qaware.de

QA|WARE
SOFTWARE ENGINEERING

GitOps in Kubernetes

Lukas Buchner
lukas.buchner@qaware.de

Recap

Continuous Delivery - Definition

ContinuousDelivery

Martin Fowler
30 May 2013

Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.

martinfowler.com

Continuous delivery

From Wikipedia, the free encyclopedia

Continuous delivery (**CD**) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.[1] It aims at building, testing, and releasing software faster and more frequently. The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production. A straightforward and repeatable deployment process is important for continuous delivery.

Abgrenzung zu Continuous X

Continuous Integration (CI)

- Alle Änderungen werden sofort in den aktuellen Entwicklungsstand integriert und getestet.
- Dadurch wird kontinuierlich getestet, ob eine Änderung kompatibel mit anderen Änderungen ist.

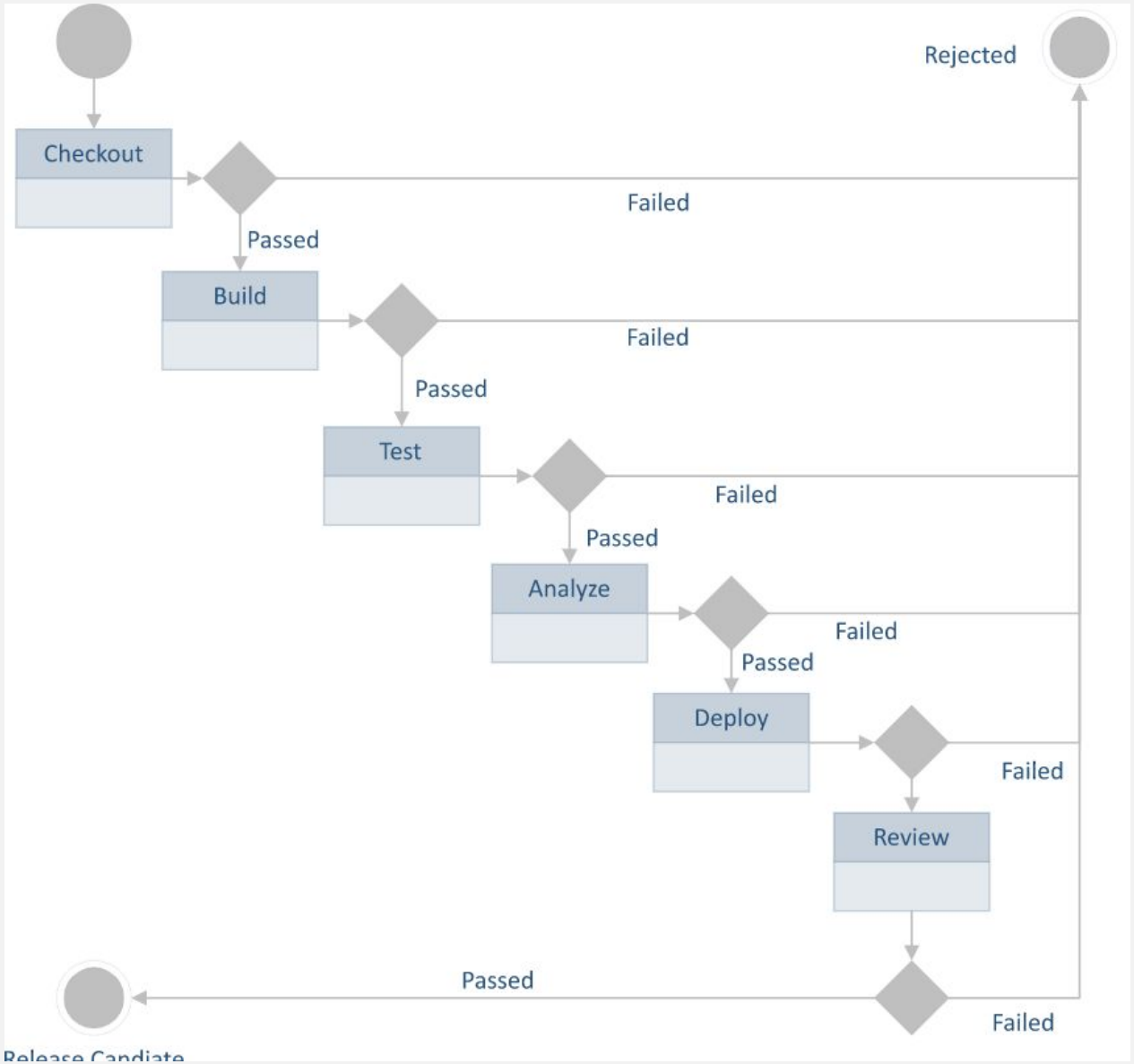
Continuous Delivery (CD)

