Johannes Scheuermann, Nils Magnus

Tanz der Container

Container-Orchestrierung mit Docker-Tools und Kubernetes

Richtig Sinn machen Container erst, wenn man sie massenweise einsetzt. Orchestrierungs-Tools organisieren das Zusammenspiel und automatisieren den Container-Betrieb. Wir zeigen an einem konkreten Beispiel, wie das mit den Docker-Werkzeugen Compose, Machine und Swarm sowie mit Googles Kubernetes funktioniert.

Erfolgreiche Webanwendungen finden schnell mehr Anwender, als die Entwickler sich das ursprünglich gedacht haben. Bei einem einfachen Webserver mit statischen Inhalten mag es ausreichen, Software und Inhalte auf mehrere Maschinen zu kopieren und einen Load-Balancer davor zu stellen. Moderne Webanwendungen enthalten allerdings auch Anwendungslogik, die ihre Daten dauerhaft in einer Datenbank speichert – da wird es komplizierter.

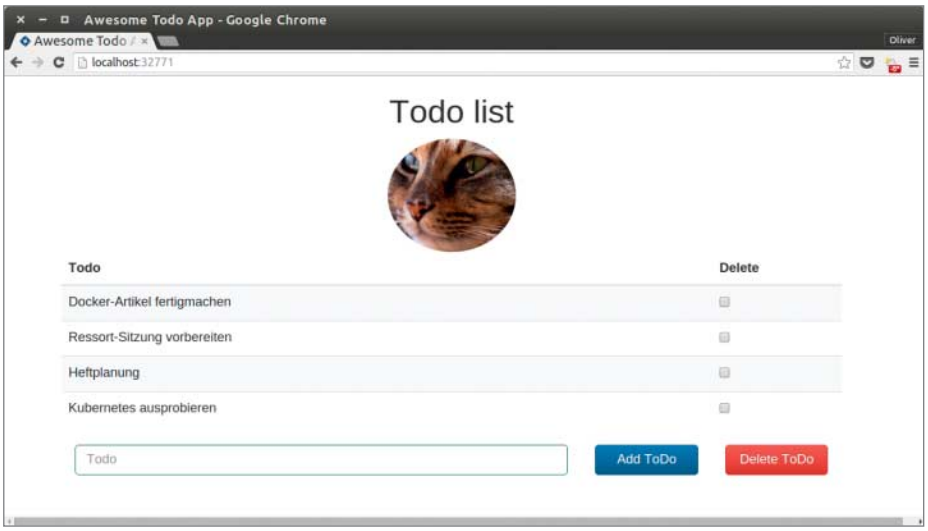
Skalierbare Anwendungen

Ein Beispiel soll das illustrieren. Ein junges Start-up kommt auf die bahnbrechende Idee, eine öffentliche To-do-Liste bereitzustellen. Jeder darf Einträge zu der Liste hinzufügen, sie nach Erledigung löschen oder einfach nachschauen, was alles zu tun ist.

Die Entwickler stellen schnell fest, dass auf einen Schreibzugriff der Datenbank ein Vielfaches an Lesezugriffen kommt – das ist bei Web-Anwendungen häufig der Fall. Ideal wäre also eine Datenbank, die auf einem Knoten die Schreibzugriffe entgegennimmt und die Daten auf viele Leseknoten verteilt, auf denen die Daten nicht verändert werden dürfen. Die NoSQL-Datenbank Redis ist so eine Datenbank mit eingebauter Replikation.

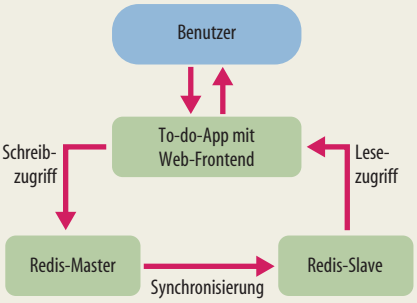
Im einfachsten Fall besteht die Merkzettel-App also aus drei Komponenten: dem Web-Frontend, dem Redis-Master für Schreibzugriffe und dem Redis-Slave für Leseanfragen. Jede Komponente läuft dabei in einem eigenen Container. Der Redis-Slave verbindet sich beim Start mit dem Redis-Master, das Frontend öffnet Verbindungen zum Redis-Master und -Slave. Der Kasten „Loslegen mit der Merkzettel-App“ beschreibt, wie Sie die Beispiel-App mit Docker zum Laufen kriegen.

Wenn immer mehr Anwender auf die To-do-Liste zugreifen, benötigt sie irgendwann mehr Web-Frontends, mehr Redis-Slaves oder beides. Wonach mehr Bedarf besteht, ist im Vorfeld schwer abzuschätzen und kann sich mit der Zeit auch ändern. Daher sollte



Die Merkzettel-App

Die To-do-Liste besteht aus drei Komponenten: der eigentlichen Web-App und zwei Instanzen der Redis-Datenbank. Container helfen bei der Skalierung.



