



kubernetes
by Google™

Cloud Computing

Kubernetes

Peyer Lars

Berger Adrian

Version 0.1

6. Dezember 2020

1	Einleitung	2
2	Übersicht	3
2.1	Was ist Kubernetes?	3
2.2	Wieso Kubernetes?	3
2.2.1	Orchestrierung	3
2.2.2	Self-Healing	3
2.2.3	Verbreitung	3
2.3	Alternativen	4
2.3.1	Openshift	4
2.3.2	Docker Swarm	4
3	Architektur	5
3.1	Master-Nodes	5
3.1.1	API Server	5
3.1.2	Scheduler	5
3.1.3	Controller manager	5
3.1.4	Etcad	6
3.2	Worker-Nodes	6
3.2.1	Kublet	6
3.2.2	Pod	6
3.2.3	Container	6
3.2.4	Service proxy	7
3.3	Schnittstellen	7
3.3.1	kubectl	7
3.3.2	Web UI	7
4	Beispiels-Applikation in einem Kubernetes Cluster	8

1 Einleitung

Dieses Dokument beinhaltet eine Übersicht zur Container-Orchestrierungsplattform Kubernetes.

Damit diese Thematik erläutert werden kann, wird ein grundsätzliches Verständnis von Container vorausgesetzt. Dabei werden hauptsächlich Grundkenntnisse zu Docker vorausgesetzt.

Eine kurze Übersicht über Docker kann unter dem folgenden Link gefunden werden:

<https://www.redhat.com/en/topics/containers/what-is-docker>

2 Übersicht

2.1 Was ist Kubernetes?

Bei Kubernetes, kurz k8s, handelt es sich um ein Open-Source Projekt, welches seinen Existenz der Idee verschiedener Google Entwickler zu verdanken hat.

Dabei war die Grundidee, ein System zu erstellen, welches die Ressourcen eines Servers automatisch idealen verwenden kann.[1]

Kubernetes ist griechisch und bedeutet Steuermann. Sowie ein Steuermann die Arbeiten auf seinem Schiff verwaltet, so verwaltet Kubernetes verschiedene Container auf einem Server. Unter Verwaltung versteht sich hierbei ebenfalls die Bereitstellung und Skalierung von Containern. [2]

Technisch spricht man dabei von einem Container-Orchestrierungs-System für Container-Applikationen.

2.2 Wieso Kubernetes?

2.2.1 Orchestrierung

Mittels Kubernetes können die Ressourcen auf dem Server je nach Auslastung automatisch skaliert und an die verschiedenen Container verteilt werden. Dadurch kann der Server bei hoher Auslastung weiterhin alle Container wie vom Entwickler vorgesehen betreiben.

Dabei beschränkt sich Kubernetes nicht auf die Verwendung von einem einzelnen Server, sondern wird im Regelfall über mehrerer Server verteilt. So können offene Anfragen stets an den Server weitergeleitet werden, welcher noch die notwendigen Ressourcen zur Verfügung hat.

2.2.2 Self-Healing

Kubernetes kennt das Prinzip von Self-Healing (Selbstheilung). Fällt ein Server oder auch nur ein laufender Container aus, erkennt Kubernetes dies und startet automatisch auf einem anderen Server den oder die fehlenden Container neu. Dadurch kann eine hohe Verfügbarkeit von Applikationen gewährleistet werden, da Abstürze im System automatisch behoben werden.

2.2.3 Verbreitung

Kubernetes ist, allem voran dank der Unterstützung von Google, aktuell eine der am besten unterstützten Container-Orchestrierungs-Systemen. So gibt es unter anderem bei Amazon Web Services (AWS), Microsoft Azure und Google Cloud einfache Konfigurationsoptionen für Kubernetes. Ebenfalls haben in den letzten Jahren auch immer mehr kleinere Hosting-Provider damit begonnen, Kubernetes als Dienstleistung anzubieten.

2.3 Alternativen

Es gibt verschiedene Alternativen zu Kubernetes. Im folgenden stellen wir zwei der beantragsten davon vor:

