

Realisierung einer CI/CD-Pipeline für container-basierte verteilte Anwendungen

Status Quo, Herausforderungen, Realisierung am konkreten Beispiel

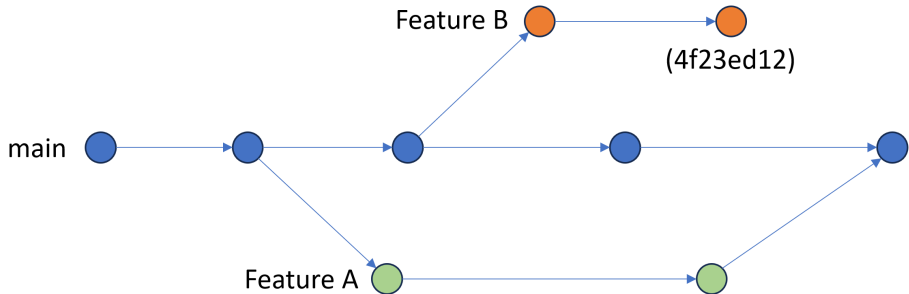
Dr.-Ing. Stephan Schneider

Dezember 18, 2024

Agenda

- 1 Container-basierte Anwendungen
- 2 CI/CD Pipeline
- 3 Beispielanwendung
- 4 Zusammenfassung

Beispiel



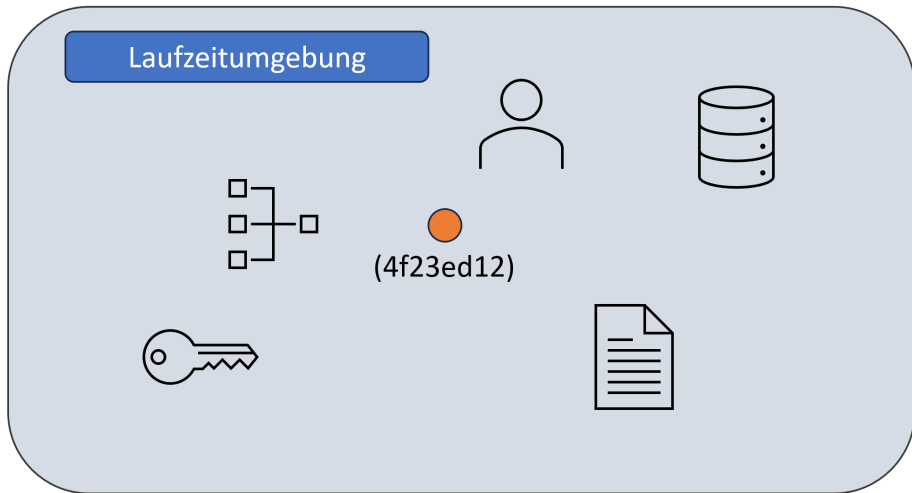
Beispiel

Laufzeitumgebung



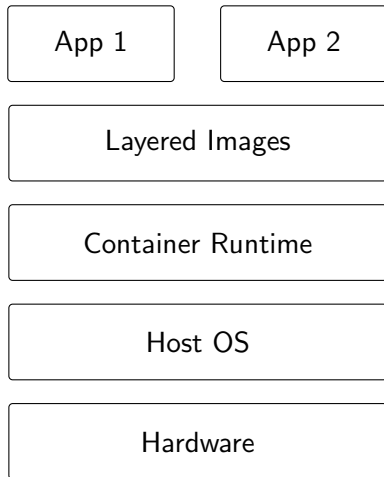
(4f23ed12)

Beispiel

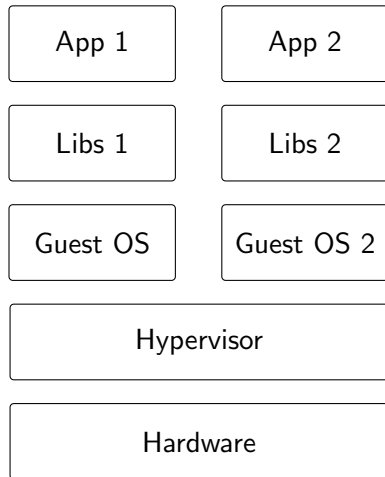


Container vs. Virtuelle Maschine

Container:



Virtuelle Maschine:



Container-basierte Anwendungen

Was sind Container?

