



# Cloud Computing Continuous Delivery und GitOps

Felix Kampfer  
felix.kampfer@qaware.de



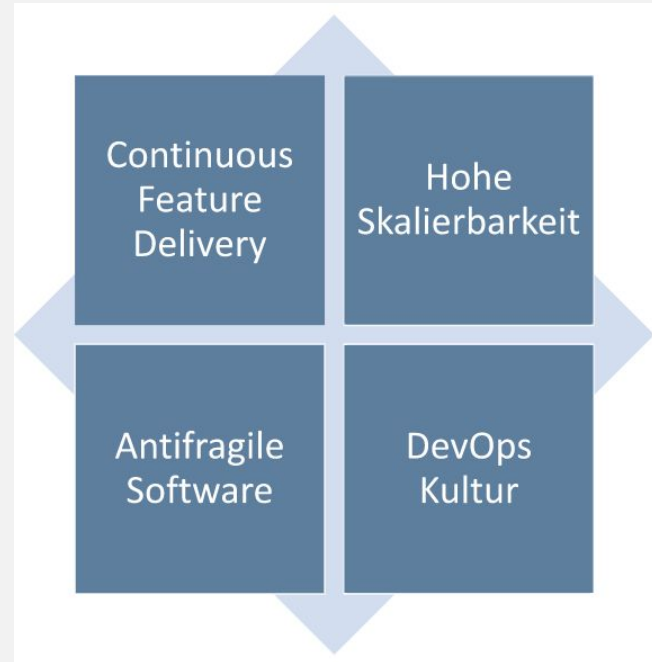
QA|WARE

# Continuous Delivery

# Treiber für Cloud-native Anwendungen



QA|WARE



# Cloud Native App Reifegradmodell

## Cloud Native

- Microservices architecture
- Domain- and API-driven design

## Cloud Resilient

- Fault-tolerant and **resilient design**
- Cloud-agnostic runtime implementation
- **Bundled metrics and monitoring (diagnosability)**
- Proactive failure testing (chaos testing)

## Cloud Friendly

- **12 factor-app-methodology**
- Horizontally scalable
- Leverages platform for high availability

## Cloud Ready

- No permanent disk access (within container)
- Self-contained application
- Platform-managed ports and networking
- Consumes platform-managed backing services



# Continuous Delivery - Definition



QA|WARE

## 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](http://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.<sup>[1]</sup> 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



QA|WARE

## 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.
- ➡ Die Möglichkeit, mit Fehlern umzugehen, muss vorhanden sein (z.B. Canary Releases)

# Kriterien für Continuous Delivery



QA|WARE

“You’re doing continuous delivery when:

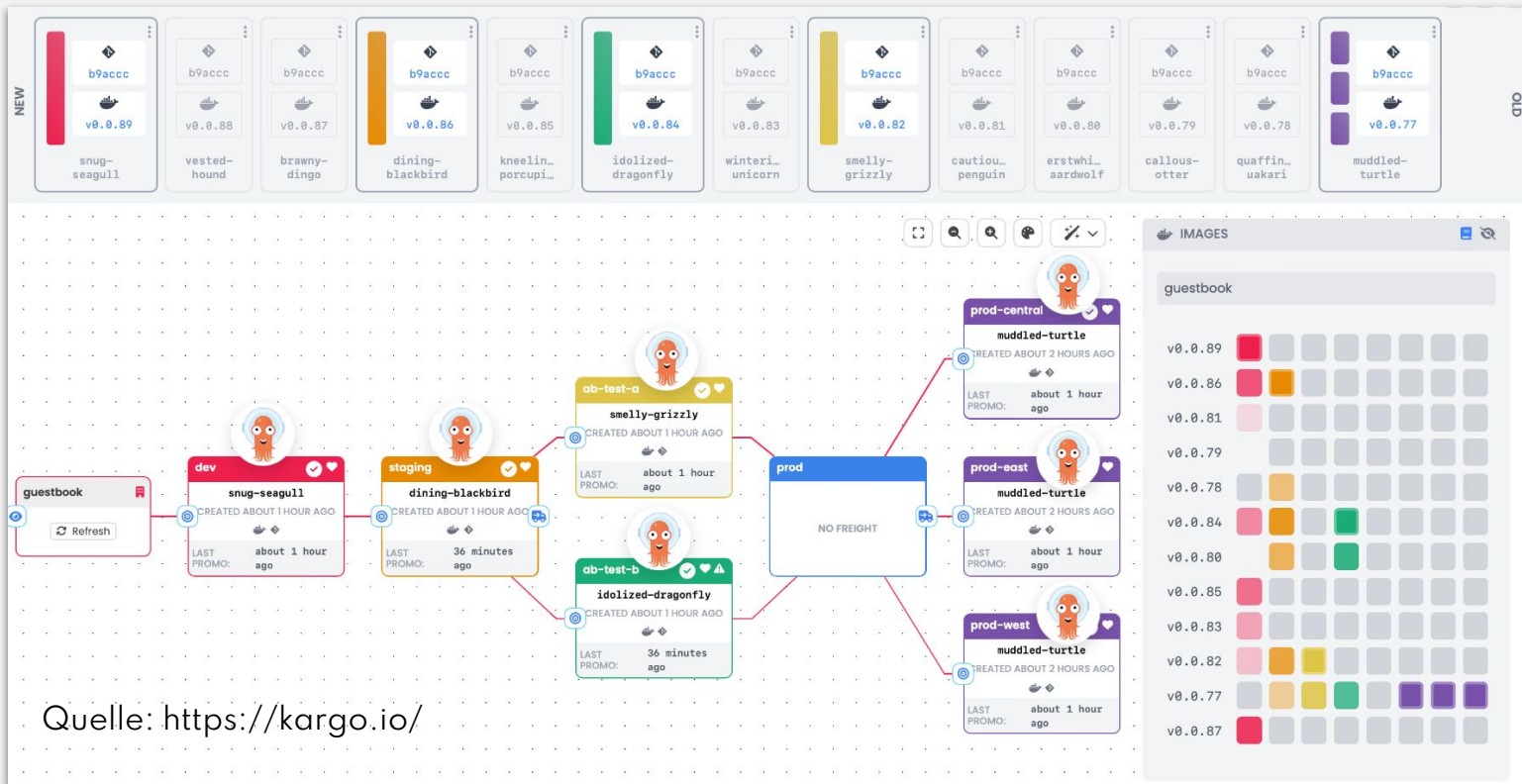
- Your software is deployable throughout its lifecycle
- Your team prioritizes keeping the software deployable over working on new features
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them
- You can perform push-button deployments of any version of the software to any environment on demand”

