



# Docker & Kubernetes

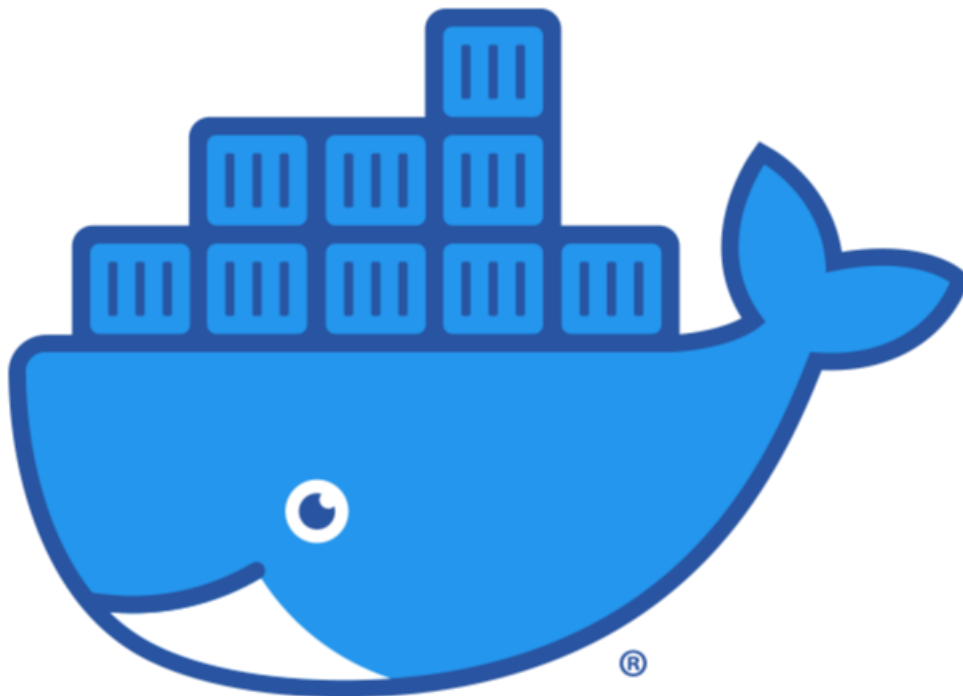
---

Tilman, Pascal, Jonas · 13.01.2025

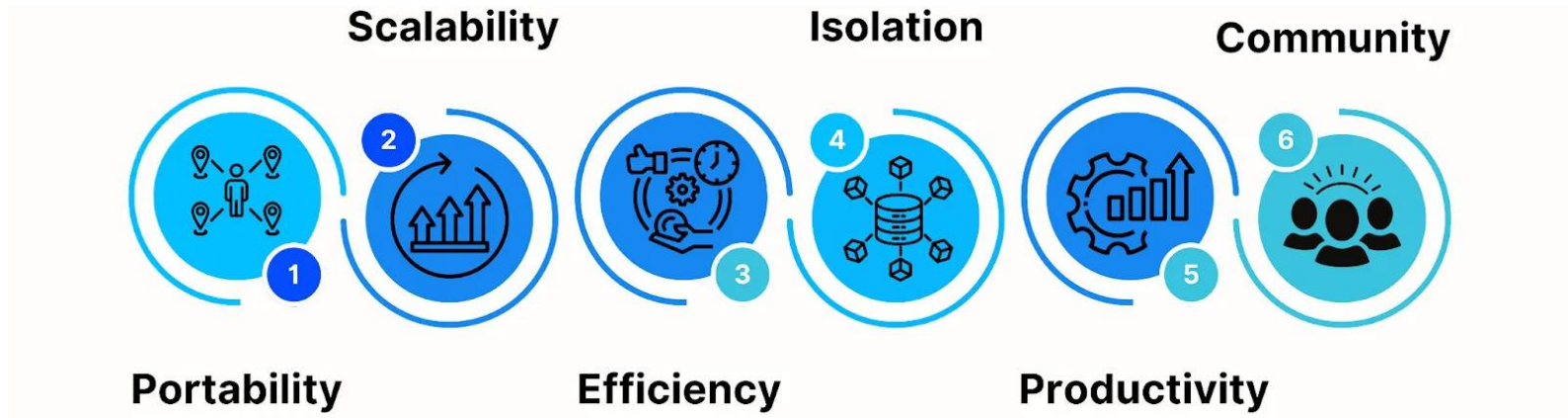


# Docker

---



# Wann Docker in Betracht ziehen?



<https://medium.com/appfoster/pros-and-cons-of-containerization-using-docker-000c36e809f8>

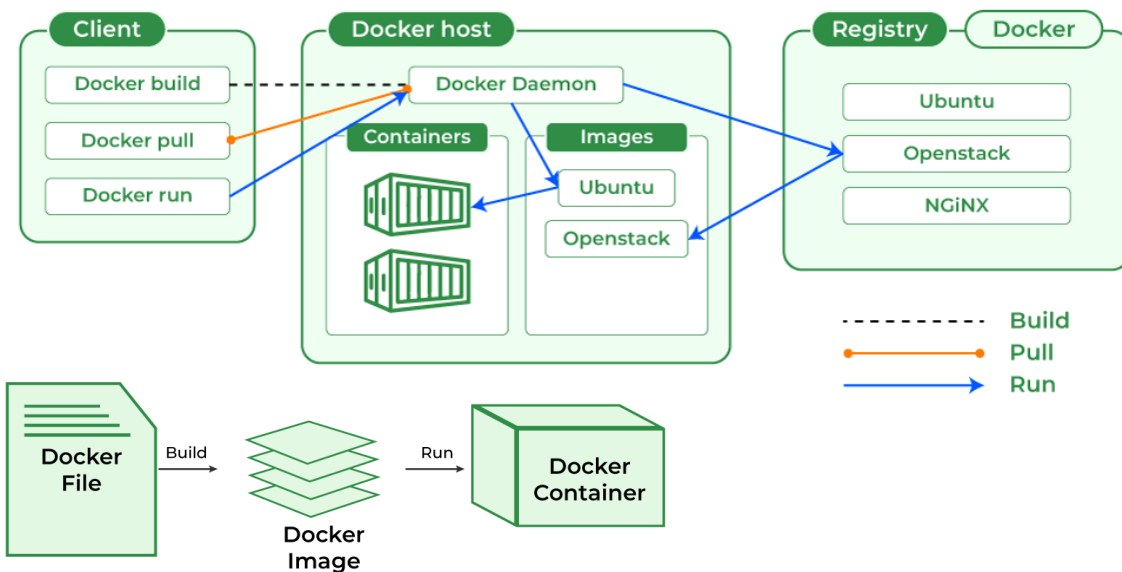


# Was ist Docker?

- Open-Source-Plattform für die Containerisierung zur Paketierung von Anwendungen mit Abhängigkeiten.

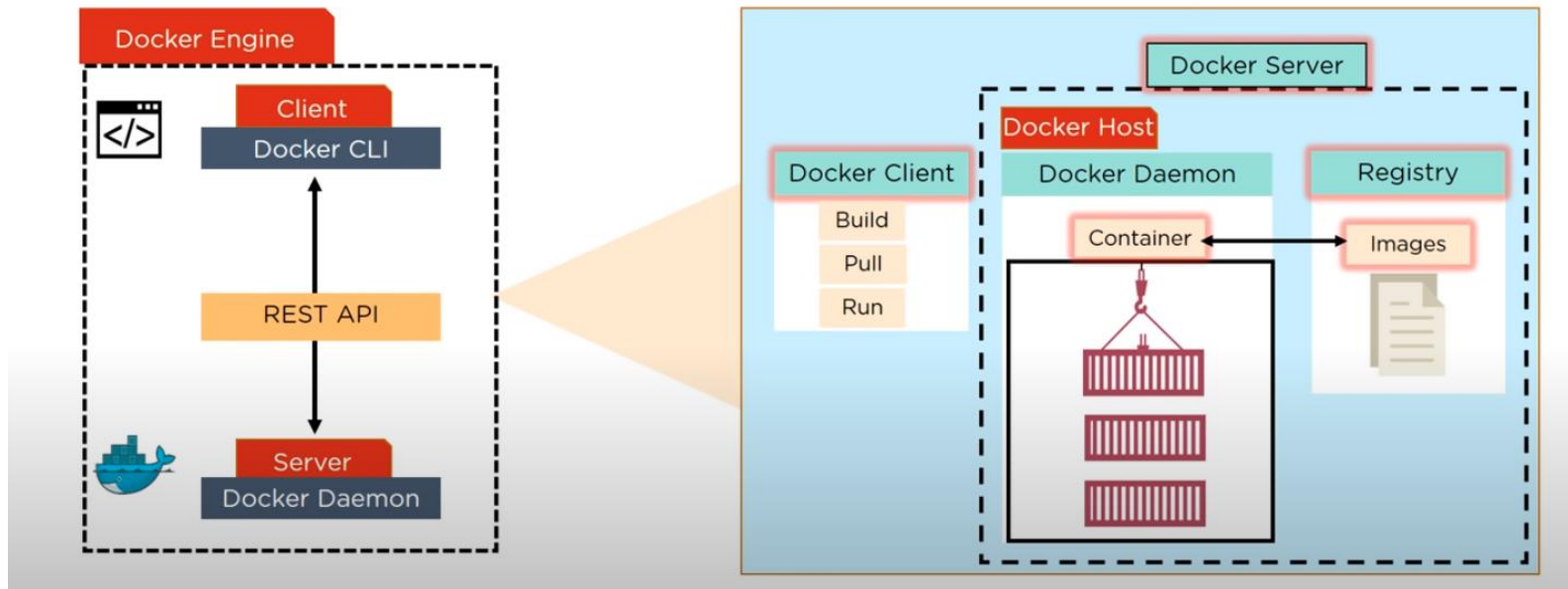
- Bestandteile:

