



Data Engineering: Docker & Kubernetes

<https://github.com/>

NickStudRepo/Docker-Kubernetes

Docker

- Überblick
- Container vs. VM
- Architektur
- Dockerfile
- Docker Compose
- Dev Container
- Portainer (Dashboard)

Kubernetes

- Überblick
- Architektur
- Konfiguration
- Helm (Paketmanager)
- Virtual IP
- Persistent Storage
- Portainer (Dashboard)



- Unternehmen die Docker/ Kubernetes verwenden



...



Docker



- Software zur Isolierung von Anwendungen durch **Containervirtualisierung**

Ein Docker Container Image
ist ein **leichtgewichtiges, eigenständiges, ausführbares Softwarepaket**, das alles
enthält, was zum Ausführen einer Anwendung erforderlich ist

- Docker ist eine Open-Source (Go)



Probleme die Docker löst



Dependency Hell:

Versionskonflikte, Komplexität der Abhängigkeitskette, Inkonsistenz, Aktualisierungen, Build Bereitstellung

Docker:

- **Portabilität:** Docker Container verpackt Anwendung und Abhängigkeiten
- **Isolation:** Container ist in sich geschlossen
- **Konsistenz:** unabhängig von der Umgebung
- **Skalierbarkeit:** horizontale Skalierung durch mehrere Container
- **Schnelle Bereitstellung:** leichtgewichtig und notwendige/minimale Abhängigkeiten
- **Entwicklungsumgebungen**
- **Ressourcenoptimierung**

Ein Container ist ein **leichtgewichtiges, eigenständiges, ausführbares Softwarepaket**, das alles enthält, was zur Ausführung einer Anwendung erforderlich ist:

Code, Laufzeit, Systemtools, Systembibliotheken und **Einstellungen**.

Ist das eine VM?



Klassischer Ansatz (ohne VM oder Container)

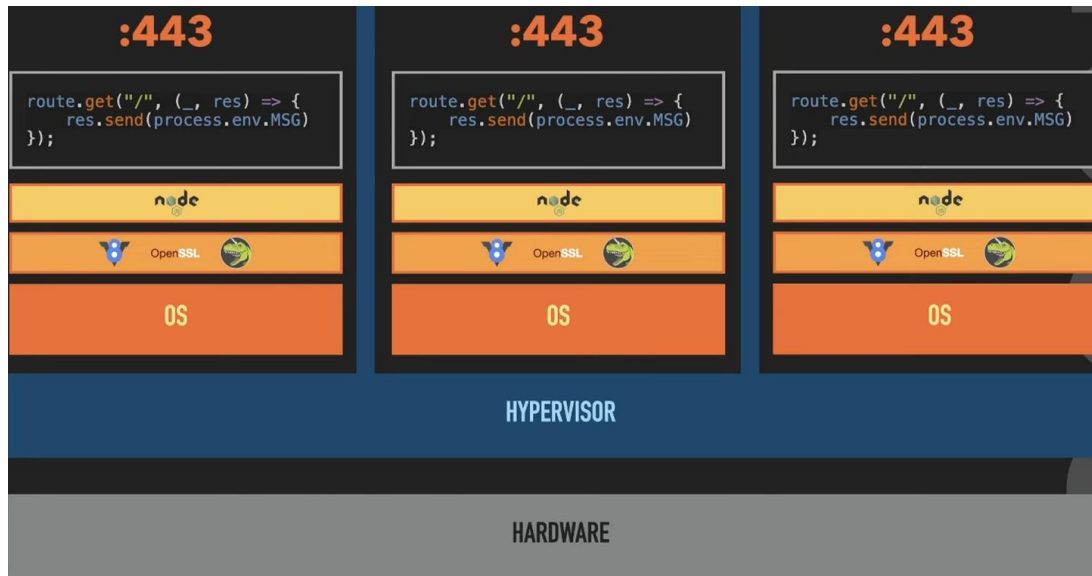


- Ressourcennutzung
- Gegenseitige Beeinflussung!



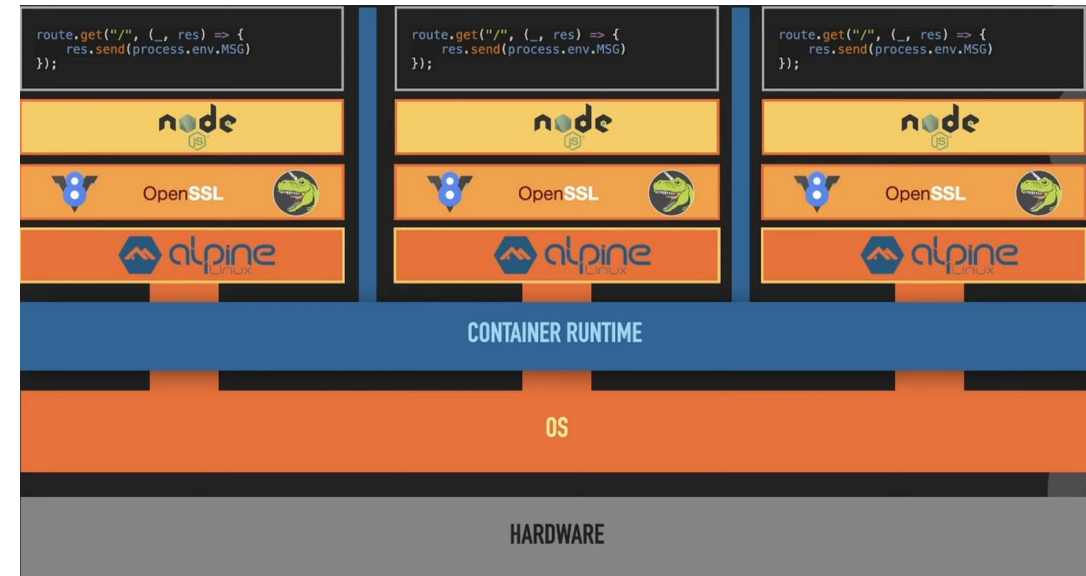


VMs



- Hypervisor verwaltet VMs
- Jede VM braucht eigenes OS
- Viele Ressourcen + Zeit

Container



- Container Runtime isoliert Container
- Nur ein OS benötigt
- OS verwaltet Isolierung der Container
- Weniger Ressourcen + schneller

Fazit Container vs. VM

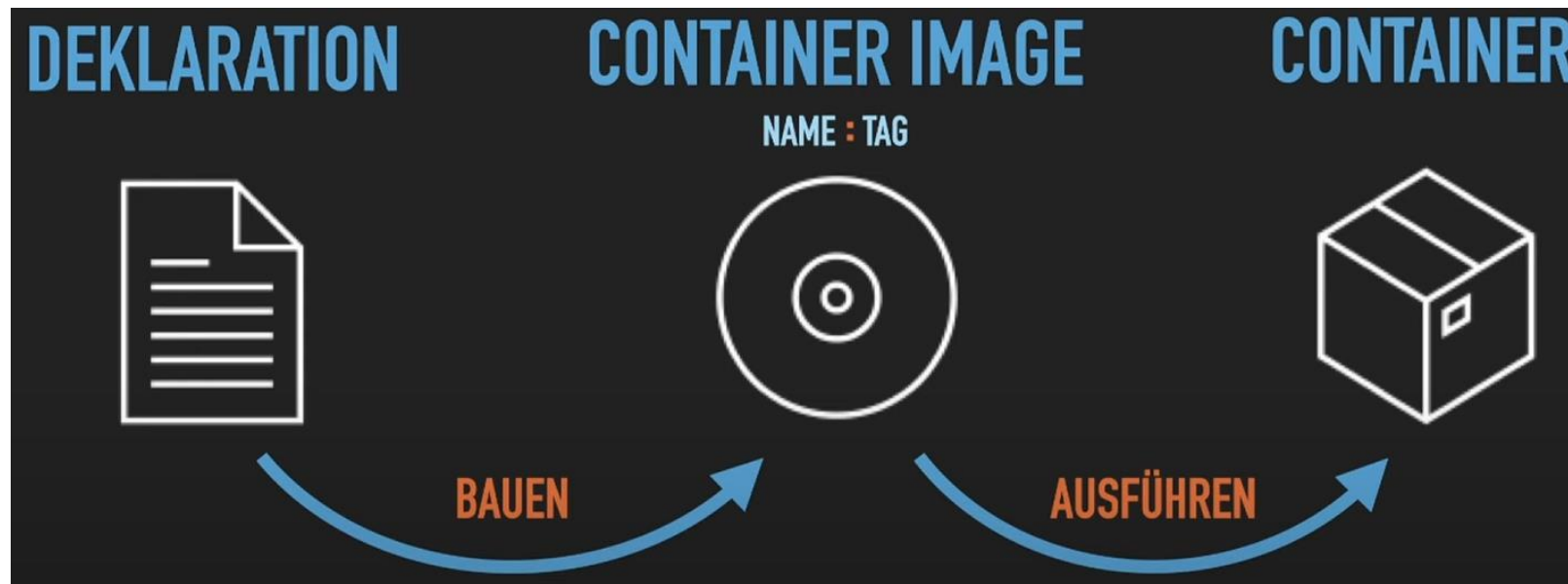
Gemeinsamkeiten:

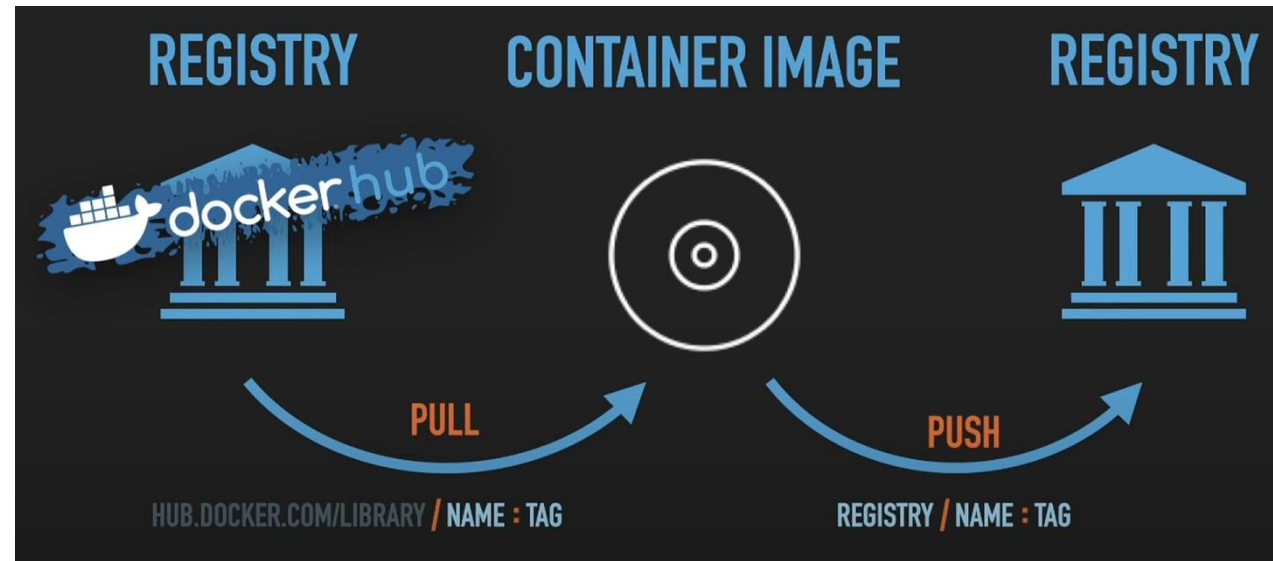
- Es gibt eine Shell in VM und Container
(muss nicht zwingend: statisch kompilierter Code)
- Eigener Prozessraum
- Eigenes Netzwerk Interface

Unterschiede:

- Container benutzt den Host Kernel
- Linux Kernel -> Linux Container; Windows Kernel -> Windows Container
- Container ist normaler Prozess auf dem Host