- Der Code *kann* zu jeder Zeit deployed werden.
- Er muss aber nicht immer deployed werden.
- D.h. der Code muss (möglichst) zu jedem Zeitpunkt bauen, getestet und ge-debugged sein.

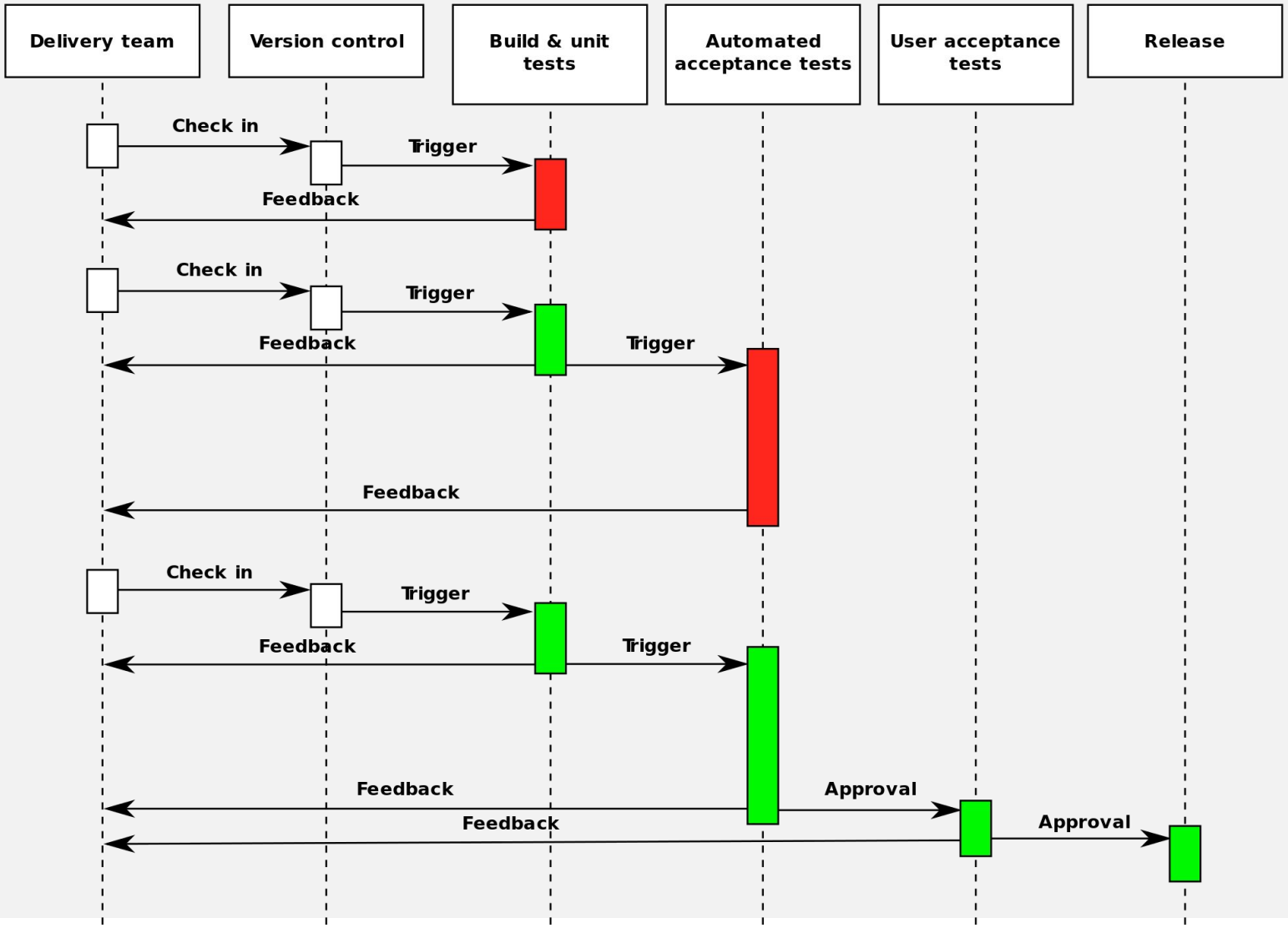
Continuous Deployment

- Jede stabile Änderung wird in Produktion deployed.
- Ein Teil der Qualitätstests finden dadurch in Produktion statt.