- Engine
- Dockerfile
- Image
- Docker Hub
- Container
- Docker Compose



<https://www.geeksforgeeks.org/architecture-of-docker/>

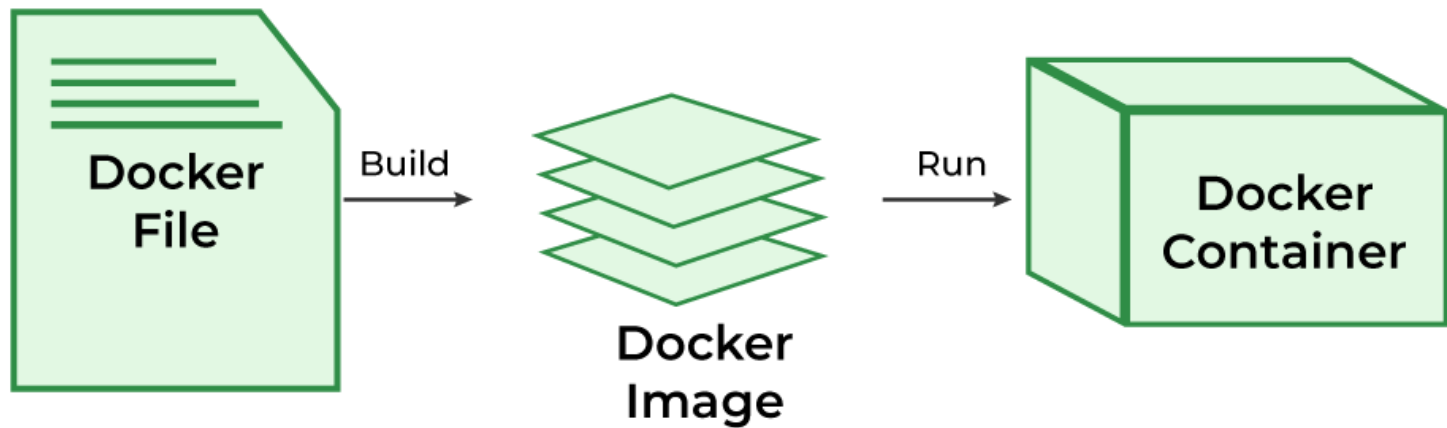
# Docker Engine & Docker Daemon



<https://www.youtube.com/watch?v=rOTqprHv1YE&t=5s>

# Build, Ship & Run

---





# Dockerfile

- einfaches, textbasiertes Skript (Definiert Docker-Image Erstellung)
- Legt Abhängigkeiten, Konfigurationen und Einrichtungsschritte fest
  - Docker Image enthält alle notwendigen Komponenten und Konfigurationen
  - Reproduzierbarkeit
- Definiert isolierte Umgebungen für Container-Ausführung
- Ermöglicht Erstellung von (identischen) Images schichtweise
- Entfall von manuellen Einrichtungen in verschiedenen Umgebungen
  - Portabilität
  - Reduktion Fehleranfälligkeit (Effizienz)

```
Dockerfile > ...
1 # Verwenden Sie das Node.js-Image in der
2 # Images sind auf https://hub.docker.com
3 FROM node:22.12.0-alpine
4
5 # Setzen Sie das Arbeitsverzeichnis im C
6 WORKDIR /app
7
8 # Kopieren Sie den Rest des Anwendungsco
9 COPY . .
10
11 # Installieren Sie die Abhängigkeiten, w
12 RUN npm install --legacy-peer-deps
13
14 # Öffnen Sie Port 3000 für den Container
15 EXPOSE 3000
16
17 # Bauen Sie die Anwendung
18 RUN npm run build
19
20 # Starten Sie die Anwendung
21 CMD ["npm", "start"]
```

# Dockerfile Befehle

---

## FROM

Erklärung: Die FROM-Anweisung legt das Basis-Image fest, auf dem das neue Image aufgebaut wird.

Beispiel: FROM ubuntu:20.04

## RUN

Erklärung: Die RUN-Anweisung führt Befehle im Image aus und erstellt eine neue Schicht.

Beispiel: RUN apt-get update && apt-get install -y nginx

## COPY

Erklärung: Die COPY-Anweisung kopiert Dateien und Verzeichnisse vom Host in das Image

Beispiel: COPY . /app

## ADD

Erklärung: Die ADD-Anweisung ähnelt COPY, bietet jedoch zusätzliche Funktionen.

Beispiel: ADD my\_archive.tar.gz /app

## CMD

Erklärung: Die CMD-Anweisung legt den Standardbefehl fest, der ausgeführt wird, wenn ein Container aus dem Image gestartet wird.

Beispiel: CMD ["nginx", "-g", "daemon off;"]

## EXPOSE

Erklärung: Die EXPOSE-Anweisung gibt an, welche Ports der Container zur Laufzeit freigibt.

Beispiel: EXPOSE 80

## WORKDIR

Erklärung: Die WORKDIR-Anweisung setzt das Arbeitsverzeichnis für nachfolgende Anweisungen.

Beispiel: WORKDIR /app

## Weitere:

- **VOLUME / ENV / ENTRYPOINT**

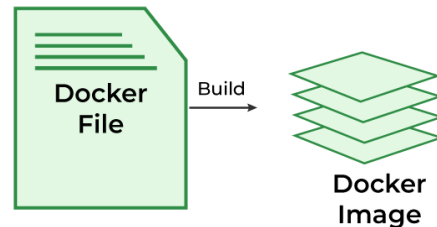


# Image

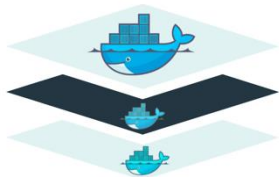


- Blueprint für die Container-Erstellung
  - Ermöglicht Erstellung von (identischen) Images schichtweise
  - Unveränderlich nach Erstellung (Konsistenz)

*docker build -t test\_app .*



- ➔ Konsistente Umgebung
- ➔ Anwendung funktioniert unabhängig von der Umgebung gleich. [Portabilität]
- ➔ Nutzbar in verschiedenen Projekten [Wiederverwendbarkeit]



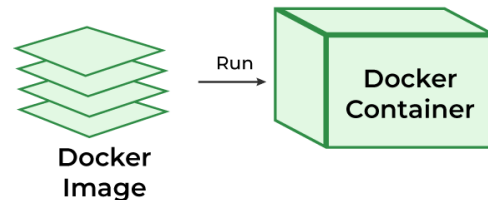
- ➔ Schichten zwischenspeicherbar & Wiederverwendbar (jede Schicht = Schritt im Dockerfile [bspw. Schicht 1: Installieren von Abh.])
- ➔ Beschleunigung des Build Prozesses (Effizienz)
- ➔ Identische Images in mehreren Instanzen skalierbar

<https://blog.packagecloud.io/what-is-a-docker-image/>

<https://www.tanium.com/blog/what-is-configuration-management/>

<https://blog.packagecloud.io/what-are-docker-image-layers/>

# Container



- Container = Instanzen von Docker Image, in welchen Anwendung ausführbar sind.

*`docker run -d -p 8080:80 my-image`*

- leichte, portable und isolierte Umgebung
  - Anwendungen und Abhängigkeiten konsistent auf verschiedenen Systemen ausführbar (Portabilität)
  - Schnell erstellbar & startbar -> einfache Replikation & Skalierung (Skalierbarkeit)
- teilen sich Host-Kernel -> keine vollständige BS-Virtualisierung notwendig (Effizienz)

# Quiz

---



# Frage 1

---

**Was ist der Hauptzweck einer Dockerfile?**

1. Die Schritte zur Erstellung eines Docker-Images definieren.
2. Den Lebenszyklus von Docker-Containern verwalten.
3. Docker-Images in einem Registry speichern.

# Frage 2

---

**Welche der folgenden Aussagen über Docker-Images ist wahr?**

1. Docker-Images sind schreibgeschützte Vorlagen zur Erstellung von Containern.
2. Docker-Images enthalten den Anwendungscode, die Laufzeitumgebung, Bibliotheken und Abhängigkeiten.
3. Docker-Images werden zur Verwaltung der Container-Orchestrierung verwendet.

# Frage 3

---

**Was gibt die FROM-Anweisung in einer Dockerfile an?**

1. Das Basis-Image, das zur Erstellung des Docker-Images verwendet wird.
2. Den Befehl, der beim Start des Containers ausgeführt wird.
3. Die Dateien, die vom Host in das Image kopiert werden sollen.

# Frage 4

---

## Wie erreichen Docker-Container Portabilität?

1. Indem sie alle notwendigen Komponenten und Konfigurationen im Container enthalten.
2. Indem sie ein vollständiges Betriebssystem innerhalb des Containers ausführen.
3. Indem sie den Kernel des Host-Betriebssystems teilen.

# Frage 5






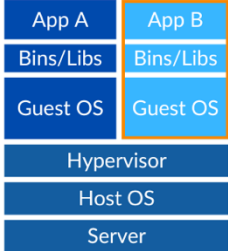
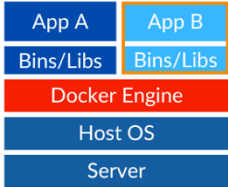
---

**Welche der folgenden Vorteile bieten Docker-Container?**

1. Leichtgewichtig und effiziente Ressourcennutzung.
2. Starke Isolation mit separaten Betriebssystemen.
3. Einfache horizontale Skalierung von Anwendungen.



# Docker (Containers) vs. Virtual Machine

					
	Betriebssystem	Sicherheit	Performance	Portabilität	Boot-Zeit
	Jeweils eigenes BS	Starke Isolation (BS)	Ressourcen-intensiv, Geringe Effizienz	Starr, Komplexe Integration	Einige Minuten
	Teilen sich Kernel	Geteilte Ressourcen	Leichtgewichtig Bessere Ressourcen-effizienz	Sehr mobil Einfacher Einsatz	Wenige Sekunden

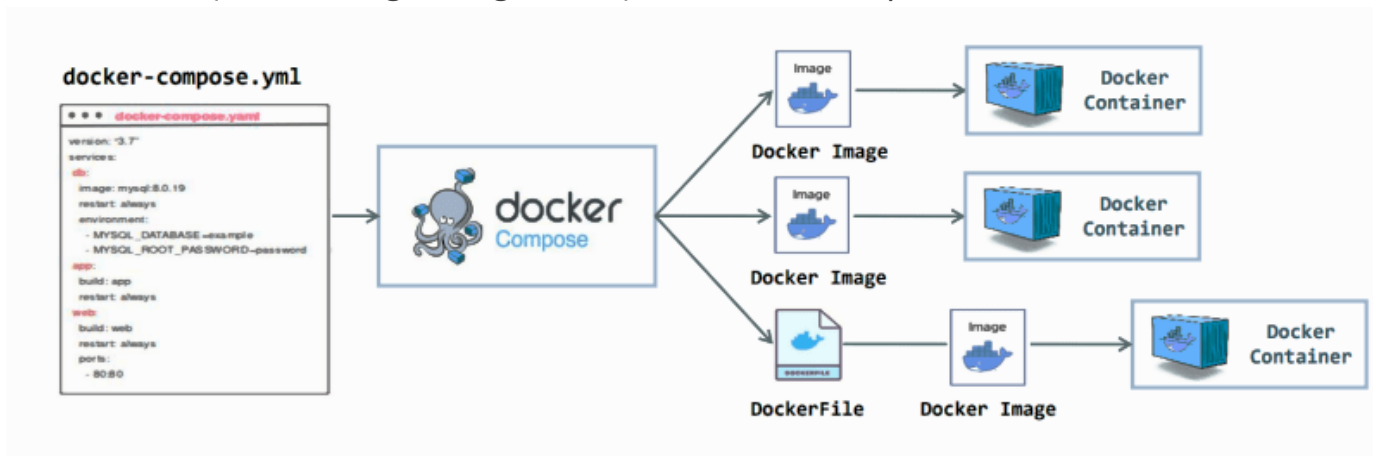
| [https://www.flaticon.com/de/kostenloses-icon/leistung\\_7172779](https://www.flaticon.com/de/kostenloses-icon/leistung_7172779) | <https://iconsout.com/de/icon/portabilitat-1499938>



# Docker Compose

Was tun bei Anwendungen mit mehreren Diensten?