nach M. Fowler / Continuous Delivery working group at ThoughtWorks

# “Push-Button Deployment”



QA|WARE





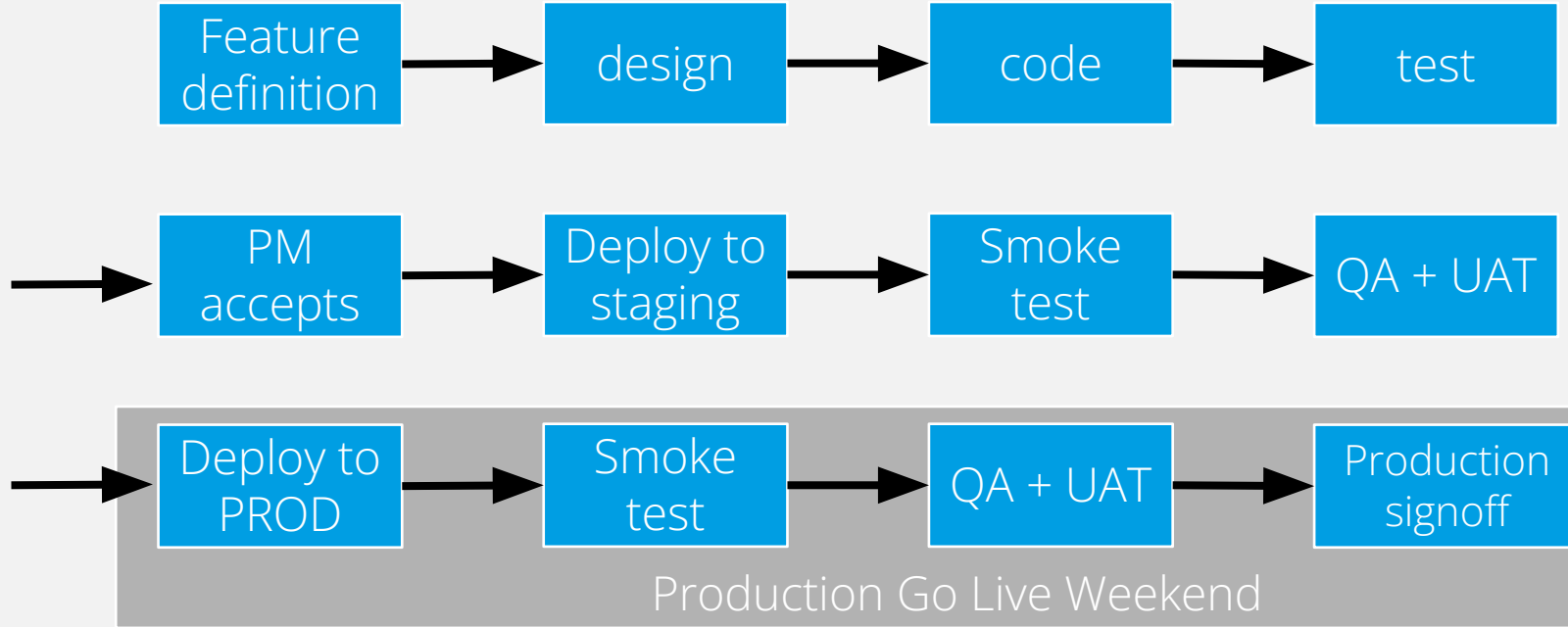
# Pipelines



# Beispiel: Entwicklung ohne Continuous Delivery



QA|WARE

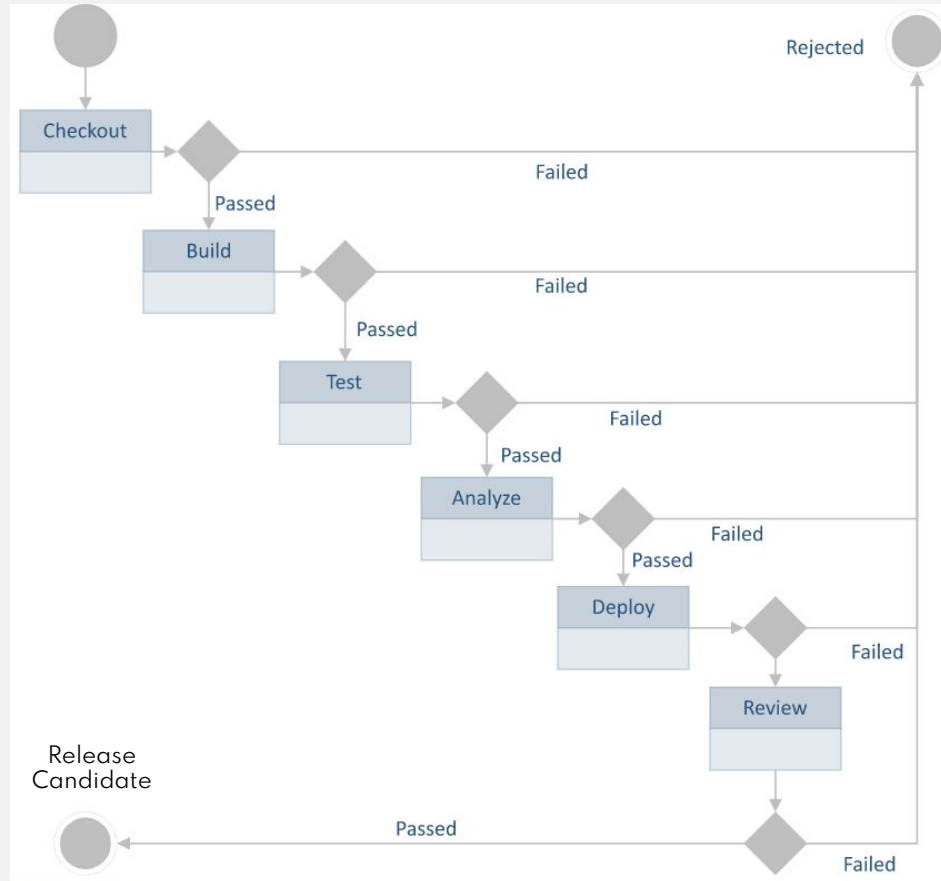


<https://www.scaledagileframework.com/continuous-delivery-pipeline/>

# Beispiel: Continuous-Delivery-Pipeline



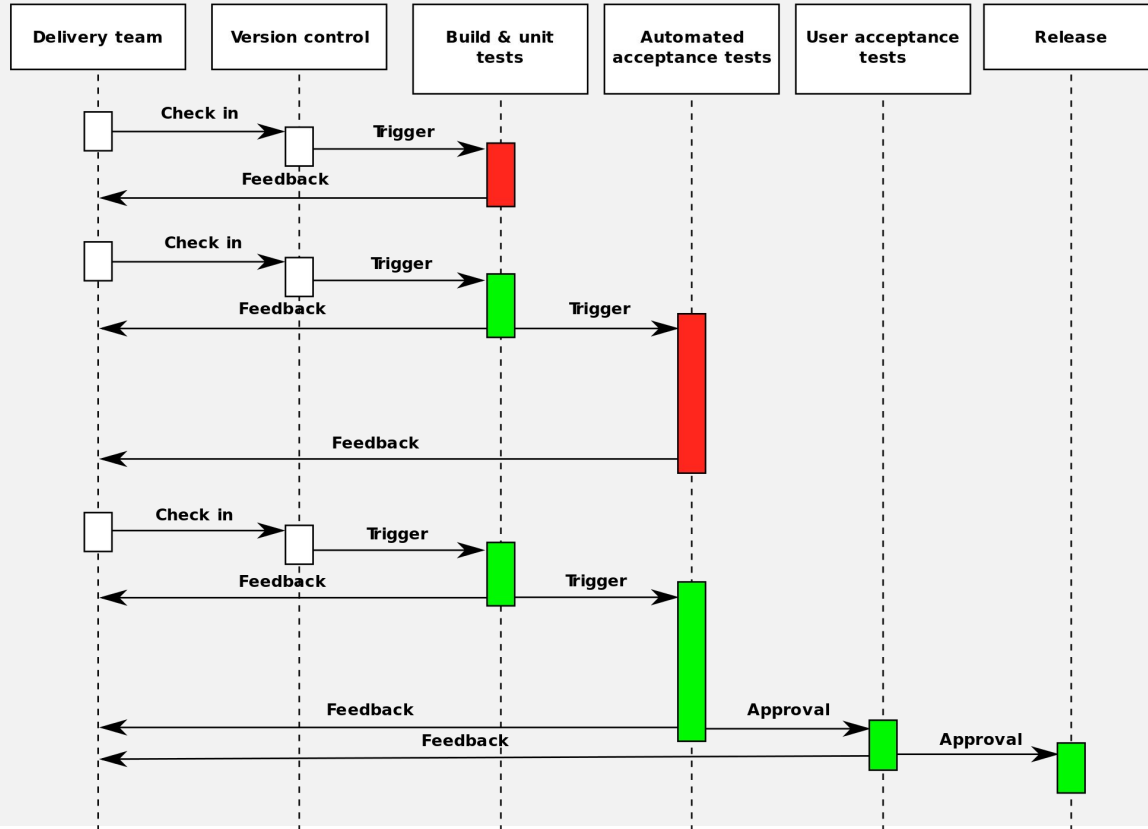
QA|WARE



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



QA|WARE



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

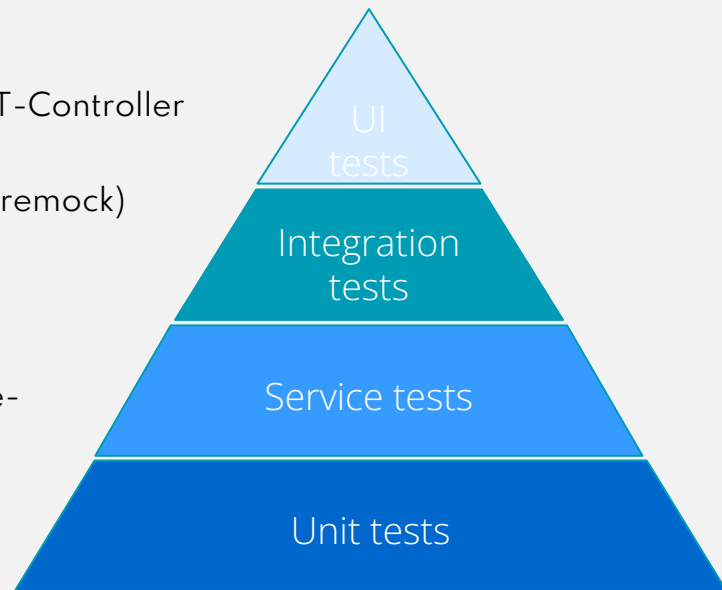
# Beispiel-Aufbau einer Test-Pyramide für sofortiges Feedback bei Fehlern