Beispiel: Continuous-Delivery-Pipeline

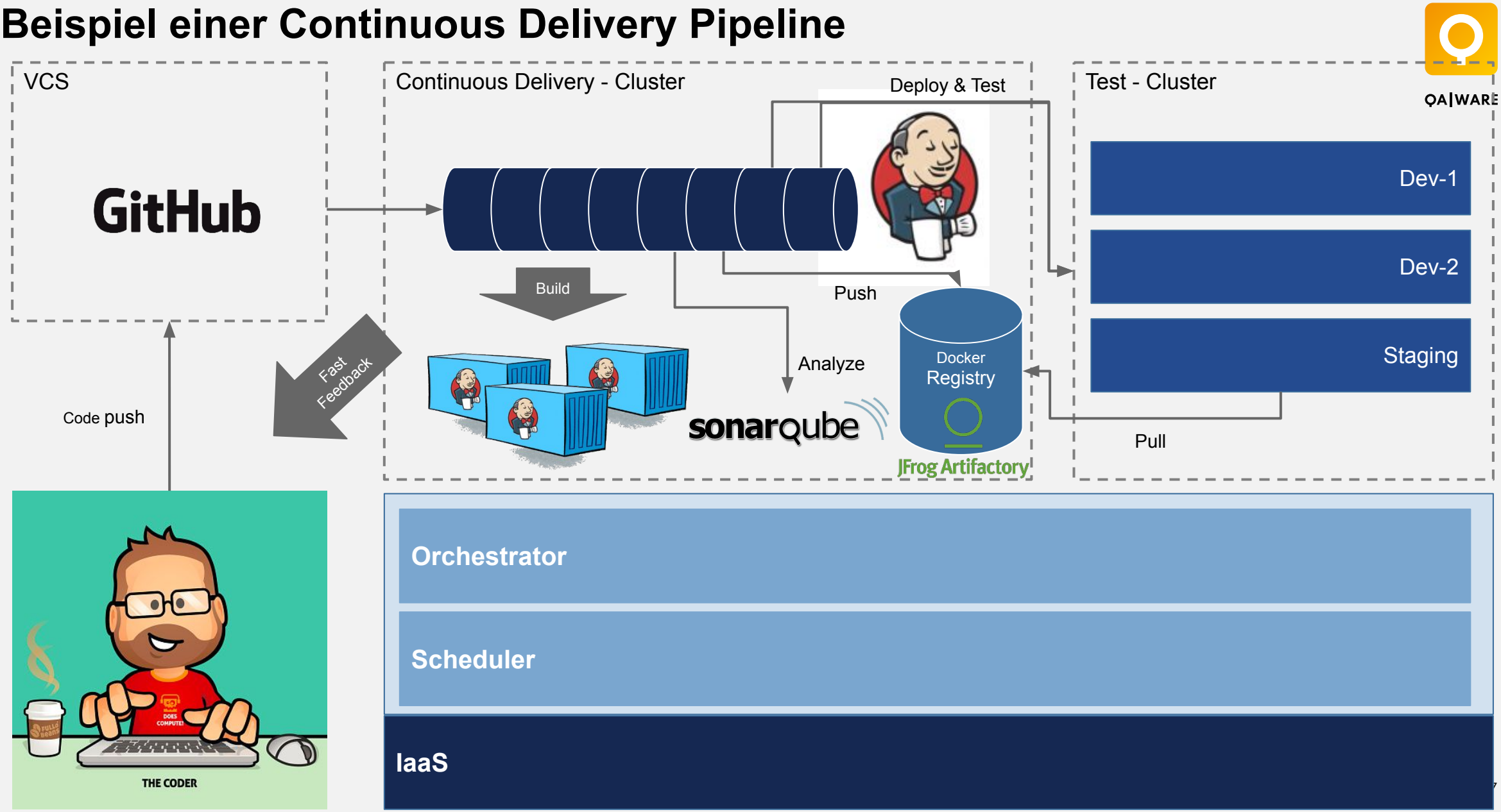


Wichtig ist ein schnelles Feedback an das Entwicklerteam, damit Fehler zügig behoben werden