- ➔ Docker Compose: Tool zum Definieren & Ausführen von Multi-Container-Docker-Anwendungen
- YAML-Datei (Anwendungskonfiguration) -> Je Dienst 1 separater Container



<https://blog.devops.dev/what-and-why-of-docker-compose-dc95314c74b8?gi=061ff2cb3bf0>

# Docker Hub

Content Management System

Data Science

Databases & Storage

Languages & Frameworks

Integration & Delivery

Internet of Things

Machine Learning & AI

Message Queues

Monitoring & Observability

Networking

Operating Systems

Security

Web Servers

Developer Tools

Web Analytics

Machine Learning & AI

tensorflow/tensorflow

Official Docker images for the machine learning framework TensorFlow (http://...)

☆2.7K ±50M+

pytorch/pytorch

PyTorch is a deep learning framework that puts Python first.

☆1.3K ±10M+

langchain/langchain

Building applications through composability

☆224 ±50M+

ollama/ollama

The easiest way to get up and running with large language models.

☆986 ±10M+

Trending this week

homeassistant/amd64-add...

☆158 ±5M+

paketobuildpacks/build

☆44 ±50M+

Most pulled images

memcached

Free & open source, high-performance, distributed memory object caching...

☆2.3K ±1B+

nginx/nginx

Official build of Nginx.

☆10K+ ±1B+

Cloud-basierter registry services zum Speichern und Freigeben von Images

- push Images
- pull Images

Registry/Hub

A registry Stores many static images

Container Engine

Pull

Images

Static, Persisted Container Image

Run

Container

Image-instance running an app process

Container Engine

Dockerfile

all commands to assemble an image

Build

Images

Static, Persisted Container Image

Run

Container

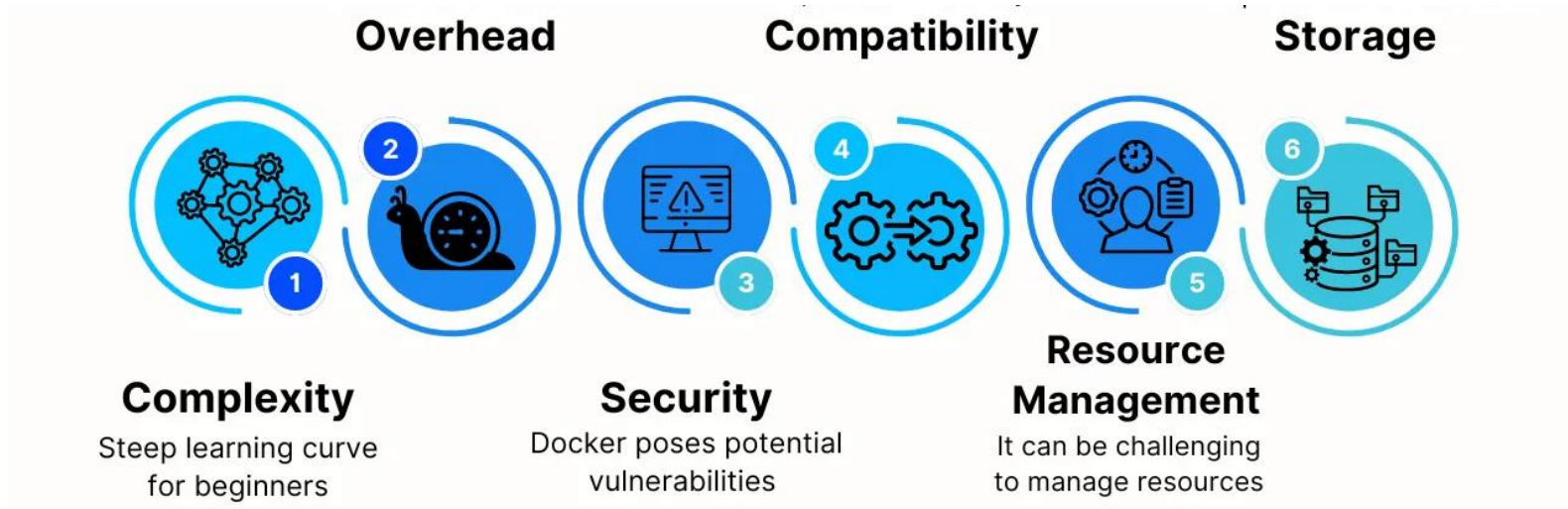
Image-instance running an app process in same system

Push

Registry/Hub



# Grenzen von Docker



# Docker - key points

---



## Portabilität

- Dockerfile (Schrittdefinition + Erstellung konsistenter & reproduzierbarer Images)
- Container (Komponenten & Konfiguration inbegriffen -> auf jedem System ausführbar)



## Skalierbarkeit

- Docker Compose (Definiert und verwaltet Multi-Container-Anwendungen. + Ermöglicht die einfache Orchestrierung und Skalierung von Diensten)

➤ **Kubernetes**



## Effizienz

- Docker Engine (Teilt den Kernel des Host-Betriebssystems)
- Images (schichtweise Aufbau -> zwischenspeicher- & wiederverwendbar)



## Isolation

- Docker Container (-> isolierte Laufzeitumgebung für Anwendungen + Trennung von Anwendungen und ihre Abhängigkeiten)

---

# Übung zu Docker



<https://dev.to/pulkit30/getting-started-with-docker-4b71>



# Kubernetes

---

- ❖ Herausforderungen
- ❖ Kubernetes to the rescue!
- ❖ Was ist Kubernetes?
- ❖ Was bietet Kubernetes?
- ❖ Grundlagen Kubernetes
- ❖ Kubernetes Komponenten
- ❖ Kubernetes in der Praxis
- ❖ Zusammenfassung



# Herausforderungen

---

## I. Traffic-Spikes und Leerlauf

- Die Infrastruktur von Data-Pipelines und der Datenhaltung ist oftmals starken Schwankungen im Traffic ausgesetzt.
- An Wochenenden laufen Services im „Leerlauf“.
- Montag morgens werden Services mit Anfragen „überflutet“.
- Services im Leerlauf erzeugen Kosten, ohne einen wirtschaftlichen Nutzen zu erbringen.
- Überlastete Services bergen die Gefahr von Datenverlust oder wirtschaftlichem Schaden.





# Herausforderungen

---

## II. Komplexe Infrastruktur

- Infrastruktur für Datenverarbeitung und Datenhaltung kann über den Verlauf der Zeit komplex und unübersichtlich werden.
- Bereitstellen der Infrastruktur kostet immer mehr Zeit.
- Infrastruktur wird immer schwerer wartbar.
- Entwickler können leicht den Überblick über die Infrastruktur verlieren.



# Herausforderungen

---

## III. Replizierbare Infrastruktur

- Kopien (Replikate) bestehender Infrastruktur müssen einfach und ohne großen Zeitaufwand erstellbar sein.
- Dies erlaubt es, Backup-Systeme zu erstellen, oder Kopien bewährter Infrastruktur bei verschiedenen Kunden zu deployen.



# Herausforderungen

---

## IV. Hohe Verfügbarkeit von Services

- Im Betrieb muss eine hohe Verfügbarkeit der Services garantiert werden.
- Ausfälle können zum Verlust von Daten oder finanziellen Verlusten führen.
- Ausfälle einzelner Services müssen abgefangen werden.



# Kubernetes

---

- ❖ Herausforderungen
- ❖ Kubernetes to the rescue!
- ❖ Was ist Kubernetes?
- ❖ Was bietet Kubernetes?
- ❖ Grundlagen Kubernetes
- ❖ Kubernetes Komponenten
- ❖ Kubernetes in der Praxis
- ❖ Zusammenfassung



# Kubernetes to the rescue!

---





# Kubernetes to the rescue!

- Mithilfe von Docker und Containerisierung erlaubt Kubernetes, diese Herausforderungen zu überwinden.
- I. Traffic-Spikes und Leerlauf:
  - Automatische horizontale Skalierung von Services
  - Loadbalancing und Routing
- II. Komplexe Infrastruktur:
  - Infrastruktur wird deklarativ und übersichtlich mit Quellcode beschrieben
  - Erlaubt Versionskontrolle und einfache Dokumentation von Infrastruktur und Service-Architektur
- III. Replizierbare Infrastruktur:
  - Durch deklarative Beschreibung der Infrastruktur lässt sich Infrastruktur immer in derselben Konfiguration ausliefern
- IV. Hohe Verfügbarkeit:
  - Automatisches Neustarten von „abgestürzten“ Containern
  - Automatisches Loadbalancing, um Überlastungen zu vermeiden



# Kubernetes

---

- ❖ Herausforderungen
- ❖ Kubernetes to the rescue!
- ❖ Was ist Kubernetes?
- ❖ Was bietet Kubernetes?
- ❖ Grundlagen Kubernetes
- ❖ Kubernetes Komponenten
- ❖ Kubernetes in der Praxis
- ❖ Zusammenfassung



# Was ist Kubernetes?

---

- Open-Source-Plattform ...
  - Zur Verwaltung von containerisierten Services
  - Deklarativen Erstellung digitaler Infrastruktur
  - Automatisierten Verwaltung von container-basierten, verteilten Services
- „Docker-Compose auf Steroiden“.
- Baut auf Containern (Docker, Podman, etc.) als zentrale Technologie auf.
- Nach dem Linux-Kernel das größte Open-Source Projekt der Welt.
- In dieser Präsentation: Eine sehr simple Übersicht über die wichtigsten Aspekte von Kubernetes.





# Kubernetes

---

- ❖ Herausforderungen
- ❖ Kubernetes to the rescue!
- ❖ Was ist Kubernetes?
- ❖ Was bietet Kubernetes?
- ❖ Grundlagen Kubernetes
- ❖ Kubernetes Komponenten
- ❖ Kubernetes in der Praxis
- ❖ Zusammenfassung



# Was bietet Kubernetes?

---

## Service Discovery

- Kubernetes übernimmt Netzwerkkonfiguration innerhalb des Clusters.