Ein schlichtes Web-Frontend erlaubt den Zugriff auf die To-do-Liste. Für die Programmlogik dahinter sorgt ein kleines Go-Programm.

die Zahl der Instanzen der Teilkomponenten möglichst flexibel sein.

Da nicht feststeht, wie viele Instanzen der einzelnen Komponenten es gibt und welche IP-Adressen sie haben, verwenden solche skalierbaren Webanwendungen Platzhalter statt fester IP-Adressen. Orchestrierungs-Software ersetzt diese Platzhalter durch konkrete Werte und verbindet die Komponenten miteinander.

Es existieren mehrere Orchestrierungs-Frameworks für Docker, die sich in ihrer Komplexität unterscheiden, darunter die Tools aus dem Hause Docker selbst, das CoreOS-Werkzeug Fleet sowie das von Google entwickelte Kubernetes. Im Folgenden zeigen wir, wie die Merkzettel-App mit den Docker-Tools Compose, Machine und Swarm sowie mit Kubernetes orchestriert und über mehrere Hosts in einem Cluster verteilt wird.

Hausmittel statt Handarbeit

Anfang 2015 haben die Docker-Entwickler die Orchestrierungs-Tools Compose, Machine und Swarm veröffentlicht. Compose startet die Container einer Anwendung nach den Angaben einer Datei im YAML-Format. Die Compose-Datei docker-compose.yml des Beispiels definiert drei Services („todoApp“, „redisSlave“, „redisMaster“) und legt für jeden Service fest, welches Container-Image Compose starten soll. Den Redis-Containern werden Namen zugeordnet, über die andere Container sie ansprechen können; der Frontend-Container macht den Port 3000 für andere Container und den Host zugänglich.

Alle drei Container werden dem gemeinsamen Docker-Netzwerk „todoapp_network“ zugeordnet, das der Befehl `docker network create` anlegt (siehe Textkasten „Container managen mit Compose“). Docker weist jedem Container eine eigene, private IP-Adresse zu. Über Docker-Netzwerke lassen sich seit Docker 1.9 für jede Anwendung eigene Netzbereiche definieren, innerhalb derer die Container und ihre Hosts sich untereinander erreichen.

Wenn alle Container auf dem gleichen Host laufen, besteht das Docker-Netz nur aus einer Bridge. Bei mehreren Hosts arbeitet es als ein Overlay-Netz: Container des gleichen Netzes können andere Container auch auf anderen Hosts kontaktieren. Dazu nutzen alle Teilnehmer dieses Netzwerks einen gemeinsamen Key-Value-Store wie Etcd, Consul oder

Loslegen mit der Merkzettel-App

Die auf mehrere Container verteilte Merkzettel-Anwendung läuft auf jeder aktuellen Linux-Distribution mit einem Kernel ab 3.16 – wir haben mit Debian 8 Jessie getestet. Nötig ist lediglich Docker ab Version 1.9, für die Demonstration des Cluster-Betriebs außerdem VirtualBox.

Bei Debian 8 befindet sich das Docker-Paket namens „docker.io“ im Backports-Repository, CentOS 7 hält das Docker-Paket im Extras-Repository vor. Ubuntu 14.04 LTS enthält eine veraltete Docker-Version, hier richten Sie am besten die aktuelle Docker-Version über das Installationsskript auf get.docker.com ein. Umsichtige Anwender prüfen das Skript, bevor sie es ausführen.

Die Merkzettel-App finden Sie auf GitHub:

```
git clone https://github.com/inovex/docker-orchestration
```

Im Unterverzeichnis container-src befinden sich die Dockerfiles für den Redis-Slave und das Web-Frontend; als Redis-Master kommt das unveränderte Image „redis“ aus dem Docker-Hub zum Einsatz. Das Image „redis-slave“ ergänzt das Orginal-Redis-Image lediglich um ein Bash-Skript, das dem Slave beim Start mitteilt, wo sich der Redis-Master befindet.

Die in Go geschriebene To-do-App liegt im Unterverzeichnis web-frontend; sie enthält bereits einen Webserver. Für das HTML-Frontend sorgen die Dateien in public. Im Unterverzeichnis bin befindet sich die ausführbare To-do-App. Das Dockerfile packt sie zusammen mit dem HTML- und JavaScript-Code sowie dem nur wenige MByte großen Alpine Linux in ein Image namens „todo-app-web“.

Die fertigen Container-Images mit dem Redis-Slave und dem Frontend haben wir auf den Docker-Hub hochgeladen. Die Befehle

```
sudo docker network create todoapp_network
sudo docker run --net=todoapp_network --name=redis-master -d redis
sudo docker run --net=todoapp_network --name=redis-slave -d ↵
                                                johscheuer/redis-slave:v1
sudo docker run --net=todoapp_network --name=frontend -d -p 3000:3000 ↵
                                                johscheuer/todo-app-web:v1
```

legen ein virtuelles Docker-Netz für die Anwendung an, laden die Images vom Docker-Hub herunter und starten die Anwendung. `sudo docker inspect frontend` gibt im Feld IPAdress die IP-Adresse aus, unter der Sie auf Port 3000 das Web-Frontend erreichen.