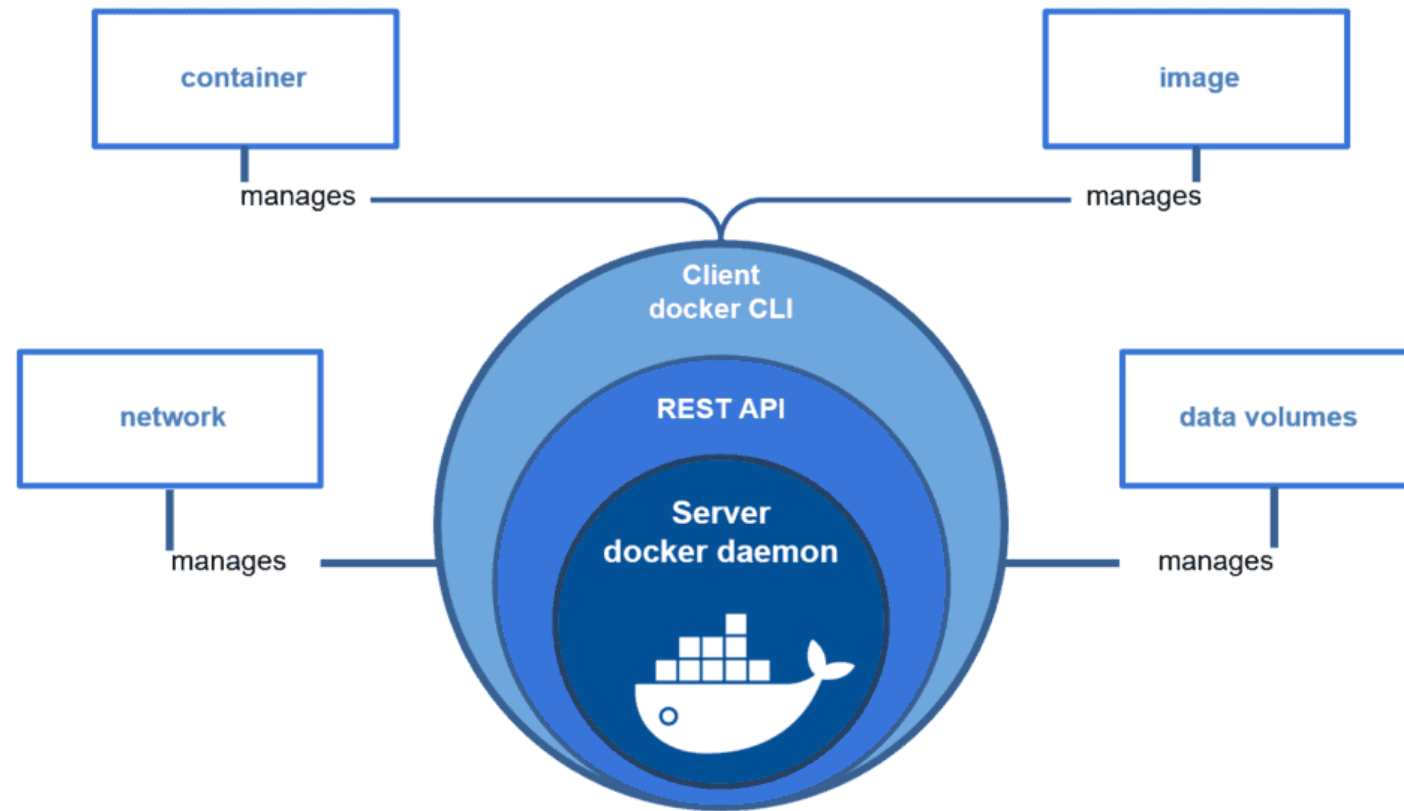
Container: Lebenszyklus



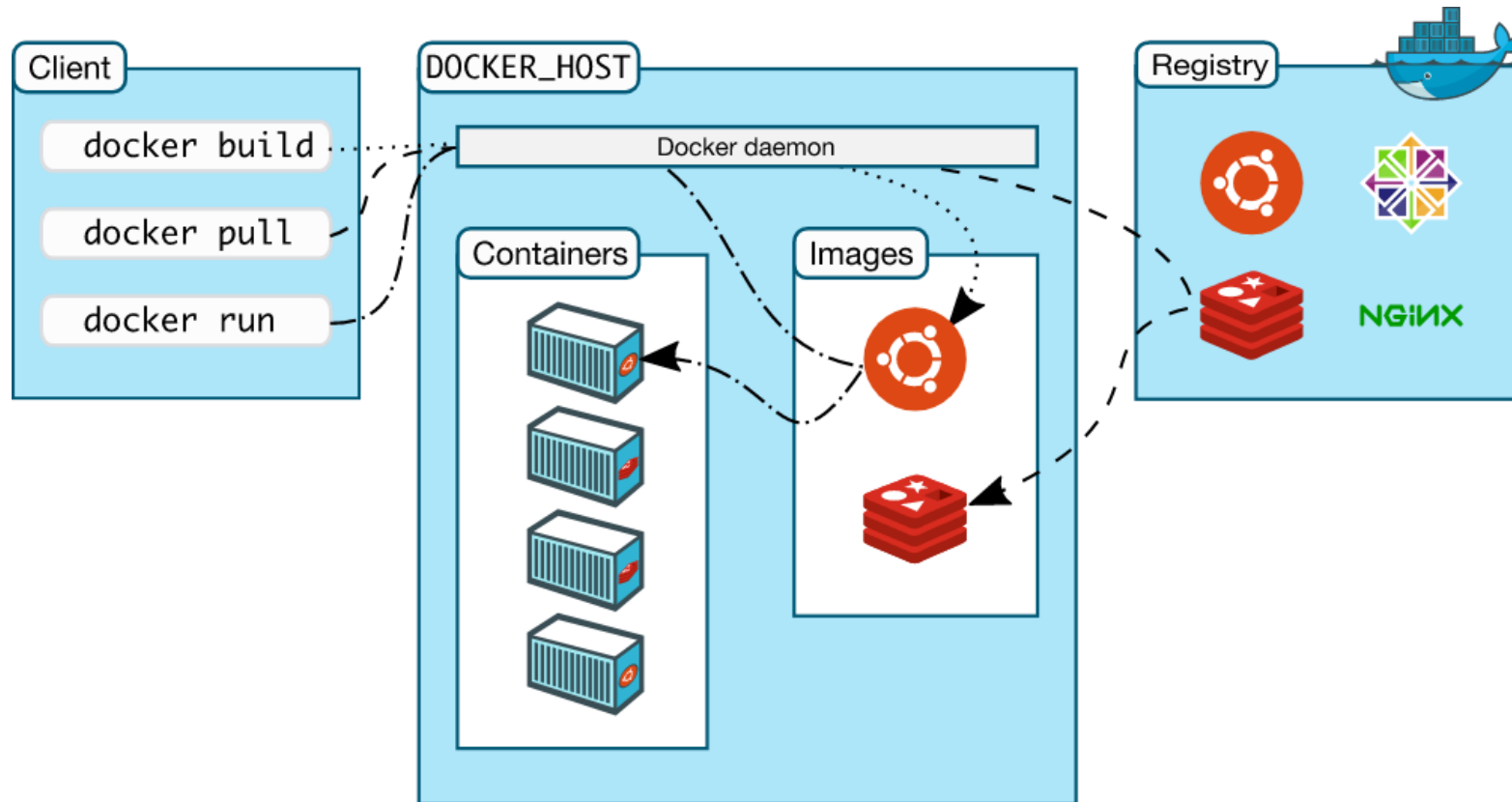


Öffentliche oder Private Registry

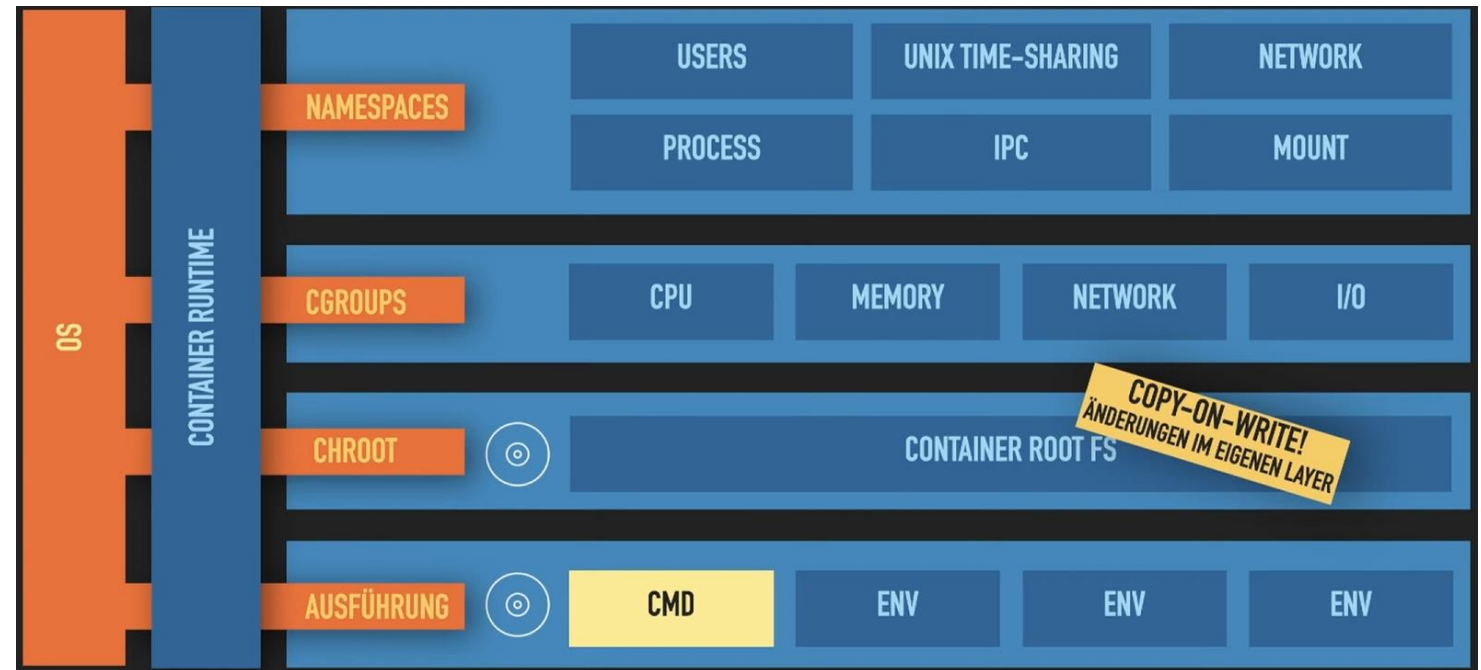
Docker Architektur



Docker Architektur



- Host OS (Linux Kernel Befehle)
- Container Runtime verwendet Befehle von OS (nichts neues)
- Isolierung über Namespaces, cgroups und chroot



- **Namespaces:** Was kann ein Container sehen?
Abgrenzung von Prozessen + kein Zugriff aufs Hostsystem
- **Cgroups** (Control Groups): Was kann ein Container benutzen?
Verwaltet Ressourcenzugriffe
- **Chroot** (Change Root): Ändert Root Directory für den aktuell laufenden Prozess
Setzt Root Dir auf Root Dir des Containers, Zugriff nur auf Dateien innerhalb des Root Dirs

→ **Isolation von Containern und Prozessen**

- Host Windows: hat namespaces, cgroups nachgebaut, + Hyper-V (VMs im Hintergrund)