## Load Balancing

- Automatische Lastverteilung auf die einzelnen Services.
- Interner Load-Balancer verteilt die Last innerhalb des Clusters auf freie Services.

## Storage Orchestration

- Automatisches Einbinden von Dateisystemen.
- Automatisches Einbinden von Cloud-Storage.

## Automatische Rollouts und Rollbacks

- Gewünschter Zustand des Systems muss lediglich in Konfiguration beschrieben werden.

## Ressourcen-Zuweisung für einzelne Container

- Kontrolle, wie viel CPU und/oder RAM ein einzelner Container erhält.



# Was bietet Kubernetes?

---

## Secret- und Konfigurationsmanagement

- Umgebungsvariablen, Schlüssel und Konfigurationen können sicher verwaltet werden.
- Umgebungsvariablen und Schlüssel können im laufenden Betrieb ausgetauscht werden.
- Konfigurationen und Secrets müssen nicht in separaten Dateien oder in Container-Images mitgeliefert werden.

## Horizontales Skalieren von Services

- Automatisches Skalieren von Services bei definierten Lastgrenzen.
- Zusätzliche Services werden automatisch gestartet, sollte die Last zu hoch werden.
- Nicht benötigte Kopien werden bei niedriger Last gestoppt.
- Spart Ressourcen bei niedriger Last.
- Zusätzliche Leistung wird nur benötigt, wenn Last dies auch erfordert.
- Erlaubt das Einsparen von Geld und Ressourcen und den Ausgleich von Lastspitzen.

## Self-Healing

- Abgestürzte Services werden automatisch neu gestartet.
- Kubernetes überwacht den Zustand der einzelnen Container.
- Backup-Container werden automatisch zwischengeschaltet.



# Kubernetes

---

- ❖ Herausforderungen
- ❖ Kubernetes to the rescue!
- ❖ Was ist Kubernetes?
- ❖ Was bietet Kubernetes?
- ❖ Grundlagen Kubernetes
- ❖ Kubernetes Komponenten
- ❖ Kubernetes in der Praxis
- ❖ Zusammenfassung



# Grundlagen Kubernetes

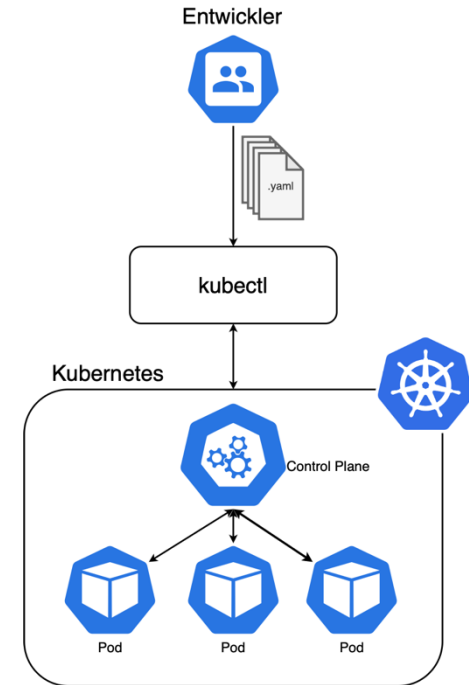
---

- Automatisiert die Verwaltung und Steuerung von Container-Applikationen.
- Überwacht den Status der einzelnen Container.
- **Self-Healing:** Container werden automatisch neu gestartet, sollten sie abstürzen.
- **Skalierbar:** Eine Applikation kann auf mehreren Replikaten laufen. Macht Load-Balancing möglich.
- **Sicher:** Container kommunizieren innerhalb eines internen Netzwerkes. Zugang von Außen ist eingeschränkt.
- **Automatische Verwaltung:** Es muss lediglich der Zustand eines Schwarms an Containern beschrieben werden.



# Grundlagen Kubernetes

- Container werden in einem sogenannten „Cluster“ von Kubernetes verwaltet.
- Kubernetes übernimmt den Aufbau und die Steuerung des Clusters.
- Kubernetes regelt die Kommunikation der Container innerhalb des Clusters.
- Kubernetes regelt die Kommunikation mit der Außenwelt.





# Grundlagen Kubernetes

- Zentrales Feature: **Deklarativer Ansatz**
- Der Aufbau und gewünschte Zustand eines Clusters wird in **.yaml**-Dateien beschrieben.
- Tatsächliches Herstellen des Zustands wird von Kubernetes übernommen.

```
apiVersion: apps/v1 # for k8s versions before
1.9.0 use apps/v1beta2 and before 1.8.0 use
extensions/v1beta1
kind: Deployment
metadata:
  name: redis-master
spec:
  selector:
    matchLabels:
      app: redis
      role: master
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
        - name: master
          image: registry.k8s.io/redis:e2e # or
just image: redis
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      ports:
        - containerPort: 6379
```



# Grundlagen Kubernetes

---

- **Command-Line-Tool: kubectl:**
  - Ermöglicht die Interaktion mit Kubernetes
  - Erlaubt es Entwicklern einfache Interaktion mit den Interfaces eines Clusters
  - Erlaubt es, Aktionen und Befehle automatisiert auszuführen mithilfe von Scripts



```
kubectl config set-context --current --namespace=myexamplnamespace
```





# Grundlagen Kubernetes

- **Erweiterbar:**

- Zahlreiche Addons und Plugins erlauben es, Cluster immer weiter auszubauen
- Erlauben Anbindung an APIs von Cloud-Anbietern
- Anbindung an spezielle Services
- Zusätzliche Konfiguration
- Feinere Kontrolle über den Cluster
- ...

```
minikube addons list
```

ADDON NAME	MAINTAINER
ambassador	3rd party (Ambassador)
auto-pause	minikube
cloud-spanner	Google
csi-hostpath-driver	Kubernetes
dashboard	Kubernetes
default-storageclass	Kubernetes
efk	3rd party (Elastic)
freshpod	Google
gcp-auth	Google
gvisor	minikube
headlamp	3rd party (kinvolk.io)
helm-tiller	3rd party (Helm)
inaccel	3rd party (InAccel [info@inaccel.com])
ingress	Kubernetes
ingress-dns	minikube
inspektor-gadget	3rd party (inspektor-gadget.io)
istio	3rd party (Istio)
istio-provisioner	3rd party (Istio)
kong	3rd party (Kong HQ)
kubeflow	3rd party
kubevirt	3rd party (KubeVirt)
logviewer	3rd party (unknown)
metallb	3rd party (MetalLB)
metrics-server	Kubernetes
nvidia-device-plugin	3rd party (NVIDIA)
nvidia-driver-installer	3rd party (NVIDIA)
nvidia-gpu-device-plugin	3rd party (NVIDIA)
olm	3rd party (Operator Framework)
pod-security-policy	3rd party (unknown)
rty (marcnuri.com)	



# Kubernetes Komponenten

---

## Pop-Quiz:

### 1. *Kubernetes* ist...

- a. Eine Plattform zur Orchestrierung von Applikationen
- b. Eine Alternative zu Docker
- c. Ein Framework zur Wartung von Infrastruktur





# Kubernetes

---

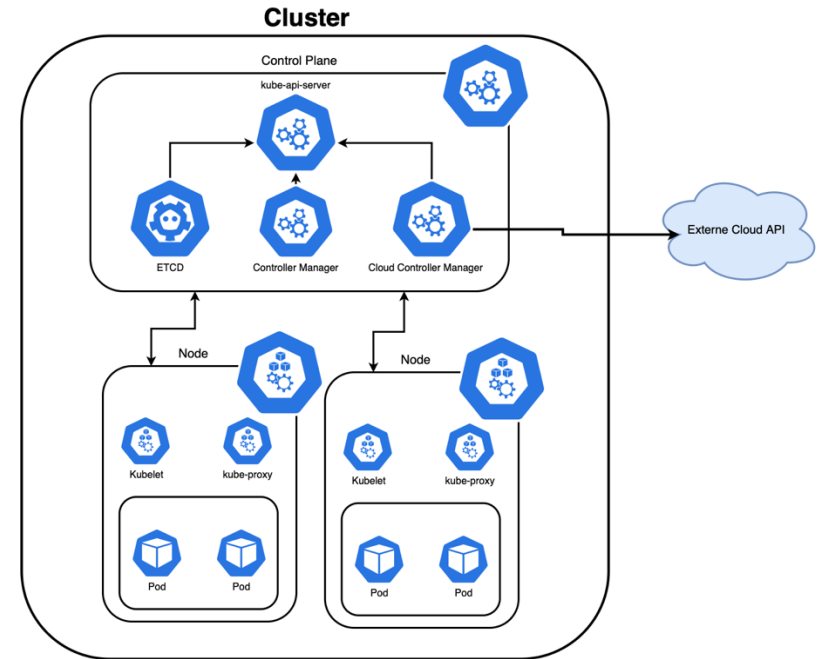
- ❖ Herausforderungen
- ❖ Kubernetes to the rescue!
- ❖ Was ist Kubernetes?
- ❖ Was bietet Kubernetes?
- ❖ Grundlagen Kubernetes
- ❖ Kubernetes Komponenten
- ❖ Kubernetes in der Praxis
- ❖ Zusammenfassung