- Leichtgewichtige Methoden zur Isolation von Dateisystemen, CPU- und Speicherressourcen, System- und Netzwerregeln
- Bieten jedoch eine geringere Isolation als virtuelle Systeme

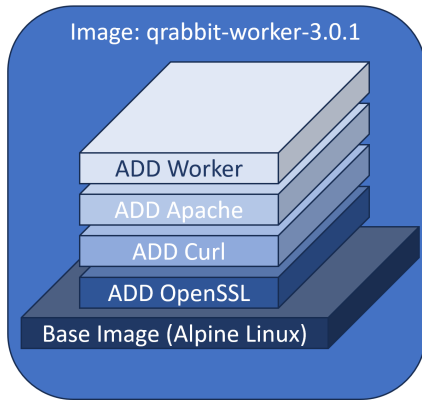
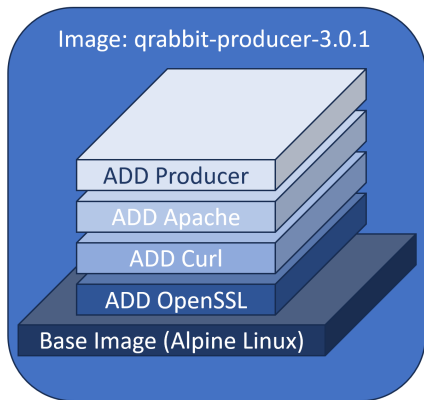
Vorteile:

- Geringer Overhead im Vergleich zu VMs
- Keine Hardware-Emulation
- Direkter Zugriff auf Ressourcen des Hostsystems (GPU, Netzwerk, Storage)

Nachteile:

- Kein abweichendes OS möglich
- Keine Änderung der Kernelkonfiguration
- Schwache Isolation

Layered Images



- **1979** UNIX : *chroot*
- **1999** BSD Jail
- **2000** Linux virtual environment
- **2004** Linux BSD Jail / Solaris Zones
- **2005** Linux OpenVZ
- **2008** Linux Containers (LXC)
- **2013** Docker (bis 2014 ein Wrapper für LXC)

- **1979** UNIX : *chroot*
- **1999** BSD Jail
- **2000** Linux virtual environment
- **2004** Linux BSD Jail / Solaris Zones
- **2005** Linux OpenVZ
- **2008** Linux Containers (LXC)
- **2013** Docker (bis 2014 ein Wrapper für LXC)

Container sind keine neue Technologie und Docker ist nicht die einzige Container-Runtime.

