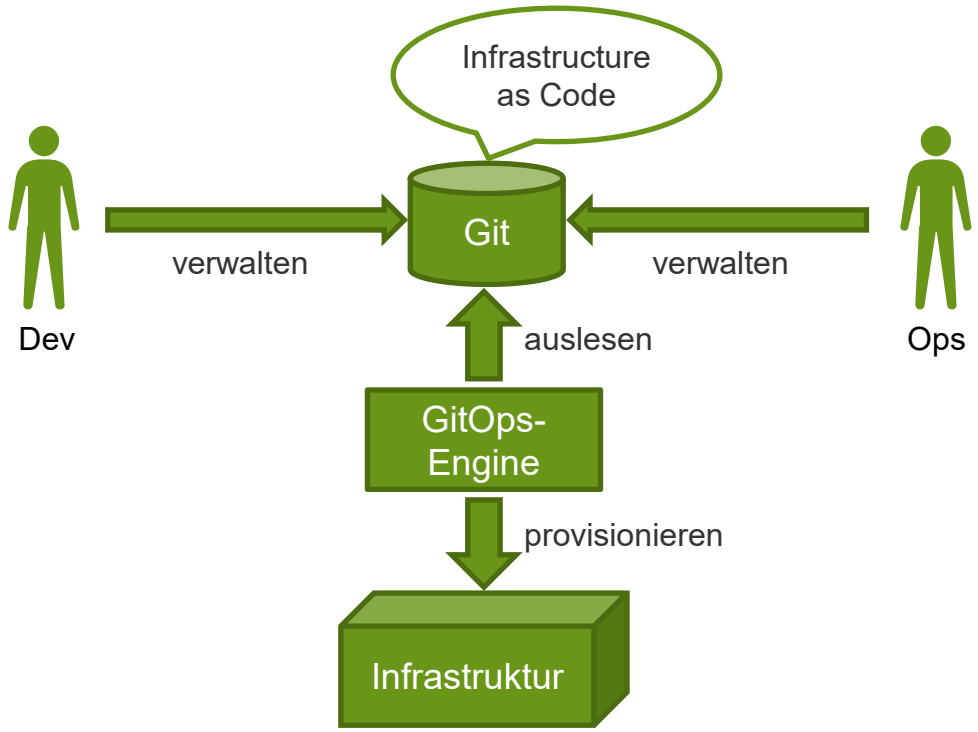
#GITOPS ALS DREH- UND ANGELPUNKT UNSERER CI/CD-PIPELINE

Klaus Dorninger
IT-Services der Sozialversicherung GmbH

#WARUM GITOPS?



Was ist GitOps?

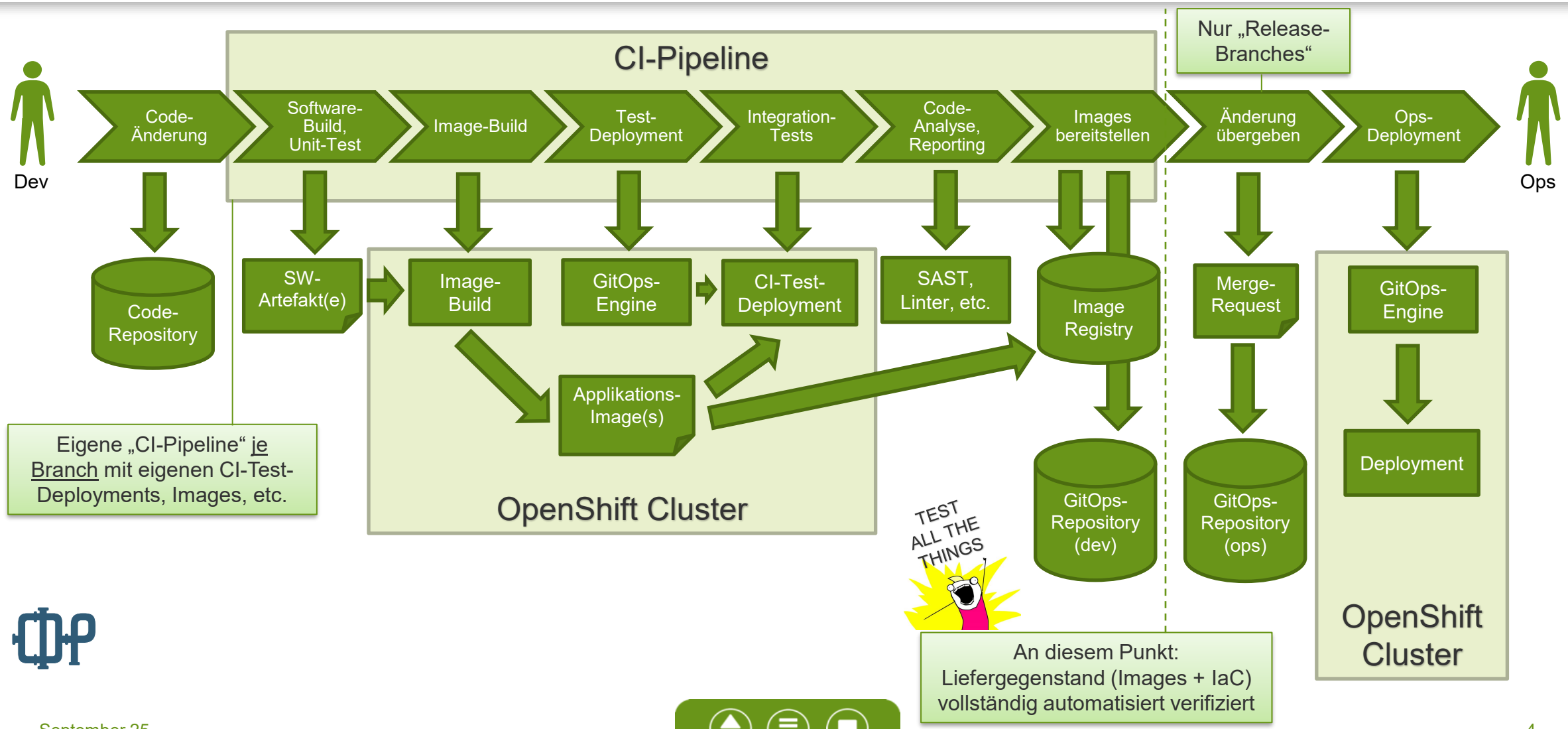
„Git als (Single-)Source-of-Truth von Infrastructure-as-Code (IaC) verwenden“
… sinnvollerweise mit Automatisierung