<https://learn.microsoft.com/en-us/archive/msdn-magazine/2017/april/containers-bringing-docker-to-windows-developers-with-windows-server-containers>

Deklaration: Dockerfile

Dockerfile:

```
# Das Basisimage für unseren Container / oder FROM scratch
FROM node:18-alpine

# Anlegen und Wechsel in das Verzeichnis /app im Container
WORKDIR /app

# Kopieren der Projekt-Metadaten in das Verzeichnis /app..
COPY package.json .
# .. und des lokalen Verzeichnisses ./src in das Verzeichnis /app im Container
COPY src ./

# Installation der Projektabhängigkeiten mit dem Node Package Manager (npm)
RUN npm install

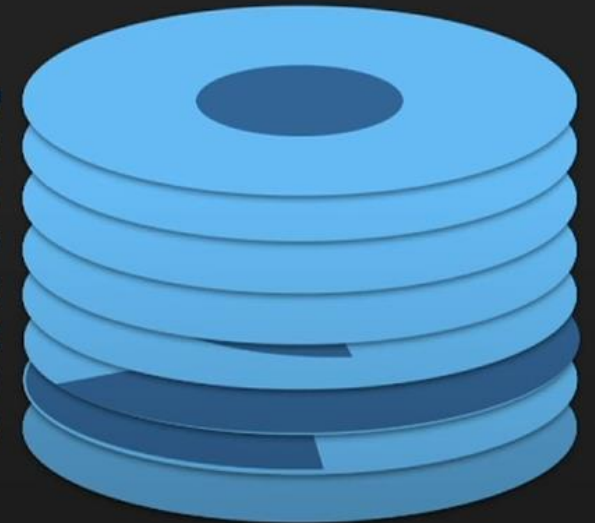
# Umgebungsvariablen für die Applikation setzen
ENV PORT=3000
ENV MSG="Hello World"

# Explizite Dokumentation: Die Applikation lauscht auf Port :3000
EXPOSE 3000

# Ausführen der Applikation als letzter Befehl
CMD ["node", "index.js"]
```

CONTAINER FILE SYSTEM

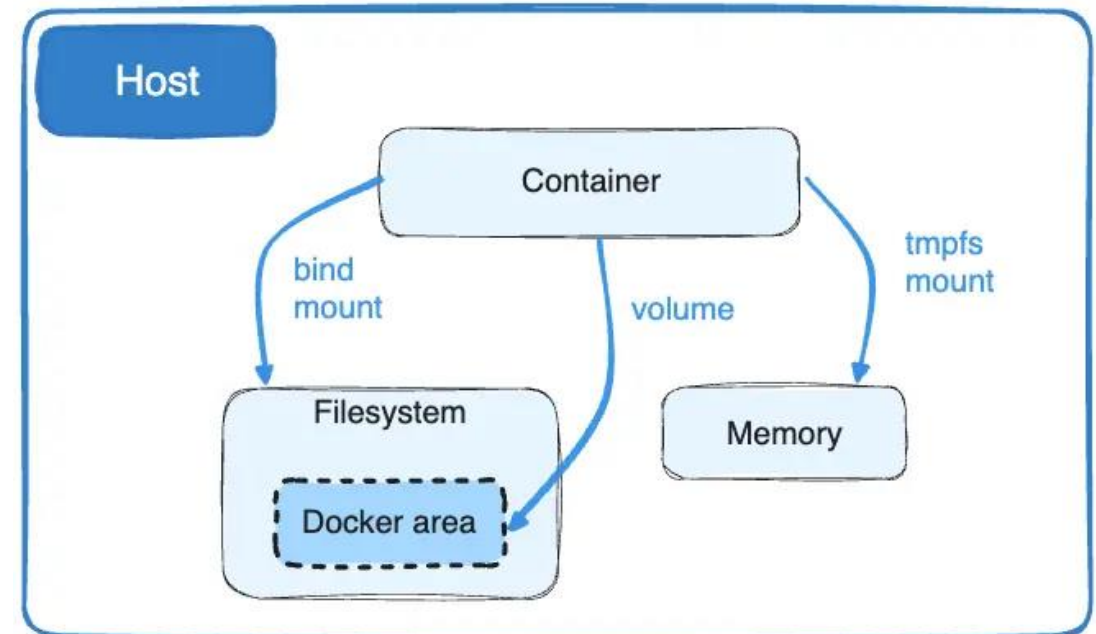
SHA256: E3DAF0...469AB9
SHA256: 4F3245...2F0865
SHA256: 1DD081...8700C0
SHA256: DEB2BF...E3DAF0
SHA256: 5FB98B...938007
SHA256: E58E34...CF5D10
SHA256: F944B3...570829
SHA256: DD565F...844F62





Demo: Dockerfile

Volumes werden genutzt,
da Container standard-
mäßig nicht persistent sind



- Konfigurationsdatei in YAML-Format
- Deklarativer Ansatz
- Kann in Versionierungssystemen verwaltet werden
- Nützlich für System aus mehreren Containern



Demo :

Docker Compose

- Entwicklungs- und Testumgebungen
- Continuous Integration/Continuous Deployment (CI/CD)
- Microservices-Architektur
- Legacy-Anwendungen

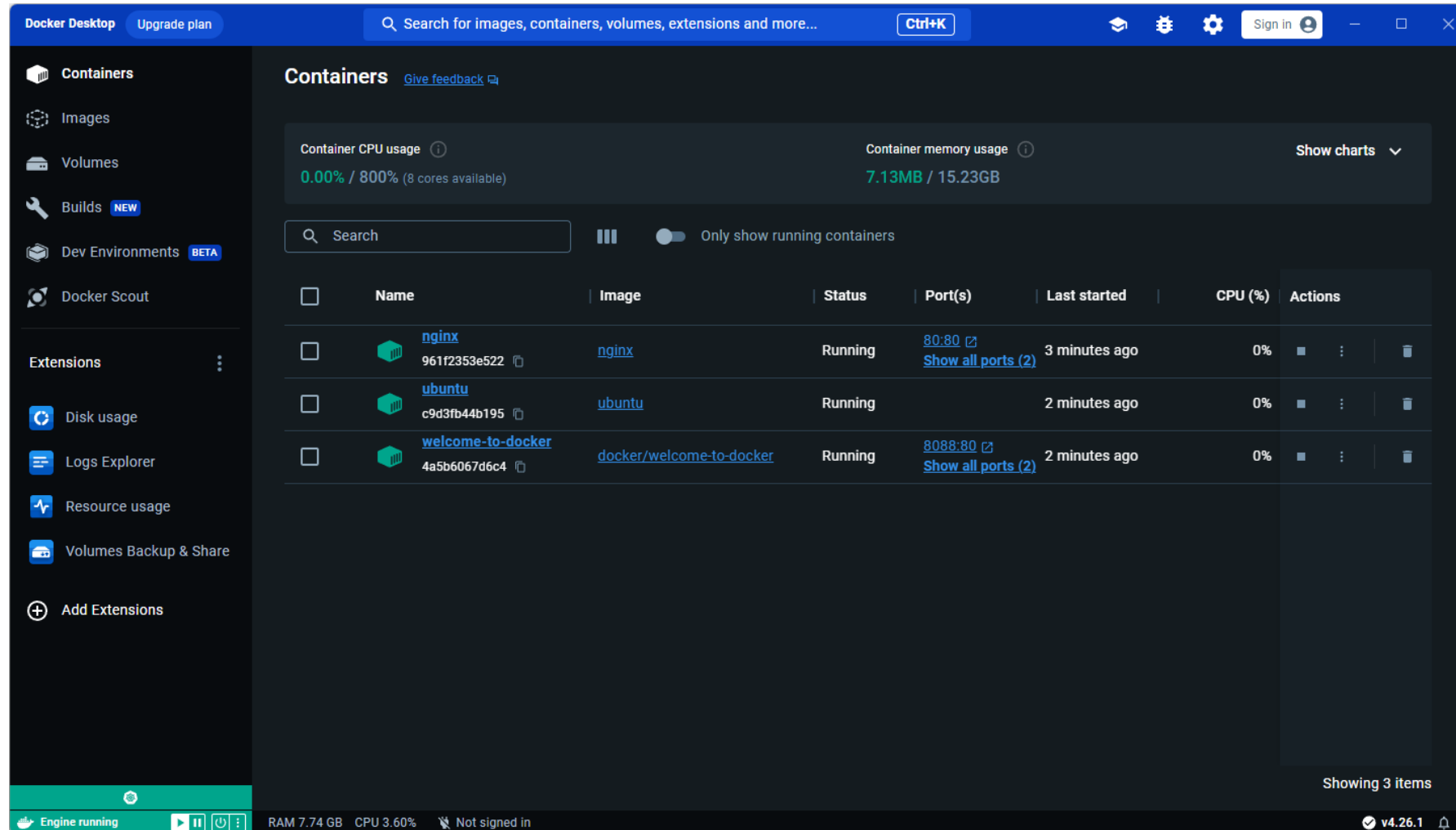
Dev Container werden genutzt, um das Problem „Es funktioniert auf meinem Rechner“ zu lösen

