QAWARE SOFTWARE ENGINEERING

Hochschule München University of Applied Sciences

Cloud Computing

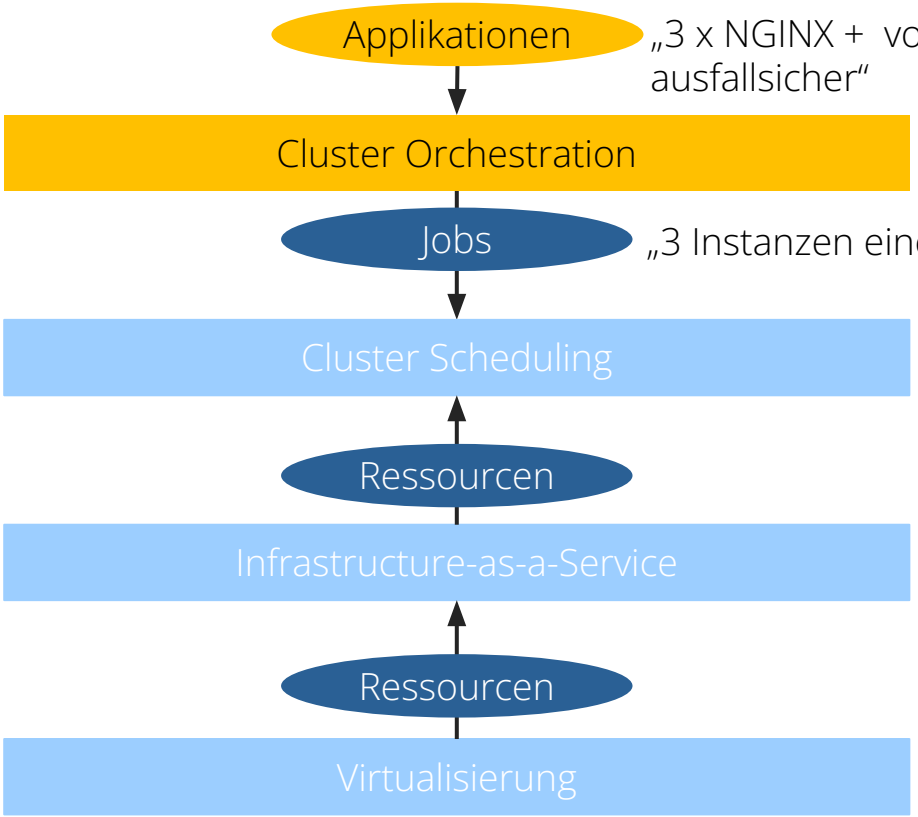
Orchestrierung

Johannes Ebke

2023-05-17

Orchestrierung - Definition, Aufgaben, Konzepte

Big Picture: Wir sind nun auf Applikationsebene



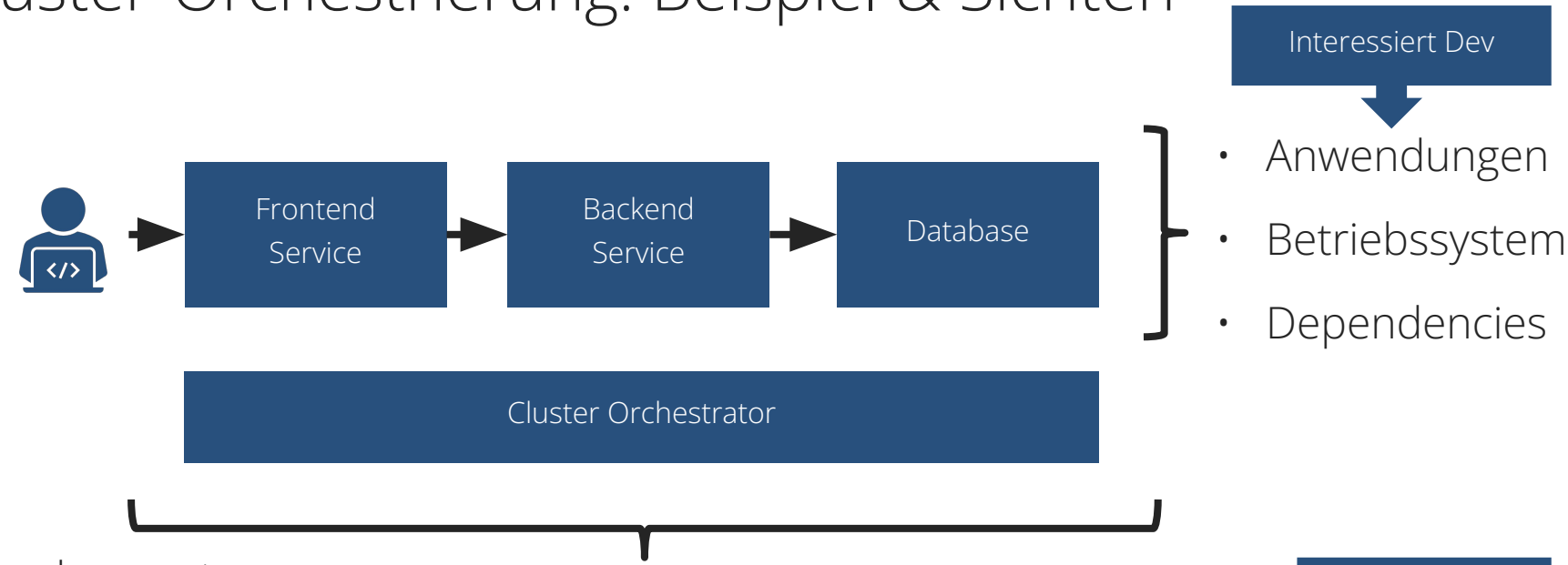
„3 x NGINX + vorgelagerter HAproxy - ausfallsicher“