Container managen mit Compose

Compose verwaltet Anwendungen, die aus mehreren Containern bestehen. Laden Sie das Programm von github.com/docker/compose/releases herunter, machen Sie es mit `chmod +x` ausführbar und speichern Sie es unter dem Namen docker-compose in /usr/local/bin.

Im Unterverzeichnis 1-Docker-Tools der Merkzettel-App befindet sich die YAML-Datei docker-compose.yml, die den Aufbau der Anwendung aus drei Containern beschreibt. In diesem Verzeichnis startet der Befehl

```
sudo docker-compose up -d
```

die Anwendung. Dazu muss das im ersten Kasten beschriebene Docker-Netwerk „todo_network“ bereits existieren. Der Aufruf lädt die Images „redis“, „redis-slave“ und „todo-app-web“ vom Docker-Hub herunter, sofern nötig. Das Compose-Unterkommando `ps` prüft den Status der Container, `sudo docker-compose logs` zeigt die Logs der Container an.

Die dynamisch zugewiesene öffentliche IP-Adresse des Web-Frontends, das intern an Port 3000 lauscht, liefert der Befehl

```
sudo docker-compose port todoApp 3000
```

Über diese Adresse erreicht man das Web-Frontend, in dem sich neue Aufgaben anlegen und erledigte löschen lassen.

Zookeeper als Dienste-Verzeichnis. Wenn ein Netzpaket an einen Container auf einem entfernten Host gehen soll, schreibt es die Docker-Engine mit Netfilter so um, dass es den Weg zum richtigen Host findet. Daher müssen die einzelnen Hosts eines Netzes untereinander ungehindert kommunizieren können.

Viel zu tun

Wenn die To-do-Liste so großen Erfolg hat, dass das Web-Frontend nicht mehr hinterher kommt, lässt sich die Zahl der Frontends mit einem Compose-Aufruf auf drei erhöhen:

```
docker-compose scale todoApp=3
```

Compose startet anhand der Konfiguration des todoApp-Services in der YAML-Datei zwei weitere Container (einer läuft ja bereits) und verbindet sie mit der Datenbank. Jede Instanz des Frontends ist dabei über einen eigenen Port auf dem Host zu erreichen; denjenigen des zweiten Web-Frontends gibt der folgende Befehl aus:

```
docker-compose port --index=2 todoApp 3000
```

Jetzt kann ein vorgeschalteter Load-Balancer die Anfragen auf die verschiedenen Instanzen des Frontends verteilen.

Nicht alles skaliert von allein

Solange nur eine Instanz des Frontends existiert, könnte man dessen Port 3000 in der Compose-Datei einem festen Port auf dem Host zuordnen, über den man die App erreicht. Da aber nicht mehrere Container denselben Host-Port verwenden können, verhindert eine feste Port-Zuordnung die Skalierung, daher muss die Zuordnung dynamisch erfolgen.

Außerdem darf die Compose-Datei den Containern keine Container-Namen zuweisen – diese müssen auf einem Docker-Host (und auch in einem Swarm-Cluster, dazu gleich mehr) einzigartig sein. Diese Einschränkung entfällt mit der Anfang Februar vorgestellten Version 1.6 von Compose: Docker trägt die Container dann mit ihrem Servicenamen in sein integriertes DNS ein. Allerdings fehlt noch ein Load-Balancing, das Anfragen automatisch auf die Instanzen eines Dienstes verteilt. Stürzt jedoch ein Redis-Slave ab, trägt Docker automatisch den nächsten noch laufenden Slave in sein DNS ein, sodass Anfragen weiter beantwortet werden.

Mit dem Eintrag `restart: always` in der Compose-Datei startet Compose Container neu, wenn sie abstürzen oder sich aus einem anderen Grund beenden. Die Dokumentation beschreibt einige weitere Schlüssel, die weitere Eigenschaften von Containern und ihre Beziehungen untereinander definieren.

Ein Schwarm von Hosts

Compose verwaltet Container nur auf einem Docker-Host. Wer mehrere Rechner als Docker-Host nutzen will, muss sie erst einmal mit der Docker-Engine bestücken. Das Docker-Tool Machine provisioniert neue Docker-Hosts sowohl auf bereits installierten Linux-Rechnern als auch auf virtuellen Maschinen.

Machine kann selbst virtuelle Maschinen unter verschiedenen lokalen Virtualisierungsplattformen wie VMware und Hyper-V sowie bei diversen Cloud-Anbietern neu anlegen. Darauf wird dann Boot2Docker installiert, ein abgespecktes Linux mit Docker-Engine. Bei VMs und Rechnern, auf denen bereits ein Linux installiert ist, installiert Machine Docker per SSH.