- Gleiche Umgebung für alle Entwickler



Demo: Dev Container

Dashboard: Docker Desktop



- Einfache Container-Verwaltung
- Headless
- Unterstützt externe Docker-Instanzen
- GitOps



Demo: Portainer



The screenshot displays the Portainer Community Edition dashboard. The left sidebar contains navigation links: Home, local (selected), Dashboard, App Templates, Stacks, Containers, Images, Networks, Volumes, Events, and Host. Below these are settings for Users, Environments, Registries, and Authentication logs. The main content area shows an 'Environment summary' section with a 'Dashboard' tab. The 'Environment info' section lists details: Environment (local), URL (/var/run/docker.sock), GPU (none), and Tags (-). Below this, five resource statistics are shown: 18 Stacks, 25 Containers (13 running, 10 stopped, 2 healthy, 0 unhealthy), 26 Images (5.7 GB), 7 Volumes, and 11 Networks. The bottom left corner indicates 'portainer.io Community Edition 2.19.4'.

Upgrade to Business Edition

portainer.io COMMUNITY EDITION

Home

local

Dashboard

App Templates

Stacks

Containers

Images

Networks

Volumes

Events

Host

Settings

Users

Environments

Registries

Authentication logs

portainer.io Community Edition 2.19.4

Environment summary

Dashboard

Environment info

Environment	local 4 8.2 GB - Standalone 24.0.7
URL	/var/run/docker.sock
GPU	none
Tags	-

18 Stacks

25 Containers
13 running 10 stopped
2 healthy 0 unhealthy

26 Images
5.7 GB

7 Volumes

11 Networks

- Daemon-freie Architektur
- Vollständig kompatibel zu Docker-CLI
- Rootless Betrieb
- Verbesserte Sicherheit

- Linux Containers (LXC) und erweiterter Daemon (LXD)
- Näher am traditionellen VM-Modell
- Systemcontainer statt Anwendungscontainer
- Anpassbares Sicherheitsmodell

- Effiziente Containerisierung
- Vergleich mit VMs
- Dockerfile und Docker Compose
- Entwicklungsumgebung mit Dev Container
- Dashboards zur vereinfachten Verwaltung

Im Rahmen dieser Übung wird Docker inklusive Dockerfile und Docker Compose genutzt, um ein System aus Frontend, Backend und Datenbank zu containerisieren.

<https://github.com/NickStudRepo/Docker-Kubernetes>

Im Rahmen dieser Übung wird ein Dev Container für
NodeJS in Visual Studio Code erstellt.

<https://github.com/NickStudRepo/Docker-Kubernetes>



Kubernetes



Kubernetes (K8s):

- Open Source (von Google entwickelt)
- Orchestrierung von Containern
- Deklarativer Ansatz (Soll zustand)



kubernetes

- **Container-Orchestrierung**: Bereitstellung, Aktualisierung, Ressourcenverwaltung und Skalierung
- **Geräte- und Netzwerkübergreifende Orchestrierung**
- Kubernetes oft in **Cloud** oder als **Hybrid** eingesetzt (AWS oder Microsoft Azure)

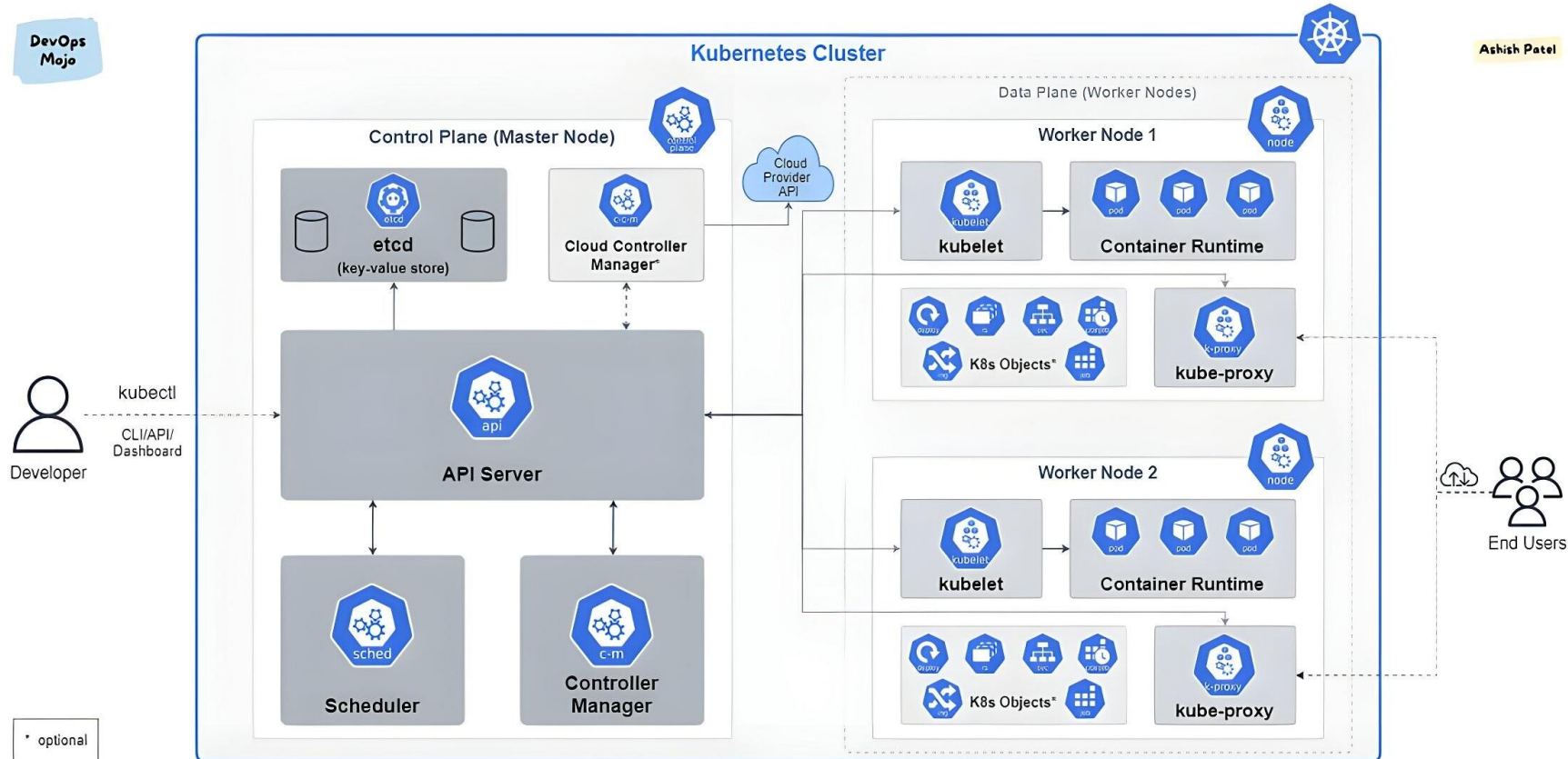
Features:

- **Load Balancing**: automatisches Verteilen von Lasten
 - **Automatisches Skalieren** (hoch und runterfahren von Containern)
 - **Selbstheilung** und **Ausfallsicherheit** durch Redundanzen
 - **Rolling Updates** und **Rollbacks**
- Ressourcenmanagement

→ **Keine Downtime!**

- **Kubernetes Cluster:** Gruppe von verbundenen Computern/ Nodes
- **Master Node** (Control Plane): zentraler Steuerungsknoten
- **Worker Node:** ausführenden Knoten, haben Container Runtime, enthalten Pods
- **Pod:** ist Wrapper um Container, enthält mind. 1 Container
Grundlegende/ kleinste Einheit die verwaltet wird

Kubernetes: Architektur



- **etcd**: verteilte Datenbank (Key, Value Store)
Speichert Zustand des Clusters, Konfigurationen
- **API Server**: Rest Schnittstelle (Interaktion mit Devs + Auth), Node Kommunikation, Überwachung ob Nodes im gewünschten Zustand sind
- **Scheduler**: Verteilung von Pods auf Nodes (Lastenausgleich), Ressourcenmanagement auf Nodes
- **Controller Manager**: Überwacht Controller (Replica Controller, Namespace Controller), Reagiert im Fehlerfall
- **Cloud Controller Manager**: ermöglicht Integration mit Cloud (Auth)
- **kubelet**: Master Kommunikation, Pod Management (Start, Stopp), Pod Networking (innerhalb eines Pods)
- **kube-proxy**: Netzwerkmanagement (Pod- und Node-übergreifend), Load Balancing (Verteilung des Netzwerkverkehrs auf die Nodes), ermöglicht jedem Pod stabilen Endpunkt (Service)
- **K8s objects**: ConfigMap und Secrets: wer darf mit Pod kommunizieren
- **Useranfragen an kube-proxy**: welcher Node antwortet ist Requestabhängig + Load Balancing Algorithmus (Round Robin ist default)

Kubernetes Konfiguration



- .yaml (Deklarativ)
- Deployment
- Service
- Secret
- ConfigMap

```
! webapp.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: webapp-deployment
5    labels:
6      app: webapp
7  spec:
8    replicas: 3
9    selector:
10     matchLabels:
11       app: webapp
12    template:
13     metadata:
14       labels:
15         app: webapp
16     spec:
17       containers:
18         - name: webapp
19           image: node:21-bullseye
20           ports:
21             - containerPort: 3000
```

```
39  apiVersion: v1
40  kind: Service
41  metadata:
42    name: webapp-service
43  spec:
44    type: NodePort
45    selector:
46      app: webapp
47    ports:
48      - protocol: TCP
49        port: 3000
50        targetPort: 3000
51        nodePort: 30100
52
```


- **Deployment:** Definiert wie Container bereitgestellt werden + Skalierung (Replika)
- **Service:** Definiert stabilen Endpunkt für Gruppe von Pods (Netzwerkschnittstelle)
- **Secret:** Verwaltet sensible Informationen (Auth)
- **ConfigMap:** Key Value Paare an Konfigurationen



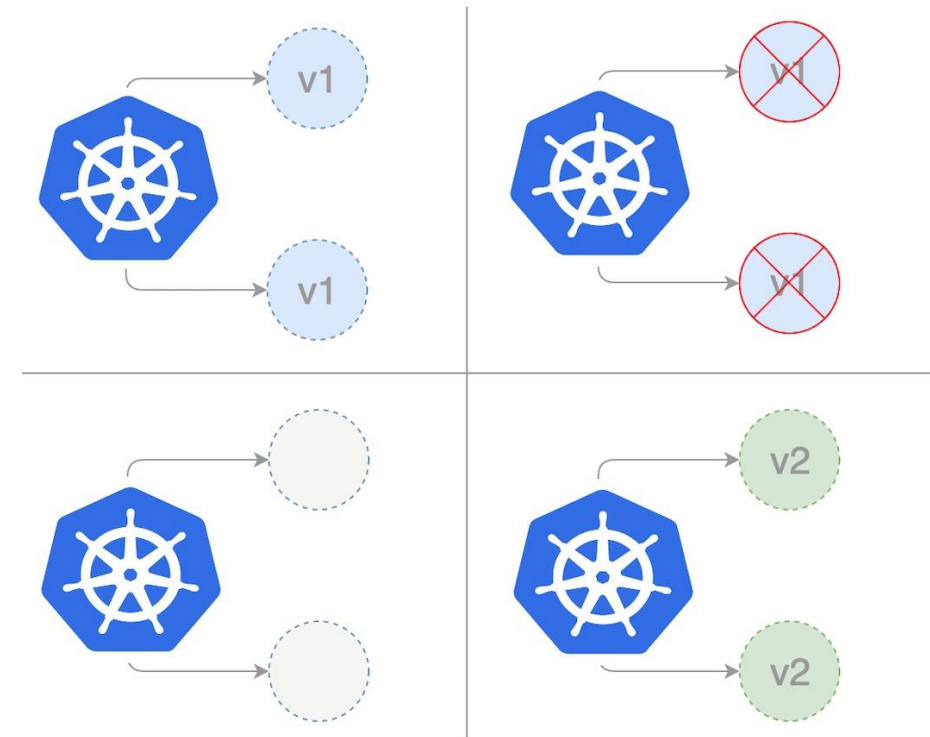
Demo: Kubernetes

Konfiguration eines einfachen Clusters

Deployment Strategie: Recreate



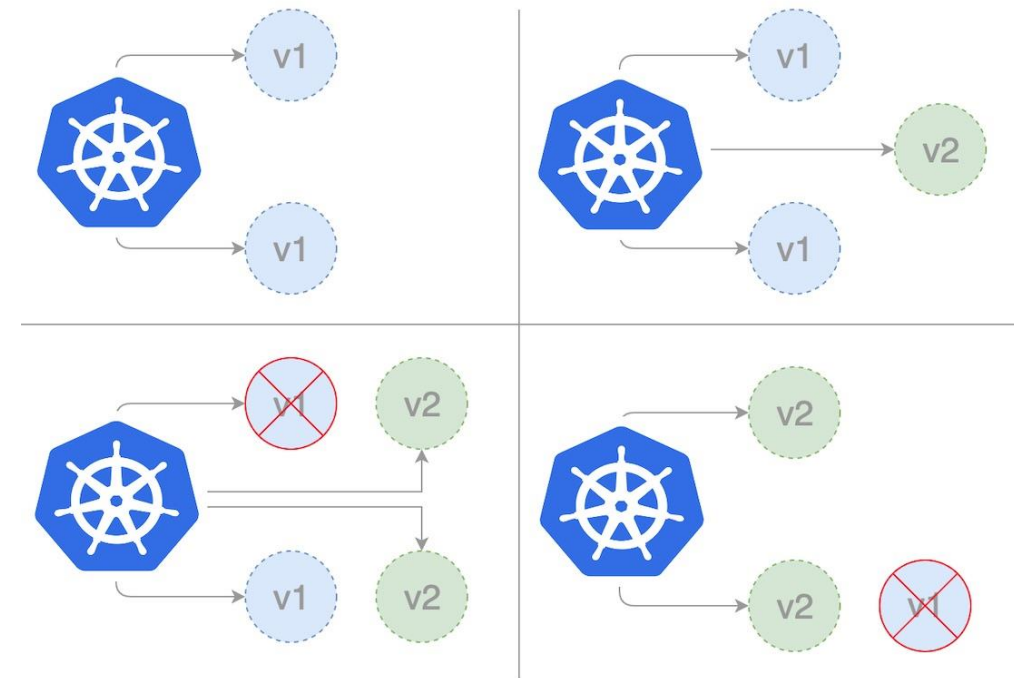
- Downtime während Update
 - Manuelles Zurücksetzen
- bei Fehlern