How would you design your infrastructure if you couldn't login? Ever.

Kelsey Hightower

„3 Instanzen eines NGINX Containers“

Cluster-Orchestrierung: Beispiel & Sichten



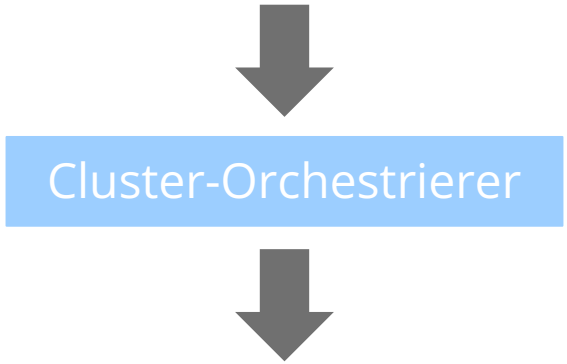
- Deployment
- Networking (Loadbalancing, Service Discovery, …)
- Scaling
- „Insight“ (Logging, Analytics, Rescheduling, Failure Recovery, …)

Interessiert (Dev)Ops

Cluster-Orchestrierung

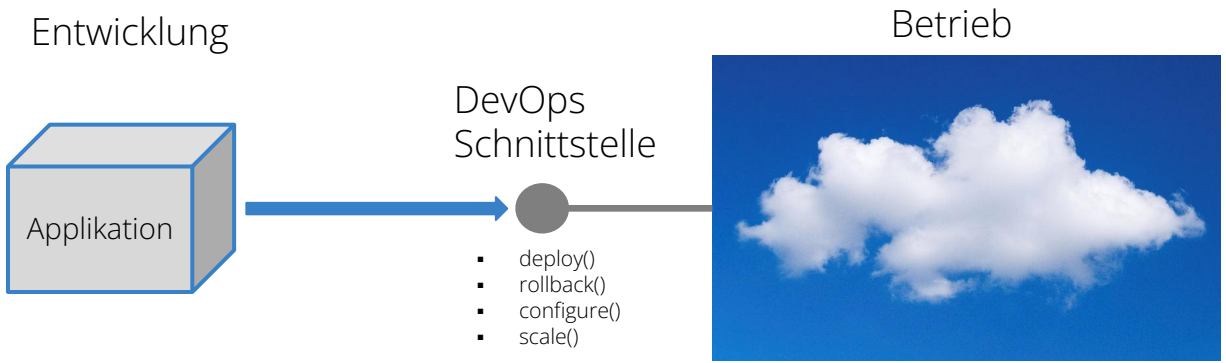
- Ziel: Eine Anwendung, die in mehrere Betriebskomponenten (Container) aufgeteilt ist, auf mehreren Knoten laufen lassen.
  „Running Containers on Multiple Hosts“.
  DockerCon SF 2015: Orchestration for Sysadmins
- Führt Abstraktionen zur Ausführung von Anwendungen mit ihren benötigten Schnittstellen und Betriebskomponenten 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.

