Peter Roßbach
(peter.rossbach@bee42.com)
ist Infracoder, Systemarchitekt und Berater vieler Websysteme. Sein besonderes Interesse gilt dem Design und der Entwicklung von komplexen Infrastrukturen. Er ist Apache Tomcat-Committer und Apache-Member. Mit der bee42 solutions GmbH realisiert er Infrastrukturprodukte und bieten Schulungen auf der Grundlage des Docker-Ökosystems, aktuellen Web-Technologien, NoSQL-Datenbanken und Cloud-Plattformen an.

Docker-Container-Orchestrierung

Das Docker-Ökosystem entwickelt sich enorm schnell. Eine Aufteilung der eigenen Software in Microservices und das Deployment auf verschiedenen Ablaufplattformen erfordern neue Strategien für den Betrieb und die Entwicklung von Software. Erzeugen, Betreiben und Ändern von vielen Maschinen und Containern auf dem eigenen Notebook, im Datacenter oder in der Cloud stellen uns vor gewaltige neue Herausforderungen. Unsere Produkte werden ständig erweitert und an verschiedene Nutzungen angepasst. Das Docker-Ökosystem bietet hier vielversprechende Orchestrierungswerkzeuge, um automatische Erzeugung, Skalierung und Deployment elegant zu realisieren. Der Artikel stellt den praktischen Nutzen der neuesten Docker-Projekte „machine“, „swarm“ und „compose“ vor.

DevOps

Systeme bereitzustellen und mit Software zu bestücken, ist eine oft als lästig empfundene Tätigkeit. Entwicklung und Betrieb nutzen für diese Aufgabe meist sehr unterschiedliche Werkzeuge und Methoden.

Das Ziel scheint, die Verzögerung von Auslieferungen möglichst auszudehnen, indem die Teams noch in verschiedene Abteilungen und Firmen aufgeteilt werden. Wenn die Aufgaben noch mehr Verzögerung benötigen, kann man die Teams noch mit Outsourcing, Arbeiten in verschiedenen Zeitzonen und mit unterschiedlichen Personen aus verschiedenen Kulturregionen ausbremsen. Eine Kultur ausgeprägter Meetings ist natürlich auf allen Ebenen dieses Prozesses förderlich. Wer in einem solchen Umfeld Software entwickelt, bereitstellt und betreibt, hat es nicht leicht, neue Wege zu gehen.

Die Erwartungshaltung der Kunden und Auftraggeber ist allerdings, dass wir schneller Software liefern, die Qualität steigt und die Kosten gesenkt werden. Änderungen sollen direkt nach der Entwicklung in die Produktion geliefert werden. Direkt am Arbeitsplatz des Entwicklers entsteht das fertige Produkt und das Entwicklungsteam ist direkt für die Auslieferung und den Betrieb seiner Software verantwortlich. DevOps-Prinzipien und Microservice-Architekturen sollen helfen, diese Anforderungen umzusetzen.

Daraus folgt, dass wir unsere Organisationen und Herstellungsprozesse für dieses Ziel grundsätzlich verändern müssen. Der Druck der Internet-Giganten, der Cloud-Provider und vieler sehr erfolgreicher Internet-Startups der letzten Jahre konfrontieren uns alle mit einer neuen Ära der IT.

Die schnelle und einfache Verfügbarkeit von fast beliebigen IT-Ressourcen und

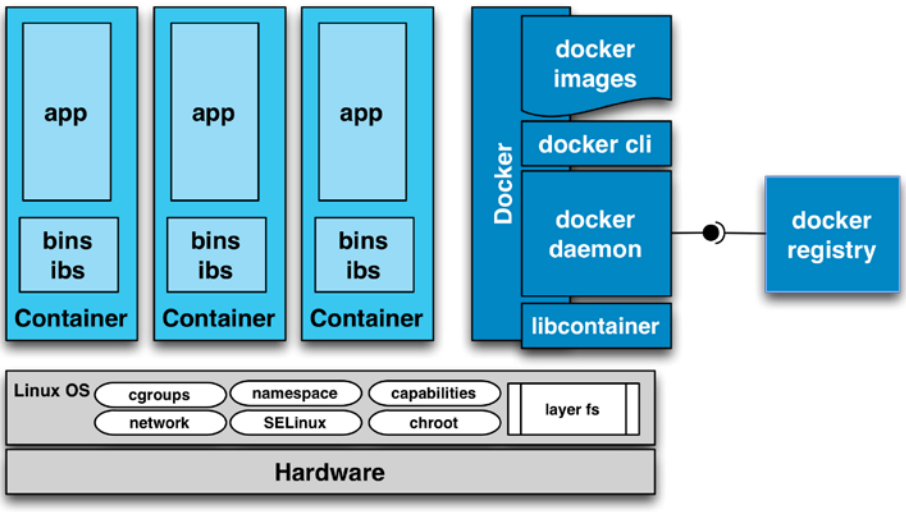

Abb. 1: Docker-Architektur

nutzbaren Services in der Cloud sind hervorragende Möglichkeiten, die Lieferung von Software zu beschleunigen. Alles wird quasi zur Software. Die Zuteilung von CPU, Memory, Netzwerk oder Storage wird beschrieben und von entsprechenden Services bereitgestellt. In dieser Cloud-Welt dauert die Bereitstellung vieler Maschinen in der Regel wenige Minuten und sie kosten Cents pro Stunde. Zusätzlich gibt es jede Menge fertige Services.