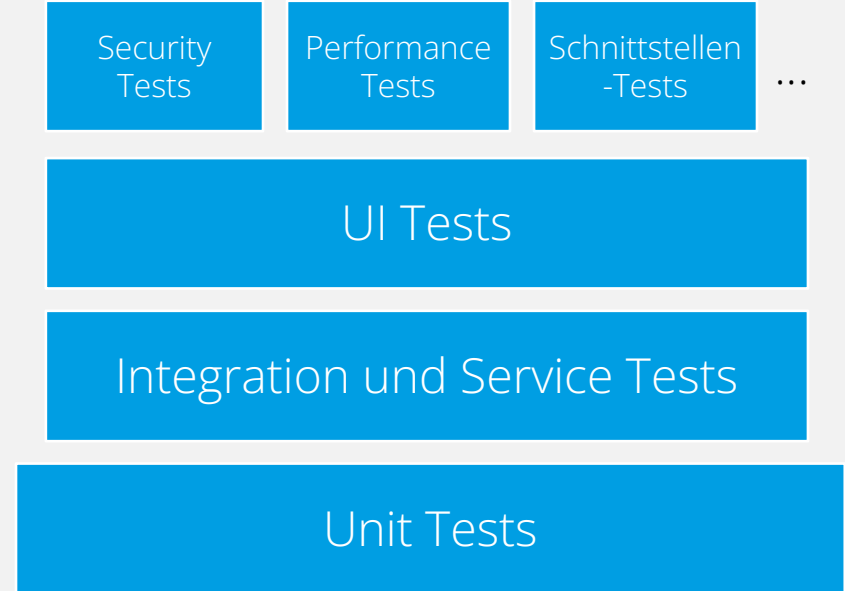
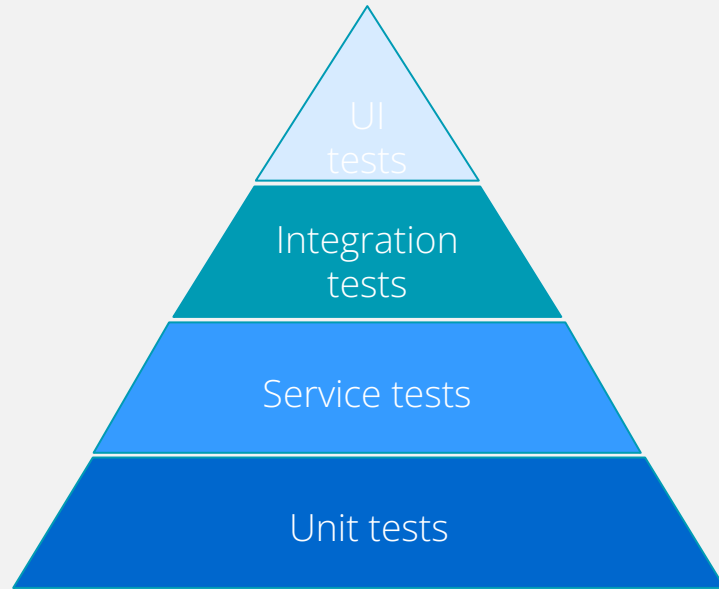
QA|WARE

- Unit Tests: Die klassischen Unit Tests (z.B. JUnit, Mockito)
- Service Tests: Tests eines einzelnen Microservices, inkl. der REST-Controller und Client-Calls (z.B. JUnit, Spring MVC Tests, Wiremock)
  - Mocks der anderen Microservices notwendig (z.B. mit Wiremock)
- Integration Tests:
  - Testet die Integration mehrerer Services und deren Interaktion (z.B. JUnit, Spring MVC Tests)
  - Performance Tests: Testet, ob es signifikante Performance-Änderungen gibt (z.B. Gatling)
- UI-Tests: Testet die UI-Funktionalität und deren Zusammenspiel mit dem Backend (z.B. Selenium, Protractor)

Alle Tests sollten so oft wie möglich ausgeführt werden.  
Idealerweise bei jedem Commit!

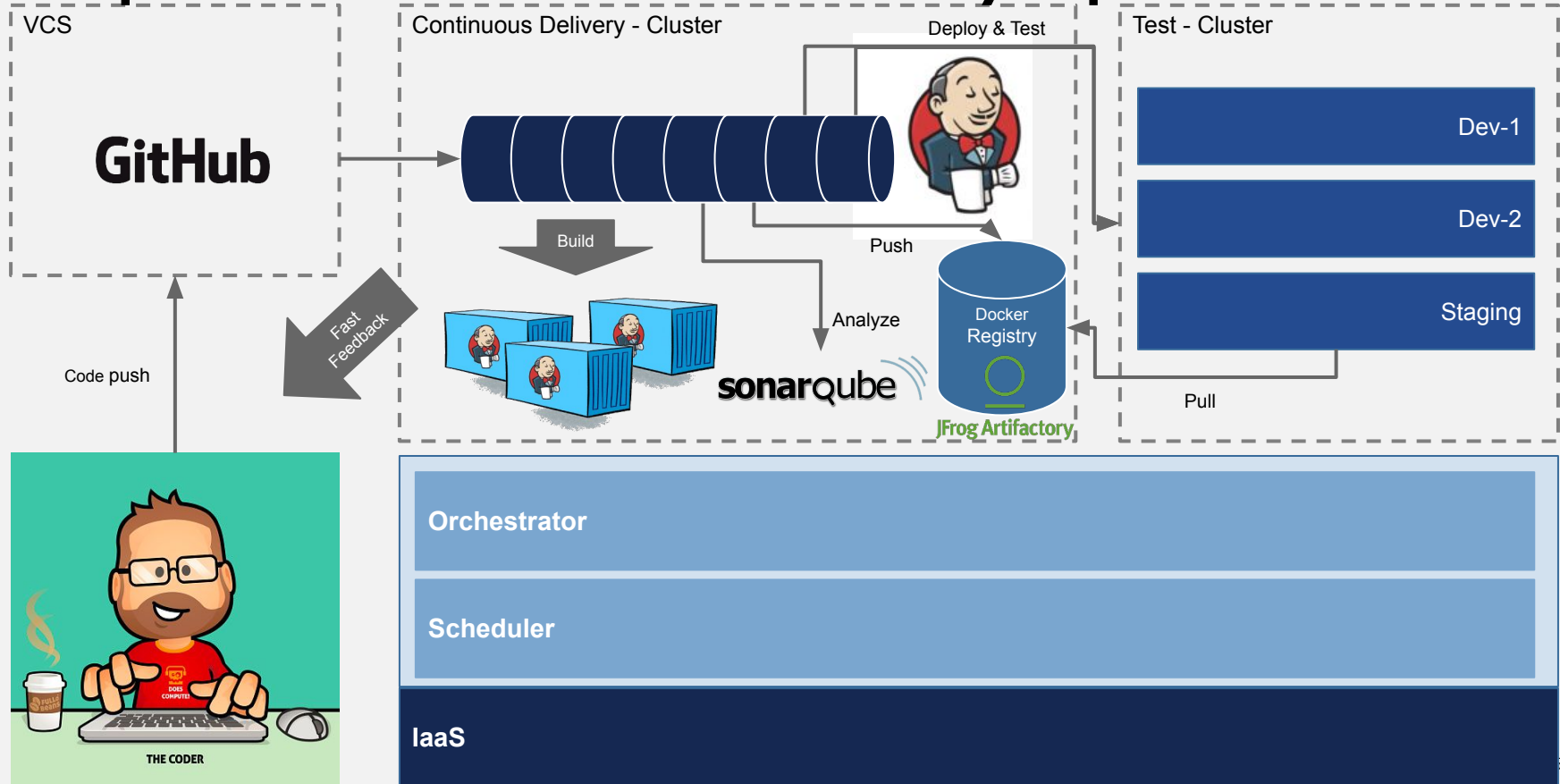


**Alle Testebenen sind heute gut automatisierbar.  
Anstelle der klassischen Testpyramide kann die  
Verteilung auch so aussehen:**