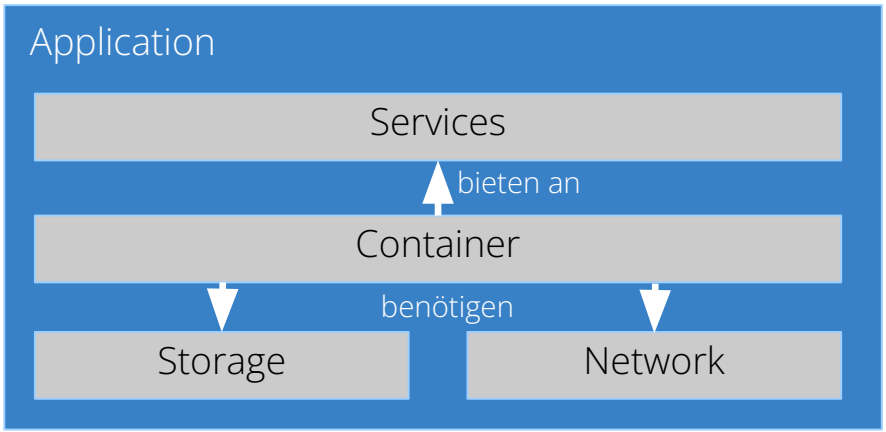


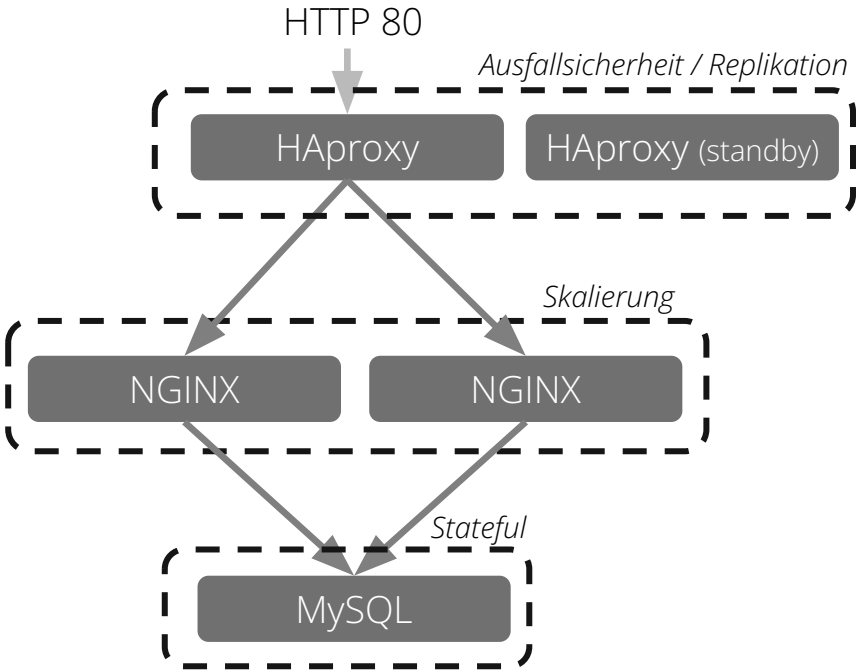
Steuerungsaktivitäten im Cluster:
- Start von Containern auf Knoten (□ Scheduler)
- Verknüpfung von Containern
- …

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



Dafür benötigt er für jede Anwendung eine Blaupause





Analogie 1: Lego Star Wars

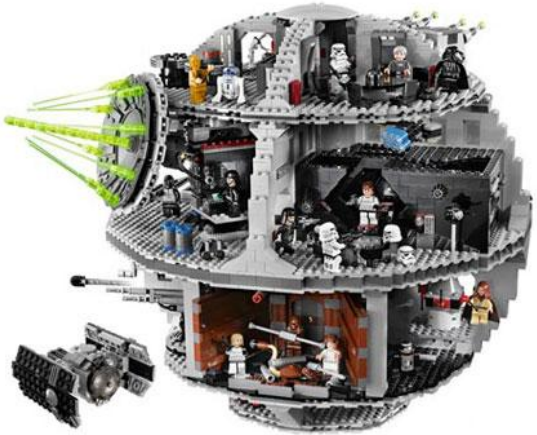
Services (angeboten)

Container

Quelle: Lego

Services (benötigt)

