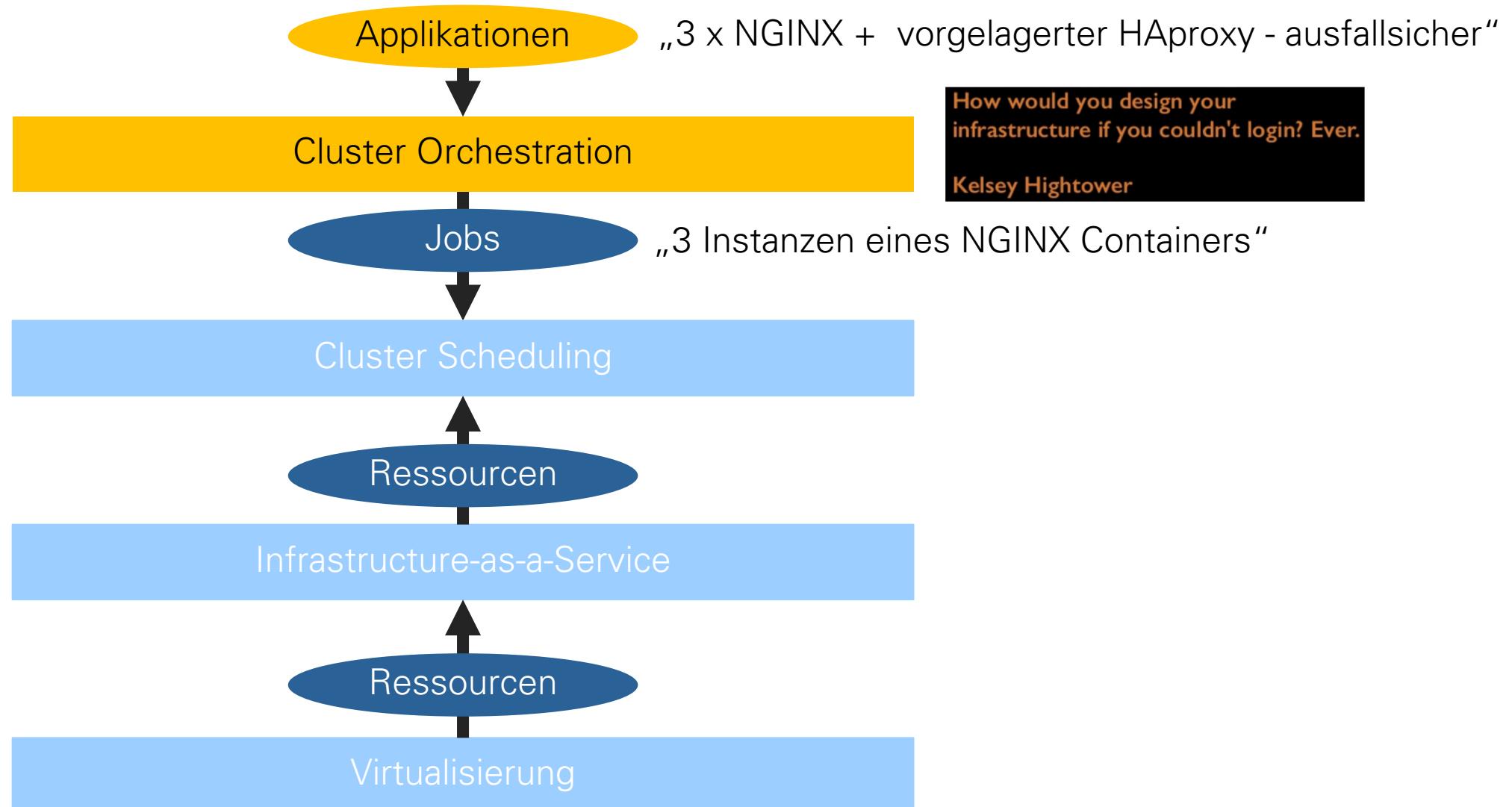


Kapitel 7: Cluster Orchestrierung

Vorlesung

CLOUD
COMPUTING

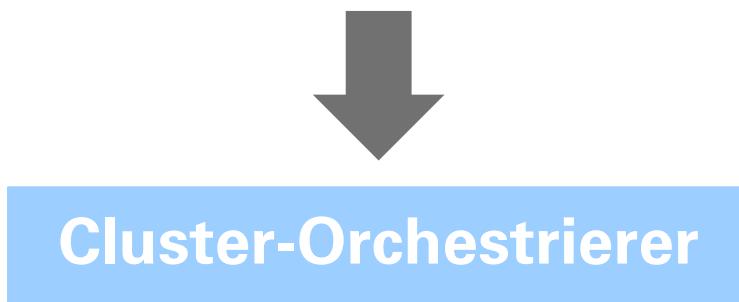
Das Big Picture: Wir sind nun auf Applikationsebene.



Cluster-Orchestrierung

- Eine Anwendung, die in mehrere Betriebskomponenten (Container) aufgeteilt ist, auf mehreren Knoten laufen lassen.
„Running Containers on Multiple Hosts“.
DockerCon SF 2015: Orchestration for Sysadmins
- Führt Abstraktionen zur Ausführung von Anwendungen mit ihren Services in einem großen Cluster ein.
- Orchestrierung ist keine statische, einmalige Aktivität wie die Provisionierung sondern eine dynamische, kontinuierliche Aktivität.
- Orchestrierung hat den Anspruch, alle Standard-Betriebsprozeduren einer Anwendung zu automatisieren.

Blaupause der Anwendung, die den gewünschten Betriebszustand der Anwendung beschreibt: Betriebskomponenten (Container), deren Betriebsanforderungen sowie die angebotenen und benötigten Schnittstellen.