Multilayer-Architektur

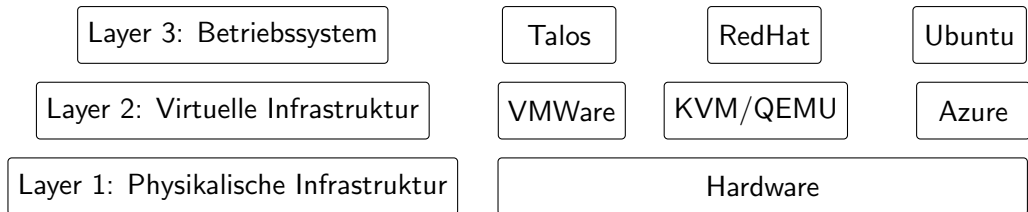
Layer 1: Physikalische Infrastruktur

Hardware

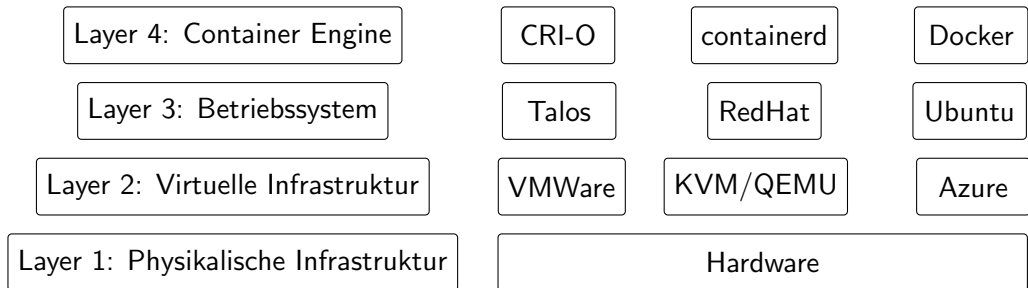
Multilayer-Architektur



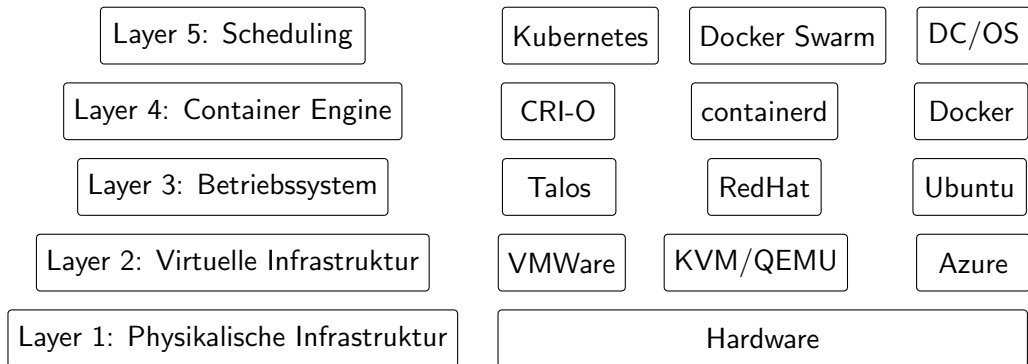
Multilayer-Architektur



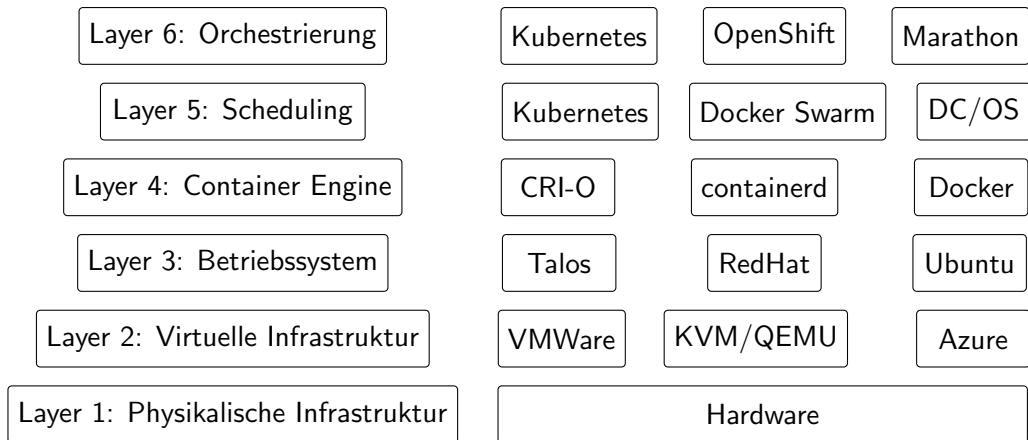
Multilayer-Architektur



Multilayer-Architektur



Multilayer-Architektur



Definition

CI : engl. *Continuous Integration*

CD : engl. *Continuous Delivery* oder *Continuous Deploy*

Definition

CI : engl. *Continuous Integration*

CD : engl. *Continuous Delivery* oder *Continuous Deploy*

Was soll kontinuierlich integriert werden?

Definition

CI : engl. *Continuous Integration*

CD : engl. *Continuous Delivery* oder *Continuous Deploy*

Was soll kontinuierlich integriert werden? **Softwareanpassungen!**

Definition

CI : engl. *Continuous Integration*

CD : engl. *Continuous Delivery* oder *Continuous Deploy*

Was soll kontinuierlich integriert werden? **Softwareanpassungen!**

Wo liegt der Unterschied zwischen *Delivery* und *Deploy*?

Definition

CI : engl. *Continuous Integration*

CD : engl. *Continuous Delivery* oder *Continuous Deploy*

Was soll kontinuierlich integriert werden? **Softwareanpassungen!**

Wo liegt der Unterschied zwischen *Delivery* und *Deploy*?

Beim Continuous Delivery findet die Aktivierung im Produktivsystem manuell statt.

Verfügbare CI/CD-Lösungen

- Jenkins
- Gitlab
- Travis-CI
- TeamCity
- CircleCI
- (uvm.)

Jede dieser Anwendungen besitze Stärken und Schwächen die im jeweiligen Projektkontext bewertet werden müssen.

Die nachfolgenden Beispiele wurden mit GitLab und dessen CI/CD-Möglichkeiten erstellt.