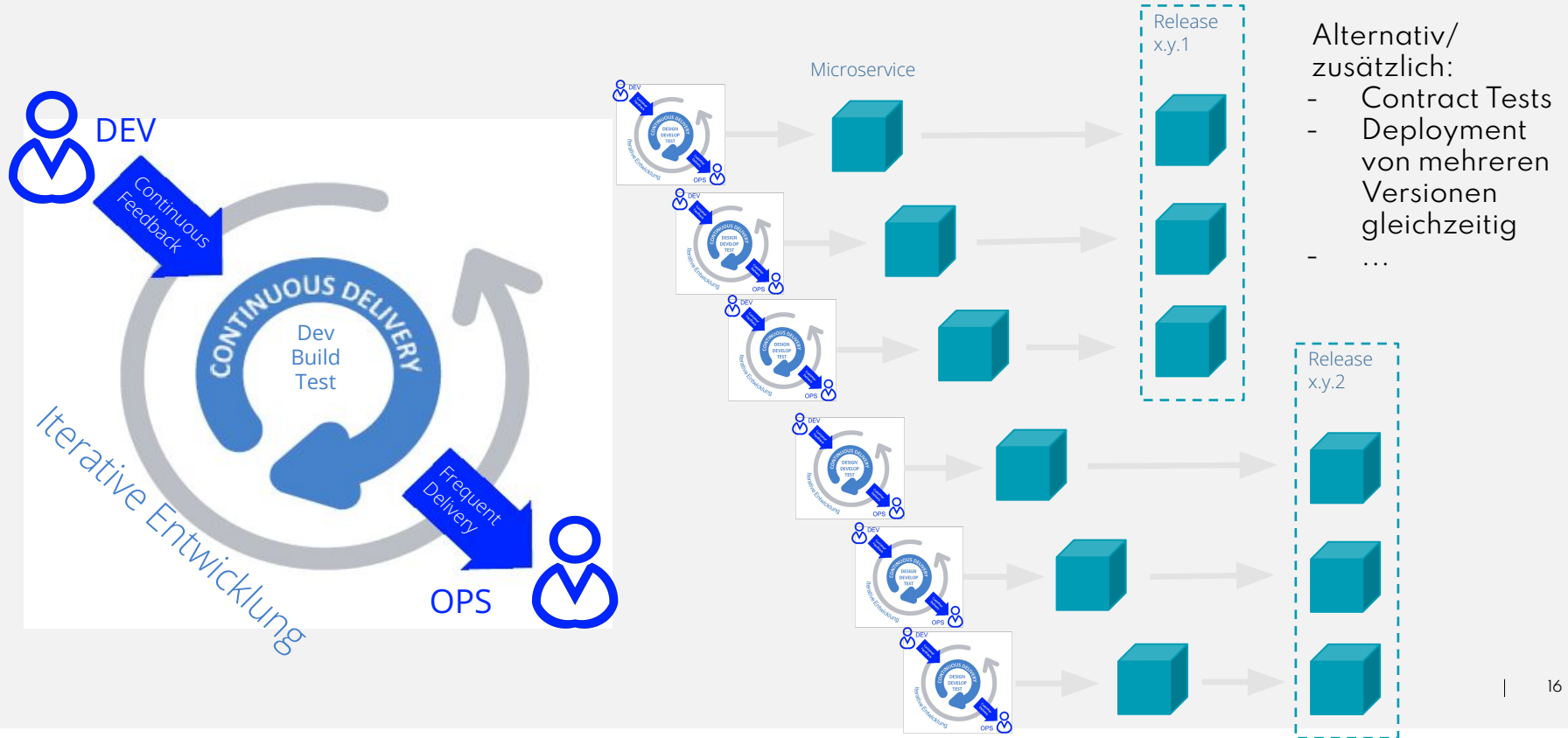
# Beispiel einer Continuous Delivery Pipeline



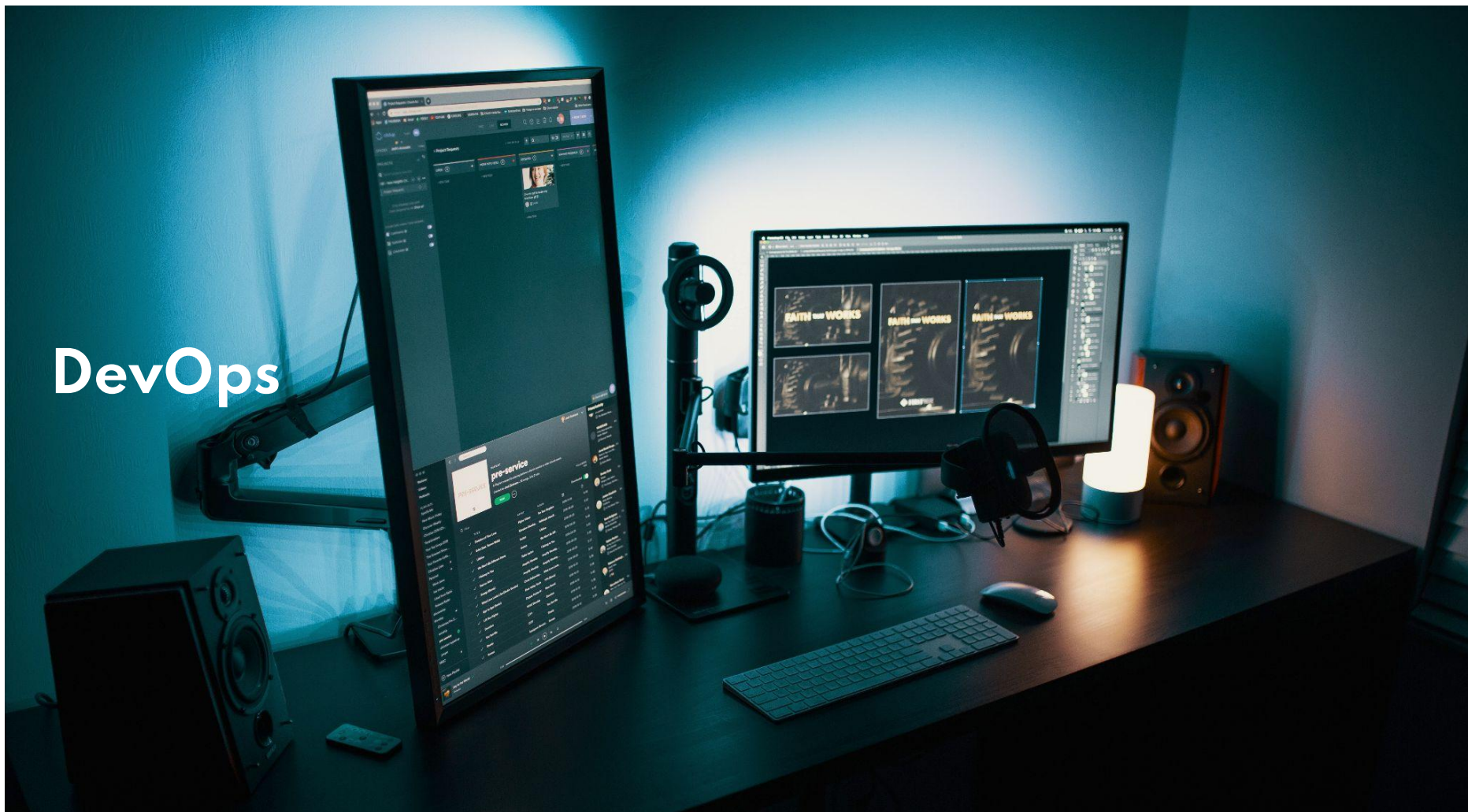
# Continuous Delivery in Microservice-Architekturen: Build und Release Modell für Release-Einheiten