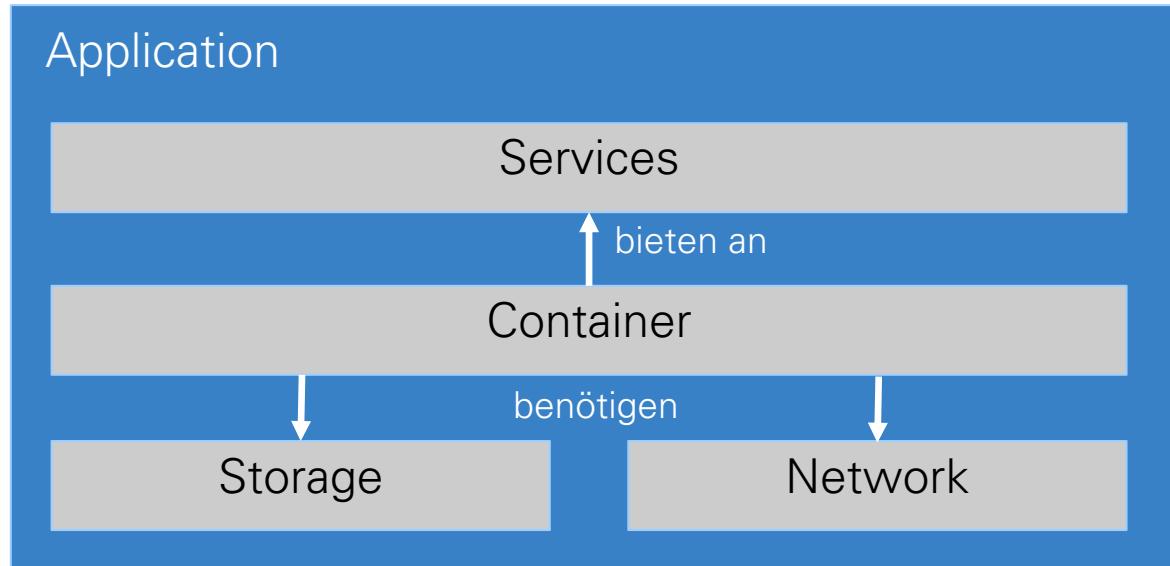


Cluster-Orchestrator

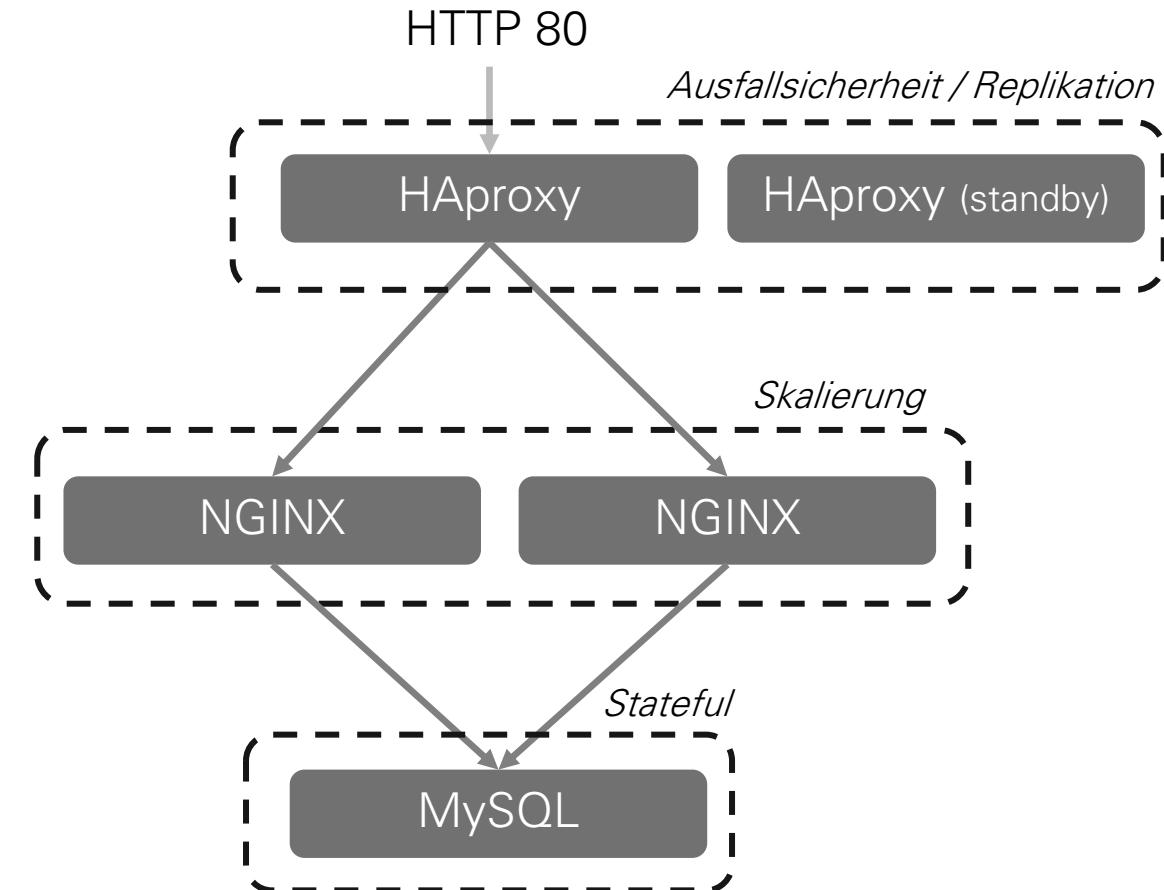
Steuerungsaktivitäten im Cluster:

- Start von Containern auf Knoten (→ Scheduler)
- Verknüpfung von Containern
- ...

Blaupause einer Anwendung (vereinfacht)



Metamodell



Modell

Wir kennen bereits einen Cluster-Orchestrator

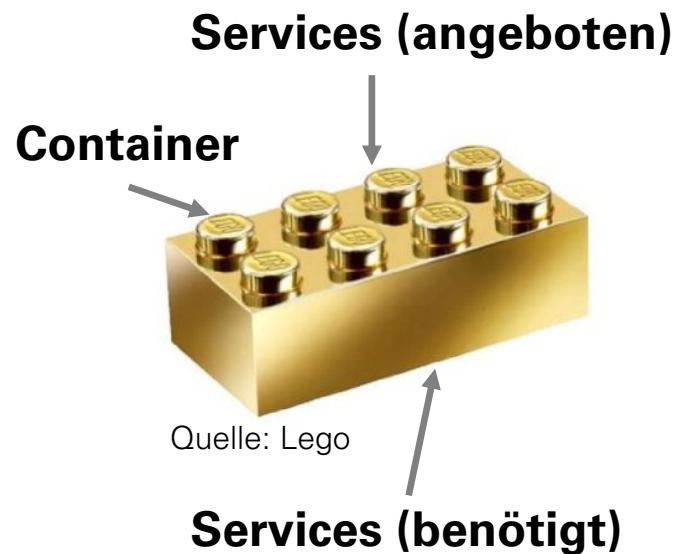


MARATHON

ID	Command	Memory (MB)	CPUs	Instances
eat-22-pancakes	echo "pancakes for every meal" && sleep 500	16	0.1	1 / 1
healthy-app	python -m SimpleHTTPServer \$PORT0	10	0.1	1 / 3
meaning	sleep 42	16	0.1	1 / 1
sleepy-time.4321	sleep 4321	16	0.3	6 / 10