Docker

Das Open-Source-Projekt Docker wurde 2013 gestartet, um nun alle unsere Software einfach in Linux-Container zu verpacken, zu verteilen und ablaufen zu lassen. Die Idee von Solomon Hykes, dem Gründer von dotCloud und Docker Inc., hat eine enorme Verbreitung gefunden. Das Docker-Ökosystem wächst unaufhaltsam und die Erfolgsstorys verbreiten sich in einer enormen Geschwindigkeit.

Schon viele Cloud-Provider sind heute in der Lage, Docker-Container zu betreiben und arbeiten an neuen Container-Service-Modellen. In den letzten zwei Jahren haben Red Hat, Google, Amazon, VmWare, Parallels, Mircosoft, IBM und viele andere geholfen, das Projekt voranzutreiben. Das Projekt Docker hat mehr als 750 Maintainer und betreut mittlerweile eine Vielzahl von Teilprojekten. Im Umfeld sprießen fast wöchentlich neue Projekte aus dem Boden. Das Mantra „Build, Ship and Run, Any App, Anywhere“ fasziniert. Im Kern nutzt Docker die Linux-Kernel-Virtualisierung und isoliert die einzelnen Prozesse kontrolliert voneinander (vgl. Abbildung 1).

Das spart Ressourcen und beschleunigt enorm den Start von Prozessen. Wer nun noch eine Microservice-Architektur verwendet und konsequent an einer voll automatischen Delivery-Pipeline arbeitet, kann die Lieferung von Software gewaltig verkürzen.

Damit verschiedene Umgebungen mit der Software bestückt werden können, werden die Images in einer Docker-Registry bereitgestellt. Alleine in den öffentlichen Docker-Registries wie Docker-Hub sind mehr als 45.000 solcher Images schon für alle möglichen Einsatzzwecke vorhanden. Ob Datenbanken, Programmiersprachen, Messages-Queues, Log-Server, Monitoring-Lösungen, Webserver oder Tools, vieles existiert schon. Natürlich können auch eigene Registries genutzt werden.

Mit dieser Docker-Unterstützung ist es wesentlich leichter, neue Software auf ihre Verwendbarkeit zu überprüfen. Die neuen Docker-Teilprojekte „machine“, „swarm“ und „compose“ zielen nun auf die Orchestrierung von Containern auf vielen Maschinen ab (vgl. Abbildung 2).

Docker Machine

Das Projekt „machine“ erleichtert die Bereitstellung von Maschinen auf verschiedenen Clouds, Virtualisierungslösungen und dem eigenen Notebook. Die erste Version Preview Version 0.1.0 existiert seit Februar 2015 [1].

Docker Machine erzeugt auf der jeweiligen Maschine einen Docker-Host. Dieser wird auf der Basis eines vorgefertigten Images des jeweiligen Docker Machine-Drivers mit einem SSH-Zugang erstellt. Dort wird Docker installiert und mit entsprechenden Zertifikaten für die TLS-Kommunikation mit dem Docker Daemon gestartet.

Die Ausstellung der Zertifikate geschieht noch ausschließlich auf dem Rechner, der die Docker-Maschinen kontrolliert. Für die Ansteuerung der diversen Provider müssen entsprechende Credentials bereitstehen. Das Werkzeug Docker Machine ermöglicht die lokale Kontrolle über alle seine Docker-Hosts (vgl. Abbildung 3).