„Continuous delivery process diagramm“
by Grégoire Détrez
Creative Commons 4.0

Beispiel einer Continuous Delivery Pipeline



GitOps

Die Idee

1 Declarative

A system managed by GitOps must have its desired state expressed declaratively.

2 Versioned and Immutable

Desired state is stored in a way that enforces immutability, versioning and retains a complete version history.

3 Pulled automatically

Software agents automatically pull the desired state declarations from the source.

4 Continuously reconciled

Software agents continuously observe actual system state and attempt to apply the desired state.

Die Vorteile

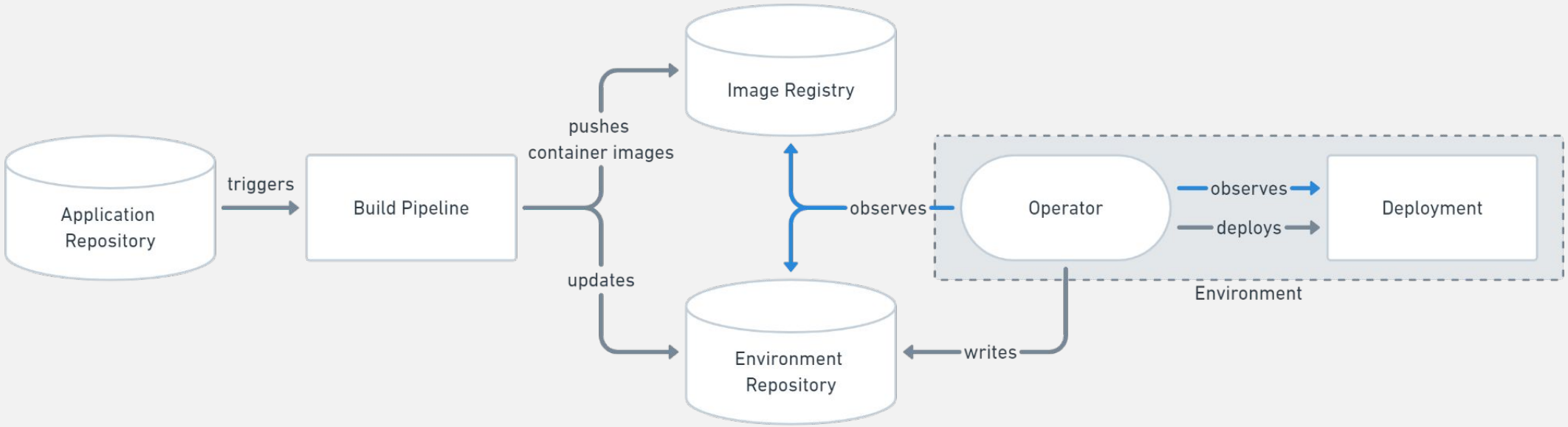
- Ermöglicht im Idealfall einen beliebigen Systemzustand in der Historie wiederherzustellen, e.g. einfacher Rollback
- Forciert Pipelines
- Bietet Transparenz von Änderungen, im Falle von Git ist auch ersichtlich wer etwas geändert hat
- Stellt sicher, dass der Systemzustand nicht vom Zielzustand abweicht