Quelle: <https://mesosphere.com>

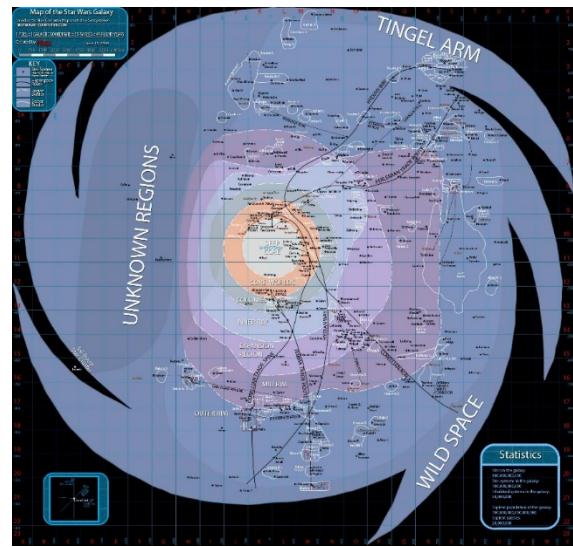
Analogie 1: Lego Star Wars



Cluster-Orchestrizer



Cluster-Scheduler

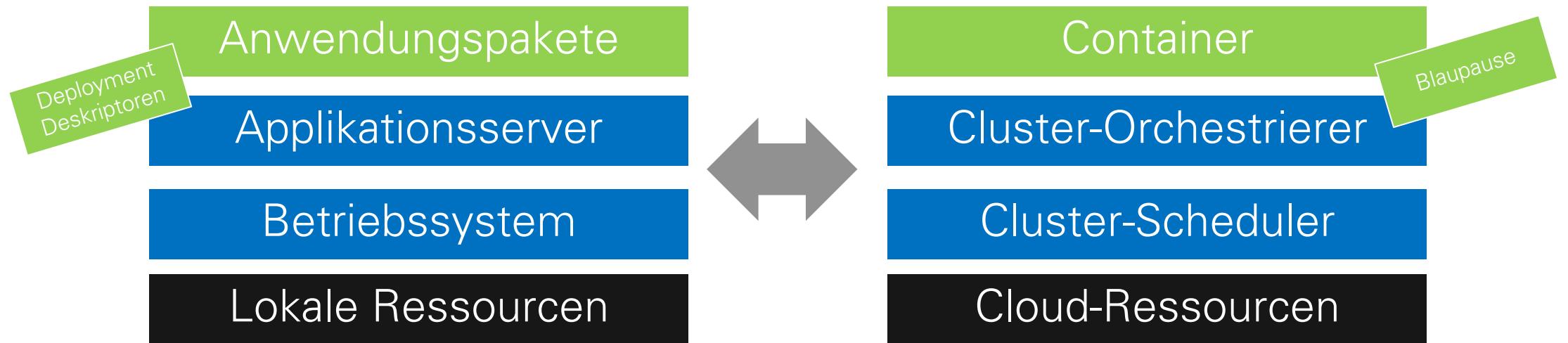


Quelle: wikipedia.de

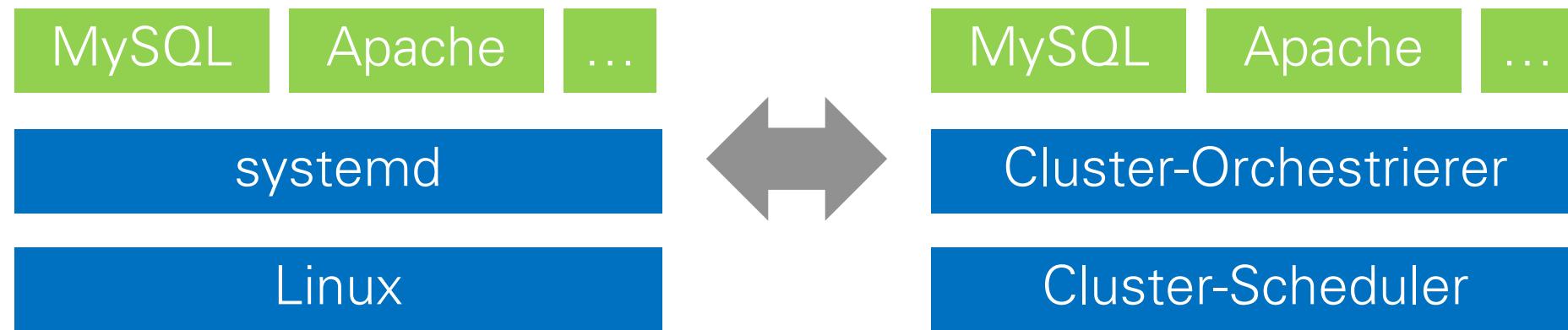


Blaupause

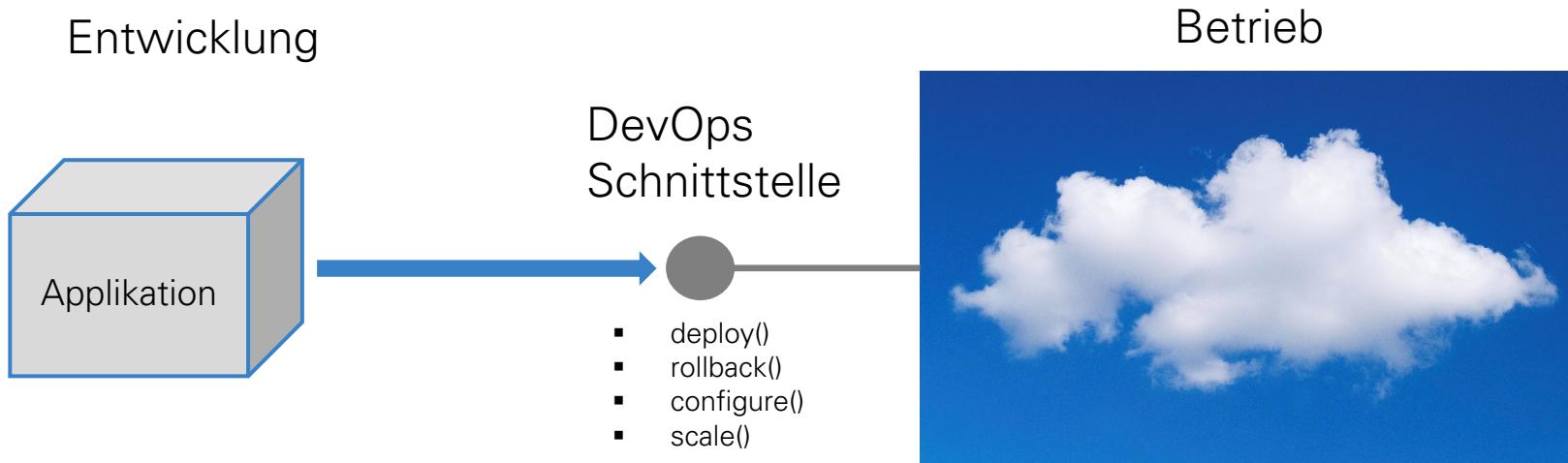
Analogie 2: Applikationsserver



Analogie 3: Betriebssystem



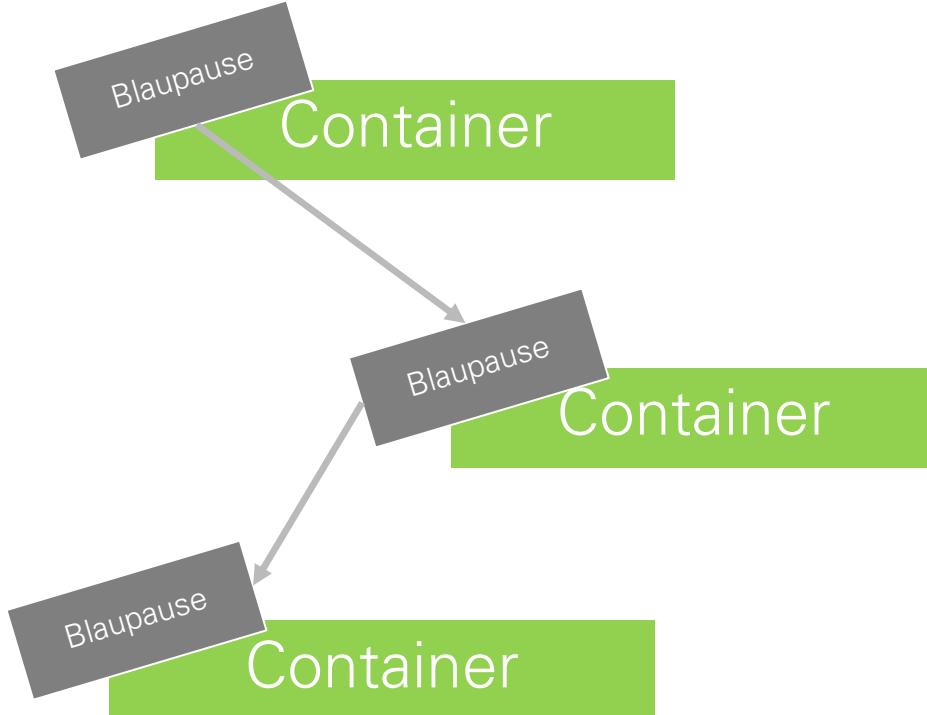
Ein Cluster-Orchestrator bietet eine Schnittstelle zwischen Betrieb und Entwicklung für ein Cluster an.



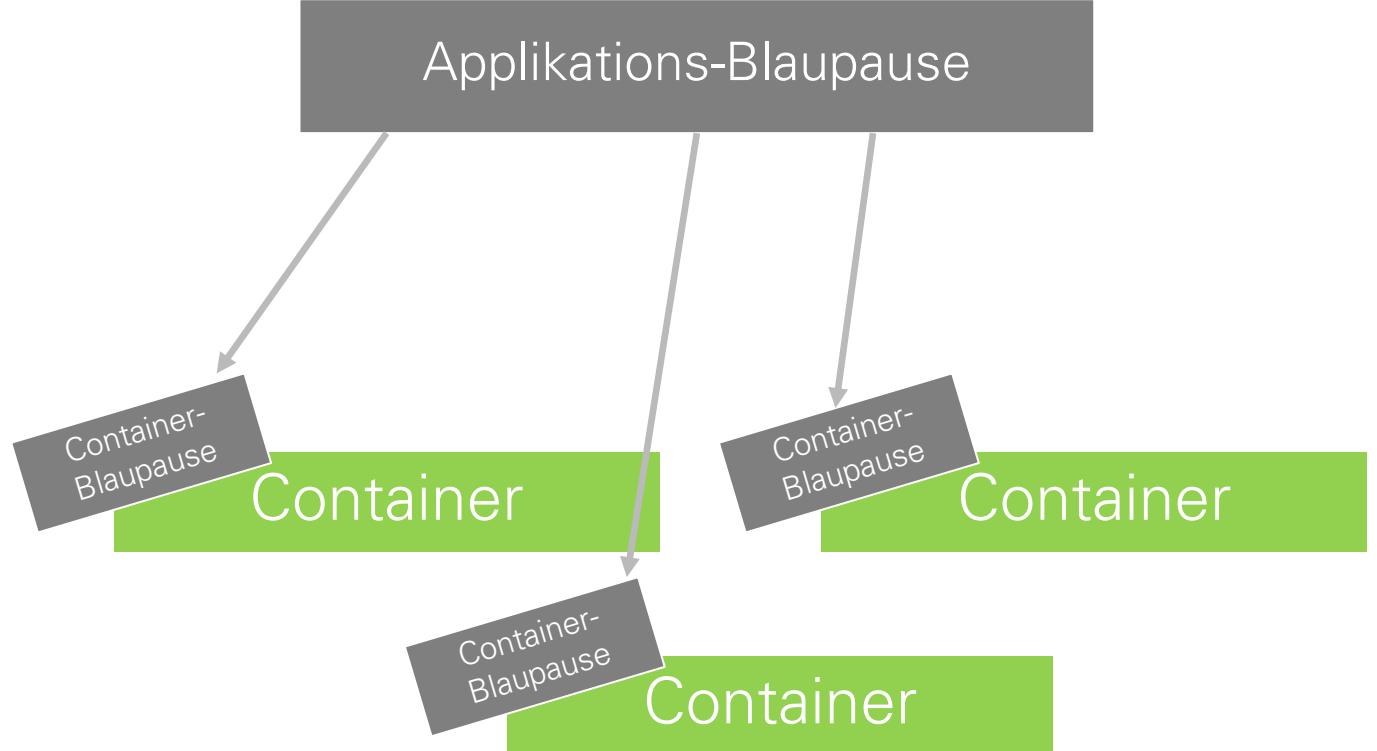
Ein Cluster-Orchestrator automatisiert vielerlei Betriebsaufgaben für Anwendung auf einem Cluster.

- Scheduling von Containern mit applikationsspezifischen Constraints (z.B. Deployment- und Start-Reihenfolgen, Gruppierung, ...)
- Aufbau von notwendigen Netzwerk-Verbindungen zwischen Containern.
- Bereitstellung von persistenten Speichern für zustandsbehaftete Container.
- (Auto-) Skalierung von Containern.
- Re-Scheduling von Containern im Fehlerfall (Auto-Healing) oder zur Performance-Optimierung.
- Container-Logistik: Verwaltung und Bereitstellung von Containern. Package-Management: Verwaltung und Bereitstellung von Applikationen.
- Bereitstellung von Administrationsschnittstellen (Remote-API, Kommandozeile).
- Management von Services: Service Discovery, Naming, Load Balancing.
- Automatismen für Rollout-Workflows wie z.B. Canary Rollout.
- Monitoring und Diagnose von Containern und Services.