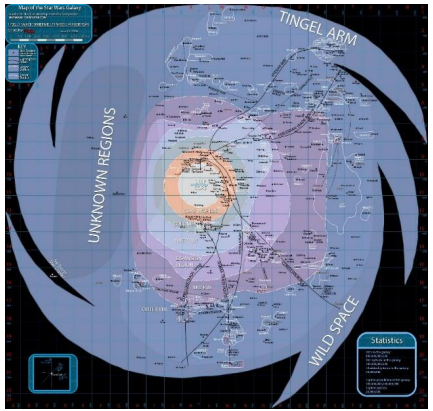
Cluster-Orchestrierer



Quelle: Lego

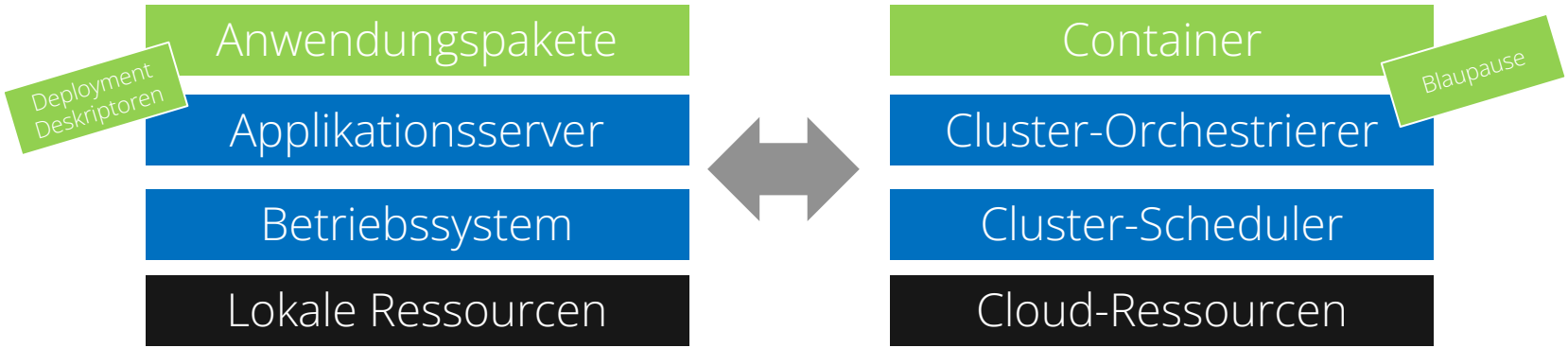
Blaupause

Cluster-Scheduler



Quelle: wikipedia.de

Analogie 2: Applikationsserver



Cluster-Orchestrierer automatisieren verschiedene Betriebsaufgaben für Anwendungen auf einem Cluster (1/2):

- 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.

Cluster-Orchestrierer automatisieren verschiedene Betriebsaufgaben für Anwendungen auf einem Cluster (2/2):

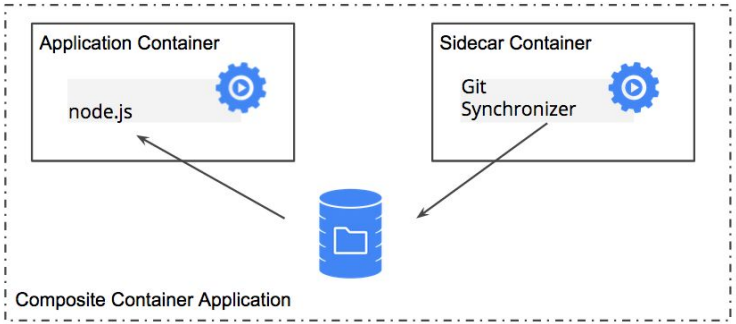
- Scheduling von Containern mit applikationsspezifischen Constraints (z.B. Deployment- und Start-Reihenfolgen, Gruppierung, ...)
- Aufbau von notwendigen Netzwerk-Verbindungen zwischen Containern.
- Bereitstellung von persistenten Speichern für zustandsbehaftete Container.
- (Auto-)Skalierung von Containern.
- Re-Scheduling von Containern im Fehlerfall (Auto-Healing) oder zur Performance-Optimierung.

Übung: Orchestrierungsmuster - Separation of Concerns mit modularen Containern

Sidecar Containers

Sidecar containers extend and enhance the "main" container, they take existing containers and make them better. As an example, consider a container that runs the Nginx web server. Add a different container that syncs the file system with a git repository, share the file system between the containers and you have built Git push-to-deploy. But you've done it in a modular manner where the git synchronizer can be built by a different team, and can be reused across many different web servers (Apache, Python, Tomcat, etc). **Because of this modularity, you only have to write and test your git synchronizer once and reuse it across numerous apps**. And if someone else writes it, you don't even need to do that.