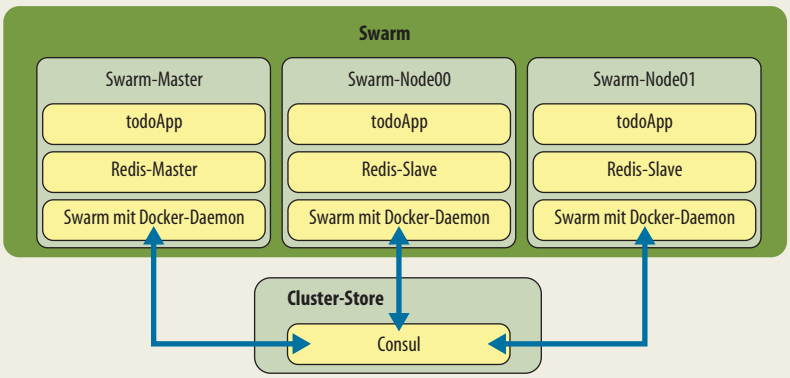
Die entfernten Docker-Engines steuert man über das Netzwerk mit den lokalen Docker-Kommandozeilentools. Sie erfahren über Umgebungsvariablen, auf welchem Host sie arbeiten sollen. Die Verbindung sichert Docker mit SSL/TLS ab.

Um mehrere Docker-Hosts zu einem Cluster zusammenzufügen, muss auf allen Hosts der von Docker bereitgestellte Container „swarm“ starten. Er enthält den Docker-Daemon und lässt sich mit den üblichen Docker-Tools ansprechen. Swarm verteilt dann die Container der Anwendung über die Knoten.

Die Hosts können sich in unterschiedlichen Netzen befinden; Overlay-Netze sorgen dafür, dass sie sich finden. Die Knoten benötigen einen gemeinsamen Key-Value-Store (Etcd, Zookeeper oder Consul), der als Discovery-Dienst fungiert. Einer der Knoten muss über die entsprechende Option von Machine als Swarm-Master ausgezeichnet werden; über ihn spricht man den komplet-

Merkzettel-App im Docker-Swarm

Im Beispiel sind ein Container mit dem Redis-Master, zwei Container mit Redis-Slaves und drei Container mit dem Web-Frontend über drei virtuelle Maschinen verteilt, die gemeinsam einen Swarm bilden. Der Key-Value-Store Consul sorgt für das Service Discovery.



ten Cluster an. Der Kasten rechts erklärt die Installation und Benutzung von Machine und das Aufsetzen eines Docker-Clusters.

Google Kubernetes

Die Docker-Tools überlassen dem Cluster-Verwalter noch einiges an Handarbeit. Einen Schritt weiter in Sachen Automatisierung geht Kubernetes. Die Google-Software kümmert sich um die Verteilung der Container auf die Hosts des Clusters, den Ersatz ausgefallener und den Start zusätzlicher Container, die Gruppierung zusammengehöriger Container sowie das Load-Balancing. Aus dem umfassenden Ansatz ergeben sich nützliche Eigenschaften für den Produktivbetrieb. So kann Kubernetes neue Versionen von Anwendungskomponenten als Rolling Updates im laufenden Betrieb einspielen – und ein Rollback zur alten Version durchführen, wenn etwas nicht funktioniert.

Man merkt der Open-Source-Software an, dass die Entwickler im Vorfeld über ihre Architektur nachgedacht haben. Daher überrascht es nicht, dass sich Kubernetes gut für den Einsatz in größeren Cluster-Umgebungen eignet; unser Beispiel nutzt nur einige grundlegende Funktionen. Trotzdem lohnt das Einarbeiten in die Architektur der flexiblen, beständig weiterentwickelten Software auch bei überschaubaren Anwendungen – nicht zuletzt wegen der großen Community.

Pods, Controller, Services

Kubernetes arbeitet mit drei zentralen Abstraktionen: Pods, Replication Controller und Services. Ein Pod enthält mindestens einen, oft aber auch mehrere Container, die direkt über die Localhost-Adresse kommunizieren können und aus dem gleichen Ressourcenpool (CPU, RAM, I/O) bedient werden. Durch das Hantieren mit Pods statt Containern tragen die Entwickler dem Umstand Rechnung, dass in verteilten Anwendungen Container oft direkt gekoppelt sind, etwa ein Webserver, der mit einem Cache betrieben wird. In der Merkzettel-App läuft jeder Container in einem eigenen Pod, denkbar wäre jedoch auch, den Redis-Slave und das Frontend in einen gemeinsamen Pod zu stecken.

Einem Pod kann man eine beliebige Zahl von Labels wie „frontend“, „backend“ oder „v5“ anheften. Was wie eine Erinnerungshilfe für schusselige Admins aussieht, entpuppt sich als mächtiges Hilfsmittel zur Steuerung des Kubernetes-Clusters.