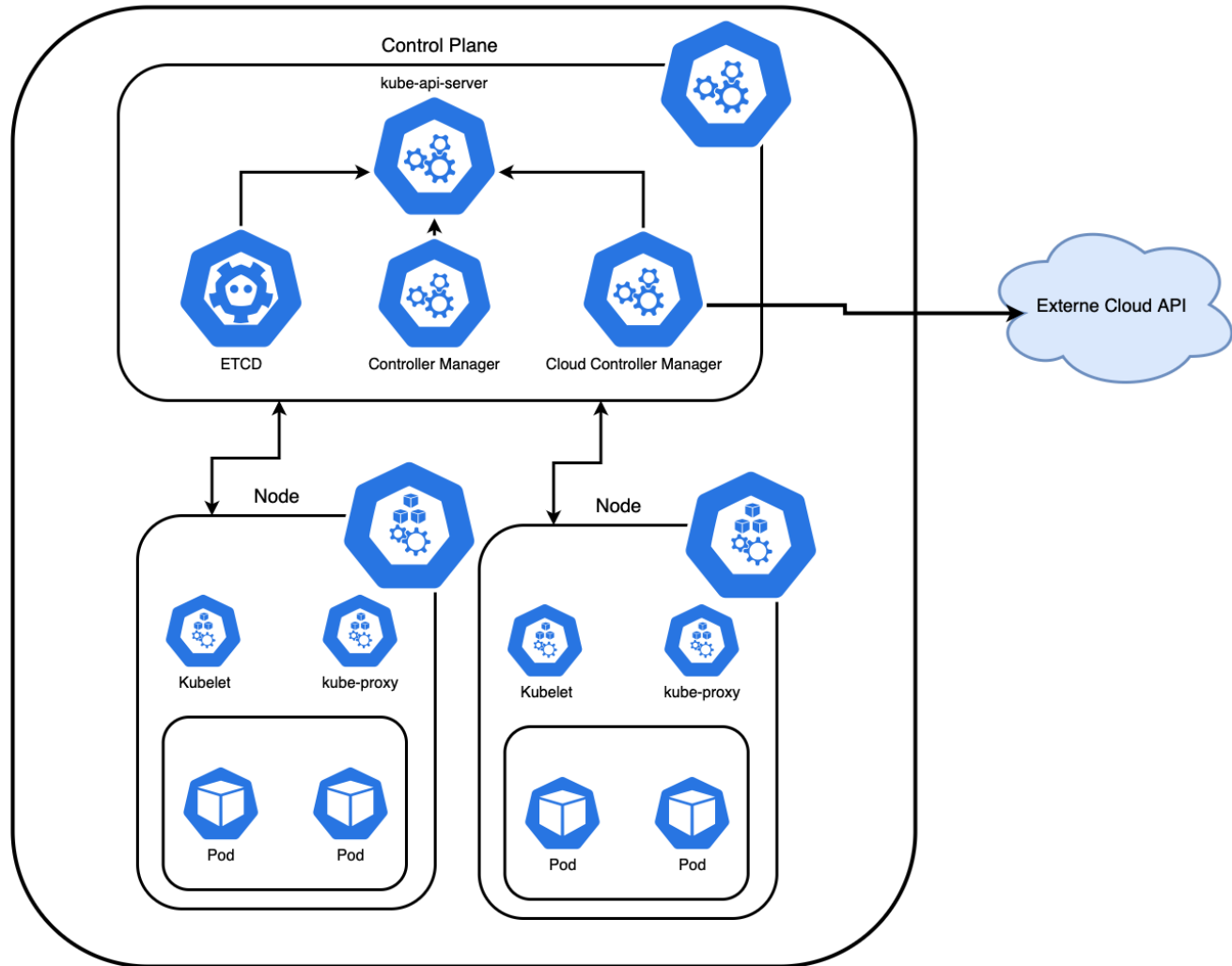
# Kubernetes Komponenten

## Zentraler Begriff: Der Cluster

- Wird deklarativ mithilfe von YAML-Files konfiguriert.
  - Besteht aus mehreren Komponenten.
1. Control Plane:
    - Enthält mehrere Kubernetes-Services zur Verwaltung und Steuerung des Clusters
  2. Nodes:
    - Enthalten die eigentlichen Container-Services
    - Container werden innerhalb von sogenannten Pods verwaltet



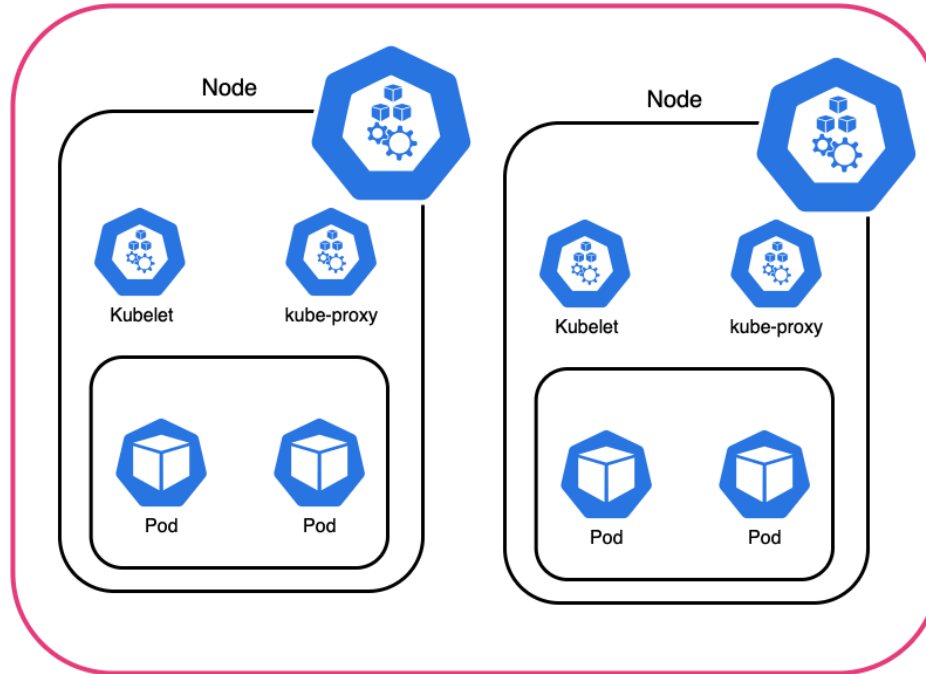
# Cluster





# Kubernetes Komponenten

## I: Namespace





# Kubernetes Komponenten

---

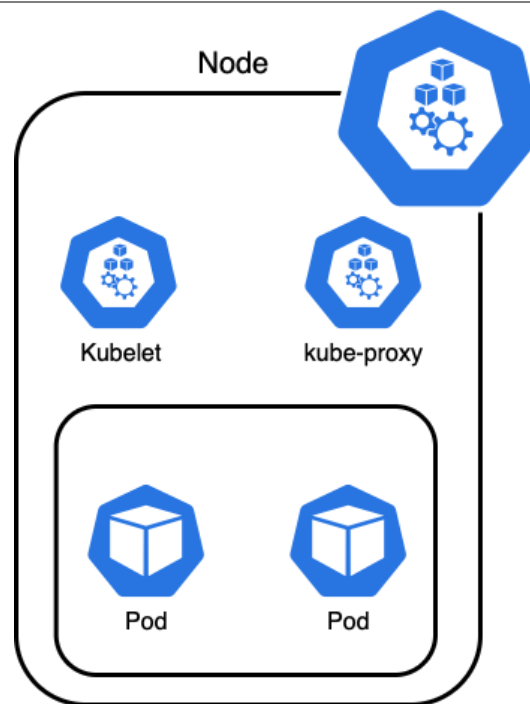
## I: Namespace

- Ein Cluster wird in sogenannte Namespaces unterteilt.
- Teilt Gruppen von Ressourcen in abgegrenzte Teilbereiche auf.
- Namen für Ressourcen müssen nur innerhalb des Namespaces einzigartig sein.
- Erlaubt es vielen verschiedenen Personen, Teams oder Projekten, einen einzelnen Cluster zu benutzen und zu verwalten.



# Kubernetes Komponenten

## II: Nodes:







# Kubernetes Komponenten

---

## II: Nodes:

- Auch Worker-Nodes genannt
- Enthalten sog. Pods
- Mehrere Nodes können auf einem physischen Server laufen, oder weit verteilt auf vielen einzelnen Servern.
- Nodes können virtuelle Maschinen sein, oder Container.
- Eine Node kann mehrere Pods gemeinsam verwalten.
- Jede Node wird mit einem einzigartigen Namen identifiziert.
- Jede Node enthält:
  1. Kubelet: Stellt sicher, dass Pods laufen und registriert die Node bei der Control-Plane
  2. Kube-Proxy: Verwaltet Netzwerk Regeln (optional)
  3. Pods: Enthalten die Container-Services

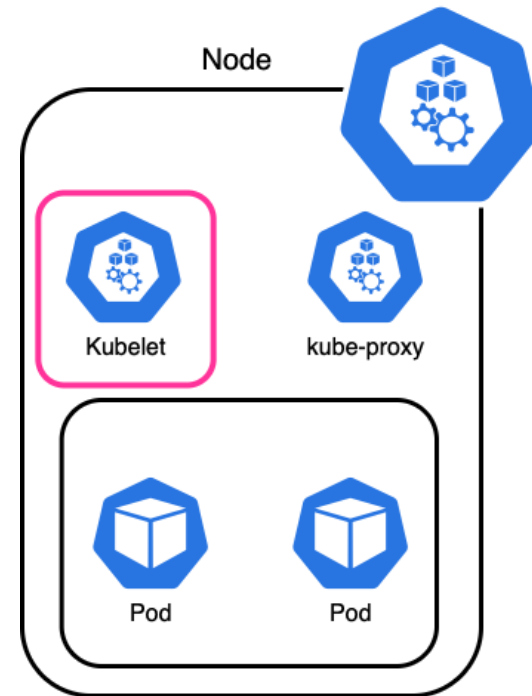


# Kubernetes Komponenten



## II: Nodes: kubelet

- Prozess, welcher innerhalb einer Node läuft.
- Stellt sicher, dass alle Container der Node innerhalb eines Pods laufen.
- Erhält die deklarativen Spezifikationen für einen Pod von der Control-Plane.
- Stellt sicher, dass Pods nach den erhaltenen Spezifikationen erzeugt werden und laufen.

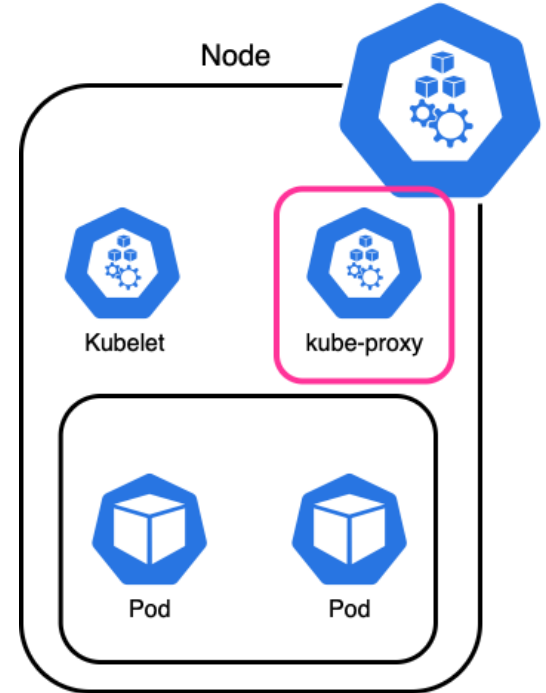




# Kubernetes Komponenten

## II: Nodes: kubeproxy

- Network-Proxy innerhalb einer Node.
- Optionale Komponente, welche Netzwerk-Regeln überwacht.
- Erlaubt es bestimmte Pods von Außen zugänglich zu machen.
- Erlaubt es auch, die Kommunikation eines Pods einzuschränken.