GitOps im K8s Umfeld

ArgoCD

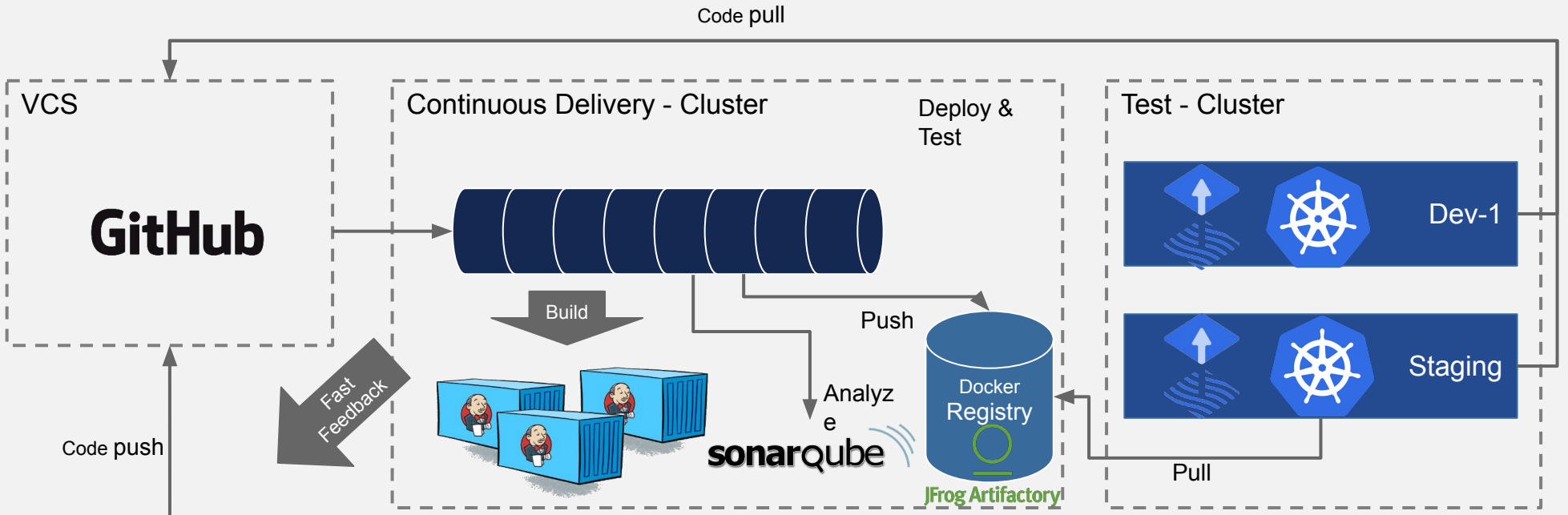
Flux

Pull-based deployments mit GitOps



https://www.gitops.tech/#what-is-gitops

GitOps mit Flux



Unterschiede in der Architektur

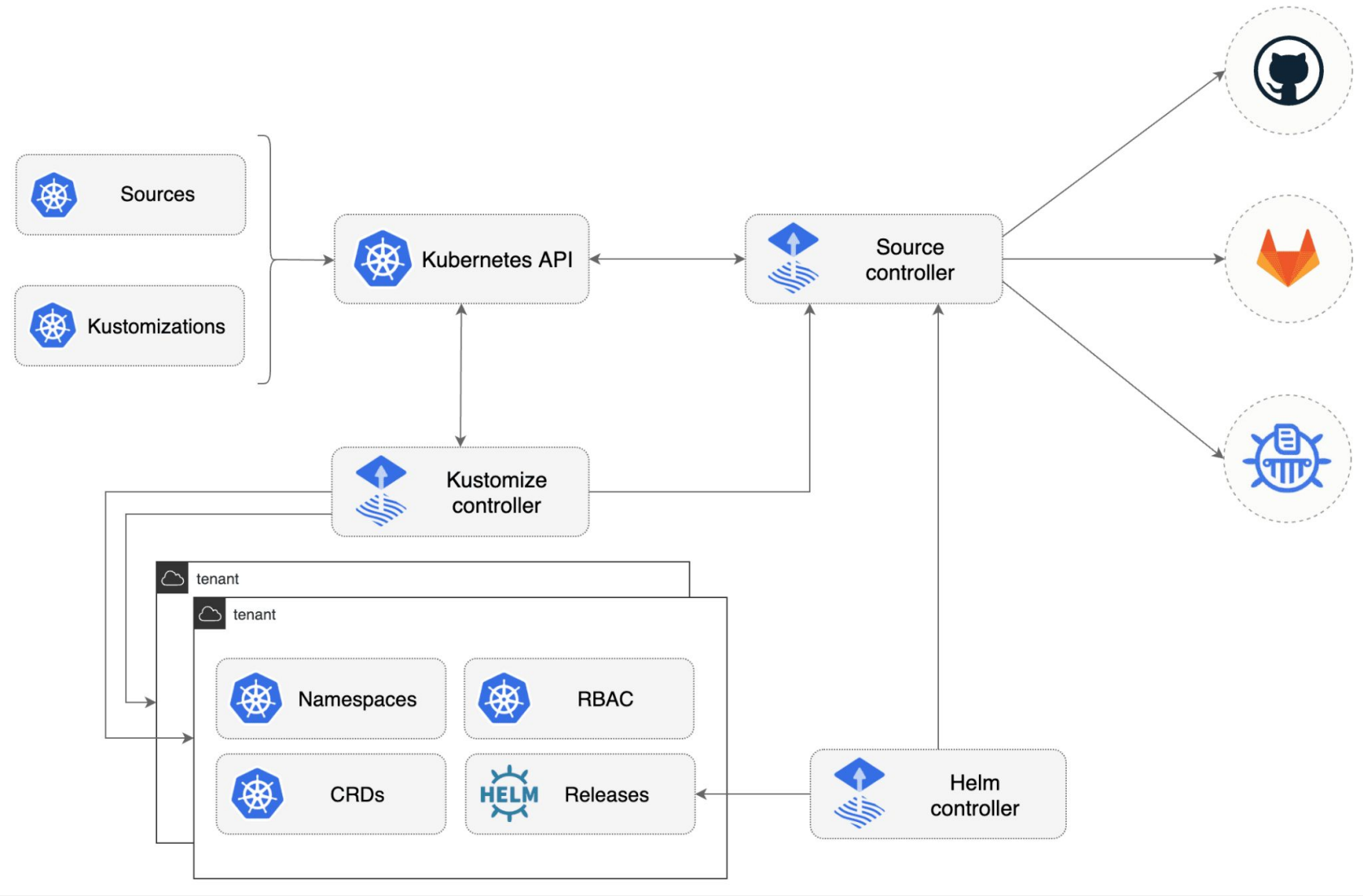
- CI / CD Pipeline braucht keinen cluster Zugriff mehr