Um Ausfälle zu kompensieren oder um Last auf mehrere Instanzen zu verteilen, laufen in einem Cluster mehrere gleichartige Pods. Wegen sich ändernder Nachfrage oder Abstürzen sind im Betrieb gelegentlich neue Pods zu starten oder überzählige zu beenden. Ein Replication Controller sorgt dafür, dass immer die gewünschte Zahl von Pods läuft. Den Sollwert gibt man für einen Satz von Labels an, indem man etwa sechs Instanzen von Pods mit den Labels „frontend“ und

Container verteilen mit Machine

Machine lässt sich wie Compose als Binary für verschiedene Plattformen von github.com/docker/machine/releases herunterladen und installieren. Das folgende Beispiel richtet mehrere Docker-Hosts in VirtualBox-VMs ein, die gemeinsam einen Cluster bilden. Zunächst benötigt man eine VM für den Key-Value-Store Consul:

```
docker-machine create  -d virtualbox cluster-store
docker $(docker-machine config cluster-store) run -d  -p "8500:8500" -h "consul" progrium/consul ⤸
                                                                    -server -bootstrap -ui-dir /ui
```

Die Umgebungsvariable OPTIONS speichert die Konfiguration für die weiteren Hosts:

```
OPTIONS="-d virtualbox --swarm --swarm-discovery=consul://$(docker-machine ip cluster-store):8500 ⤸
                                    --engine-opt=cluster-store=consul://$(docker-machine ip cluster-store):8500 ⤸
                                                                    --engine-opt=cluster-advertise=eth1:0"
```

Mit dieser Konfiguration lassen sich drei weitere Hosts starten. Der erste fungiert als Swarm-Master; über ihn läuft später sämtliche Kommunikation, die den Cluster im Ganzen betrifft.

```
docker-machine create ${OPTIONS} --swarm-master swarm-master
docker-machine create ${OPTIONS} swarm-node-00
docker-machine create ${OPTIONS} swarm-node-01
```

Der Befehl docker-machine ls zeigt die laufenden VMs an. Die Anweisung

```
eval $(docker-machine env --swarm swarm-master)
```

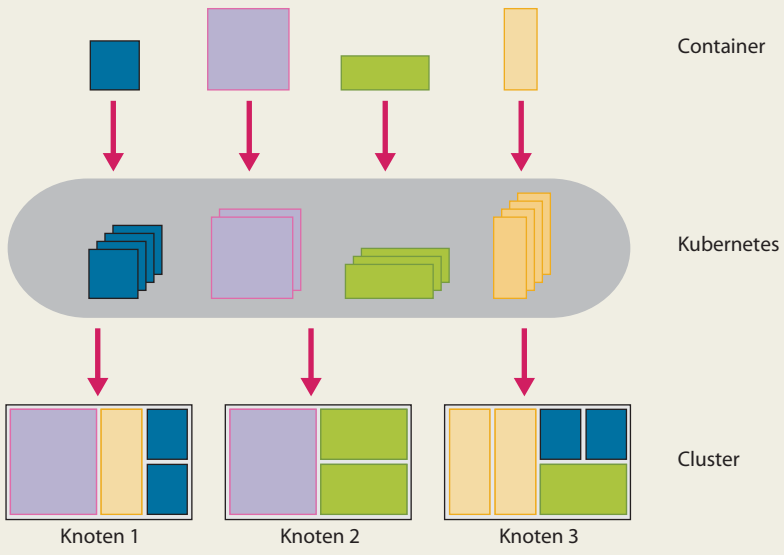
setzt eine Reihe von Umgebungsvariablen, die dafür sorgen, dass die Docker-Tools alle Aktionen statt über die lokale Docker-Engine nun auf dem Swarm-Master ausführen. Jetzt zeigt docker ps die Container an, die im Swarm-Cluster laufen. Nach diesen Vorbereitungen startet

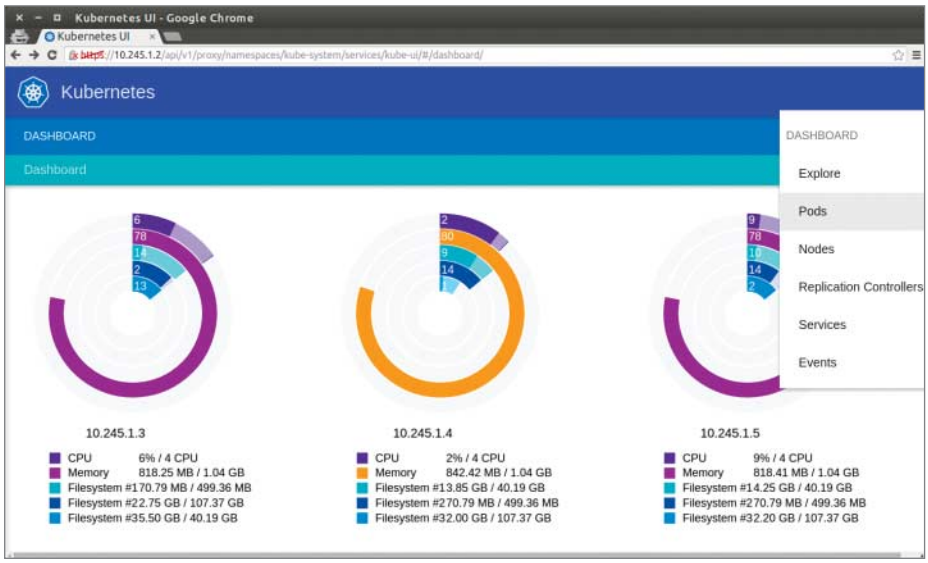
```
docker-compose up -d
```

die Merkzettel-App verteilt auf den angelegten drei VMs. Swarm entscheidet dabei selbstständig, auf welchem Host es welchen Container aktiviert. Der Admin kann dem Cluster jedoch mit dem Befehl docker swarm manage eine Strategie vorgeben. Die Standard-Strategie „spread“ sorgt für eine gleichmäßige Verteilung über die Hosts.