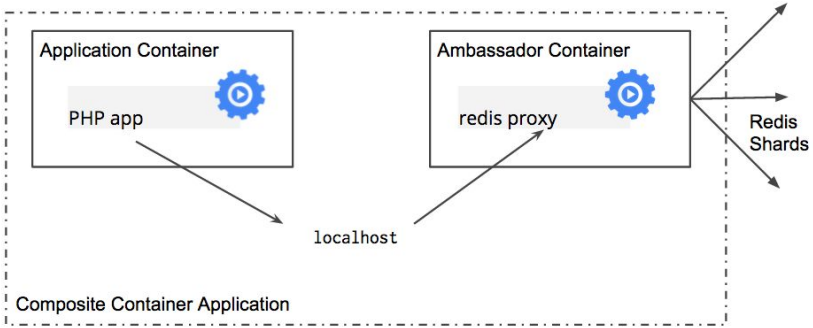
Quelle: https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/



Ambassador containers

Ambassador containers proxy a local connection to the world. As an example, consider a Redis cluster with read-replicas and a single write master. You can create a Pod that groups your main application with a Redis ambassador container. **The ambassador is a proxy is responsible for splitting reads and writes and sending them on to the appropriate servers**. Because these two containers share a network namespace, they share an IP address and your application can open a connection on "localhost" and find the proxy without any service discovery. **As far as your main application is concerned, it is simply connecting to a Redis server on localhost**. This is powerful, not just because of separation of concerns and the fact that different teams can easily own the components, but also because in the development environment, you can simply skip the proxy and connect directly to a Redis server that is running on localhost.

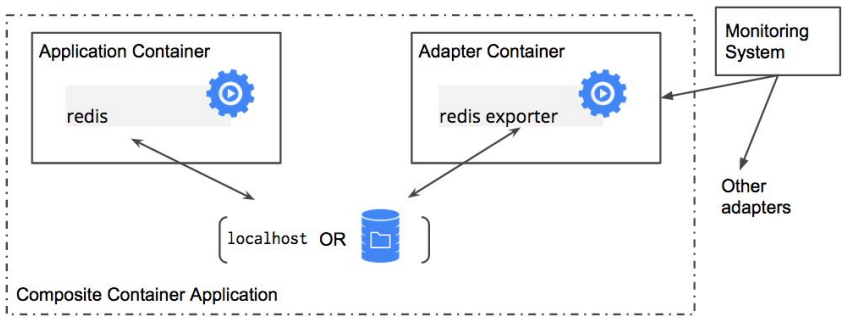
Quelle: https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/



Adapter containers

Adapter containers standardize and normalize output. Consider the task of monitoring N different applications. Each application may be built with a different way of exporting monitoring data. (e.g. JMX, StatsD, application specific statistics) but every monitoring system expects a consistent and uniform data model for the monitoring data it collects. **By using the adapter pattern of composite containers, you can transform the heterogeneous monitoring data from different systems into a single unified representation** by creating Pods that groups the application containers with adapters that know how to do the transformation. Again because these Pods share namespaces and file systems, the coordination of these two containers is simple and straightforward.