1-Level- vs. 2-Level-Orchestrierung



1-Level-Orchestrierung
(Container-Graph)



2-Level-Orchestrierung
(Container-Repository mit zentraler Bauanleitung)

1-Level- vs. 2-Level-Orchestrierung

Plain Docker

```
FROM ubuntu  
ENTRYPOINT nginx  
EXPOSE 80
```

```
docker run -d --link  
nginx:nginx
```

1-Level-Orchestrierung
(Container-Graph)

weba:

```
image: qaware/nginx  
expose:  
- 80
```

webb:

```
image: qaware/nginx  
expose:  
- 80
```

haproxy:

```
image: qaware/haproxy  
links:  
- weba  
- webb  
ports:  
- „80:80“  
expose:  
- 80
```

2-Level-Orchestrierung
(Container-Repository mit zentraler Bauanleitung)

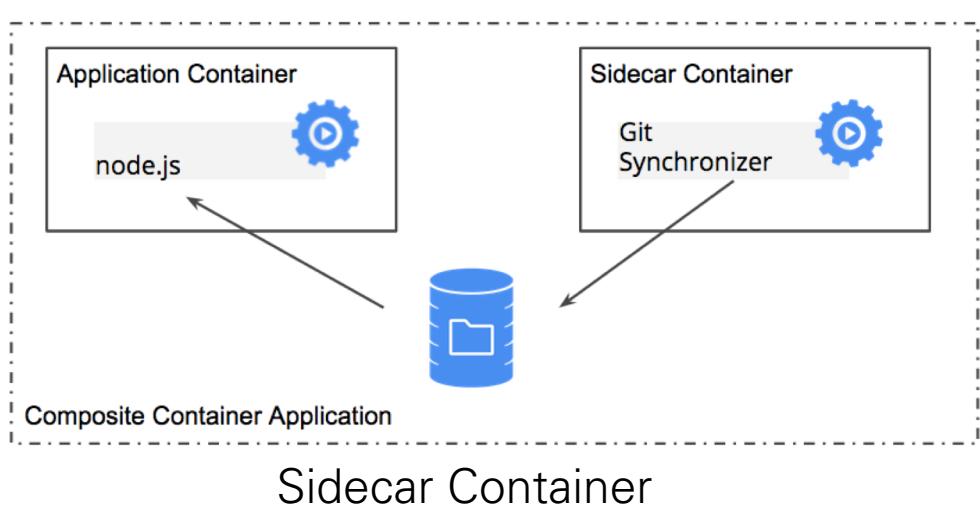
Docker Compose

<https://docs.docker.com/compose/compose-file>

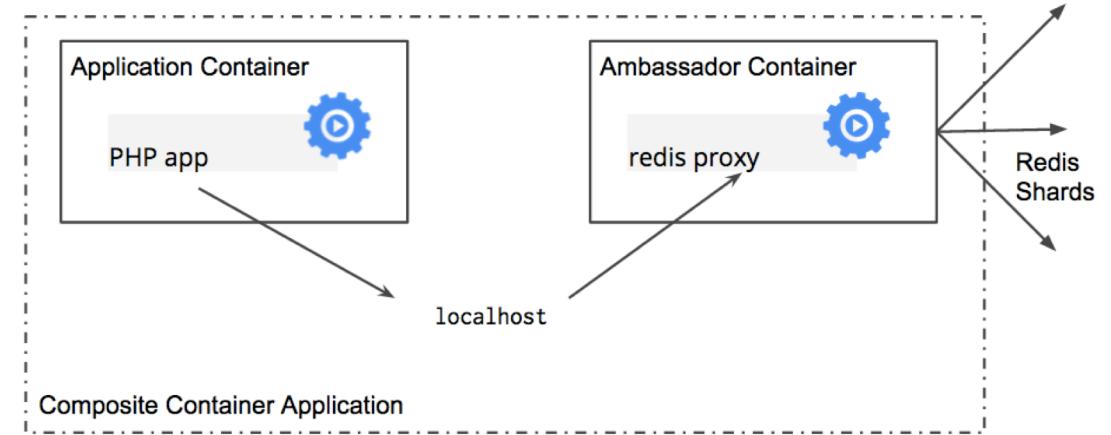
```
FROM ubuntu  
ENTRYPOINT nginx  
EXPOSE 80
```

```
FROM ubuntu  
ENTRYPOINT haproxy  
EXPOSE 80
```