Kubernetes: Orchestrierung im großen Stil

Kubernetes verteilt Container auf einen Cluster und sorgt dabei auch gleich für die Replikation der Dienste.





Über ein Web-Interface stellt Kubernetes unter anderem Informationen zur Ressourcenauslastung bereit.

„v1“ startet. Ab der Kubernetes-Version 1.1 vom Herbst 2015 lässt sich die Zahl der Pods per Autoscaling auch über Leistungsparameter wie die Last oder den Speicherverbrauch steuern.

Services schließlich ermöglichen den Zugriff auf Pods, ohne wissen zu müssen, wo sie genau laufen – das ändert sich ja dynamisch. Welche Pods ein Service bedient, legt der Admin über die Pod-Labels fest. Services dienen sowohl zur Kommunikation innerhalb des Clusters als auch zum Veröffentlichen von Diensten außerhalb des Clusters. Da ein Service auf die Pods mehrerer Replication Controller verweisen kann, lassen sich darüber sehr einfach Szenarien realisieren, bei denen beispielsweise einige Anfragen an eine neuere Programmversion geschickt oder zwei Varianten in einem A/B-Test miteinander verglichen werden.

Endpoints, eine Kombination aus IP-Adresse und Port des im Pod laufenden Dienstes, ermöglichen den Zugriff auf einen Service. Neue Services trägt Kubernetes in einen internen DNS-Server ein, der in einem eigenen Pod läuft. Nun können alle anderen Pods auf ihn über DNS zugreifen.

Innenansicht

Ein Kubernetes-Cluster besteht aus mindestens einem Master und mehreren Knoten. Der Kubernetes-Master stellt drei Komponenten bereit: Der API-Server als Schnittstelle zwischen Benutzer und Cluster nimmt Befehle über REST-Aufrufe oder das Kommandozeilentool kubectl entgegen. Der Scheduler verteilt Pods auf Knoten. Der etwas unglücklich bezeichnete Controller-Manager kümmert sich um die restlichen Aufgaben im Cluster, so verwaltet er die Endpoints zum Zugriff auf den Cluster.

Auf jedem Knoten verwaltet ein sogenanntes Kubelet die Pods und deren Abhängigkeiten wie Container, Docker-Images und Ressourcen. Der Kube-Proxy implementiert eine NAT-ähnliche Funktion und schreibt IP-Adressen auf die eigentlichen Pod-Adressen um. Zudem sorgt er für die Kommunikation zwischen den Pods und implementiert das Load-Balancing. Der cAdvisor schließlich stellt Statistiken zur Ressourcennutzung bereit.

Kubernetes nutzt Etcd als verteilten Speicher für Konfigurationsinformationen und legt ein Overlay-Netzwerk über die bestehende Netzwerkinfrastruktur. Dazu wird ein Tool wie Flannel, Weave oder Calico genutzt. Mit dem Overlay-Netzwerk stellt Kubernetes sicher, dass es keine Konflikte zwischen den Portfreigaben der Container gibt.

Das Aufsetzen des Masters mit seinen diversen Diensten, das Aktivieren des Overlay-Networks sowie das Provisionieren der Nodes sind recht komplexe Aufgaben, die ein mitgeliefertes Skript namens kube-up.sh übernimmt. Der Textkasten links beschreibt, was zu tun ist.

Auf Skalierung ausgelegt

Services und Replication Controller für die Komponenten der eigenen Anwendung definiert man über JSON- oder YAML-Dateien. Die JSON-Dateien der Merkzettel-App finden Sie im Unterverzeichnis 2-kubernetes des Beispiels. Der Start erfolgt über das Kommandozeilentool Kubectl oder einen Aufruf des REST-API:

```
kubectl create -f redis-master-controller.json
```

Das Kommando `kubectl get pods` prüft, ob der Redis-Master-Pod erfolgreich gestartet ist. Es gibt für jeden Pod seinen Namen, die Anzahl der laufenden Container, den Status („Running“ bei einem erfolgreichen Start), die Zahl der Restarts und die Laufzeit aus. Laufen mehrere Container in einem Pod, muss man sich den Pod-Status genauer anschauen, da die Container unterschiedliche Zustände haben können. Damit die nachfolgenden Pods den Redis-Master-Pod finden, startet man mit

```
kubectl create -f redis-master-service.json
```

den dazugehörigen Service.

Nun kann man analog einen Replication Controller für die Redis-Slaves, im Beispiel mit zwei Pods, und einen Service dazu starten. Schließlich ist der Replication Controller für das Web-Frontend an der Reihe. Die Datei todo-app-controller.json legt fest, dass drei Pods mit dem Frontend starten sollen.