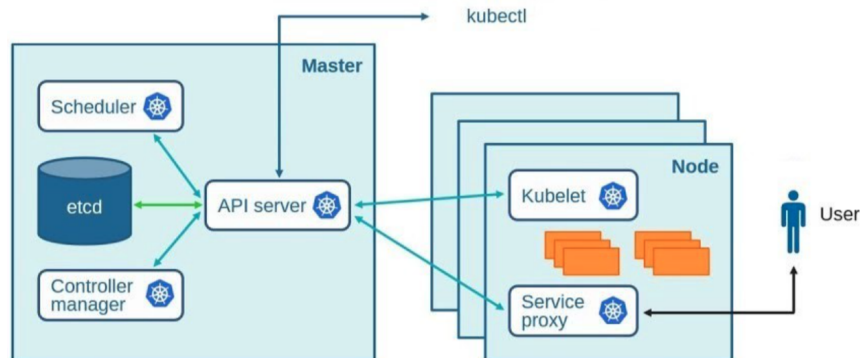
2.3.1 Openshift

Bei OpenShift handelt es sich um ein Projekt von Red Hat. Im Gegensatz zu Kubernetes ist OpenShift nicht Open-Source sondern wird von Red Hat als kostenpflichtigen Service angeboten. Dabei basiert dies auf dem Open-Source Tool OKD, was wiederum auf Kubernetes aufbaut. OpenShift ist somit im wesentlichen eine Erweiterung zu Kubernetes, die das Deployment und Monitoring vereinfacht. Für Einsteiger bietet dies oft eine einfacheren Einstieg in die Orchestrierungswelt als Kubernetes. Dafür sind die Konfigurationsoptionen für Kubernetes zahlreicher. [3]

2.3.2 Docker Swarm

Docker Swarm ist verwendet als Engine direkt Docker. Dadurch ist die Einrichtung von Swarm meist einfacher als das Aufsetzen eines Kubernetes Clusters. Swarm ist generell simpler zu konfigurieren, da es weniger Funktionalitäten bietet als Kubernetes. Ebenfalls ist kein automatisches Skalieren innerhalb von Swarm möglich. [4]

3 Architektur



3.1 Master-Nodes

Kubernetes funktioniert nach dem Master-Worker Prinzip. Der Master Node ist dabei der Hauptknotenpunkt von Kubernetes. Er kontrolliert und koordiniert alle Aktionen, die im Cluster erfolgen. Fällt der Master-Node in einem Cluster aus, so wird dieser disfunktional. Die Node Server warten auf Befehle vom Master und führen diese anschliessend aus. Folgende Hauptkomponenten sind Bestandteil eines Kubernetes Master-Nodes und werden auch Control-Plane genannt. [5]

3.1.1 API Server

Der API-Server ist die Drehscheibe des Kubernetes-Cluster und ist zuständig für die Entgegennahme und Verteilung von Aufträgen wie z.B. Deployments oder Konfigurationen. Alle Komponenten des Clusters (Master- und Worker-Node) sind an den API Server angeschlossen und tauschen so untereinander Status-Informationen über den Cluster-Zustand aus. [5]

3.1.2 Scheduler

Der Scheduler kennt alle verfügbaren Ressourcen im Cluster und den Zustand aller verbundenen Worker-Nodes. Wird über den API Server ein neues Deployment in Auftrag gegeben, sucht sich der Scheduler einen passenden Worker-Node raus, welcher über die benötigten Ressourcen (CPU, RAM, DISK) verfügt. Er meldet die passenden Worker-Nodes dem API-Server, welcher dann das Deployment auf den ausgewählten Nodes ausführt. [5]

3.1.3 Controller manager

Der Controller manager besteht eigentlich aus vier Subkomponenten, welche zu einem Controller zusammengeführt werden.

Der **Node-Controller** überwacht den Zustand aller Worker-Nodes im Cluster. Fällt eine Node aus, schaut der Controller ob die Node neu gestartet werden kann oder ob eine neue Node erstellt werden muss.

Der **Replication-Controller** ist zuständig dafür, dass alle Deployments gemäss Konfiguration repliziert werden. Sollte ein Worker-Node ausfallen, werden über den Replication-Controller neue Replikationen auf anderen Worker-Nodes gestartet.

Der **Endpoint-Controller** baut die Verbindung zwischen allen Pods und Services her. Services werden dazu verwendet, dass Pods untereinander kommunizieren können.

Die **Service Account und Token Controllers** sind zuständig um neue Standardkonten beim Erstellen von Namespaces (Gruppierungen von Pods) zu kreieren. Zudem verwalten sie die Zugriffstokens des API Servers damit nicht einfach jeder auf den Cluster zugreifen kann. [5]

3.1.4 Etcd

Etcd ist ein externer hoch konsistenter Key-Value Store, welcher speziell auf Verteilte Systeme ausgelegt ist und die wichtigsten Daten des Kubernetes Cluster ablegt. Es werden sämtliche Konfigurationen und Zustände des Systems im Etcd Store abgelegt. Dadurch empfiehlt es sich diesen Store zu replizieren, denn bei einem Ausfall sind alle Cluster Daten verloren und der Cluster somit unbrauchbar. [5]