CI/CD-Pipeline

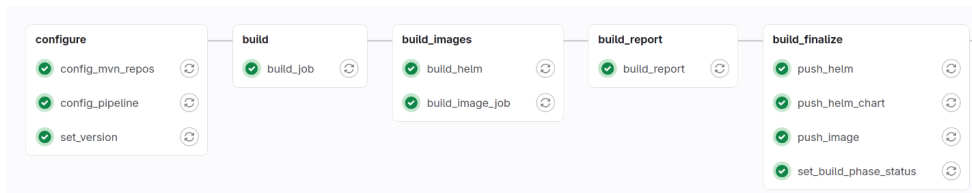
Update .gitlab-ci.yml file

✓ Passed sts created pipeline for commit 302027e1 1 minute ago, finished just now

For main

latest 16 jobs 1 minute 3 seconds, queued for 2 seconds

Pipeline Needs Jobs 16 Tests 0



Beispielanwendung

- Die Beispielanwendung besteht aus drei Java-Microservices

Beispielanwendung

- Die Beispielanwendung besteht aus drei Java-Microservices
- **Producer** erstellt Arbeitsaufträge und sendet diese an eine asynchrone Warteschlange.
 - ▶ **WAIT**: Kommando bewirkt einen Wartezyklus von 5 Sekunden im Worker
 - ▶ **NOOP**: Kommando bewirkt eine Ausführung eines Nullbefehls
 - ▶ **FIBONACCI**: Kommando bewirkt die Berechnung einer Fibonacci-Zahl im Worker
 - ▶ **KILL**: Kommando bewirkt, dass der Worker beendet wird.

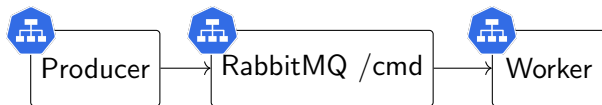
Beispielanwendung

- Die Beispielanwendung besteht aus drei Java-Microservices
- **Producer** erstellt Arbeitsaufträge und sendet diese an eine asynchrone Warteschlange.
 - ▶ **WAIT**: Kommando bewirkt einen Wartezyklus von 5 Sekunden im Worker
 - ▶ **NOOP**: Kommando bewirkt eine Ausführung eines Nullbefehls
 - ▶ **FIBONACCI**: Kommando bewirkt die Berechnung einer Fibonacci-Zahl im Worker
 - ▶ **KILL**: Kommando bewirkt, dass der Worker beendet wird.
- **Worker** nimmt die Kommandos aus der Warteschlange und bearbeitet diese
 - ▶ Abgearbeitete Kommandos werden in die Warteschlange **results** eingestellt
- **Writer** Der Prozess nimmt die Ergebnisse aus der Warteschlange **results** und persistiert diese.