Quelle: https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/



Kubernetes

kubernetes

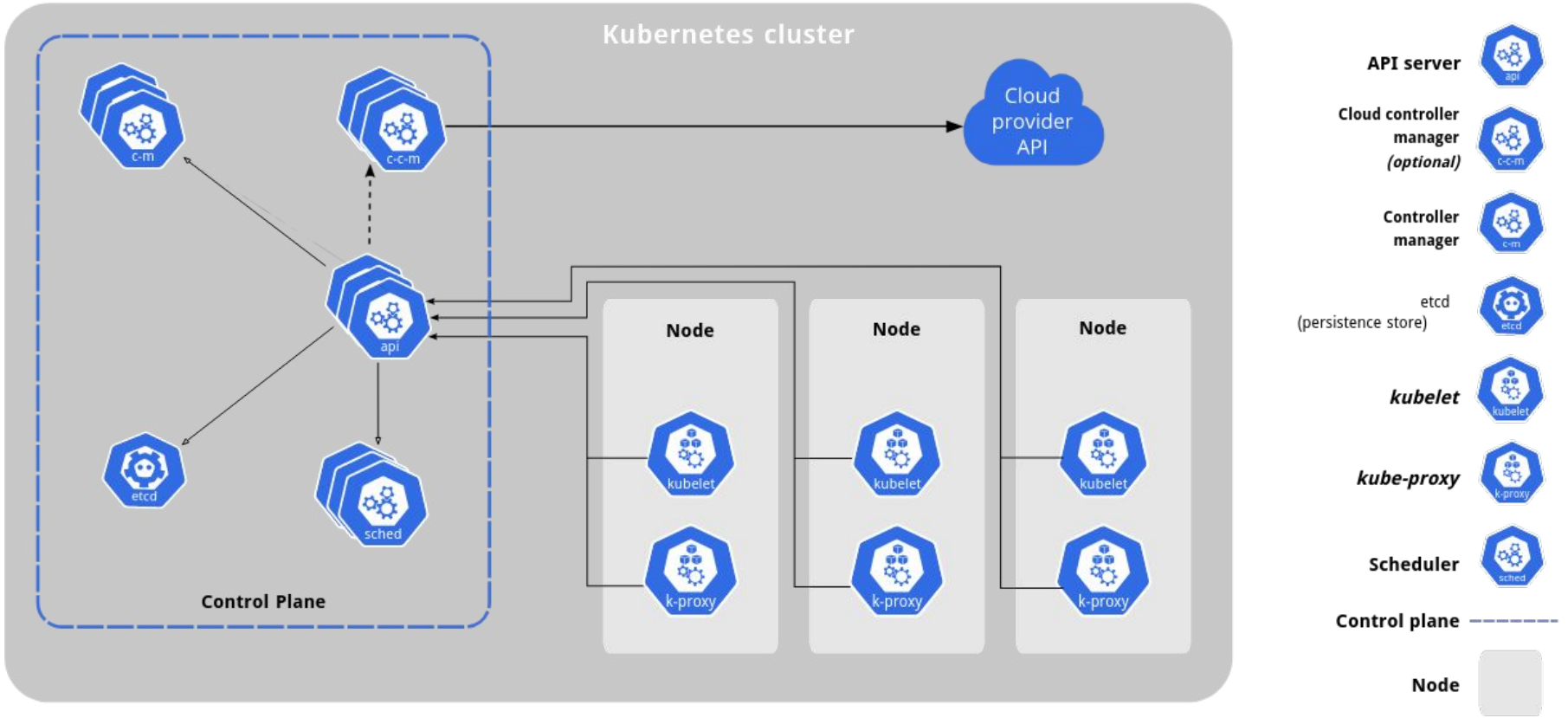
Kubernetes (auch als K8s bezeichnet) ist ein **Open-Source**-System zur **Automatisierung** der **Bereitstellung**, **Skalierung** und **Verwaltung** von **Container-Anwendungen**, das ursprünglich von Google entworfen und an die Cloud Native Computing Foundation (CNCF) gespendet wurde. Es zielt darauf ab, eine „Plattform für … Anwendungscontainer[n] auf **verteilten Hosts**“ zu liefern. Es unterstützt eine Reihe von Container-Tools, einschließlich **Docker**.

Quelle: Wikipedia

Wieso?!

- Ungelöst: Effiziente Nutzung der Hardware in einem Rechenzentrum
  - Ein Server pro Team? Was passiert, wenn ein Team keinen ganzen braucht?
- Virtualisierung und Elastizität löst das Problem der Hardware-Beschaffung
- Ungelöst: Development & Deployment der Anwendungen
- Trend: Weg vom Monolithen, hin zu Microservices
  - Verstärkt das Problem des Deployments
  - Wie finden sich die Microservices gegenseitig?
- Trend: Zero-downtime Deployments
- Lösung: Kubernetes als Standard
  - Anwendung in Container verpacken
  - Deployment beschreiben
  - Kubernetes kümmert sich um den Betrieb: startet neu, skaliert, etc.
  - Zero-Downtime und Rollbacks inkl.

Architektur von Kubernetes



Aufgaben der Bausteine auf einem Node

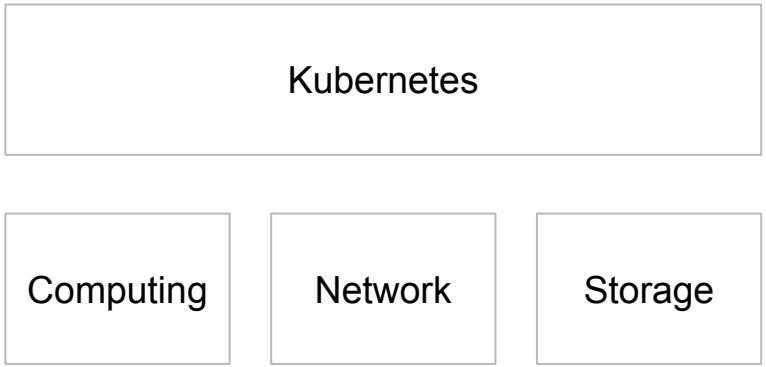
- **Pods** are the smallest deployable units of computing that you can create and manage in Kubernetes.
- Kubernetes runs your workload by placing containers into Pods to run on **Nodes**. A node may be a virtual or physical machine, depending on the cluster.
- **Container runtime**: “The container runtime is the software that is responsible for running containers. Kubernetes supports several container runtimes, one of them is Docker”
- **kubelet**: “An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.”
- A **service** is an abstract way to expose an application running on a set of Pods as a network service.
- **kube-proxy**: “kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.”