QA|WARE



# DevOps

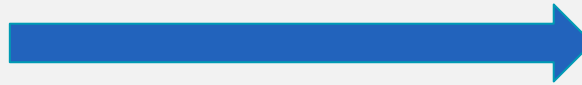
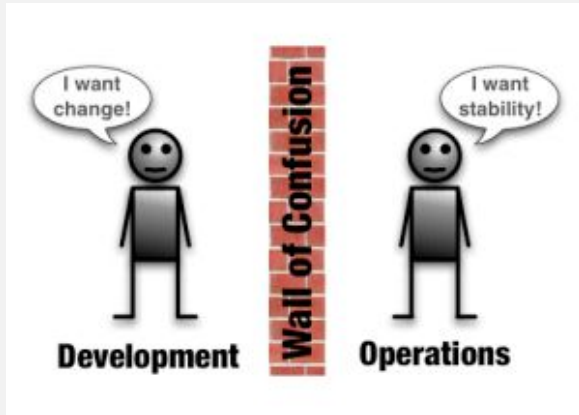


# Was ist eigentlich DevOps?



QA|WARE

**DevOps** ist die **verbesserte Integration** von **Entwicklung und Betrieb** durch mehr **Kooperation und Automation** mit dem Ziel, Änderungen schneller in Produktion zu bringen und die MTTR dort gering zu halten. DevOps ist somit eine Kultur.

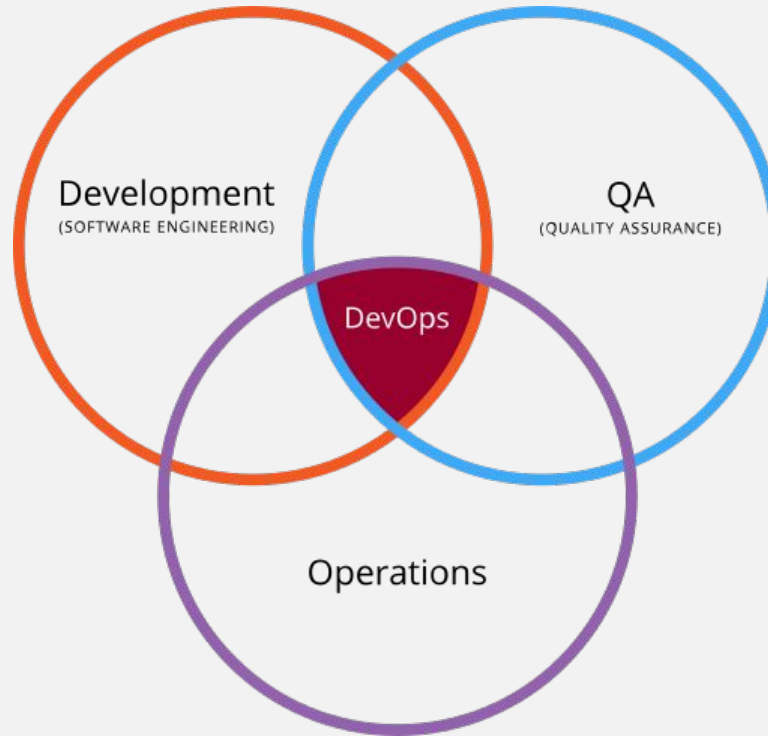


MVP + Feature-Strom

Pro Feature:

- Minimaler manueller Post-Commit-Anteil bis PROD
- Diagnostizierbarkeit des Erfolgs eines Features
- Möglichkeit Feature zu deaktivieren / zurückzurollen

# DevOps verbindet DEvelopment, OPerations und Quality Assurance

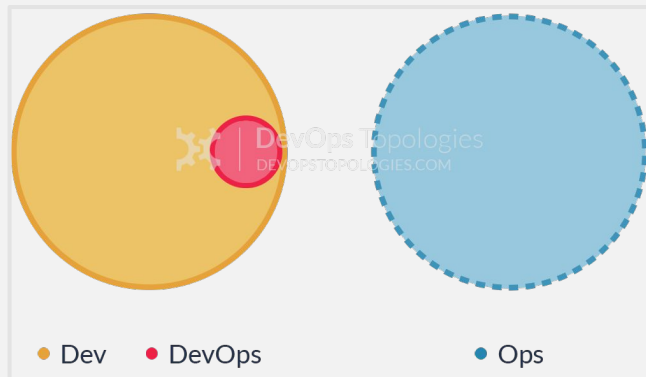


# DevOps Topologies


Nicht überall wo DevOps draufsteht, ist auch DevOps drin.

Zahlreiche Anti-Types und Types finden sich auf:

<https://web.devopstopologies.com/>



“Anti-Type C: Dev Don't Need Ops”

 DevOps Topologies

Anti-Types Team Topologies

## DevOps Anti-Types

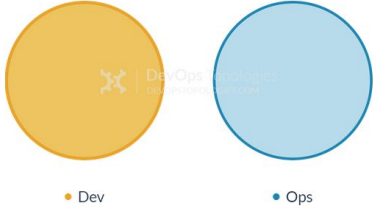
It's useful to look at some bad practices, what we might call 'anti-types' (after the ubiquitous 'anti-pattern').

A: Dev vs Ops B: DevOps Silo C: No Ops Needed D: Tools Team E: SysAdmin F: Embedded Ops G: Dev vs DBA H: Fake SRE

### Anti-Type A: Dev and Ops Silos

This is the classic 'throw it over the wall' split between Dev and Ops. It means that story points can be claimed early (DONE means 'feature-complete', but not working in Production), and software operability suffers because Devs do not have enough context for operational features and Ops folks do not have time or inclination to engage Devs in order to fix the problems before the software goes live.

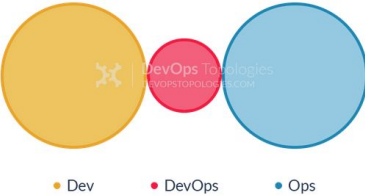
We likely all know this topology is bad, but I think there are actually worse topologies; at least with Anti-Type A (Dev and Ops Silos), we know there is a problem.



### Anti-Type B: DevOps Team Silo

The DevOps Team Silo (Anti-Type B) typically results from a manager or exec deciding that they "need a bit of this DevOps thing" and starting a 'DevOps team' (probably full of people known as 'a DevOp'). The members of the DevOps team quickly form another silo, keeping Dev and Ops further apart than ever as they defend their corner, skills, and toolset from the 'clueless Devs' and 'dinosaur Ops' people.

The only situation where a separate DevOps silo really makes sense is when the team



QA|WARE

Image based on work at [devopstopologies.com](https://web.devopstopologies.com/) - licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/).