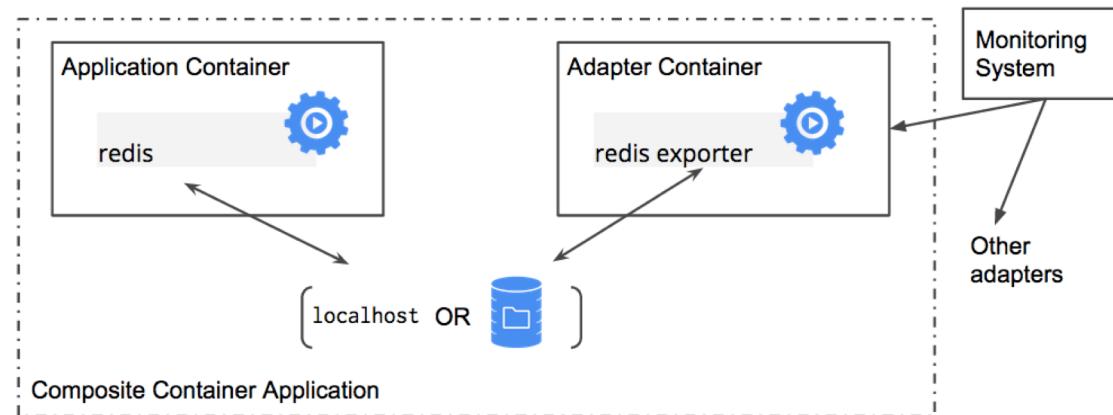
Orchestrierungsmuster



Sidecar Container



Ambassador Container



Adapter Container

Cluster Orchestrierer

- Kubernetes
- Apache Marathon & Chronos
- Docker Compose & Swarm

Kubernetes

Josef Adersberger @adersberger · Jul 21

Google spares no effort to lauch
#kubernetes @ #OSCON



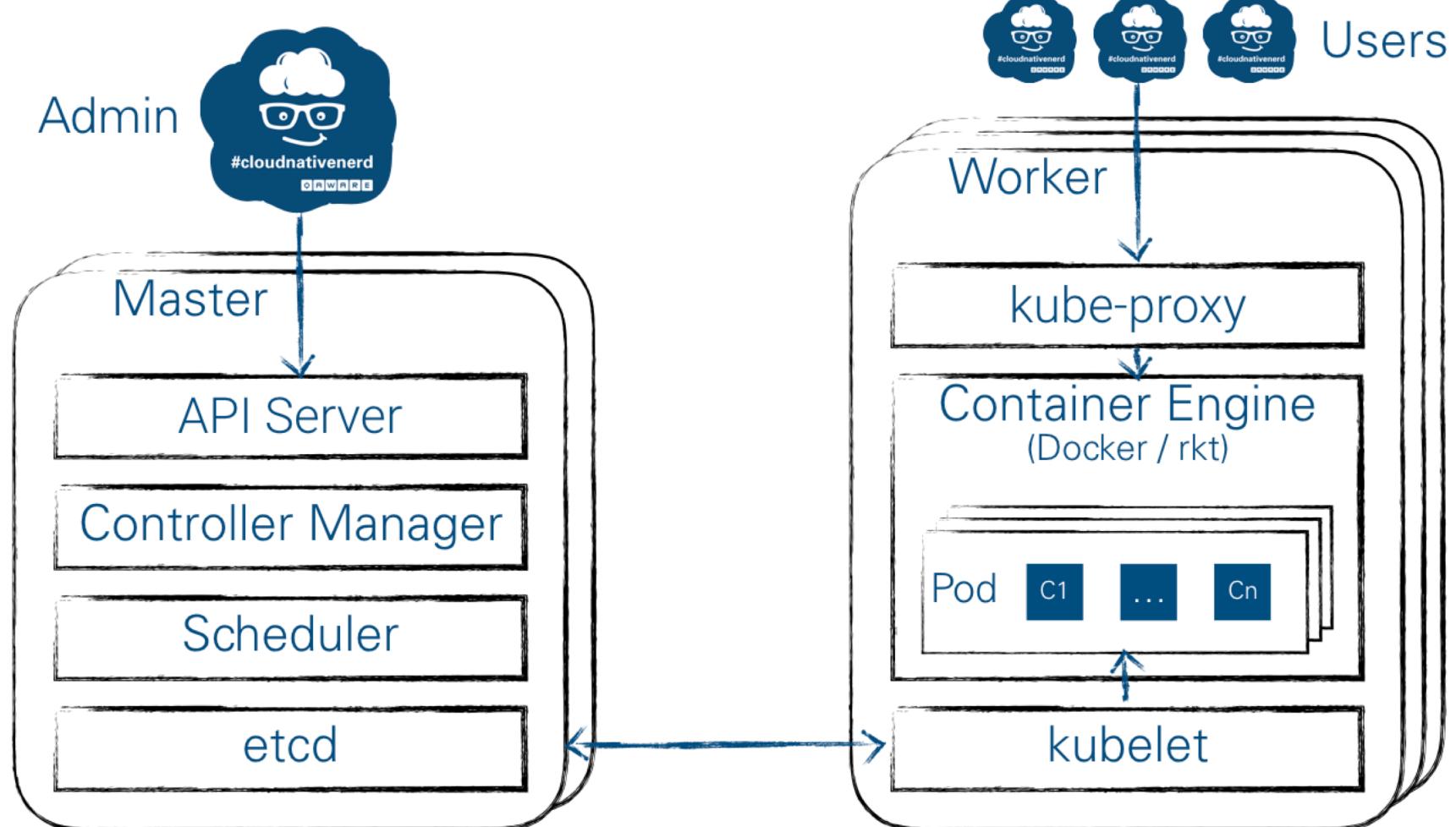
kubernetes by Google

Manage a cluster of Linux containers as a single system to accelerate Dev and simplify Ops.

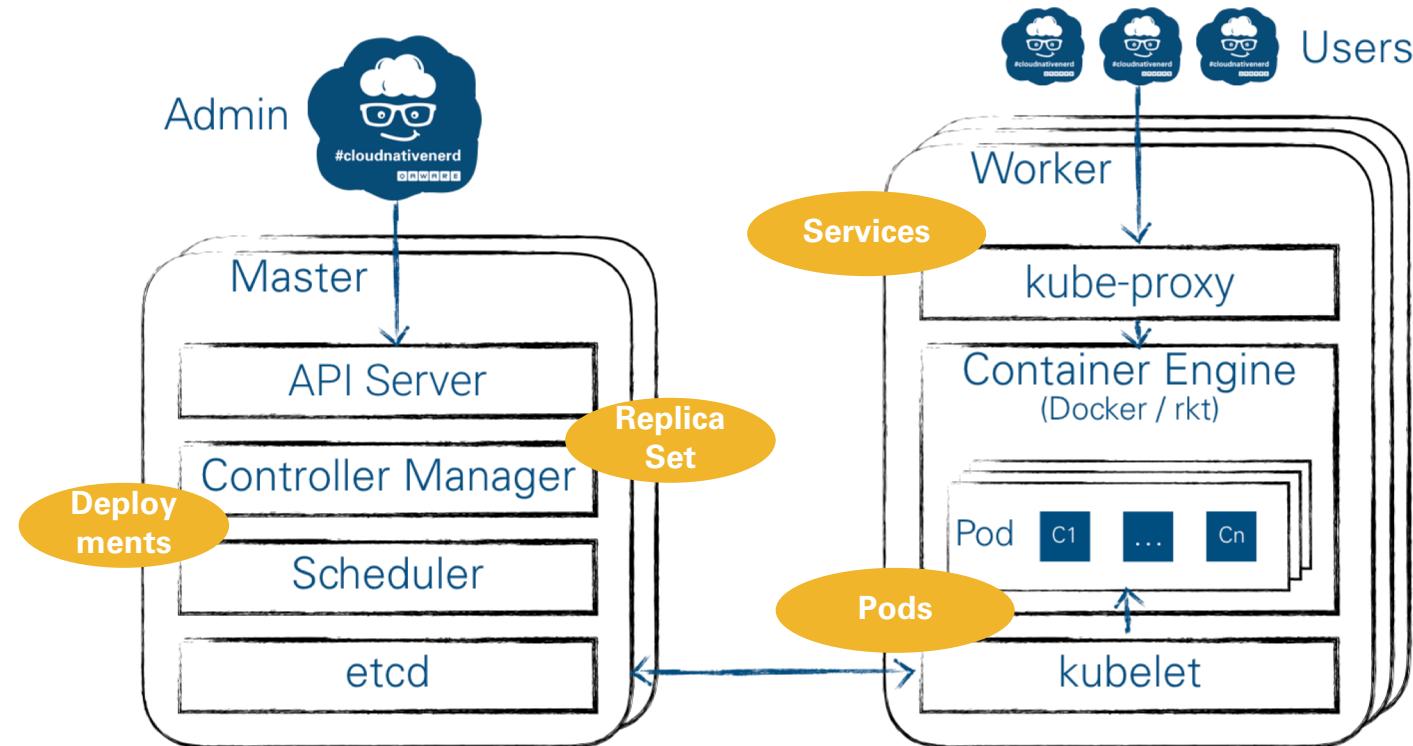
Kubernetes

- Cluster-Orchestrator auf Basis von Docker-Containern, der eine Reihe an Kern-Abstraktionen für den Betrieb von Anwendungen in einem großen Cluster einführt. Die Blaupause wird über YAML-Dateien definiert.
- Open-Source-Projekt, das von Google initiiert wurde. Google will damit die jahrelange Erfahrung im Betrieb großer Cluster der Öffentlichkeit zugänglich machen und damit auch Synergien mit dem eigenen Cloud-Geschäft heben.
- Seit Juli 2015 in der Version 1.0 verfügbar und damit produktionsreif. Skaliert aktuell nachweislich auf 10^2 großen Clustern.
- Aktuell bereits bei einigen Firmen im Einsatz wie z.B. Google im Rahmen der Google Container Engine, Wikipedia, ebay. Beiträge an der Codebasis aus vielen Firmen neben Google – u.A. Mesosphere, Microsoft, Pivotal, RedHat.
- Soll den Standard im Bereich Cluster-Orchestration setzen. Dafür wurde auch eigens die Cloud Native Computing Foundation gegründet (<https://cncf.io>).

Architektur von Kubernetes.



Aufgaben der Kubernetes Bausteine.