Deployment Strategie: Rolling Update und Rollback



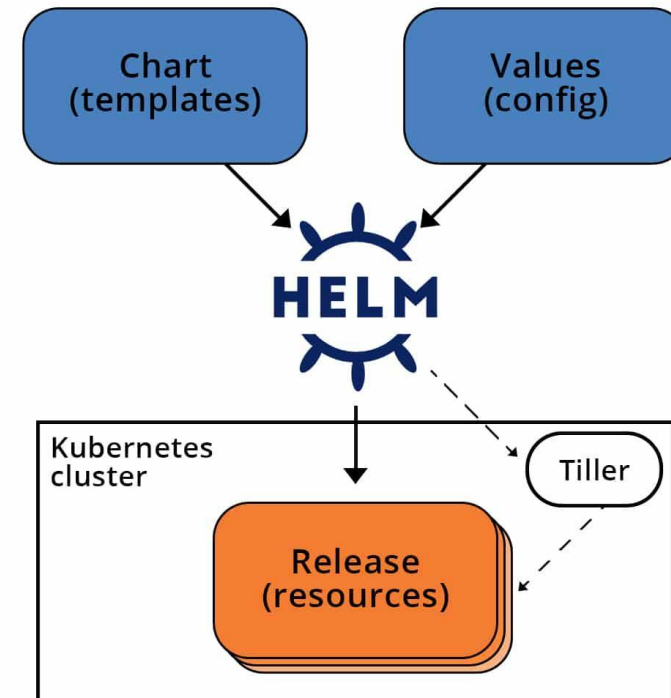
- Update ohne Downtime
- Historie für Rollback