Everything as C<>de



# Alles, was die CD-Umgebung mit Leben befüllt, kommt aus dem VCS:



QA|WARE

- Build-as-Code
  - Maven, Gradle, ...
  - Beschreibt wie die Anwendung gebaut wird
- Test-as-Code
  - Unit-, Component-, Integration-, API-, UI-, Performance-Tests
  - Beschreibt wie das Projekt getestet wird
- Infrastructure-as-Code
  - Docker, Terraform, Vagrant, Ansible, Marathon-Deployments
  - Beschreibt, wie die Laufzeitumgebungen aufgebaut werden
- Pipeline-as-Code
  - Build-Pipeline per Jenkinsfile
  - Buildklammer: Beschreibt alle Schritte bis zur lauffähigen Installation



QA|WARE

# GitOps

# Die Idee



QA|WARE

## 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



QA|WARE

- 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



QA|WARE



ArgoCD



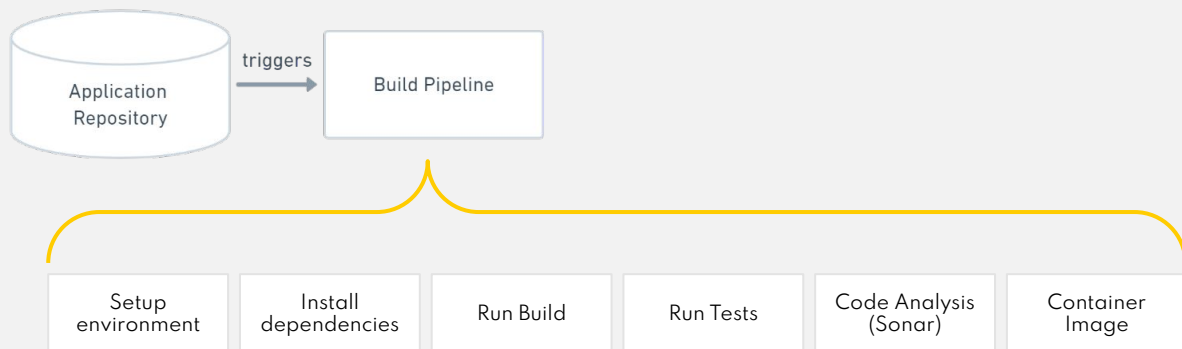
Flux



# Continuous Integration



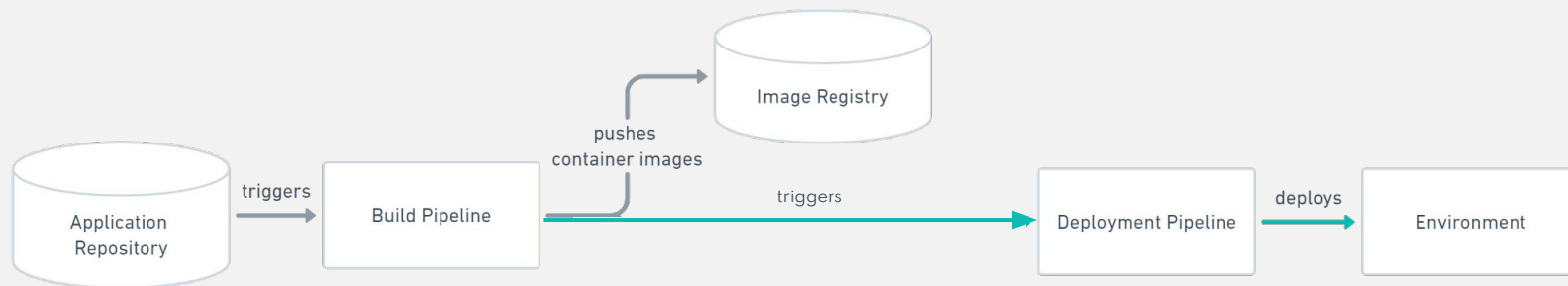
QA|WARE



# Continuous Integration mit automatisierten Deployments



QA|WARE

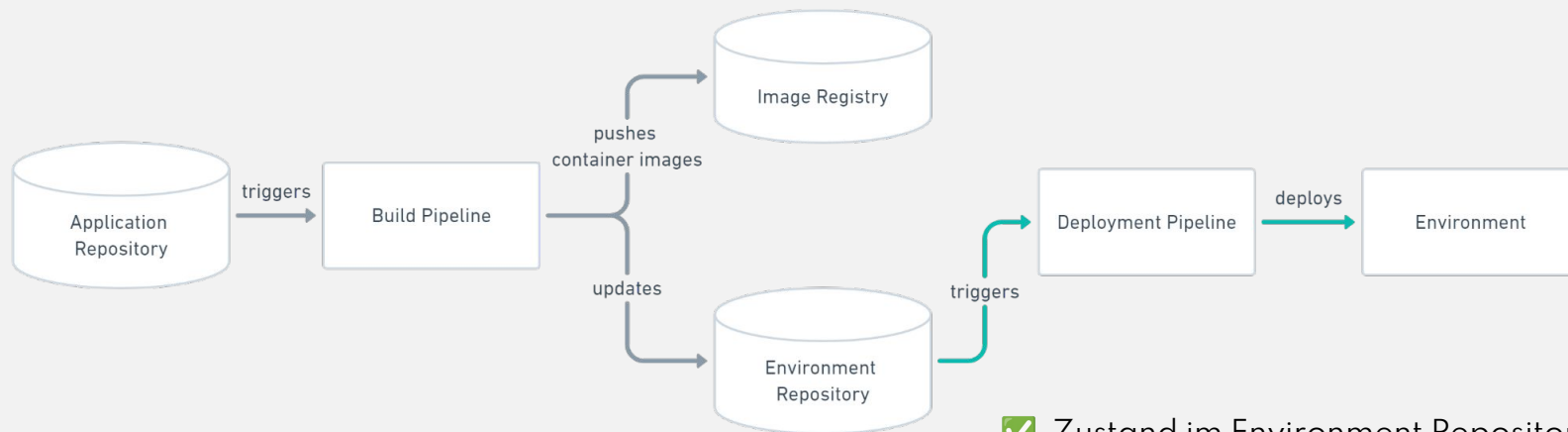


- ✓ Automatisierte Deployments
- ✗ Deployment Pipeline braucht umfassende Rechte
- ✗ Ausführung oft nicht idempotent
- ✗ Cluster-Zustand nicht fixiert

# Push-based deployments mit GitOps



QA|WARE



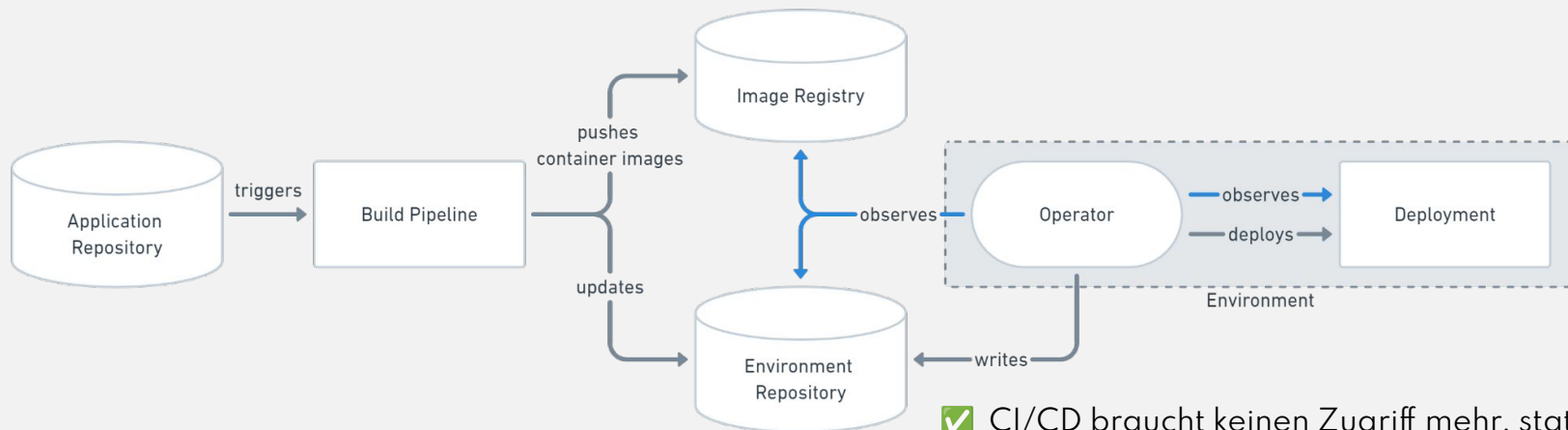
✓ Zustand im Environment Repository abgelegt und wiederherstellbar

- ✗ Deployment Pipeline braucht umfassende Rechte
- ✗ Abweichungen im Cluster außerhalb eines Deployments werden nicht erkannt oder korrigiert
- ✗ Löschen von Ressourcen schwierig