- **API Server:** Stellt die REST API von Kubernetes zur Verfügung (Admin-Schnittstelle)
- **Controller Manager:** Verwaltet die Repilca Sets / Replication Controller (stellt Anzahl Instanzen sicher) und Node Controller (prüfen Maschine & Pods)
- **Scheduler:** Cluster-Scheduler.
- **etcd:** Stellt einen zentralen Konfigurationsspeicher zur Verfügung.
- **Kubelet:** Führt Pods aus.
- **Container Engine:** Betriebssystem-Virtualisierung.
- **kube-proxy:** Stellt einen Service nach Außen zur Verfügung.

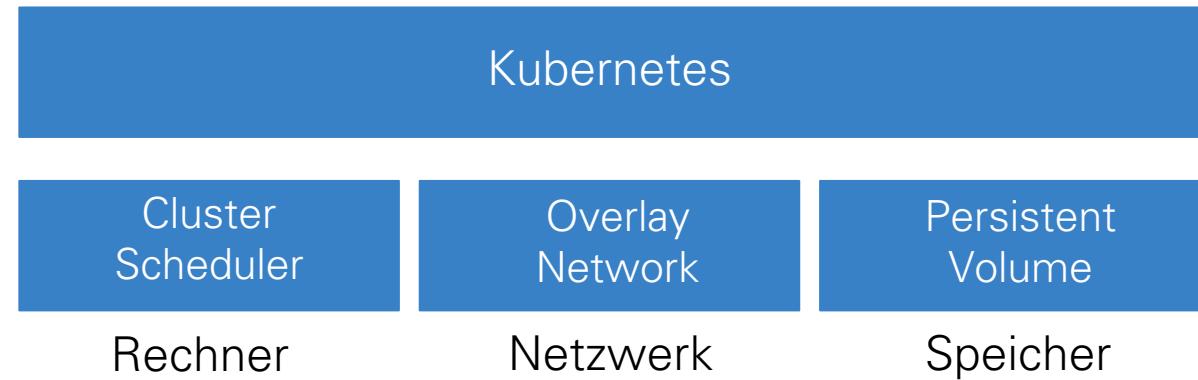
Neben einem Cluster-Scheduler setzt Kubernetes auch noch auf Netzwerk- und Storage-Virtualisierungen auf.

Netzwerk-Virtualisierung (Overlay Network)

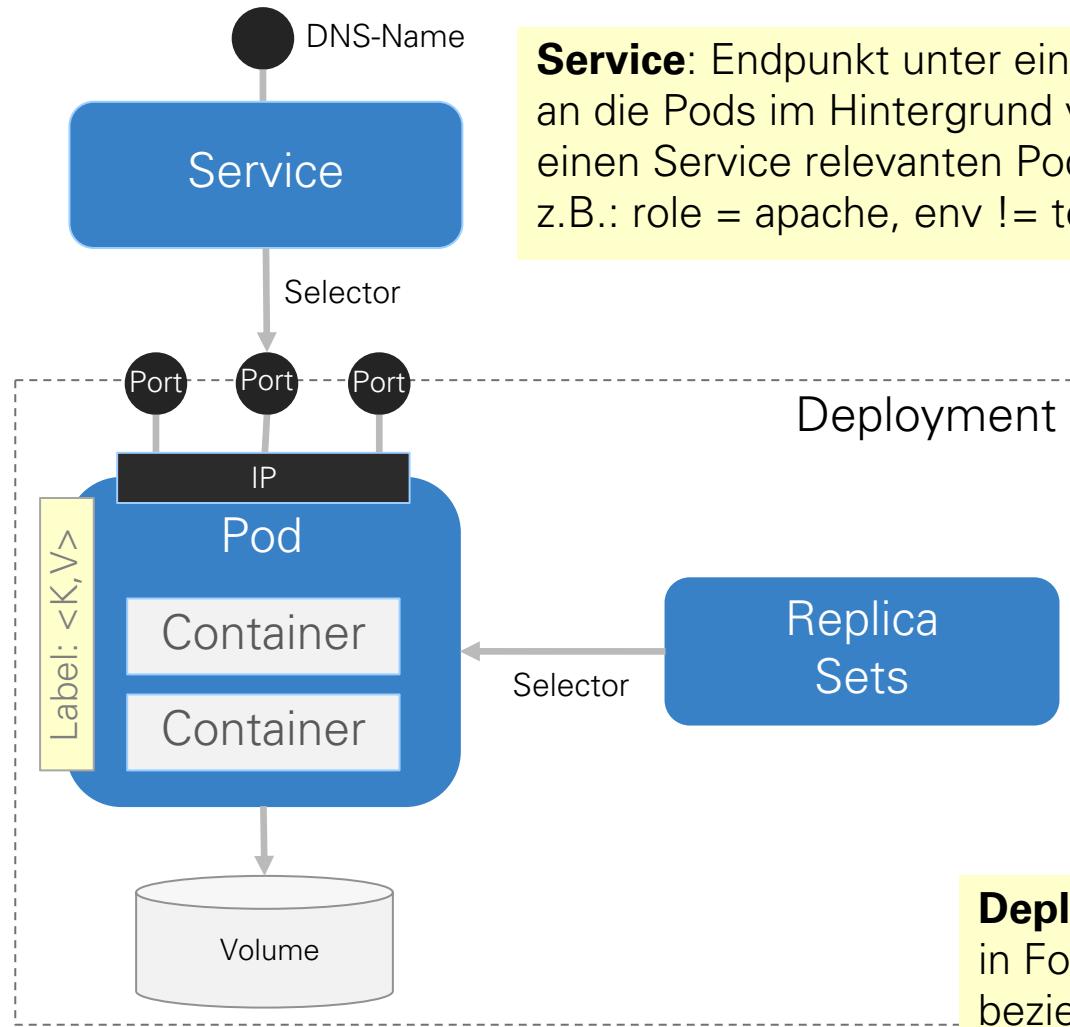
1. OpenVSwitch
2. Flannel
3. Weave
4. Calico

Storage-Virtualisierung (Persistent Volume), insbesondere zur Behandlung von zustandsbehafteten Containern.

1. GCE / AWS Block Store
2. NFS
3. iSCSI
4. Ceph
5. GlusterFS



Der Kern-Abstraktionen von Kubernetes.