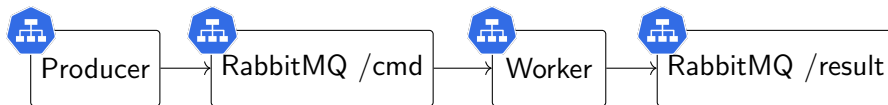
Beispielanwendung



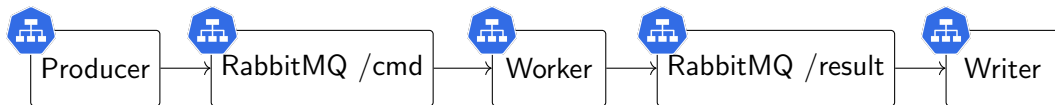
Beispielanwendung



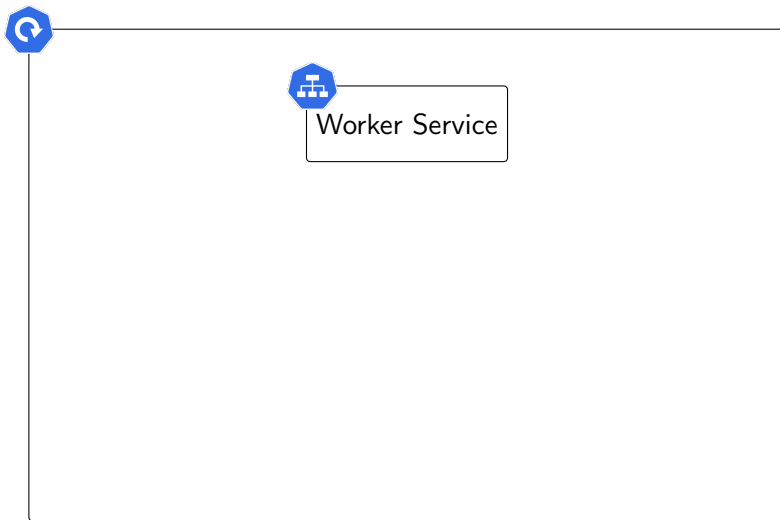
Beispielanwendung



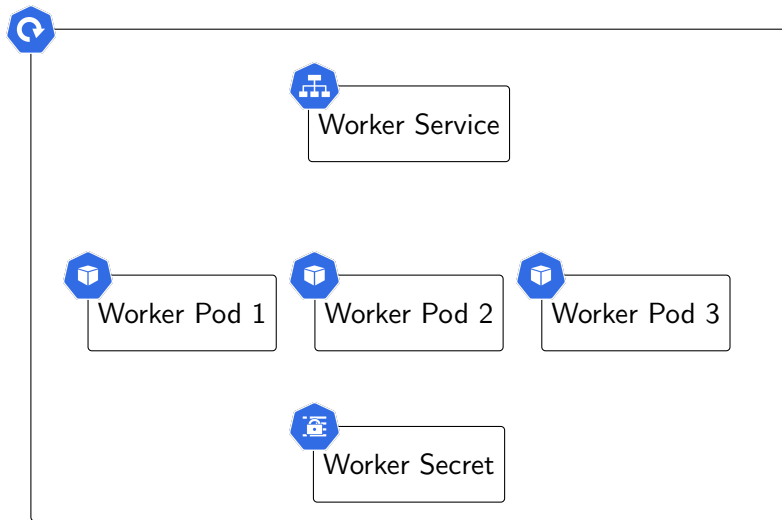
Beispielanwendung



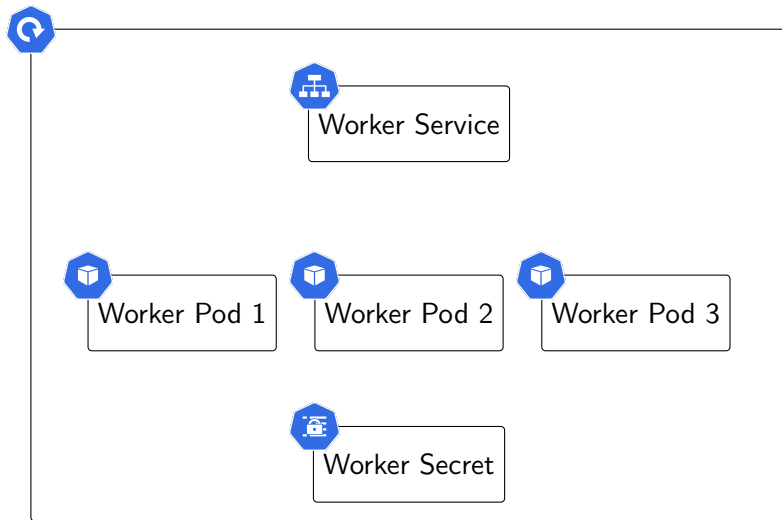
Kubernetes Deployment



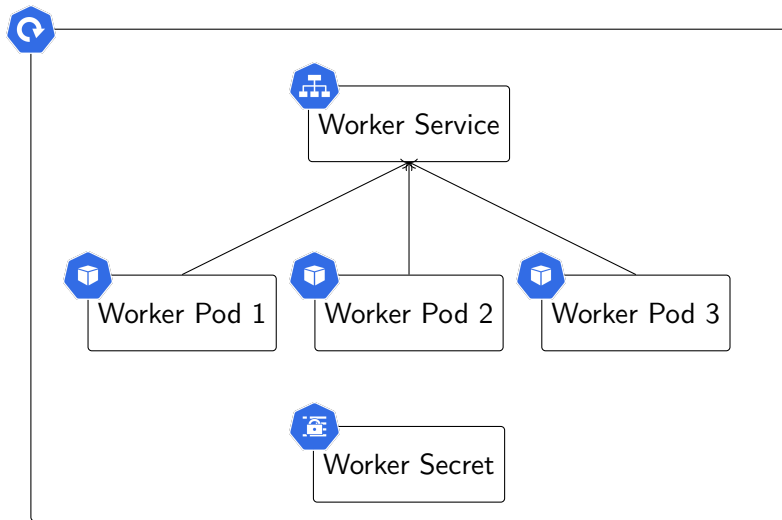
Kubernetes Deployment



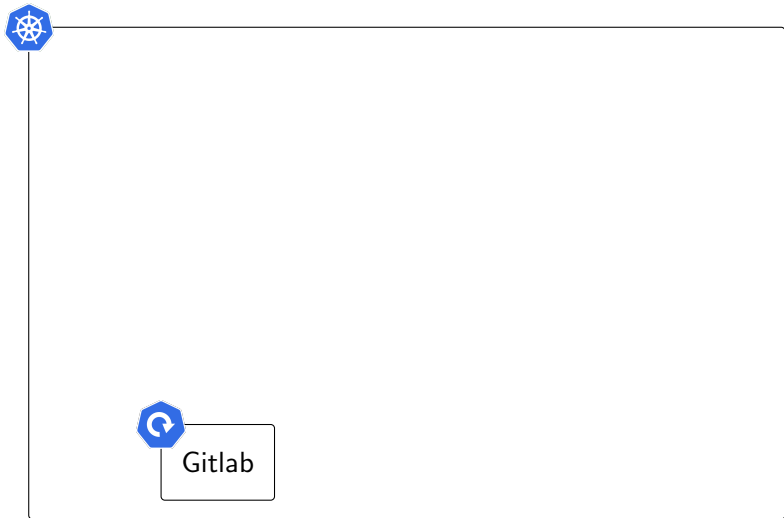
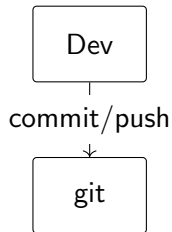
Kubernetes Deployment