# Pull-based deployments mit GitOps



QA|WARE

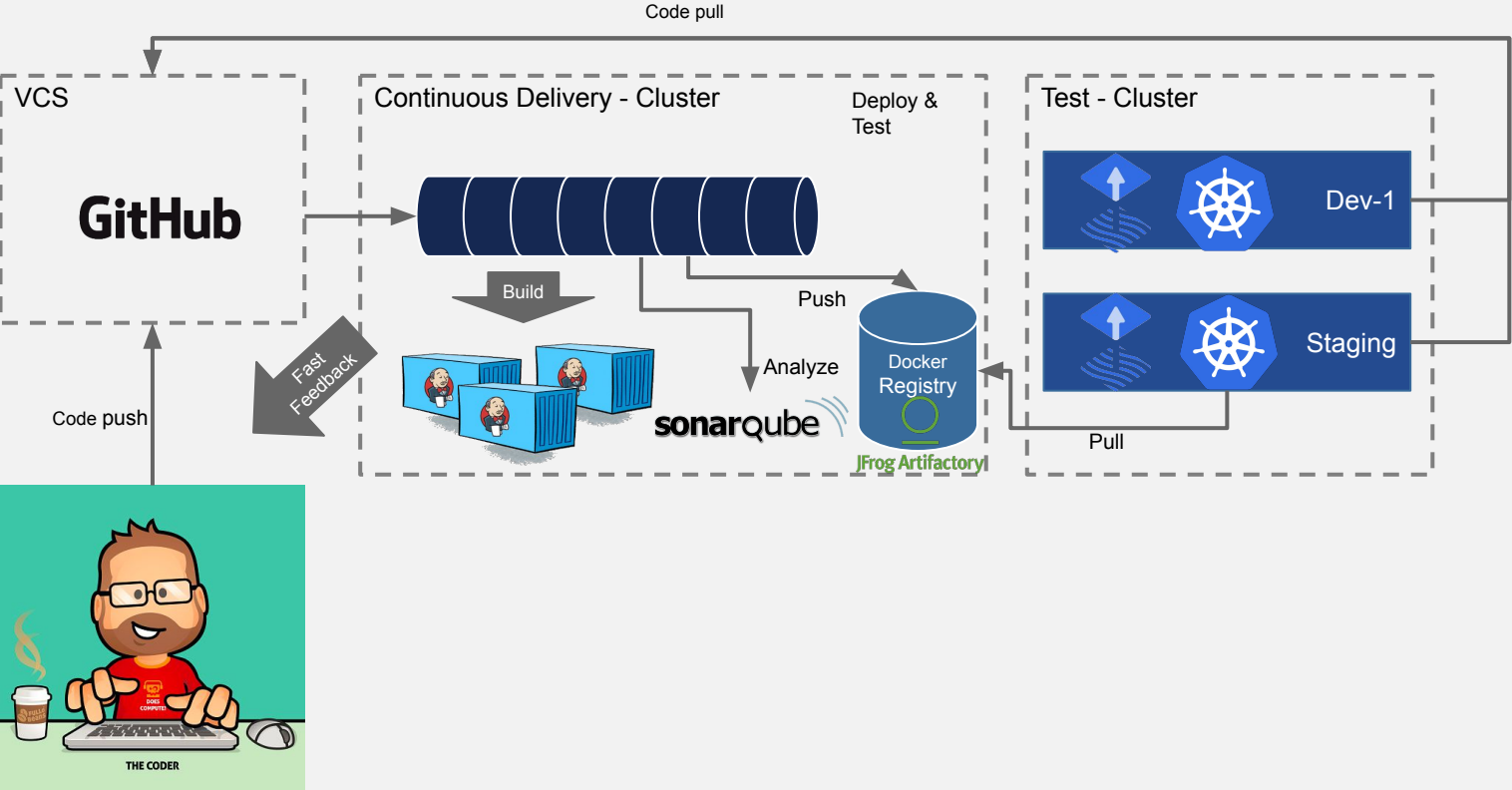


- ✓ CI/CD braucht keinen Zugriff mehr, stattdessen greift der Operator auf das Environment Repo und Registry zu
- ✓ Operator versucht den gewünschten Cluster-Zustand herzustellen, inklusive Löschen von Ressourcen
- ✗ Komplexer

# GitOps mit Flux



QA|WARE



# Unterschiede in der Architektur

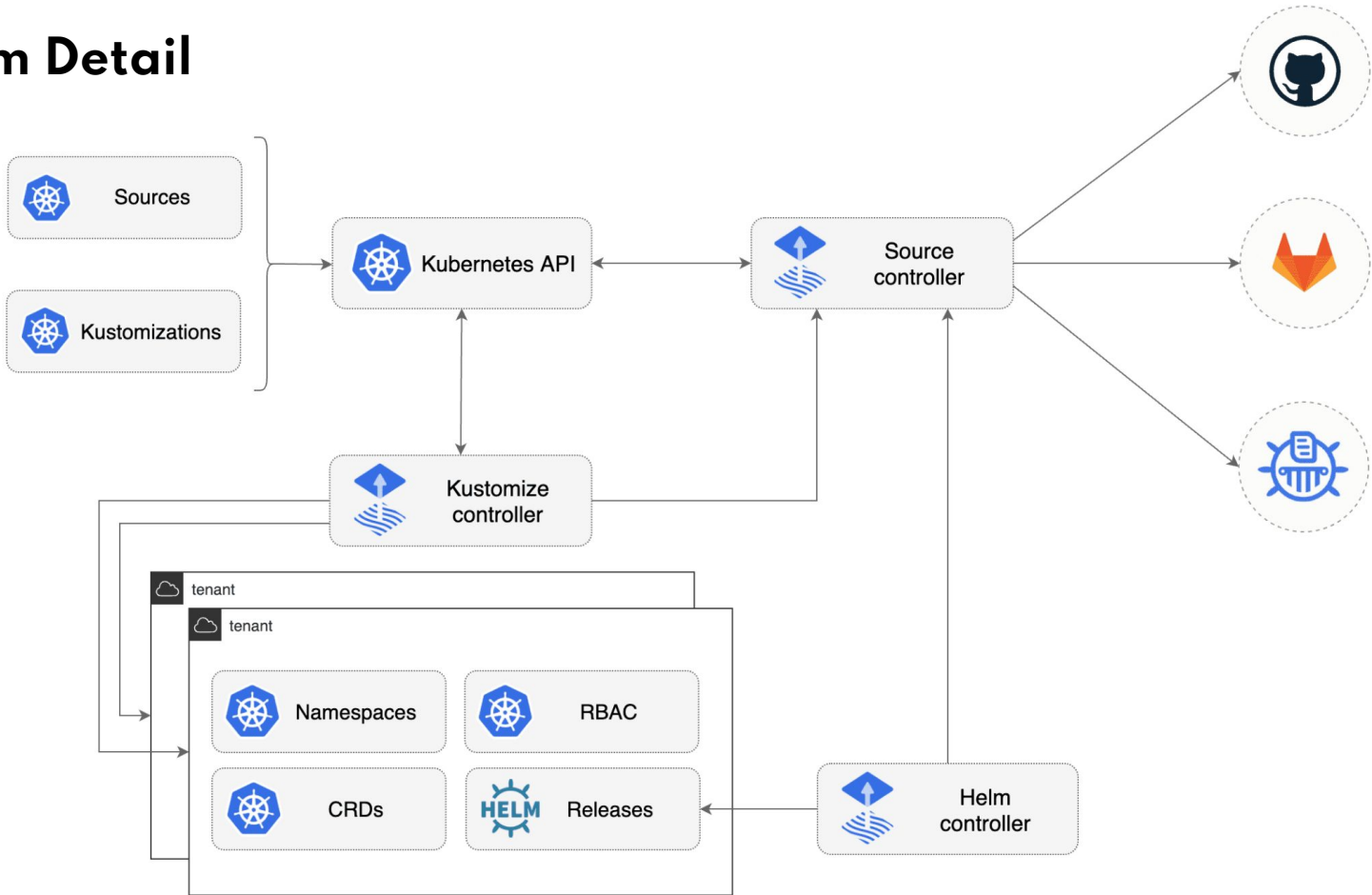


QAWARE

- 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 gleichen ihren Zustand ab (reconcile), d.h. es wird der Ist-Zustand mit dem Soll-Zustand abgeglichen und versucht den Soll-Zustand zu erreichen.
  - => Cluster konvergieren automatisch gegen den Sollzustand