Aufgaben der Bausteine in der Control Plane

- **API Server**: “The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.”
- **etcd**: “Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.”
- **scheduler**: “Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.”
- **Controller Manager**: “Control Plane component that runs controller processes.”
    - Node controller: Responsible for noticing and responding when nodes go down.
    - Replication controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
    - ...
- **Cloud Controller Manager** (optional): “A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that just interact with your cluster.”

Kubernetes virtualisiert:

- Computing durch die Container Runtime
- Network durch virtuelle IPs und Overlay Networks
- Storage durch Persistent Volumes



Live-Demo: Lokales Kubernetes mit minikube

Minikube

- Minikube installiert ein lokales Kubernetes-Cluster auf dem eigenen Rechner
- Eignet sich super zum Ausprobieren von Kubernetes
- kubectl ist ein Client für die Kubernetes API, die verwendet wird, um Kubernetes zu konfigurieren
- K9s ist eine Terminal UI für Kubernetes

Live-Demo: Pods & Deployment

Pods & Deployments

- Pods sind die kleinste schedulbare Einheit in Kubernetes
- Ein Deployment beschreibt, aus welchen Containern der Pod besteht und wie er konfiguriert ist

Übung, Aufgabe 1, 2 & 3
30 Minuten

Live-Demo: Probes

Probes

- https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes
- Liveness probes prüfen, ob der Container noch am Leben ist. Falls nicht, wird dieser neu gestartet.
- Readiness probes prüfen, ob der Container bereit ist, Traffic zu bekommen. Falls nicht, wird dieser aus dem Load Balancing genommen.
- Mit einer Startup Probe kann das Problem von langsam startenden Containern mitigiert werden

Live-Demo: Resource Constraints

Resource constraints

- https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
- Resource Requests helfen dem Scheduler, einen geeigneten Node für den Pod zu finden
- Resource Limits schützen vor amoklaufenden Containern, in dem sie den Container abschießen, sollte dieser die Memory-Limits erreichen
- Limits können auf Speicher und auf CPU gesetzt werden

Resource Constraints: CPU

Limits and requests for CPU resources are measured in cpu units. One cpu, in Kubernetes, is equivalent to 1 vCPU/Core for cloud providers and 1 hyperthread on bare-metal Intel processors.

Fractional requests are allowed. A Container with spec.containers[].resources.requests.cpu of 0.5 is guaranteed half as much CPU as one that asks for 1 CPU. The expression 0.1 is equivalent to the expression 100m, which can be read as "one hundred millicpu". Some people say "one hundred millicores", and this is understood to mean the same thing. A request with a decimal point, like 0.1, is converted to 100m by the API, and precision finer than 1m is not allowed. For this reason, the form 100m might be preferred.

CPU is always requested as an absolute quantity, never as a relative quantity; 0.1 is the same amount of CPU on a single-core, dual-core, or 48-core machine.

Quelle: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

Übung, Aufgabe 4
15 Minuten

Live-Demo: Services

Services

- https://kubernetes.io/docs/concepts/services-networking/service/
- Services machen Anwendungen in Pods für andere Pods im Kubernetes-Cluster zugreifbar
- Services verwenden Label selectors, um die Pods zu finden, auf die der Service verweist
- Services load-balancen zwischen mehreren Pods