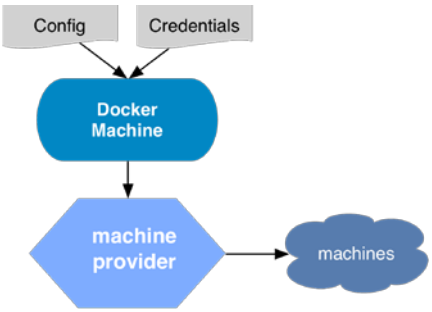

Abb. 3: Docker Machine

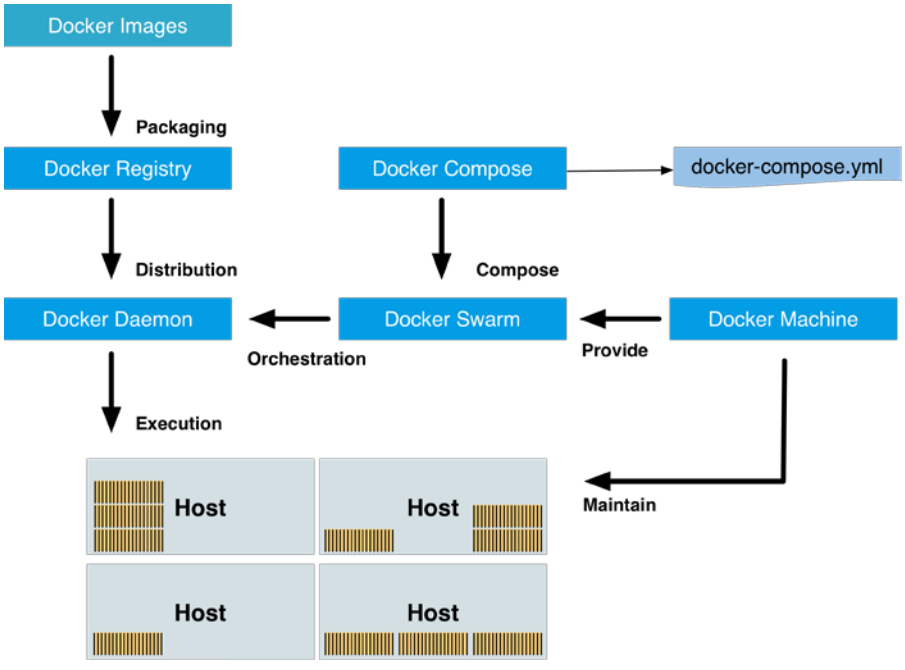

Abb. 2: Docker-Orchestrierung

Das Preview beweist schon sehr eindrucksvoll, wie einfach das Erzeugen und die Bestückung von Software auf Maschinen nun ist. Je nach Provider dauert es zwischen 30 Sekunden und drei Minuten.

Das Ziel der Softwareentwicklung ist, nicht nur Software zu schreiben, sondern Kunden zu haben, die die Software wirklich einsetzen. Um das zu erreichen, muss jeder Softwareentwickler lernen, Software richtig zu betreiben. Wenn die Kunden in Asien Latenzprobleme haben, dann werden in dieser Zone halt die entsprechende Anzahl von Hosts gestartet und später wieder gelöscht.

Das nächste Release von Docker Machine ist auf dem Weg und wird die Dynamik der Maschinen in der Cloud noch besser unterstützen. Aktuell wird ein neues Design diskutiert, das es schneller und unabhängiger erlaubt, Docker Machine zu erweitern. Docker Machine steht auf Linux, MacOS X und Windows zur Verfügung.

Docker Swarm

Eine weitere Orchestrierungsaufgabe ist es, die eigene Software auf verschiedene Maschinen zu verteilen. Dafür bietet das Projekt „swarm“ einen entsprechenden Scheduler an [2].

Docker Swarm ist als eingeschränkter Proxy für den Docker Daemon realisiert. Er kann lokal installiert werden oder lieber gleich als Container gestartet werden. Auf jedem Docker-Host wird ein Agent gestartet, der seine Erreichbarkeit in einer Service-Discovery bekannt gibt (vgl. Abbildung 4). Für Tests kann die Service-Discovery des Docker-Hub direkt genutzt werden. Für den produktiven Einsatz sind die Service Discovery-Lösungen Consul oder Ectd besser geeignet.

Als Erstes wird ein Token für jeden Cluster von Docker Swarm-Maschinen erzeugt. Dann wird beispielsweise mit dem Elastic Cloud Compute (EC2)-Provider die Swarm Master-Maschine erzeugt. Dazu sind neben den korrekten Legitimationen, ein entsprechendes Amazon Machine Image (AMI) der Amazon Web Service (AWS)-Region auszuwählen und ein Virtual Network (VPC) zu erzeugen.

```
$ SWARM_TOKEN=$(docker run swarm
create)
$ AWS_access_key=xyz-key
$ AWS_secret_key=secret-key
$ Docker Machine create --driver amazonec2
\
 --amazonec2-access-key=${AWS_access_
key} \
 --amazonec2-ami=ami-898dd9b9 \
 --amazonec2-instance-type=t2.micro \
 --amazonec2-region=us-west-2 \
 --amazonec2-root-size=16 \
 --amazonec2-secret-key=${AWS_secret_
key} \
 --amazonec2-vpc-id=vpc-2748f942 \
 --amazonec2-zone=a \
 --swarm \
 --swarm-master \
 --swarm-discovery=token://$SWARM_TO-
KEN \
 ec2-swarm-master
```

Nun kann der Schwarm angelegt werden. Aktuell werden dazu noch einige merkwürdige Fehler erzeugt, aber der Schwarm ist fast unabhängig von der Anzahl der Maschinen nach ca. drei Minuten verfügbar. Ein kurzer Test zeigt, dass alle Maschinen dem Swarm Master bekannt sind. Auf jeder Maschine läuft ein Swarm-Agent, der die Adresse seines Docker Daemon in die Service-Discovery einträgt (vgl. Abbildung 4).