  => verbesserte Security, da für die Pipeline kein high privilege credential mehr notwendig ist

- Cluster pullen jetzt aktiv Source-Repos, welche den Zielzustand enthalten
- Cluster reconcilen ihrer Zustand, d.h. es wird der Ist-Zustand mit dem Ziel-Zustand abgeglichen und versucht den Ziel-Zustand zu erreichen.

  => Cluster konvergieren automatisch gegen den Zielzustand

Flux im Detail



https://fluxcd.io/flux/components/

Flux im Detail

- Source-Controller
  - bindet verschiedene Quellen an z.B. Git-Repos, OCI-Repos etc.
  - stellt die heruntergeladenen Pakete den anderen Controllern zur Verfügung
- Helm-Controller
  - bietet eine Helm Integration für Flux. In Verbindung mit dem Source-Controller lassen sich Helm-Charts automatisch herunterladen und installieren/updaten
- Kustomize-Controller
  - nutzt die Git-Repos des Source-Controllers um kustomize auszuführen und die konfigurierten Ressourcen im Cluster zu applien. Alle Ressourcen werden getracked und können auch automatisiert wieder garbage collected werden.
- Notification-Controller
  - bietet insights in flux events z.B. Erfolge oder aufgetretene Fehler.
  - bietet Integrationen in Chat Systeme

Sources konfigurieren


```yaml
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: podinfo
  namespace: default
spec:
  interval: 5m0s
  url: https://github.com/stefanprodan/podinfo
  ref:
    branch: master
```

- konfiguriert ein **GitRepository** als Quelle im Source-Controller
- klont den **master** branch von **url** und stellt ihn den anderen controllern als tar.gz zur Verfügung
- pollt die **url** alle 5 Minuten und prüft, ob es Änderungen gibt

Kustomizations konfigurieren


```yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: podinfo
  namespace: default
spec:
  interval: 10m
  targetNamespace: default
  sourceRef:
    kind: GitRepository
    name: podinfo
  path: "./kustomize"
  prune: true
  timeout: 1m
```

- erstellt eine Kustomization im Kustomize-Controller, die über **sourceRef** das GitRepository vom Source-Controller referenziert
- der kustomize-Controller holt sich den Source-Code vom Source-Controller und applied mit kustomize (k8s-Tool) alles unter **path**

Flux in action!