Live-Demo: Ingresses

Ingress

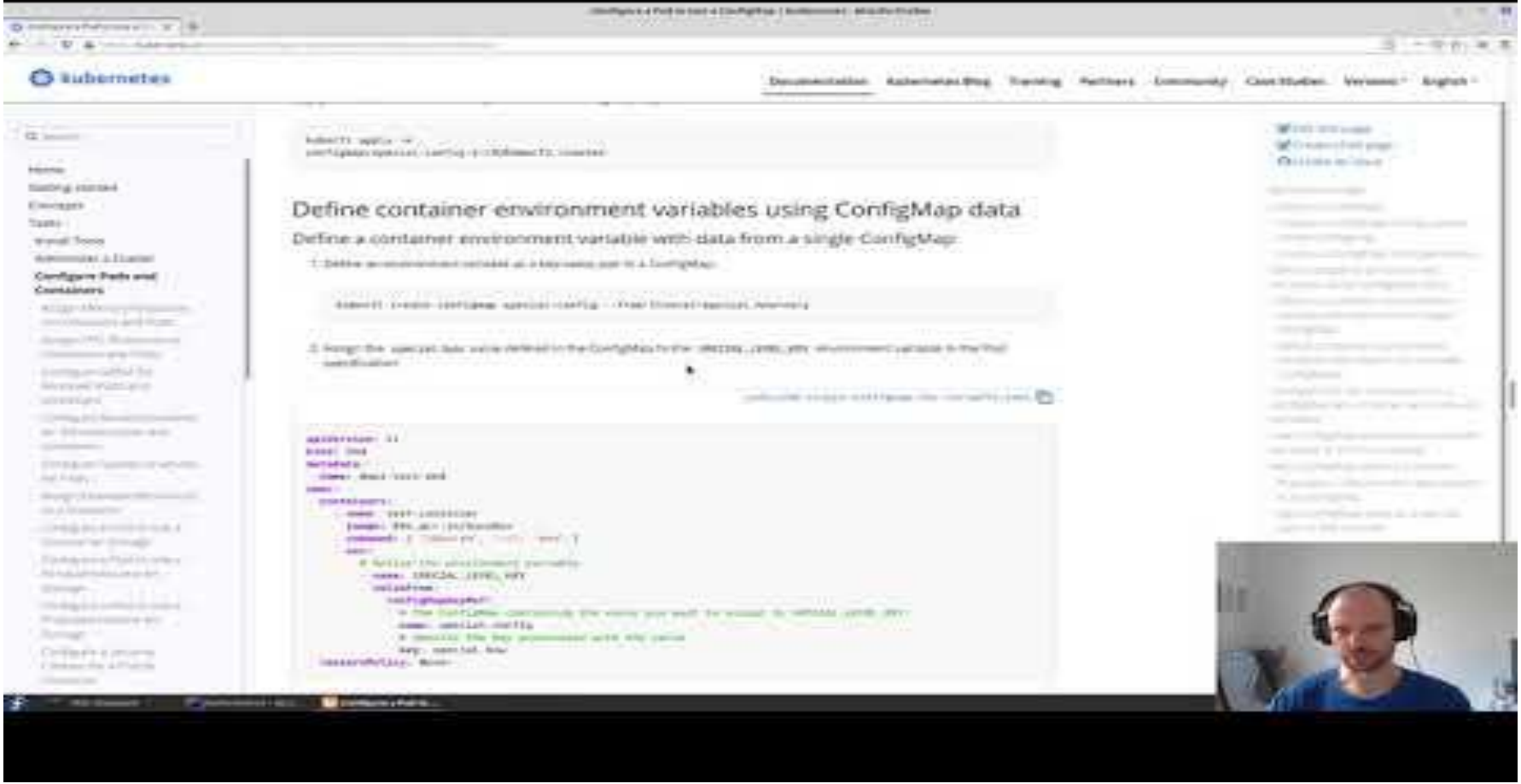
“An API object that manages **external access to the services** in a cluster, typically HTTP. Ingress may provide load balancing, SSL termination and name-based virtual hosting”

https://kubernetes.io/docs/concepts/services-networking/ingress/

- Durch einen Ingress kann ein Service von außerhalb des Clusters erreicht werden

Übung, Aufgabe 5
15 Minuten

Live-Demo: Config Maps



https://youtu.be/NbyZnJu7Wdo

Config Maps

- Config Maps werden verwendet, um eine Anwendung in Kubernetes zu konfigurieren
- Die Werte von Config-Maps können entweder als Umgebungsvariablen oder als Dateien im Container erscheinen

Übung, Aufgabe 6
15 Minuten

Live-Demo: Persistent Volumes

Volumes

- Persistent Volumes stellen Speicherplatz bereit
  - Diese können entweder statisch oder dynamisch provisioniert werden
  - Dies ist für den Admin des Kubernetes-Clusters interessant
- Persistent Volume Claims fordern ein Persistent Volume an
  - Dies ist für den Entwickler des Kubernetes-Clusters interessant
- Über ein Volume kann ein Pod ein Persistent Volume Claim verwenden
- Volume mounts lassen das Volume in einem Container über einen Linux-Mount erscheinen

Übung, Aufgabe 7
15 Minuten

Live-Demo: Helm, der Package Manager

Helm

- https://helm.sh/
- Paket-Manager für Kubernetes
- Erlaubt das Installieren von Charts in den Cluster
- Charts gibt es für alles mögliche: Redis, PostgreSQL, etc.
- Es gibt eine Suchmaschine für Charts
- Charts können über den `--set` oder den `--values` Parameter angepasst werden