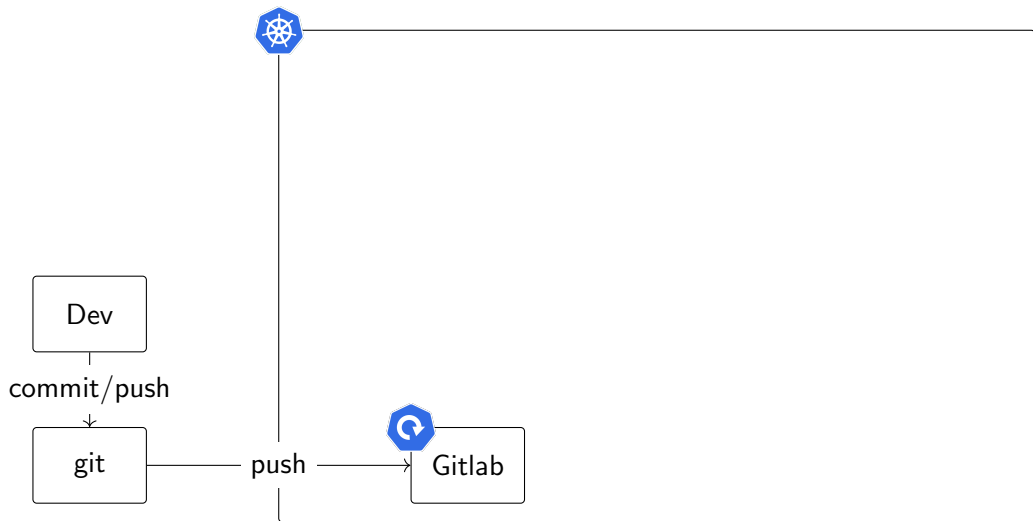
Kubernetes Deployment



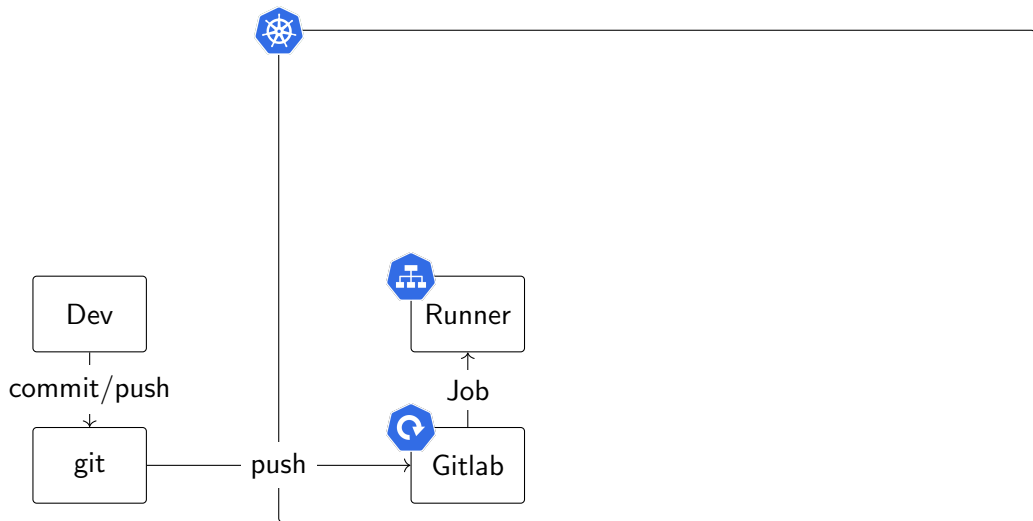
Übersicht Kubernetes



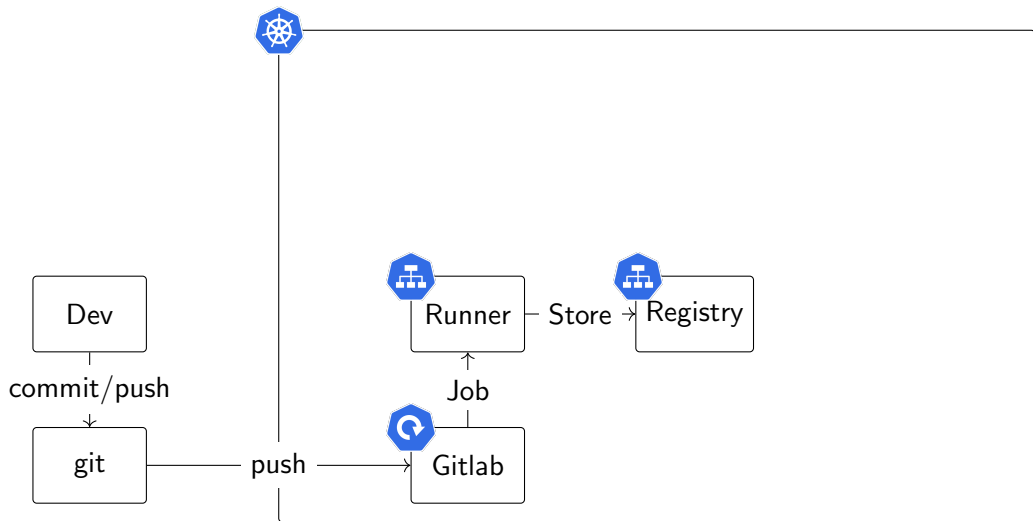
Übersicht Kubernetes



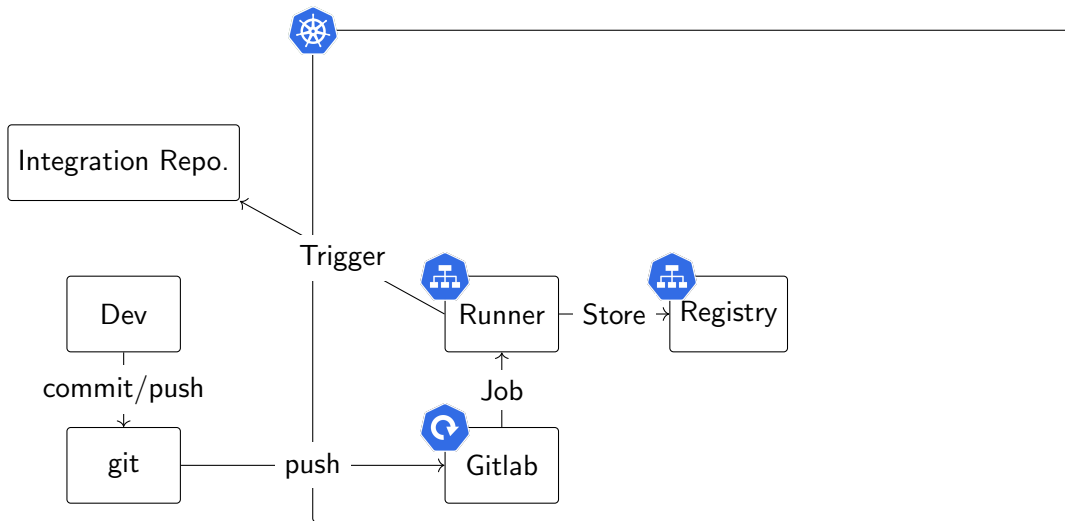
Übersicht Kubernetes



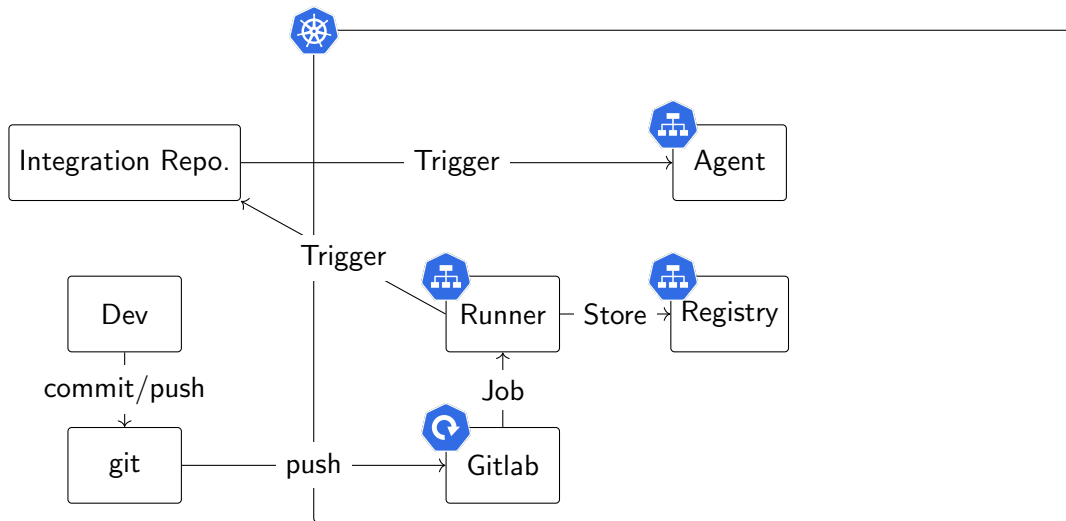
Übersicht Kubernetes



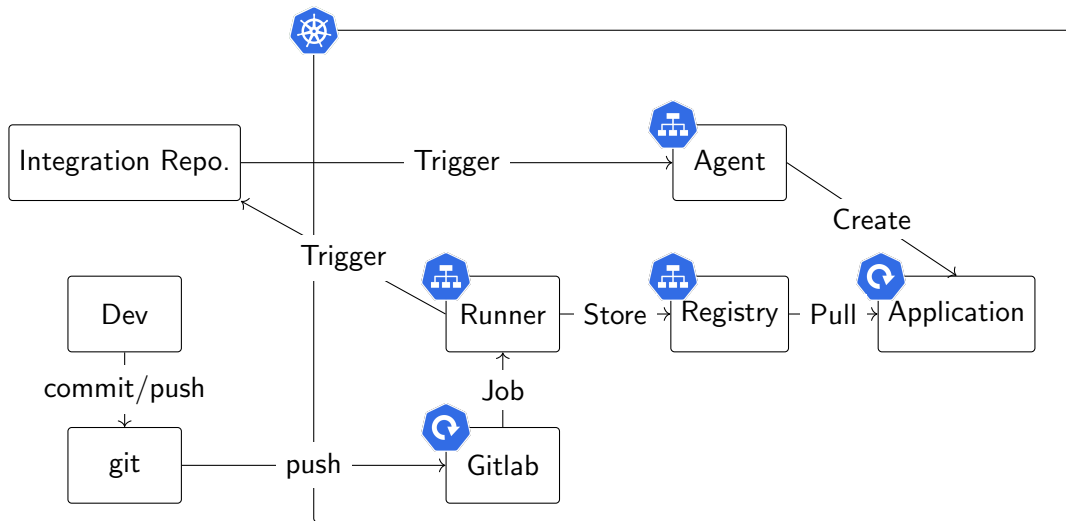
Übersicht Kubernetes



Übersicht Kubernetes



Übersicht Kubernetes



Summary

- Using CI/CD-Pipelines is highly recommended to ensure necessary software quality.

Summary

- Using CI/CD-Pipelines is highly recommended to ensure necessary software quality.
- Modern container solutions provide

Summary