# Flux im Detail



# Flux im Detail



Q|WARE

- 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 getrackt 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



QA|WARE

```
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
- klonet 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



QA|WARE

```
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**



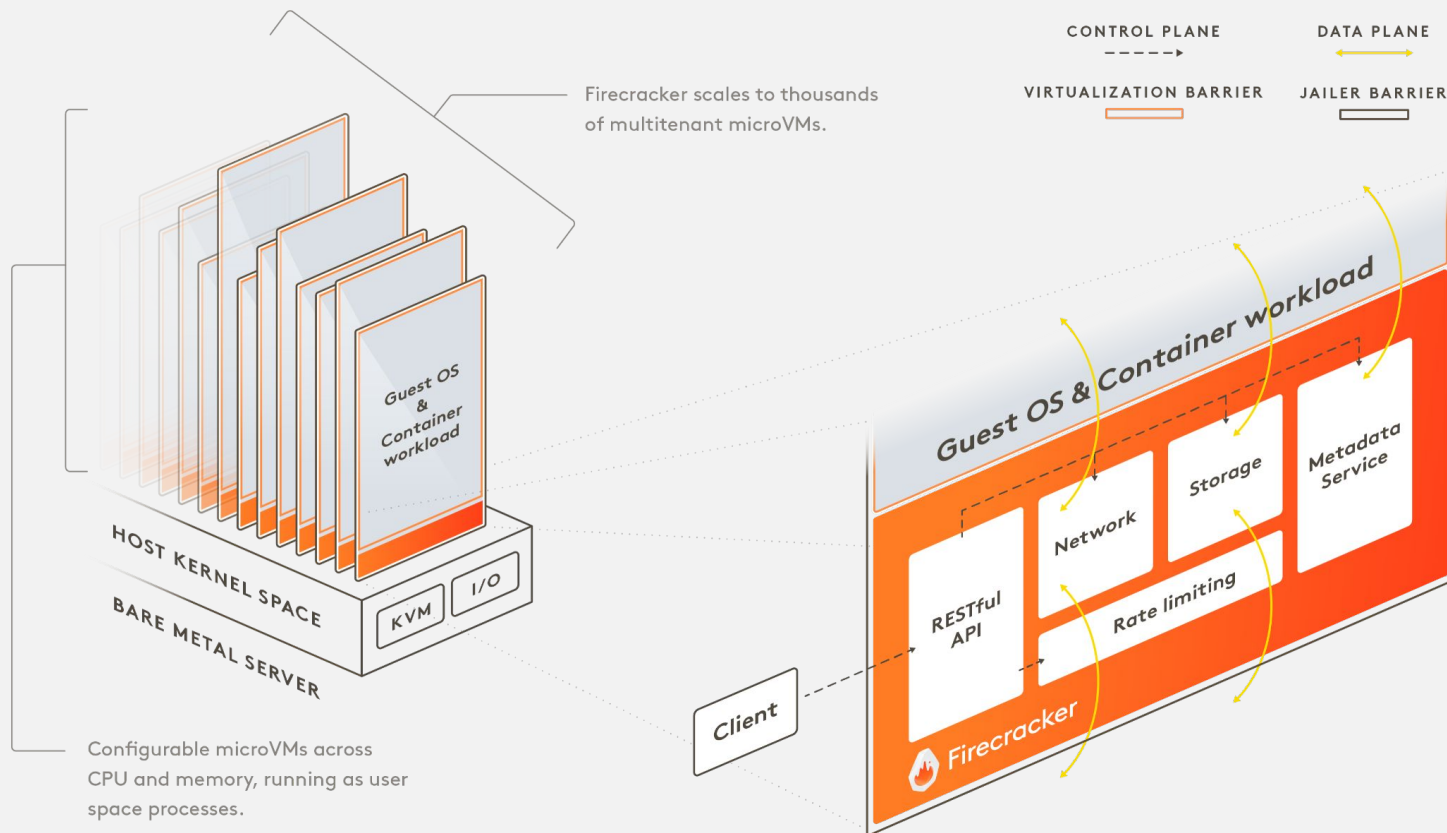
QA|WARE

# Cloud Runtimes

# Firecracker - back to VMs



QnWARE



Configurable microVMs across CPU and memory, running as user space processes.



# Firecracker



QA|WARE

- microVMs statt Container
- Minimaler Overhead, wesentlich bessere Isolation
- Schnelle Startup times
- Wird von Amazon für AWS lambda und Fargate verwendet

# Hashicorp Nomad als Orchestrator



## Vorteile

- einfach zu bedienender Workload Orchestrator
  - container
  - VMs
  - binaries
- einfacher strukturiert als Kubernetes
  - es muss nur ein binary deployed werden - gleiches binary für server und client

## Nachteile

- Ökosystem ist wesentlich schwächer
  - kein helm
  - kein gitops etc.
- Braucht für Features Service Discovery und Secrets Management weitere Produkte (Consul und Vault)
- Features für größere Firmen erfordern Lizenzen

# Webassembly - the runtime of the future?



QA|WARE



# WEBASSEMBLY

# Webassembly - Overview



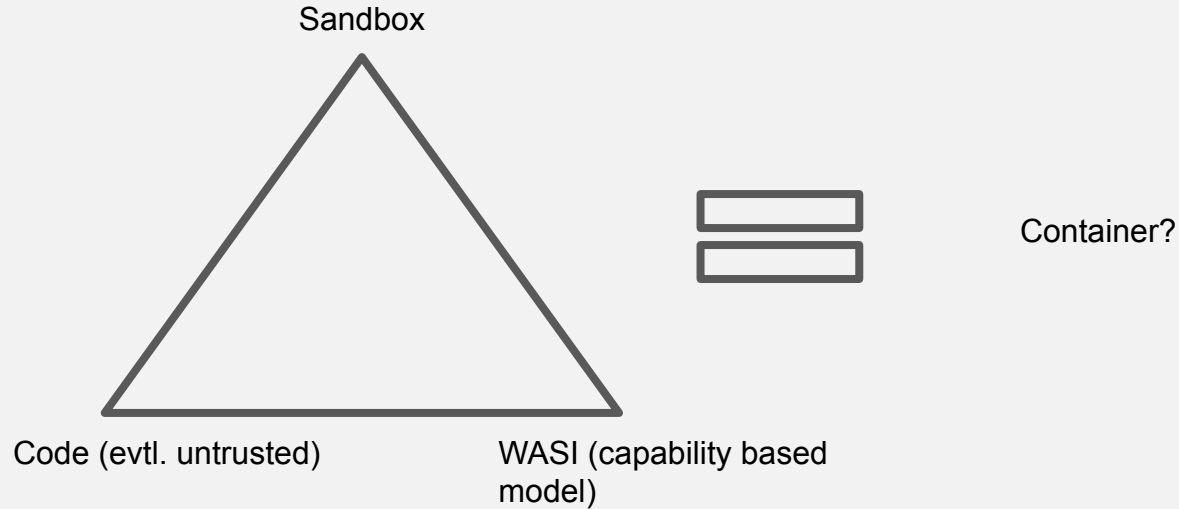
QA|WARE

- Webassembly ist eine Sammlung an Standards
  - Core Specification - Beschreibung des Bytecode Formats etc.
  - Embedding Interfaces - Beschreibung wie Webassembly in verschiedene Technologien eingebettet werden kann
  - WASI - System Interface Beschreibung. Webassembly Programme laufen in einer Sandbox. Mit WASI kann feingranular Zugriff auf System APIs gesteuert werden, z.B. file oder network access.
- Wurde ursprünglich für das Web entwickelt und ist im Browser lauffähig
- Webassembly ist als Compilation-Target verschiedenster Sprachen verfügbar
- Läuft per default in einer Sandbox => großer Unterschied zu vorherigen Versuchen einer generalisierten Plattform wie CLR oder JVM.

# Webassembly - im Backend?



QA|WARE



# Webassembly auf Kubernetes?



QA|WARE

- Es wird mittlerweile aktiv versucht, dass container runtimes WASM Module direkt supporten
  - containerd wird erweitert und kann dann WASM als “normale” Kubernetes workload ausführen
- dedizierte Projekte (viele davon sind allerdings verwaist)
  - z.B. [wasmcloud](#)