- Nachvollziehbarkeit
- Robustheit
- Skalierbarkeit
- Einfachheit

Was ist die Continuous Container Delivery Pipeline (CDP) ?

- Lösung für CI/CD von Eigenentwicklungen in der IT der SV
- Reproduzierbarer **Weg**,
  - den eine Software-Änderung durch Tools nimmt,
  - der identifizierbare, versionierte Artefakte erzeugt und in Betrieb nimmt,
  - der Container-Technologie dabei in allen Stufen zielführend einsetzt.
- Besteht aus
  - Konzepten
  - Werkzeugen
  - Dienstleistungen
  - Community

#CDP


CI-Pipeline
Nur „Release-Branches“
Dev
Code-Änderung
Software-Build, Unit-Test
Image-Build
Test-Deployment
Integration-Tests
Code-Analyse, Reporting
Images bereitstellen
Änderung übergeben
Ops-Deployment
Ops
Code-Repository
SW-Artefakt(e)
Image-Build
GitOps-Engine
CI-Test-Deployment
SAST, Linter, etc.
Image Registry
Merge-Request
GitOps-Engine
Applikations-Image(s)
Deployment
Eigene „CI-Pipeline“ je Branch mit eigenen CI-Test-Deployments, Images, etc.
OpenShift Cluster
GitOps-Repository (dev)
GitOps-Repository (ops)
TEST ALL THE THINGS
OpenShift Cluster
An diesem Punkt: Liefergegenstand (Images + IaC) vollständig automatisiert verifiziert

#REPOSITORY-DESIGN

Unsere Implementierung

- Anspruch: möglichst von CI bis in Produktion gleichförmige Vorgehensweisen und Werkzeuge einsetzen (Lieferqualität)
- Inhalt von GitOps-Repositories:
  - **nur** IaC (je Applikation), aber **vollständig** bzw. **hermetisch** (keine zusätzlichen externen Informationsquellen)
  - „Lieferstände“ (Release-Branches) und Umgebungsstände (Umgebungs-Branches)
- Modell
  - Abstrakte Basis **aller** umgebungs**un**abhängigen IaC-Teile (= Liefergegenstand) inkl. Image-Version
  - Konkrete Umgebungsanpassungen bzw. -definitionen in eigenen Verzeichnissen *je* Umgebung
  - CI: Dynamische Übergabe von Images und Namensteilen an GitOps-Engine
- Operationen: Merge
  - … ohne dass Umgebungsspezifika andere Umgebungen beeinflussen können (durch Trennung in Verzeichnisse)
  - … ohne dass umgebungsunabhängige Definitionen andere Umgebungen beeinflussen können (durch Branches)

#REPOSITORY-DESIGN

Warum Software-Code- und GitOps-Repository trennen?