- Using CI/CD-Pipelines is highly recommended to ensure necessary software quality.
- Modern container solutions provide
 - ▶ Process isolation via kernel namespaces

Summary

- Using CI/CD-Pipelines is highly recommended to ensure necessary software quality.
- Modern container solutions provide
 - ▶ Process isolation via kernel namespaces
 - ▶ Resource constraints using kernel control groups

Summary

- Using CI/CD-Pipelines is highly recommended to ensure necessary software quality.
- Modern container solutions provide
 - ▶ Process isolation via kernel namespaces
 - ▶ Resource constraints using kernel control groups
- Implementing a reliable CI/CD operation many "moving" parts must align perfectly

Summary

- Using CI/CD-Pipelines is highly recommended to ensure necessary software quality.
- Modern container solutions provide
 - ▶ Process isolation via kernel namespaces
 - ▶ Resource constraints using kernel control groups
- Implementing a reliable CI/CD operation many "moving" parts must align perfectly

Thank you for your attention!

GitHub: `https://github.com/p2k-ko/cicd_pipeline`



References I

- K. Welter: *Kubernetes*, 2024
- B. Öggl u. M. Kofler: *Docker*, 2021
- O. Liebel: *Skalierbare Container-Infrastrukturen*, 4. Aufl., 2023
- A. Jones: *Advanced GitLab CI/CD Pipelines*, 2024
- J. van Baarsen: *Gitlab Cookbook*, 2014
- D. Dossot u. I. Johansson: *RabbitMQ Essentials*, 2020
- M. Tefanko u. J. Martika: *Quarkus in Action*, 2024