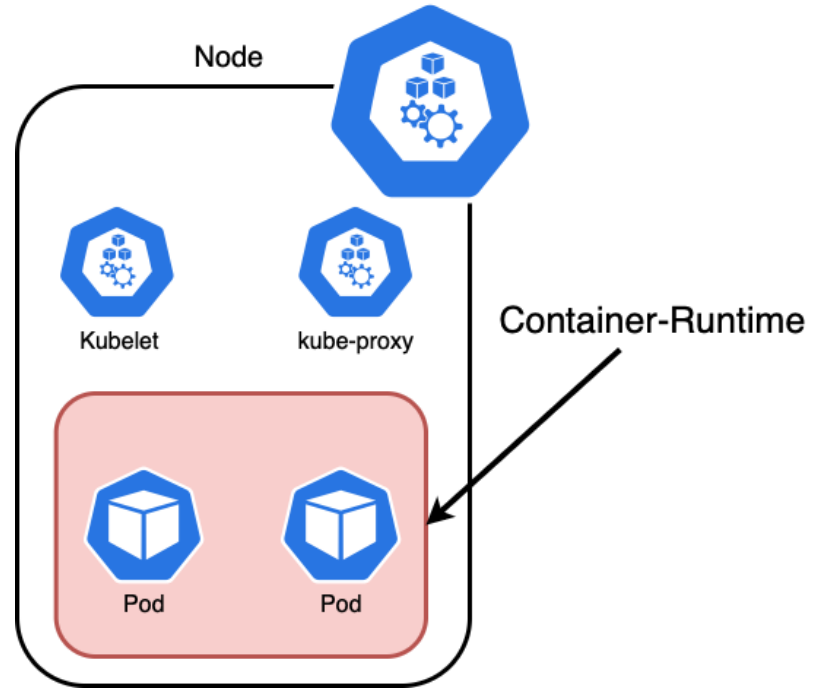




# Kubernetes Komponenten

## II: Nodes: Container-Runtime

- Zentrales Werkzeug einer Node.
- Zuständig für das Ausführen und Steuern der einzelnen Container-Images innerhalb der Pods einer Node.

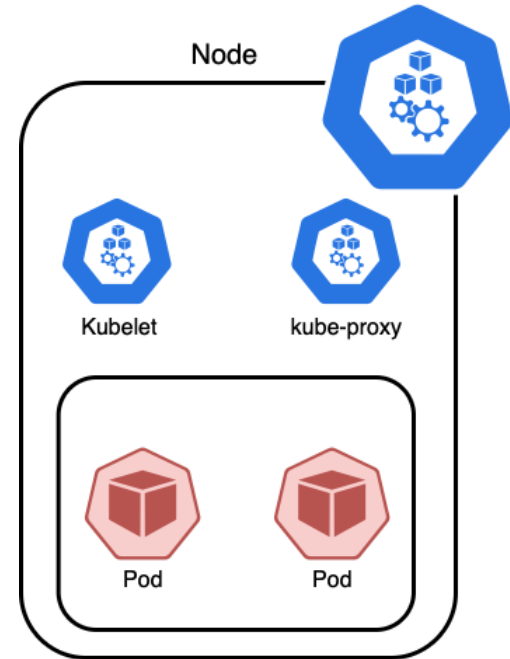




# Kubernetes Komponenten

## III: Pods:

- Jede Node verwaltet einen, oder mehrere, Pods.
- Enthält die eigentlichen Services als Container-Images.
- Ein Pod kann mehrere Container enthalten.
- Pods existieren rein als deklarativer Quellcode und werden verfügbaren Nodes von der Control-Plane zugewiesen.
- Best Practice: Ein Container pro Pod.





# Kubernetes Komponenten

---

## III: Pods:

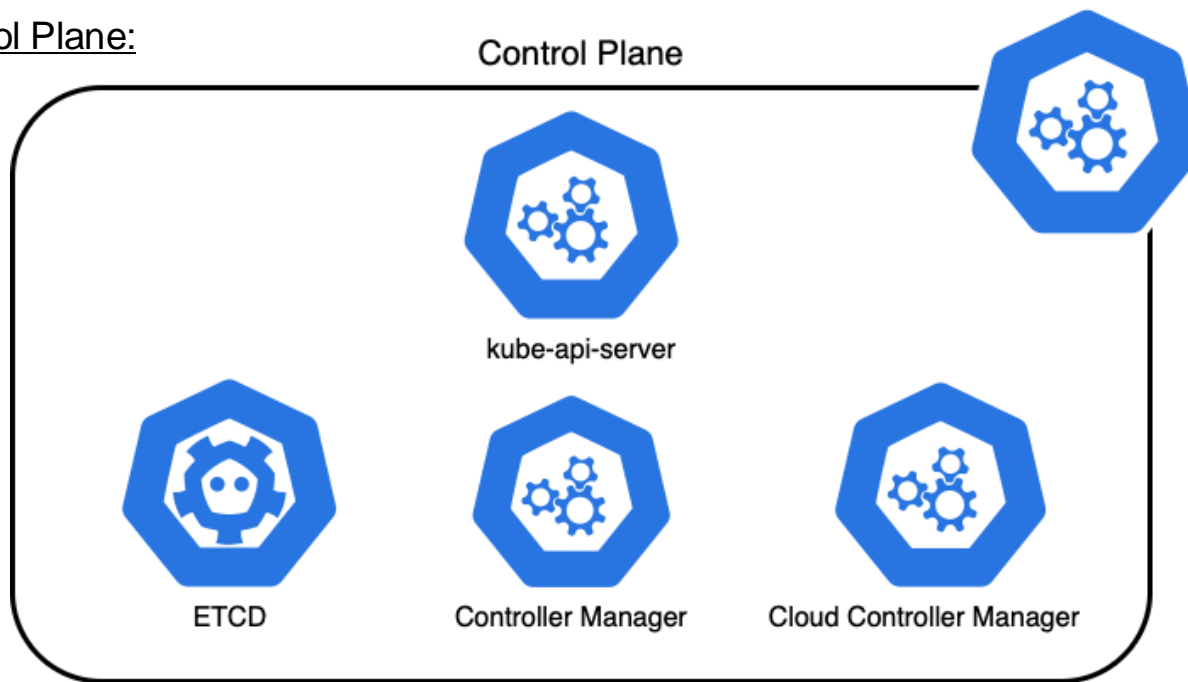
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Ein Pod, der nginx als Container enthält



# Kubernetes Komponenten

## IV: Control Plane:





# Kubernetes Komponenten

---

## IV: Control Plane:

- Verwaltet den Zustand des Clusters
- Enthält folgende Applikationen:
  1. Kube-Apiserver: Bietet HTTP-Interface zur Steuerung des Clusters
  2. Etcd: Service für Key-Value Store für Secrets und Umgebungsvariablen
  3. Kube-Scheduler: Verteilt einzelne Pods auf freie Nodes
  4. Kube-Controller-Manager: Verwaltet einzelne Controller, um API-Verhalten zu steuern
  5. Cloud-Controller-Manager: Bietet Integration mit Cloud-Providern, auf deren Infrastruktur der Cluster läuft (Optional)



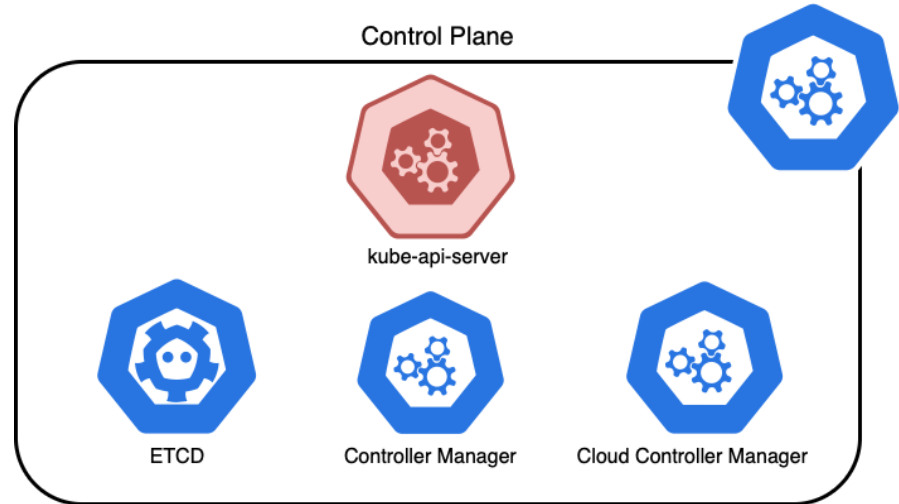


# Kubernetes Komponenten



## IV: Control Plane: Kube-APIServer

- Bietet REST-API zur Steuerung eines Clusters.
- Ist das Frontend des Clusters.
- Skaliert Horizontal bei vielen Anfragen an die API des Clusters.



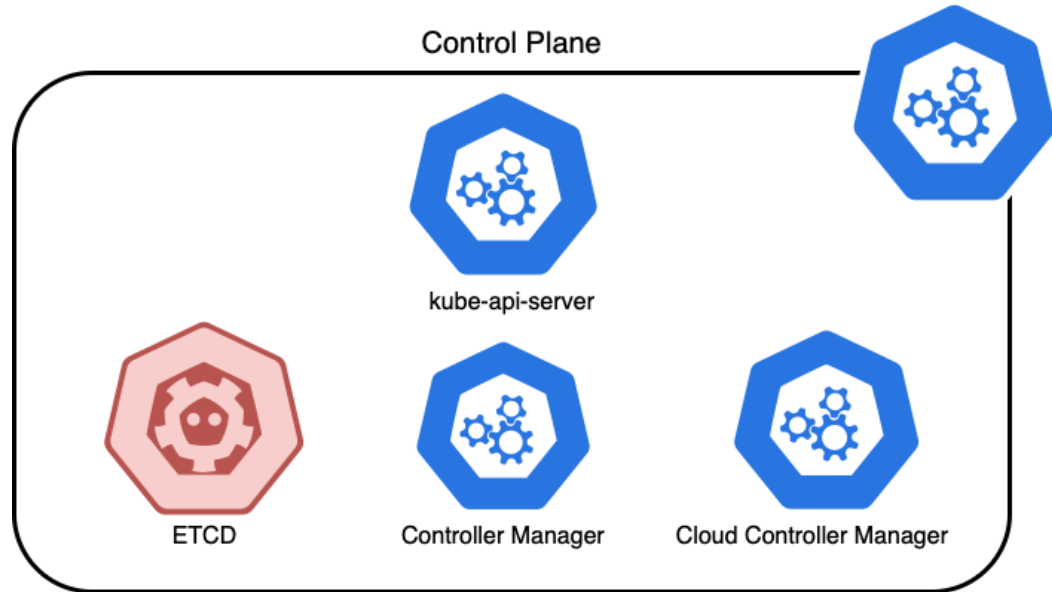


# Kubernetes Komponenten



## IV: Control Plane: Etcd

- Service, welcher Secrets und Umgebungsvariablen speichert.
- Globaler Key-Value Store.
- Gespeicherte Keys und Values können von einzelnen Nodes abgefragt werden, um deren Pods zu konfigurieren.