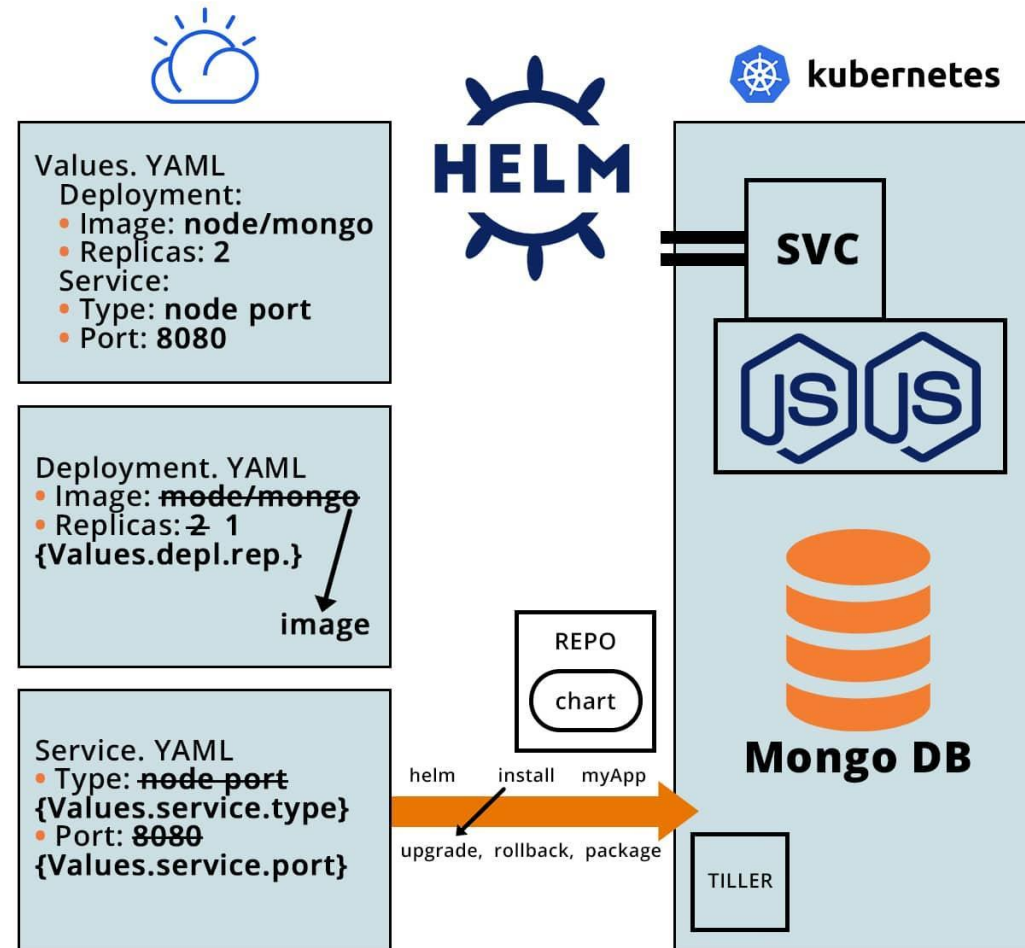




Demo: Rolling Update und Rollback

- Paketmanager für Kubernetes
- Wiederverwendbare Charts
- Anpassbar

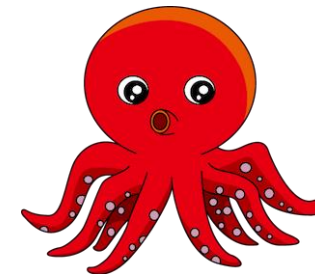




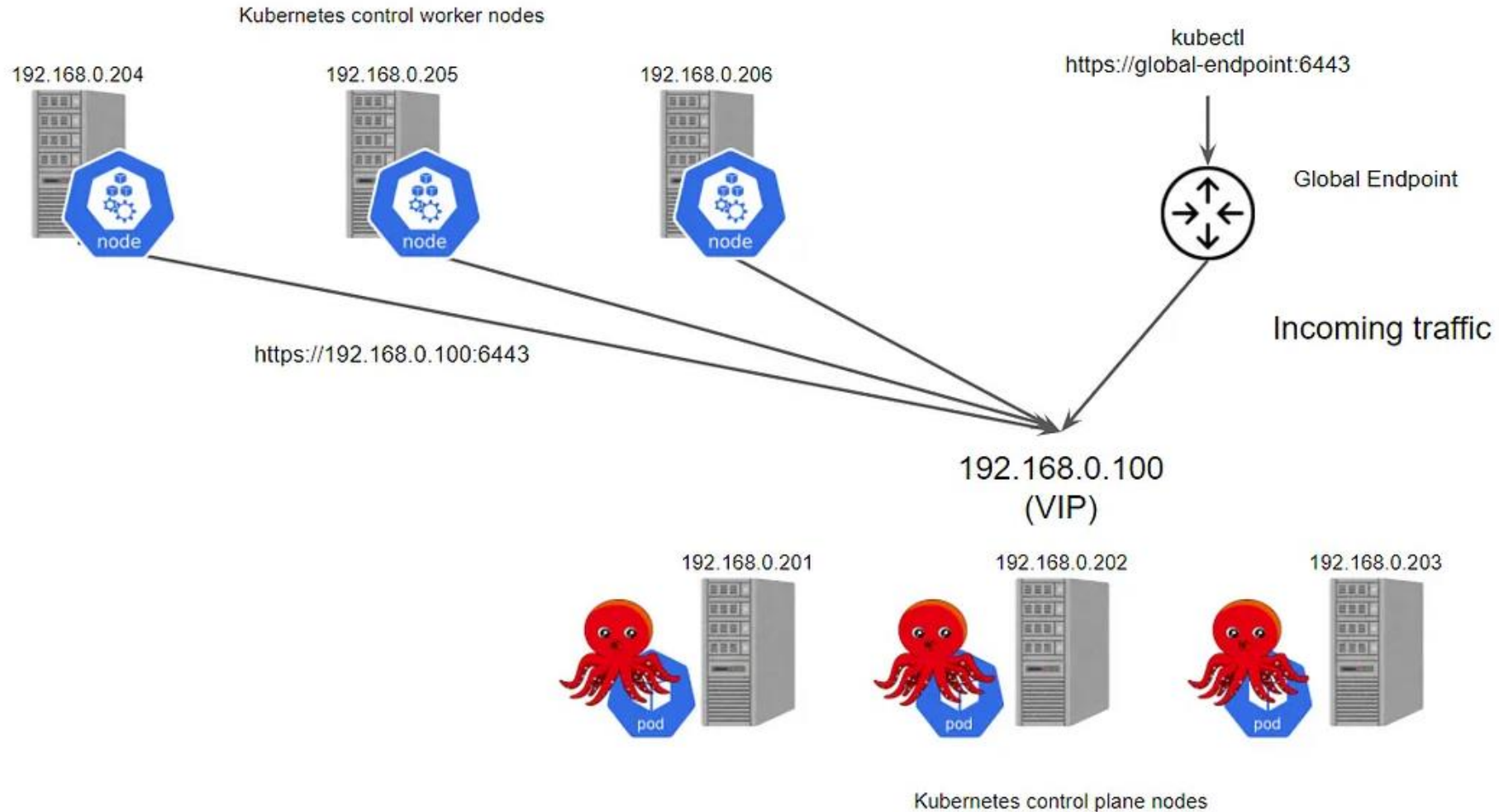


kube-vip stellt eine stabile und zuverlässige (externe) Virtual IP (VIP) für hochverfügbare Kubernetes Cluster bereit

- Failover und Redundanz
- Load Balancing



Virtual IP: kube-vip

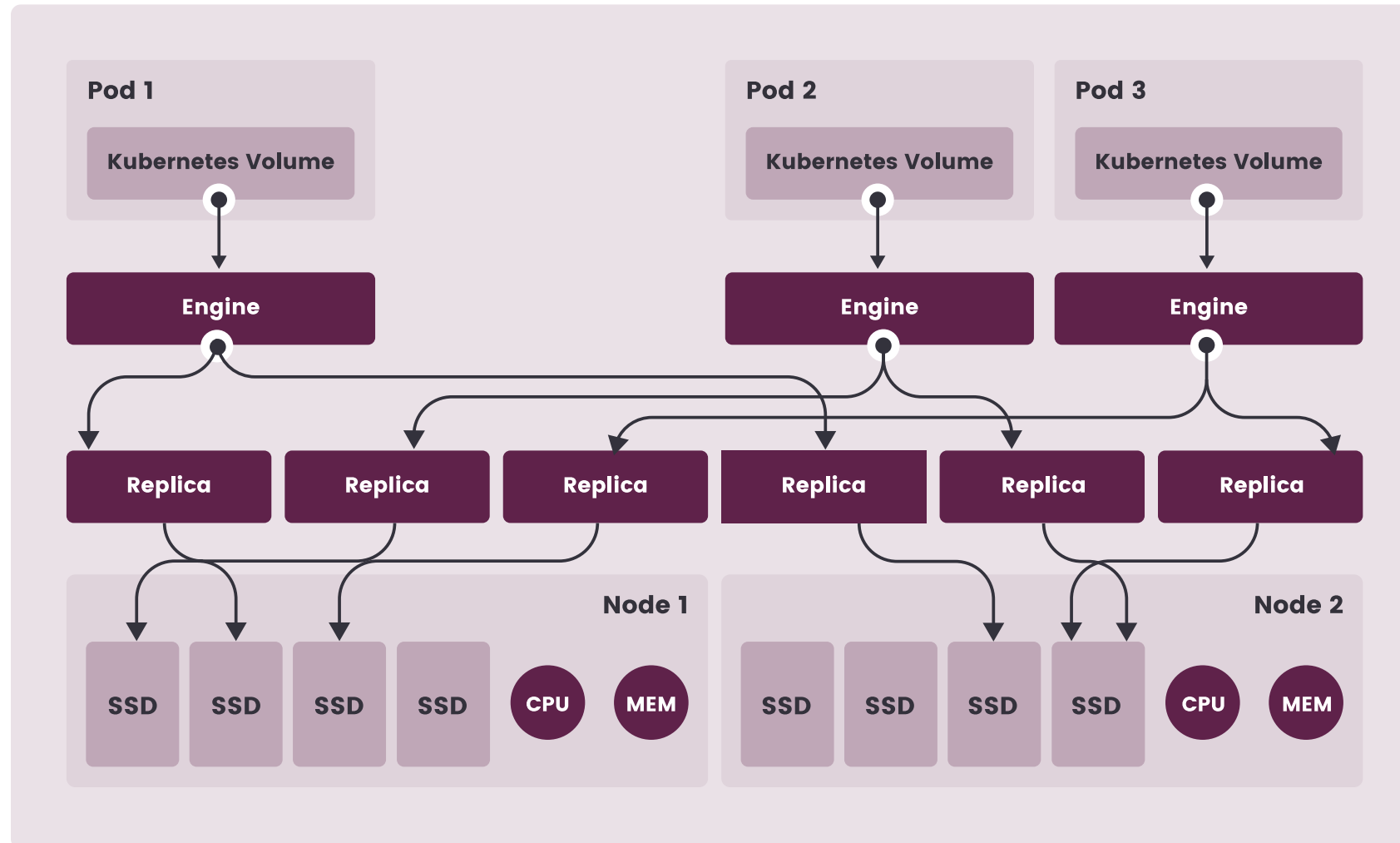


Longhorn stellt persistenten und hochverfügbaren
Speicher für containerisierte Anwendungen bereit

- Backup und Wiederherstellung
- Skalierbarkeit und Flexibilität



Persistent Storage: Longhorn



Demo: Portainer



Upgrade to Business Edition

portainer.io
COMMUNITY EDITION

Home

kubernetes-docker-des... x

kubectl shell

Dashboard

Custom Templates

Namespaces

Helm

Applications

Services

Ingresses

ConfigMaps & Secrets

Volumes

Cluster

Settings

Users

Environments

portainer.io Community Edition 2.19.4

Environment summary

Dashboard

Environment info

Environment	kubernetes-docker-desktop
URL	10.10.10.10:9001
Tags	-

5 Namespaces

9 Applications

4 Services

0 Ingresses

12 ConfigMaps

1 Secret

0 Volumes

- Integriert in Docker
- Geringerer Overhead
- Einfachheit
- Unterstützt Deklarativ und Imperativ

- Kubernetes als Orchestrierungsplattform
- Modulare und robuste Architektur
- Helm als Paketmanager
- High Availability durch Persistent Storage und Virtual IP
- Dashboards zur vereinfachten Verwaltung

Im Rahmen dieser Übungen werden mit kubectl und Helm Anwendungen deployed. Zudem, werden die Konzepte Scaling, Rolling Update und Rollback beleuchtet.

<https://github.com/NickStudRepo/Docker-Kubernetes>

- Synergie von Docker und Kubernetes
- Kubernetes eignet sich für komplexe Anwendungen
- Vereinfachung und Automatisierung
- Lernkurve und Community-Unterstützung

Was nehmt ihr aus der heutigen Vorstellung zu Docker
und Kubernetes mit?

- <https://k21academy.com/docker-kubernetes/docker-architecture-docker-engine-components-container-lifecycle/>
- <https://docs.docker.com/storage/volumes/>
- <https://www.portainer.io/hubfs/portainer-logo-black.svg>
- <https://auth0.com/blog/deployment-strategies-in-kubernetes/>
- <https://kruschecompany.com/helm-kubernetes/>
- <https://inductor.medium.com/say-good-bye-to-haproxy-and-keepalived-with-kube-vip-on-your-ha-k8s-control-plane-bb7237eca9fc>
- <https://kube-vip.io/images/kube-vip.png>
- <https://longhorn.io/img/logos/longhorn-horizontal-color.png>