3.2 Worker-Nodes

Worker-Nodes sind Arbeitstiere und stellen dem Cluster die Ressourcen zur Verfügung um Applikationen bereit zu stellen. Dabei handelt es sich meist um virtuelle Maschinen oder kleine physische Maschinen, welche zu einem grossen Netzwerk zusammen geführt werden. Ein Master-Node kann maximal bis zu 5000 Worker-Nodes unter sich verwalten. [5]

3.2.1 Kubelet

Auf jedem der Worker-Nodes läuft der Kubelet-Dienst. Dieser ist für die Kommunikation im Cluster zuständig und verbindet sich mit dem API Server damit Statusinformationen und Konfiguration ausgetauscht werden können. Dieser Dienst ist das Herzstück des Worker-Nodes und kommuniziert mit der Container Runtime (meistens Docker), welche auf dem Node installiert ist, damit Pods und Containers erstellt werden können. [5]

3.2.2 Pod

Ein Pod ist die kleinste verwaltbare Einheit in einem Kubernetes Cluster und beinhaltet einen oder mehrere Containers. Alle Container im Pod teilen sich die definierten Konfigurationen und die geteilten Speicher- und Netzwerk-Ressourcen. Es empfiehlt sich nur stark verbundene Container, welche sich gegenseitig ergänzen, in einem Pod zusammen zu fassen. In den meisten Fällen befindet sich lediglich ein Container in einem Pod. [5]

3.2.3 Container

Container werden auf Basis von einem Container-Abbild gestartet und beinhalten alles was die Applikation benötigt um lauffähig zu sein. Der Container verwendet zum Betrieb des Ab-

bildet die Ressource, welche ihm durch die Container Runtime zur Verfügung gestellt werden. Am weitesten verbreiten sind Docker Container, welche aus einem Docker-Image gestartet werden. In einem YAML-File (Dockerfile) wird die Applikation konfiguriert und mittels Docker zu einem Image gebaut. [5]

3.2.4 Service proxy

Damit Pods im Cluster miteinander kommunizieren können, werden Services eingesetzt. Dabei wird definiert bei welchem Pod welche Ports geöffnet sind und über welche lokale IP die Pods miteinander kommunizieren können.

Pods können zur internen Kommunikation anstelle der IP Adresse auch einfach über den Service-Name angesprochen werden. Dies ist vor allem bei replizierten Applikationen interessant, da die Anfragen-Verteilung auf die verschiedenen Replikationen nicht manuell gemacht werden muss. Zudem dient der Service Proxy für den Benutzer der Zugangspunkt einer Applikation. [5]

3.3 Schnittstellen

3.3.1 kubectl

Kubectl ist eine Applikation, welche in der Kommandozeile läuft und zur Konfiguration von Kubernetes Clustern dient. Diese Applikation ist direkt mit dem API Server verbunden und bietet eine Vielzahl an Befehlen um den Cluster zu konfigurieren oder Applikationen zu deployen. Deployment werden mittels YAML-Files geschrieben und können als Datei direkt dem Kubectl zur Übertragung an den Cluster übergeben werden. [5]

3.3.2 Web UI

Alternativ zum Kubectl kann auch das Web UI zur Konfiguration des Clusters verwendet werden. Im Web UI erhält man eine schöne visuelle Übersicht über den Cluster und sieht mittels Graphen die Auslastung der einzelnen Komponenten. Die Konfiguration mittels Kubectl scheint aber um einiges schneller zu gehen und darum ist das Web UI mehr eine schöne Ergänzung dazu. [5]

4 Beispiels-Applikation in einem Kubernetes Cluster

Für unser Beispiel haben wir eine simple Applikation gewählt, die aus den folgenden Containern besteht:

Vue.js Frontend Container Django Backend Container Postgres Datenbank Container

Diese Applikation soll nun bei Hetzner in einem Kubernetes Cluster betrieben werden.

Literatur

- [1] Red Hat. *Kubernetes (k8s) erklärt*. URL: <https://www.redhat.com/de/topics/containers/what-is-kubernetes>.
- [2] SysEleven MetaKube. *Kubernetes einfach erklärt: Antworten vom Techie für den Noobie*. 2019. URL: <https://blog.syseleven.de/kubernetes-einfach-erklaert>.
- [3] Tomasz Cholewa. *10 most important differences between OpenShift and Kubernetes*. 2019. URL: <https://cloudowski.com/articles/10-differences-between-openshift-and-kubernetes/>.
- [4] Tomasz Cholewa. *10 most important differences between OpenShift and Kubernetes*. URL: <https://cloudowski.com/articles/10-differences-between-openshift-and-kubernetes/>.
- [5] Kubernetes. *Kubernetes Documentation*. URL: <https://kubernetes.io/docs/home/>.