- Was wird versioniert?
- Unterschiedliches Branching-Modell (Feature vs. Umgebung) möglich
  - Schnittstelle an „Release-Branches“
- Einfacher:
  - Änderungen in GitOps-Repository pushen ohne CI-Pipelines zu triggern
  - Staging durch Merge(-Request)
- Nachteile:
  - Synchronisation zwischen Repositories muss außerhalb gelöst werden
  - Kein Renovate in GitOps (direkt) möglich (iSv. CI-Pipelines triggern)

#REPOSITORY-DESIGN

Warum zwei GitOps-Repositories?

- Zielgruppen: Dev vs. Ops
- Klare Zuständigkeiten
- Entwicklung kann eigene „stehende Umgebungen“ (neben CI) selbstständig verwalten
- Einfache Berechtigungen in GitLab (Repository-Ebene vs. Branch-Ebene) mit vorhandener Gruppen-Struktur
- Einfaches Merge-Request-Branching-Modell dank Fork
  - … das aber in der Verwendung für ungeübte Developer durchaus herausfordernd sein kann

#WERKZEUGE

#WERKZEUGE

ArgoCD

- 2 „zentrale“ Instanzen:
  - Nach Zielgruppe: Dev, Ops
  - Verwalten potenziell mehrere Cluster (Ops)
  - RBAC
- Manifest Rendering
  - Ursprünglich mit Kustomize
  - Mittlerweile Tanka/Jsonnet

Tanka / Jsonnet – Warum?

- vs. Kustomize
  - „vars“ deprecated
  - Keine Libraries / Common-Funktionalität auslösbar
  - Kaum DRY
  - „YAML-Soup“ vs. einzelne Aspekte („welche Zeilen in welchen Files setzen E2E-TLS auf einer Route zu den Pods meines Deployments um?“)
  - Community Support ohne viel spürbares Feedback oder Fortschritt
- Passendes Modell („Environments“ für Umgebungen und „Library“ für umgebungsunabhängigen Teil)
- Mixins!
- Libraries mit Jsonnet-Bundler (auch hermetisch möglich)
- Tatsächliche (hermetische, funktionale) Programmierung möglich (bspw. Arithmetik)
- Explizites Definieren von „Schnittstellen“ zwischen Liefergegenstand und umgebungsspezifischer Konfiguration möglich
- Grafana Labs scheint mehr dahinter zu sein hinsichtlich Community Support

#WERKZEUGE

Tanka / Jsonnet – Wie?

- Eigener ArgoCD Plugin nötig
  - unsere Implementierung ist ein Runner mit ~170 Zeilen Go-Code inkl. Discovery und TLA-Support
- Eigene Jsonnet-Library „cdp-libsonnet“
  - einheitliche Metadaten aller Objekte analog Kustomize (labels, name-Suffix/Prefix)
  - Häufig verwendete Funktionen einfach zugänglich („Deployment für Komponente x“)
  - Querschnittliche Aspekte per Mixins mit „einer Code-Zeile“
    - „Serving-TLS-Certs an passender Stelle für Java-Applikation mounten“
    - „typische Probes für JBoss-EAP mit XP anbringen“
  - Semantische Abhängigkeiten zwischen Objekten nutzen
    - „Service für Deployment auf Port x“
    - „Route für Service“
  - Image-Versionen (= Version der Applikation) an einer Stelle im Repo festgeschrieben (in CI dynamisch übergeben)

#WERKZEUGE

SealedSecrets

- Operator von Bitnami
- Verschlüsseln Kubernetes-Secrets mittels Keys, die nur im Cluster dem Operator selbst zur Verfügung stehen
- → hermetische / vollständige Daten von Applikationen in Git
- → lesender Zugriff auf Git auch für Gruppen, die Secrets selbst nicht sehen dürfen, möglich
- Einfache Installation und Administration
- Lassen sich einfach aus vorhandenem Secret-Store/Parameter-Verwaltungssystem oder mittels CLI/API erzeugen
- Ohne Plugin in ArgoCD verwendbar

#WERKZEUGE

CI und Git

- GitLab
  - Bereits vorhanden im Rahmen der internen Toolchain
  - Merge-Requests und Forks möglich
- Jenkins
  - Multi-Branch-Pipelines für CI-Pipelines je Commit
- Eigenes CLI-Tool implementiert konkrete CI-Steps und Interaktionen mit
  - OpenShift API
  - Quay API
  - ArgoCD API
  - GitOps-Repository (bei Release-Builds)

Herzlichen Dank
für Ihre Aufmerksamkeit!