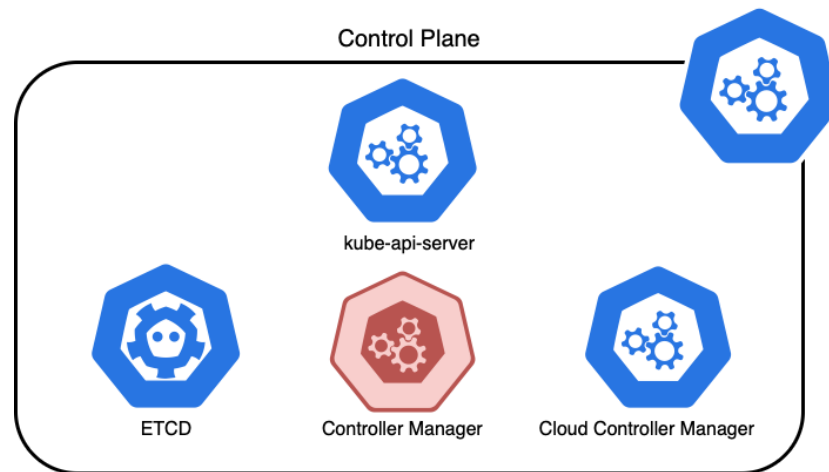


# Kubernetes Komponenten



## IV: Control Plane: Kube-Controller-Manager

- Verwaltet die Controller der Control-Plane.
- Controller laufen innerhalb eines einzelnen Prozesses auf der Control-Plane.
- Neben Default-Controllern lassen sich weitere Controller hinzufügen.
- Controller sind:
  - Node-Controller: Reagiert auf den Ausfall von einzelnen Nodes
  - Job-Controller: Reagiert auf einzelne Jobs, die einmalig ausgeführt werden müssen und kreiert Pods für diese Jobs
  - EndpointSlice-Controller: Stellt Links zwischen Services und Pods bereit
  - ServiceAccount-Controller: Erzeugt default Service Accounts für einen Cluster-Namespace
  - ...



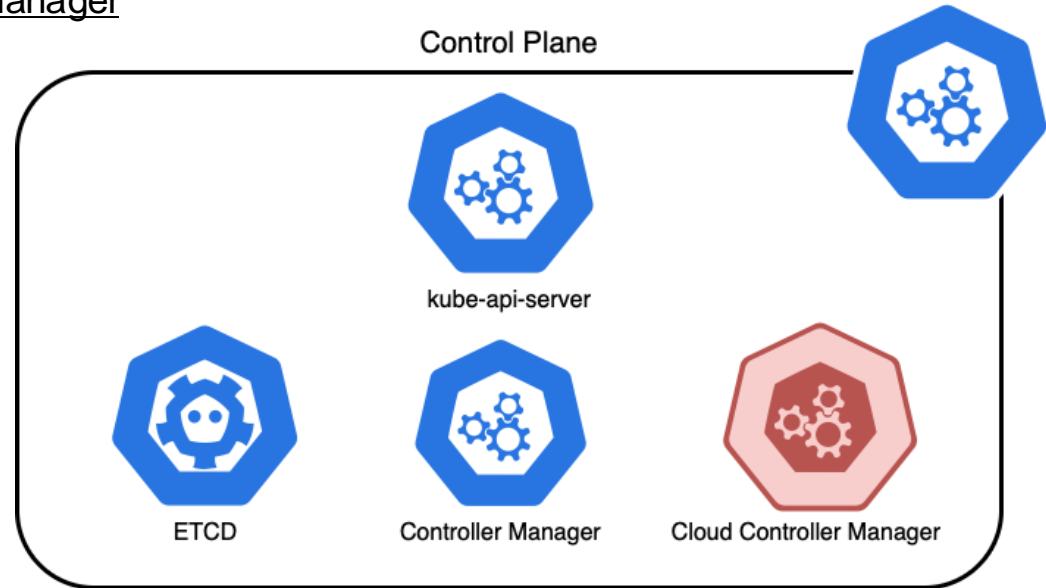


# Kubernetes Komponenten



## IV: Control Plane: Cloud-Controller-Manager

- Bietet spezifische Kontrolllogik für einzelne Cloud-Anbieter.
- Erlaubt es dem Cluster mit der API eines bestimmten Cloud-Anbieters zu interagieren
- Cloud-Controller-Manager laufen für Controller, welche spezifisch für einen bestimmten Cloud-Anbieter sind.
- Verhalten ist gleich zu Kube-Controller-Manager





# Kubernetes Komponenten

---

## V: Networking: Cluster-Netzwerk

- Kubernetes erzeugt ein Cluster-Internes Netzwerk.
- Jeder Pod innerhalb des Clusters erhält eine einzigartige IP, welche innerhalb des Clusters gilt.
- Das interne Netzwerk ermöglicht es Pods über Nodes hinweg miteinander zu kommunizieren.



# Kubernetes Komponenten

---

## V: Networking: Service-API

- Die Service-API ist Teil der Control-Plane.
- Erlaubt es, mehrere Pods zu einem sog. Service zusammenzufassen.
- Erlaubt es, einzelnen Services Hostnamen und IP-Adressen zuzuweisen.
- Ermöglicht Name-Resolution innerhalb des Cluster-Netzwerkes.



# Kubernetes Komponenten

---

## V: Networking: Services

- Mithilfe der Service-API können einzelne Pods zu einem Service zusammengefasst werden.
- Ein Service ist ein Objekt, welches einen Satz an Endpunkten (Pods) beschreibt.
- Innerhalb des Services werden Regeln (Policies) definiert, die den Zugang zu diesen Endpunkten regeln.



# Kubernetes Komponenten

## V: Networking: Services

- Jeder Pod, welcher mit demselben Label versehen ist, wie ein bestimmter Service, wird diesem zugeordnet.
- Der Service selbst erhält ebenfalls eine Cluster-IP.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app.kubernetes.io/name: proxy
spec:
  containers:
  - name: nginx
    image: nginx:stable
    ports:
      - containerPort: 80
        name: http-web-svc

---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app.kubernetes.io/name: proxy
  ports:
  - name: name-of-service-port
    protocol: TCP
    port: 80
    targetPort: http-web-svc
```





# Kubernetes Komponenten

---

V: Networking: Gateway-API



## gateway api

- Erlaubt es, einzelne Services von außen zugänglich zu machen.
- Erlaubt es, Load Balancing zu implementieren
- Besteht aus:
  - **Gateway-Class:** Definiert die Gateways und den Controller, welcher die benötigten APIs implementiert.
  - **Gateway:** Definiert die Infrastruktur, welche den Traffic verarbeitet
  - **HTTPRoute:** Beschreibt Routing-Regeln, um Traffic von einem Gateway zu einem Service zu leiten.



# Kubernetes Komponenten

## V: Networking: Gateway-Class

- Beschreibt einen Controller, welcher als Gateway agieren kann.
- Dies erlaubt es, Services innerhalb oder außerhalb des Clusters als Gateways zu markieren.



```
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: example-class
spec:
  controllerName: example.com/gateway-controller
```



# Kubernetes Komponenten

## V: Networking: Gateway-API

- Kann zur Filterung von Traffic verwendet werden.
- Erlaubt es, Traffic auf einzelne Services zu verteilen.
- Mithilfe von HTTPRoutes lässt sich das Verhalten des Gateways steuern.
- Repräsentiert einen Cloud-Load-Balancer oder einen Service innerhalb des Clusters

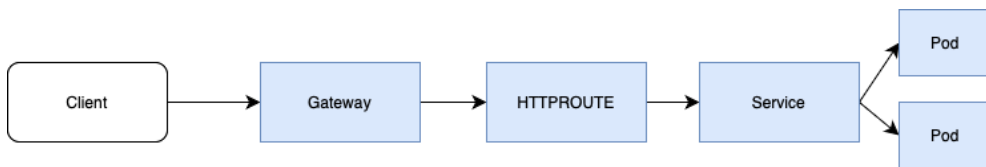
```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: example-gateway
spec:
  gatewayClassName: example-class
  listeners:
  - name: http
    protocol: HTTP
    port: 80
```



# Kubernetes Komponenten

## V: Networking: Gateway-API: HTTPRoute

- Spezifiziert das Routing-Verhalten von HTTP-Anfragen.
- Beschreibt, wie Anfragen an Services geleitet werden sollen.



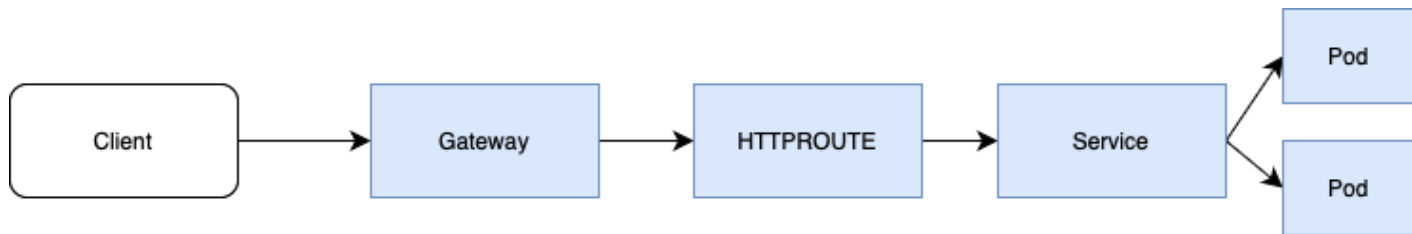
```
apiVersion:
gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-httproute
spec:
  parentRefs:
  - name: example-gateway
  hostnames:
  - "www.example.com"
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /login
    backendRefs:
    - name: example-svc
      port: 8080
```



# Kubernetes Komponenten

## V: Networking: Gateway-API: Request-Flow

1. Client schickt eine Request an den Cluster
2. Kubernetes wählt richtiges Gateway auf Basis von Gateway-Class aus
3. Gateway agiert als Reverse-Proxy und leitet Request basierend auf konfigurierter HTTPRoute weiter
4. Request erreicht Service

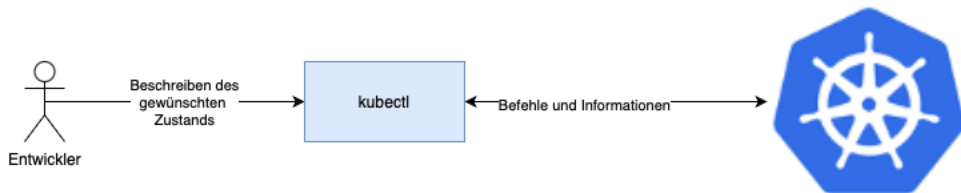




# Kubernetes Komponenten

## VII: Steuerung: kubectl

- ***kubectl*** -> CLI-Tool, um mit Kubernetes-Clustern zu interagieren.
- Steht auf allen Plattformen zur Verfügung (Windows, Linux, MacOS, OpenBSD).
- Interagiert mit der Control-Plane eines Clusters, um Steuerung zu ermöglichen.
- ***Kubectl steuert den Cluster dabei NICHT direkt, sondern schickt lediglich Anfragen an die Control-Plane, um diese zum Handeln zu bewegen.***





# Kubernetes Komponenten

## VII: Steuerung: kubectl

- Erlaubt es, Informationen über den Cluster oder seine Bestandteile zu erhalten.