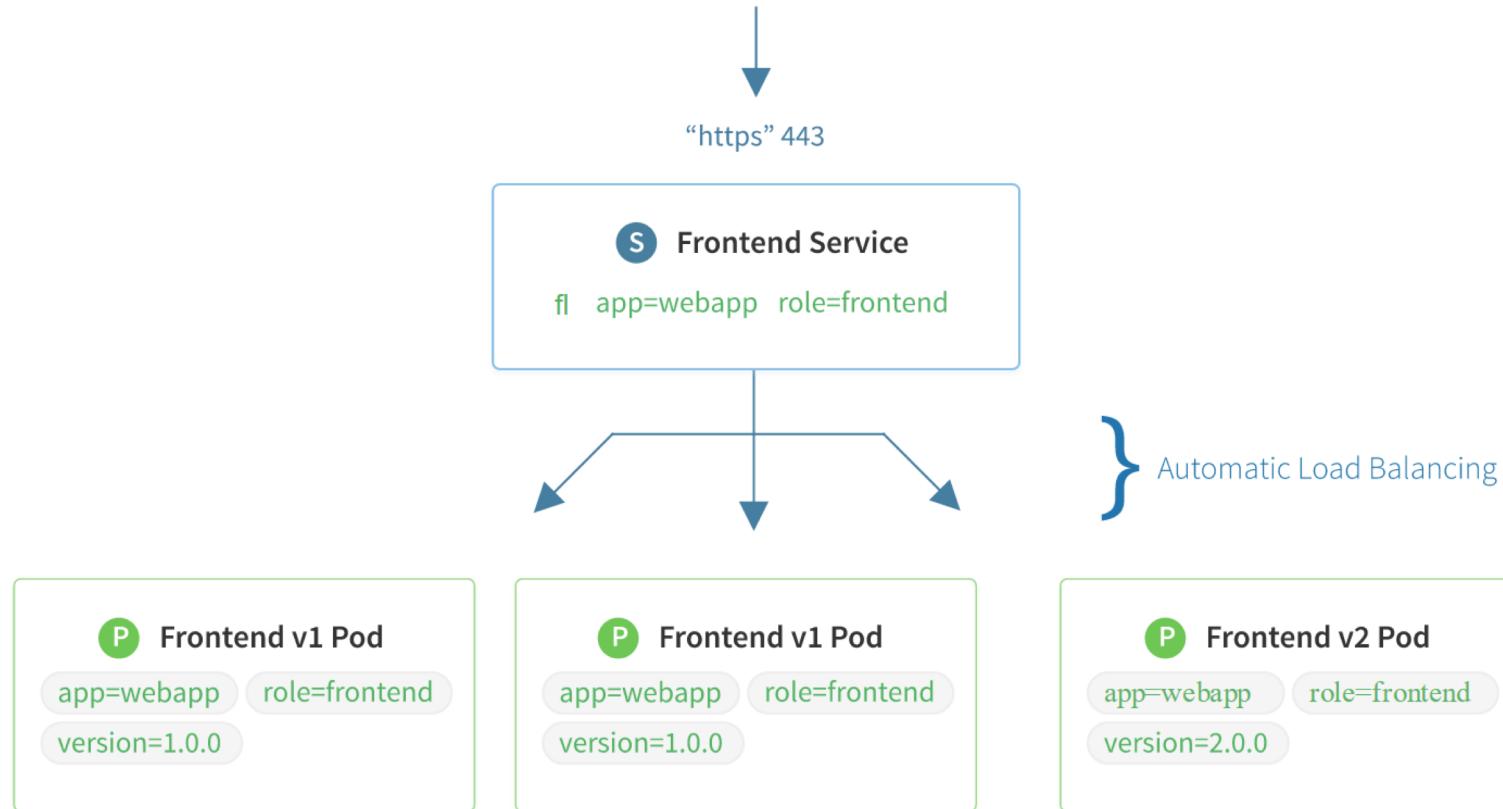
Service: Endpunkt unter einem definierten DNS-Namen, der Aufrufe an die Pods im Hintergrund verteilt (Load Balancing, Failover). Die für einen Service relevanten Pods werden über ihre Labels selektiert, z.B.: role = apache, env != test, tier in (web, app)

Pod: Gruppe an Containern, die auf dem selben Knoten laufen und sich eine Netzwerk-Schnittstelle inklusive einer dedizierten IP, persistente Volumes und Umgebungsvariablen teilen. Ein Pod ist die atomare Scheduling-Einheit in K8s. Ein Pod kann über sog. *Labels* markiert werden, das sind frei definierbare Schlüssel-Wert-Paare.

Replica Sets / Replication Controller: stellen sicher, dass eine spezifizierte Anzahl an Instanzen pro Pod ständig läuft. Ist für Reaktionen im Fehlerfall (Re-Scheduling), Skalierung und Rollouts (Canary Rollouts, Rollout Tracks, ..) zuständig.

Deployment: Klammer um einen gewünschten Zielzustand im Cluster in Form eines Pods mit dazugehörigem ReplicaSet. Ein Deployment bezieht sich nicht auf Services, da diese in der K8s-Philosophie einen von Deployments unabhängigen Lebenszyklus haben.

Ein Beispiel für das Zusammenspiel zwischen Services und Pods über Labels.



Quellen

- Services: <https://coreos.com/kubernetes/docs/latest/services.html>
- Pods: <https://coreos.com/kubernetes/docs/latest/pods.html>

Deployment Definition

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: zwitscher-service
spec:
  replicas: 3
  template:
    metadata:
      labels:
        zwitscher: service
    spec:
      containers:
        - name: zwitscher-service
          image: "hitchhikersguide/zwitscher-service:1.0.1"
          ports:
            - containerPort: 8080
          env:
            - name: CONSUL_HOST
              value: zwitscher-consul
```

Resource Constraints

```
resources:  
    # Define resources to help K8S scheduler  
  
    # CPU is specified in units of cores  
    # Memory is specified in units of bytes  
  
    # required resources for a Pod to be started  
requests:  
    memory: "128Mi"  
    cpu: "250m"  
  
    # the Pod will be restarted if limits are exceeded  
limits:  
    memory: "192Mi"  
    cpu: "500m"
```

Liveness und Readiness Probes

```
# container will receive requests if probe succeeds
readinessProbe:
  httpGet:
    path: /admin/info
    port: 8080
  initialDelaySeconds: 30
  timeoutSeconds: 5

# container will be killed if probe fails
livenessProbe:
  httpGet:
    path: /admin/health
    port: 8080
  initialDelaySeconds: 90
  timeoutSeconds: 10
```

Service Definition

```
apiVersion: v1
kind: Service
metadata:
  name: zwitscher-service
  labels:
    zwitscher: service
spec:
  # use NodePort here to be able to access the port on each node
  # use LoadBalancer for external load-balanced IP if supported
  type: NodePort
  ports:
  - port: 8080
  selector:
    zwitscher: service
```

Helm: Verwaltung von Applikationspaketen für Kubernetes.

The screenshot shows the official Helm GitHub repository page. At the top, there's a logo for "HELM by DEIS" and a "FORK ON GITHUB" button. The main title is "The package manager for Kubernetes" with the subtitle "Helm is the best way to find, share, and use software built for Kubernetes." Below this, there are two main sections: "Search" and "Install".

Search: A search bar contains the command "\$ helm search redis". The results show two charts: "redis-cluster (redis-cluster 0.0.5)" described as a "Highly available Redis cluster with multiple sentinels and standbys.", and "redis-standalone (redis-standalone 0.0.1)" described as a "Standalone Redis Master".

Install: A command line interface shows the output of "\$ helm install redis-cluster". It includes the creation of a service ("services/redis-sentinel"), a pod ("pods/redis-master"), a replication controller ("replicationcontrollers/redis"), and a sentinel ("replicationcontrollers/redis-sentinel"). The process concludes with a "Done" message.

Quelle: <https://github.com/helm/helm>