Kubernetes: Host-übergreifend orchestrieren

Das Beispiel benötigt Vagrant, ein Ruby-Tool zum Anlegen und Verwalten von virtuellen Maschinen, in Version 1.6.2 oder neuer; Sie finden es im Repository Ihrer Distribution oder auf vagrantup.com. Als Hypervisor kommt VirtualBox zum Einsatz. Als dieser Artikel entstand, war die Kubernetes-Version 1.1.7 aktuell. Sie finden sie auf github.com/kubernetes/kubernetes/releases.

Entpacken Sie das Tar-Archiv, wechseln Sie in das dabei angelegte Verzeichnis kubernetes und starten Sie den Cluster nach dem Setzen einiger Umgebungsvariablen:

```
export KUBERNETES_PROVIDER=vagrant
export NUM_NODES=3
export KUBERNETES_MEMORY=1024
export MASTER_USER=appuser
export MASTER_PASSWD=k8s-gEhEIm
./cluster/kube-up.sh
```

Der Befehl

```
./cluster/kubectl.sh get nodes
```

prüft, ob alle drei Knoten erfolgreich gestartet wurden; `kubectl.sh cluster-info` gibt Informationen über den Kubernetes-Master und den Cluster insgesamt aus.

Mit dem Kommando `kubectl get rc` lässt sich prüfen, ob alle drei Replication Controller laufen. Die Ausgabe enthält den Namen des Replication Controller, das Docker-Image und die gewünschte Zahl der Pods. Die Spalte „Selector“ zeigt die Labels des Pods an, anhand dessen der Replication Controller prüft, ob die gewünschte Zahl an Pods im Cluster läuft.

Der Service für die Frontend-Pods ist vom Typ „ClusterIP“, was auch der Default ist. Solche Services sind über den API-Server erreichbar, der auf dem Master läuft. Anwender benötigen zum Zugriff die beim Start des Clusters in Umgebungsvariablen hinterlegten User-Namen und das passende Passwort. Soll der Service ohne User-Namen und Passwort erreichbar sein, muss ein anderer Service-Typ wie „LoadBalancer“ oder „NodePort“ verwendet werden.

Die Befehle

```
kubectl get services
kubectl get endpoints
```

listen die laufenden Services und ihre Endpoints auf. Über die Endpoints kann man direkt auf die drei Instanzen der App zugreifen, sofern man diesen eine öffentliche IP-Adresse zugewiesen hat. Alternativ erreicht man die To-do-Liste über den Service „kubernetes“, der den ganzen Cluster repräsentiert. Über dessen Endpoint greift man unter der Adresse api/v1/proxy/namespaces/default/services/todo-app/ auf einen Proxy zu, der nach dem Round-Robin-Prinzip eine Verbindung zu einem Frontend-Pod der laufenden To-do-Liste liefert.

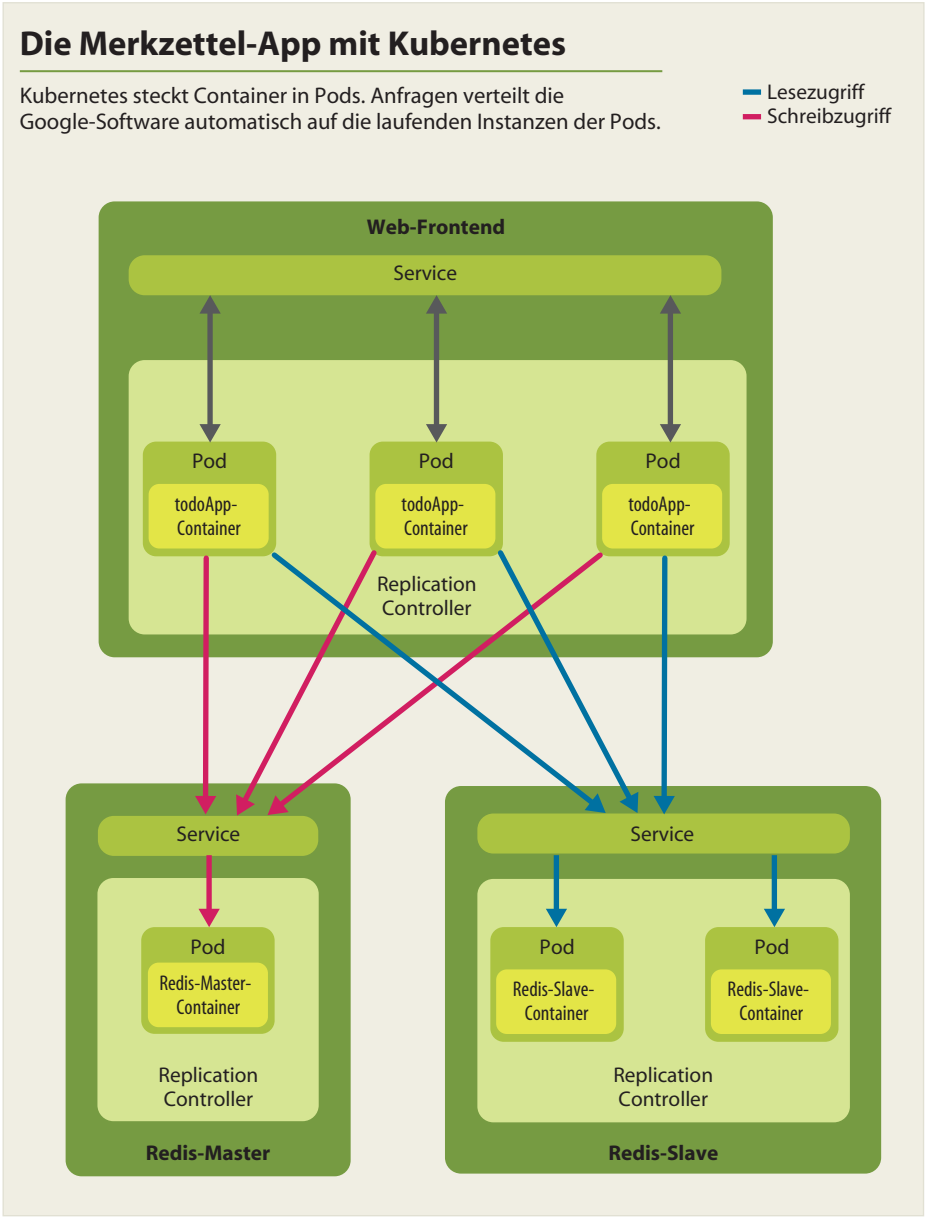
Der Vorteil der Replication Controller wird beim Skalieren deutlich: Möchte man zu den drei bestehenden Frontend-Pods zwei neue Pod-Instanzen hinzufügen, um verstärkte Anfragen an den Service zu bedienen, erhöht der Befehl

```
kubectl scale --replicas=5 rc todo-app
```