```
# List all pods in plain-text output format.
kubectl get pods

# List all pods in plain-text output format and include additional information (such as node name).
kubectl get pods -o wide

# List the replication controller with the specified name in plain-text output format. Tip: You can shorten
and replace the 'replicationcontroller' resource type with the alias 'rc'.
kubectl get replicationcontroller <rc-name>

# List all replication controllers and services together in plain-text output format.
kubectl get rc,services

# List all daemon sets in plain-text output format.
kubectl get ds

# List all pods running on node server01
kubectl get pods --field-selector=spec.nodeName=server01
```



# Kubernetes Komponenten

## VII: Steuerung: kubectl

- Nimmt deklarative Dateien mit Quellcode entgegen.
- Führt die Notwendigen Operationen aus, um den gewünschten Zustand herzustellen.



```
kubectl apply -f example-service.yaml
```







```
kubectl apply -f example-controller.yaml
```





# Kubernetes Komponenten

## Pop-Quiz:

1. Das CLI-Tool **kubect!** kann folgendes **NICHT...**
  - a. Den Status von einzelnen Komponenten abfragen. 
  - b. Das Erstellen von Services in Auftrag geben. 
  - c. Direkt eingreifen, sollte ein Pod nicht mehr reagieren. 
2. Das CLI-Tool **kubect!** interagiert mit...
  - a. Der Control-Plane von Kubernetes 
  - b. Den Kubernetes Controllern 
  - c. Dem gesamten Kubernetes-Cluster 



# Kubernetes

---

- ❖ Herausforderungen
- ❖ Kubernetes to the rescue!
- ❖ Was ist Kubernetes?
- ❖ Was bietet Kubernetes?
- ❖ Grundlagen Kubernetes
- ❖ Kubernetes Komponenten
- ❖ Kubernetes in der Praxis
- ❖ Zusammenfassung



# Kubernetes in der Praxis

---

## Praktische Anwendung

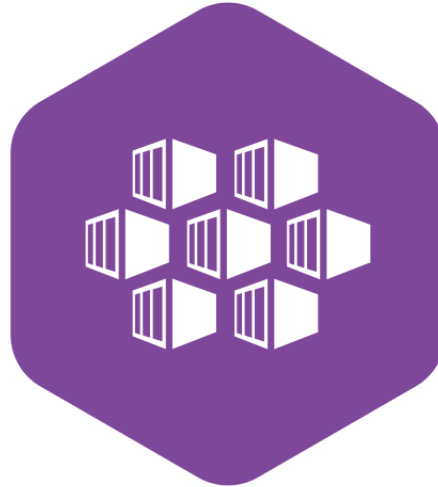
- Die Bereitstellung von Infrastruktur, auf welcher Kubernetes Nodes laufen können, ist aufwändig und komplex.
- Verschiedene Open-Source Projekte und Anbieter bieten jedoch vorgefertigte Lösungen.
- **Local / Self-Hosted:**
  - MicroK8s
  - MiniKube
  - Kubeadm
  - **Docker-Engine** bietet ebenfalls einen Kubernetes-Modus <- Übung
- **Cloud-Hosted:**
  - **AKS** – Azure Kubernetes Service
  - **Amazon EC2** – AWS selbst-verwaltete Kubernetes Cluster
  - **Amazon EKS** – Durch Amazon verwaltete Cluster
  - **GKE** – Google Kubernetes Engine
  - ...



# Kubernetes in der Praxis

---

Azure Kubernetes Service (AKS):





# Kubernetes in der Praxis

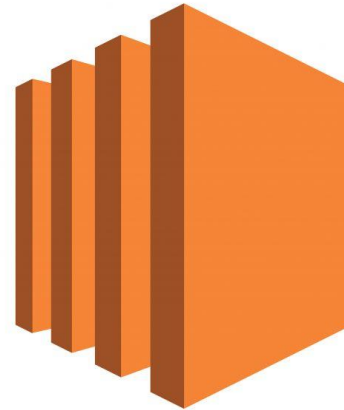
---

Amazon Elastic Kubernetes Service (EKS):

Amazon Elastic Compute Cloud (EC2):



AmazonEKS





# Kubernetes in der Praxis

---

Google Kubernetes Engine





# Kubernetes

---

- ❖ Herausforderungen
- ❖ Kubernetes to the rescue!
- ❖ Was ist Kubernetes?
- ❖ Was bietet Kubernetes?
- ❖ Grundlagen Kubernetes
- ❖ Kubernetes Komponenten
- ❖ Kubernetes in der Praxis
- ❖ Zusammenfassung



# Zusammenfassung

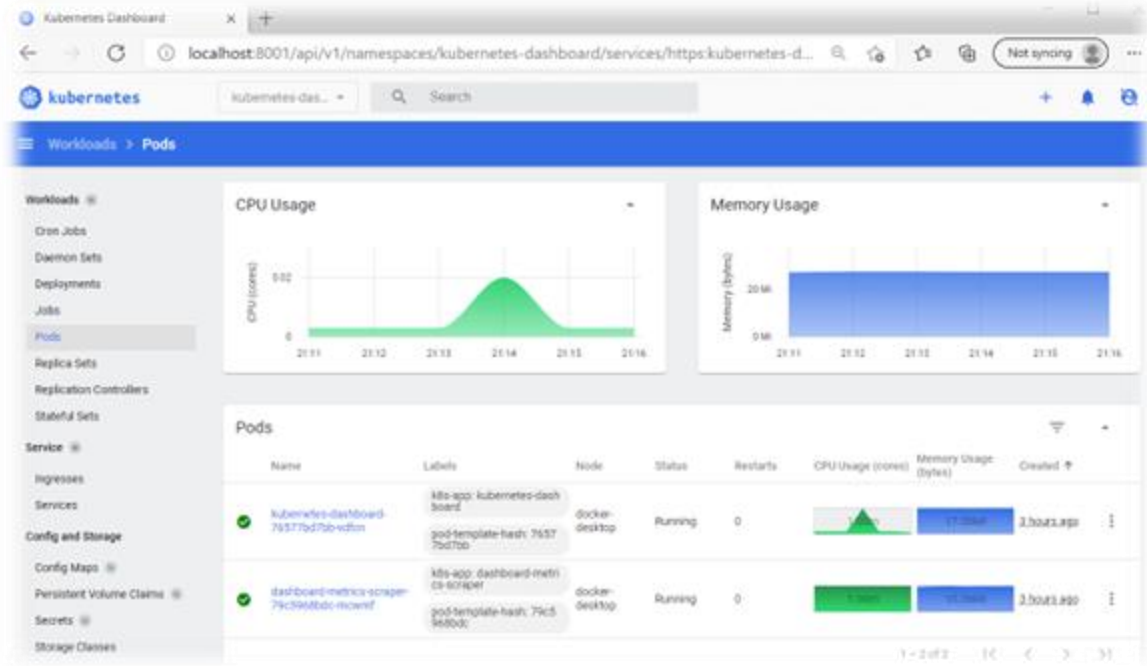
---

- Mächtige Plattform, um große Anzahl an Containern zu Provisionieren.
- Erlaubt Aufbau von komplexer Infrastruktur.
- Vereinfacht Aufbau und Wartung von komplexer Infrastruktur.
- Security, Routing, Monitoring, ... -> Alles in einem Paket.
- Grundtechnologie sind Container.
- Sehr erweiterbar und modifizierbar.
- Deklarativer Ansatz minimiert Fehlerquellen und erleichtert Reproduzierbarkeit.
- In verschiedensten Umgebungen einsetzbar. (Einzelner Rechner bis hin zu globaler Verteilung auf Rechenzentren)



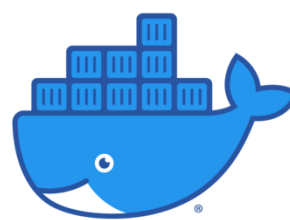


# Übungen zu Kubernetes

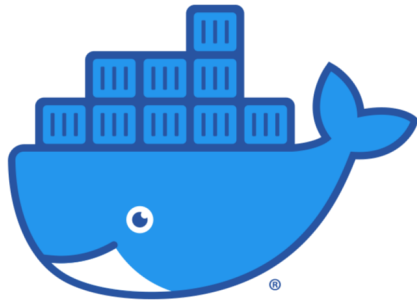




# Key Takeaways



Aspekt	Docker	Kubernetes
<b>Aufgabenbereich</b>	Fokus auf Container-Erstellung und -Ausführung	Orchestriert und verwaltet Container in Clustern
<b>Abstraktionsebene</b>	Arbeitet mit einzelnen Containern	Arbeitet mit Pods (Gruppen von Containern)
<b>Skalierung</b>	Manuelle Skalierung; eingeschränkte Funktionen ohne zusätzliche Software	Automatische Skalierung basierend auf Workload und Ressourcen
<b>Netzwerk</b>	Flaches Netzwerk für Container-Kommunikation	Komplexere Vernetzung mit Services
<b>Selbstheilung</b>	Container müssen manuell neu gestartet werden	Erkennt und startet fehlerhafte Pods automatisch neu
<b>Komplexität</b>	Einfach und schnell für kleinere Projekte	Komplexer, teurer, ideal für großskalige Anwendungen



Vielen Dank!



Icon made by [Freepik](https://www.flaticon.com) from [www.flaticon.com](https://www.flaticon.com)