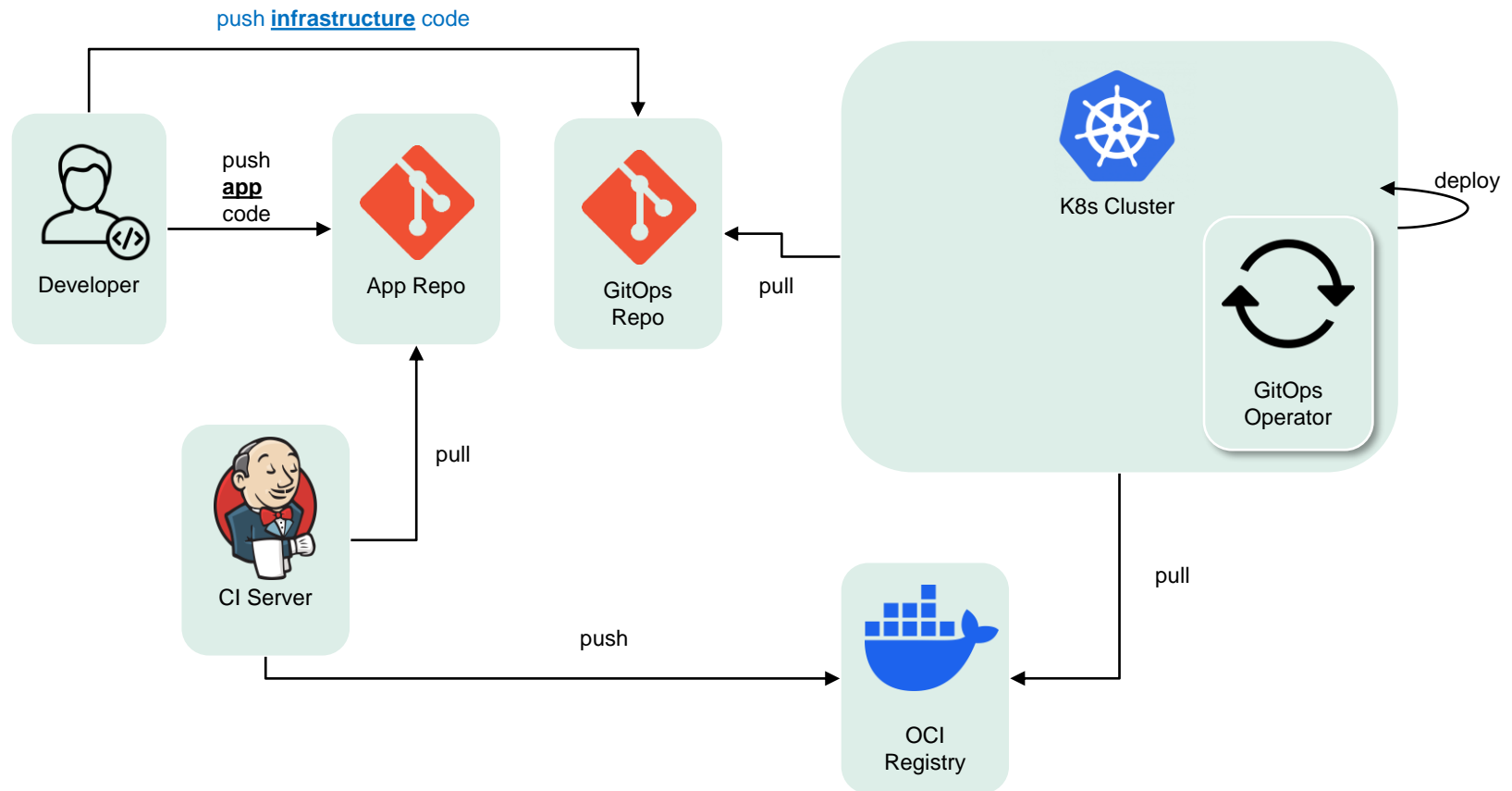
Anzahl der Repositories: Application vs. GitOps Repo