die Zahl der Frontends auf fünf, die direkt über den Service verfügbar sind. Das setzt natürlich voraus, dass noch genügend Ressourcen im Cluster vorhanden sind.

Die Merkzettel-App mit Kubernetes

Kubernetes steckt Container in Pods. Anfragen verteilt die Google-Software automatisch auf die laufenden Instanzen der Pods.



Besser orchestrieren

Für viele Anwendungen ist es sinnvoll, sie in Teilkomponenten zu zerlegen, diese in Container zu packen und bei Bedarf auf mehrere Hosts zu verteilen. Wer sich nicht in einem Verhau selbst geschriebener Shell-Skripte und ständiger Handarbeit verlieren will, benötigt schnell ein Orchestrierungs-Framework, um die Container und ihr Zusammenspiel zu beschreiben und zu steuern. Die verfügbaren Lösungen erfüllen allerdings unterschiedliche Ansprüche.

Attraktiv an dem Dreiklang aus Compose, Machine und Swarm ist die Herkunft aus dem Hause Docker, dem Hersteller der am weitesten verbreiteten Containerplattform. Damit profitieren Anwender von der engen Integration der Tools und erzielen schnelle Erfolge. Das erst kürzlich hinzugekommene Docker-Netzwerk bringt erhebliche Fortschritte für komplexere Installationen. Allerdings ist der Docker-Werkzeugkasten noch nicht ganz komplett, sodass der Admin an einigen Stellen Hand anlegen muss. Bei der rasanten Entwicklung kann sich das jedoch schnell ändern.

Der umfassende Ansatz von Kubernetes erfordert eine komplexere Architektur und damit eine längere Einarbeitungszeit. Dafür punktet die Software mit einer breiten Unterstützung auch durch andere Hersteller wie beispielsweise Red Hat, der es in sein Project Atomic integriert hat. Hauptentwickler Google preist natürlich vor allem seine Container Engine in der Google Cloud Platform an, aber auch andere Unternehmen bieten vorbereitete Kubernetes-Installationen zum Betrieb auf eigenen Servern oder als gehostete Cloud an. Langfristig hat Kubernetes gute Chancen, sich als Standard durchzusetzen, nicht zuletzt wegen der breiten Unterstützung durch namhafte Unternehmen.

Beide Frameworks befinden sich an der Schwelle zur Produktionsreife, Updates können jedoch immer noch inkompatible Änderungen bei Schnittstellen und Funktionen bringen. Zudem gibt es ein breites, unübersichtliches Angebot an alternativen Tools. In gar nicht so kleinen Nischen lauern bereits die Cluster-Software Apache Mesos mit dem Cluster-Init-System Marathon und die Container-Plattformen Deis und Rancher.

Container, Orchestrierung und Service Discovery setzen vieles von dem um, was für einen robusten Betrieb skalierbarer Anwendungen nötig ist. Letztlich entscheidet aber die Architektur der eigenen Anwendungen über den Erfolg einer Orchestrierungsplattform: Nur wenn die Anwendung ausfalltolerant und möglichst zustandslos arbeitet, profitiert sie von Containerisierung und Orchestrierung. (odi@ct.de) c't