- Application Repo



Anzahl der Repositories: Application vs. GitOps Repo

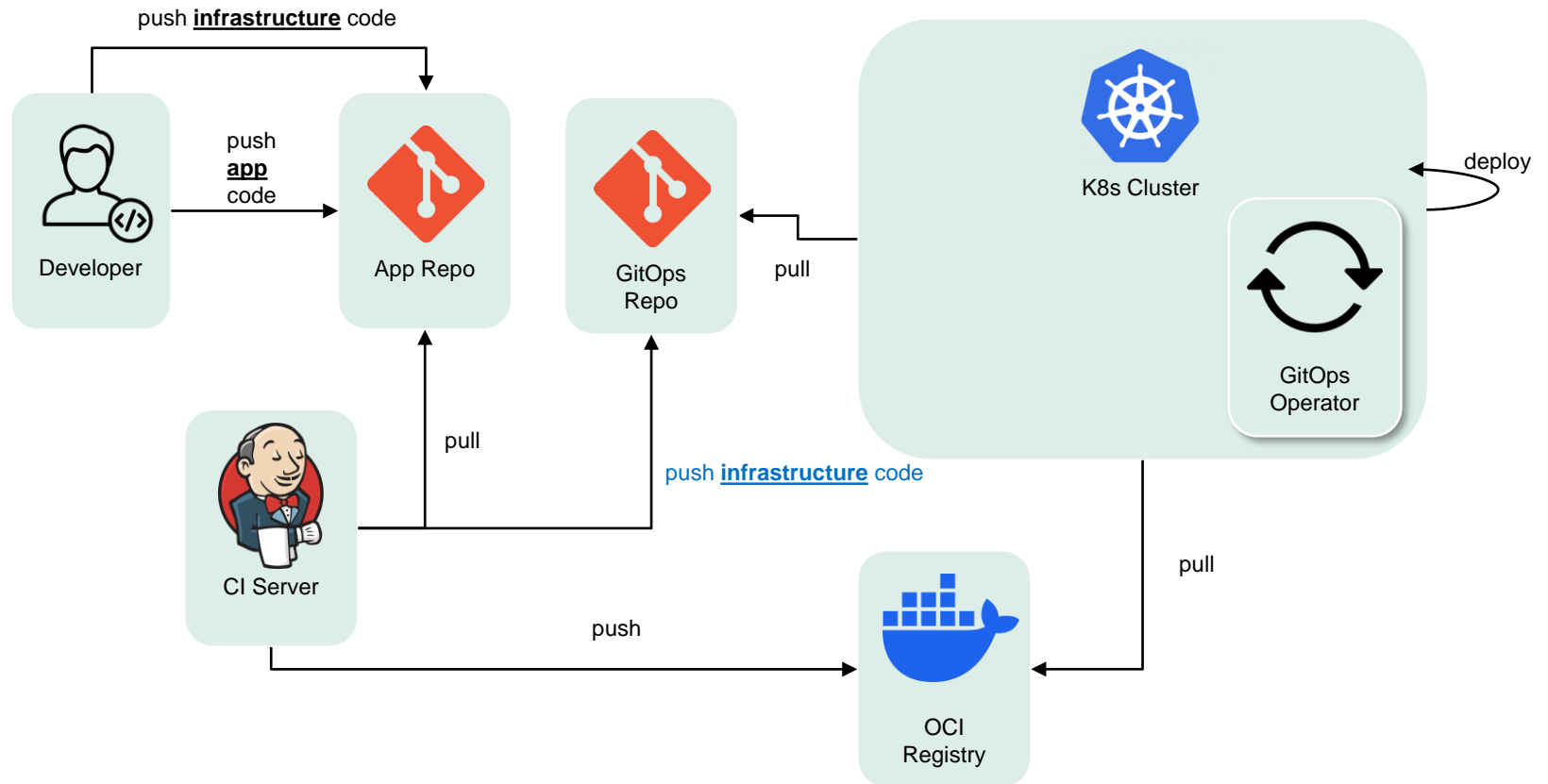
- GitOps Repo



Herausforderungen bei dediziertem GitOps Repo

- Mehrere Repos (konsistent) zu warten
- Refactorings und Tags schwerer
- Lokale Entwicklung komplizierter
- Shift-Left-Ansatz nur beim Anwendungscode
 - 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
 - Synchronisierung erforderlich (→ Konsistenz)
 - Komplexität steckt im Detail
 - ... oder eben in den CI Pipelines

Abschließende Herausforderungen

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



```
kubectl  
apply
```



```
git push
```

imgflip.com