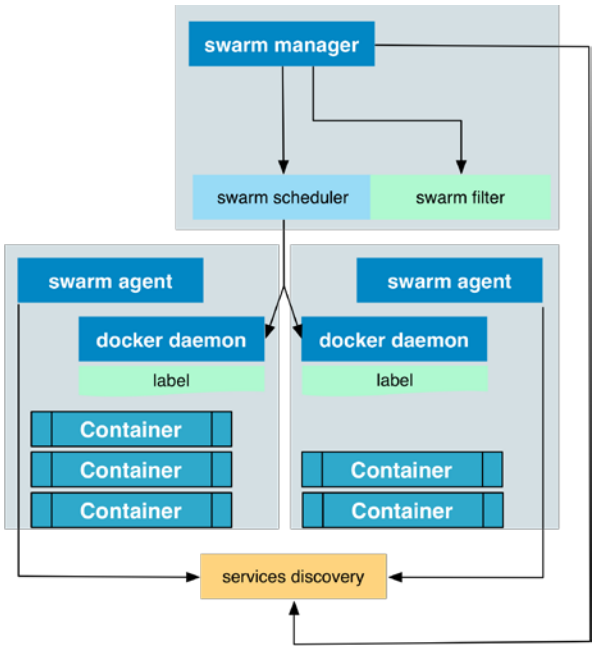

Abb. 4: Docker Swarm

```
$ export NUM_WORKER=3
$ for i in $(seq $NUM_WORKER) ; do
   Docker Machine create --driver amazonec2
\
    --amazonec2-access-key=${AWS_access_
key} \
    --amazonec2-ami=ami-898dd9b9 \
    --amazonec2-instance-type=t2.micro \
    --amazonec2-region=us-west-2 \
    --amazonec2-root-size=16 \
    --amazonec2-secret-key=${AWS_secret_
key} \
    --amazonec2-vpc-id=vpc-2748f942 \
    --amazonec2-zone=a \
    --swarm \
    --swarm-discovery=token://$SWARM_TO-
KEN \
    ec2-swarm-0$i &
 done
$ $(Docker Machine env --swarm ec2-
swarm-master)
$ docker info
Containers: 5
Nodes: 4
 ec2-swarm-master: 52.10.167.59:2376
  └Containers: 2
  └Reserved CPUs: 0 / 1
  └Reserved Memory: 0 B / 992.4 MiB
 ec2-swarm-03: 54.69.230.35:2376
  └Containers: 1
  └Reserved CPUs: 0 / 1
  └Reserved Memory: 0 B / 992.4 MiB
 ec2-swarm-01: 54.69.255.39:2376
  └Containers: 1
  └Reserved CPUs: 0 / 1
  └Reserved Memory: 0 B / 992.4 MiB
 ec2-swarm-02: 54.69.29.90:2376
  └Containers: 1
  └Reserved CPUs: 0 / 1
  └Reserved Memory: 0 B / 992.4 MiB
```

Noch kann das Werkzeug Docker Machine weder den Docker Daemon, noch den Swarm wirklich flexibel konfigurieren oder immer korrekt neu starten. Wenn das notwendig wird, muss dies zurzeit mit eigenen Skripten auf jedem Host erfolgen [7, 8].

Es ist zu berücksichtigen, dass die verschiedenen Provider-Images auf verschiedenen Linux -Distributionen beruhen. Der Swarm-Scheduler besitzt aktuell zwei verschiedene Strategien. Mit binpack wird versucht, die einzelnen Maschinen möglichst auszulasten. Die Strategie random verteilt die Container zufällig auf alle Maschinen. Die Verteilung wird im einfachsten Fall über die Anzahl, den Speicherbedarf oder die Zuordnung von CPU-Shares gesteuert.

Leider werden damit erstmal Container auch auf die Maschine des Swarm-Masters verteilt. Etwas mehr Einfluss auf die Verteilung kann mit der Vereinbarung von Constraints genommen werden.

Dafür werden die Docker Daemons durch entsprechende Label markiert. Es ist einfach, den Maschinen entsprechende Rollen für Projekte oder Services zuzuordnen und somit etwas Kontrolle in die Ver-

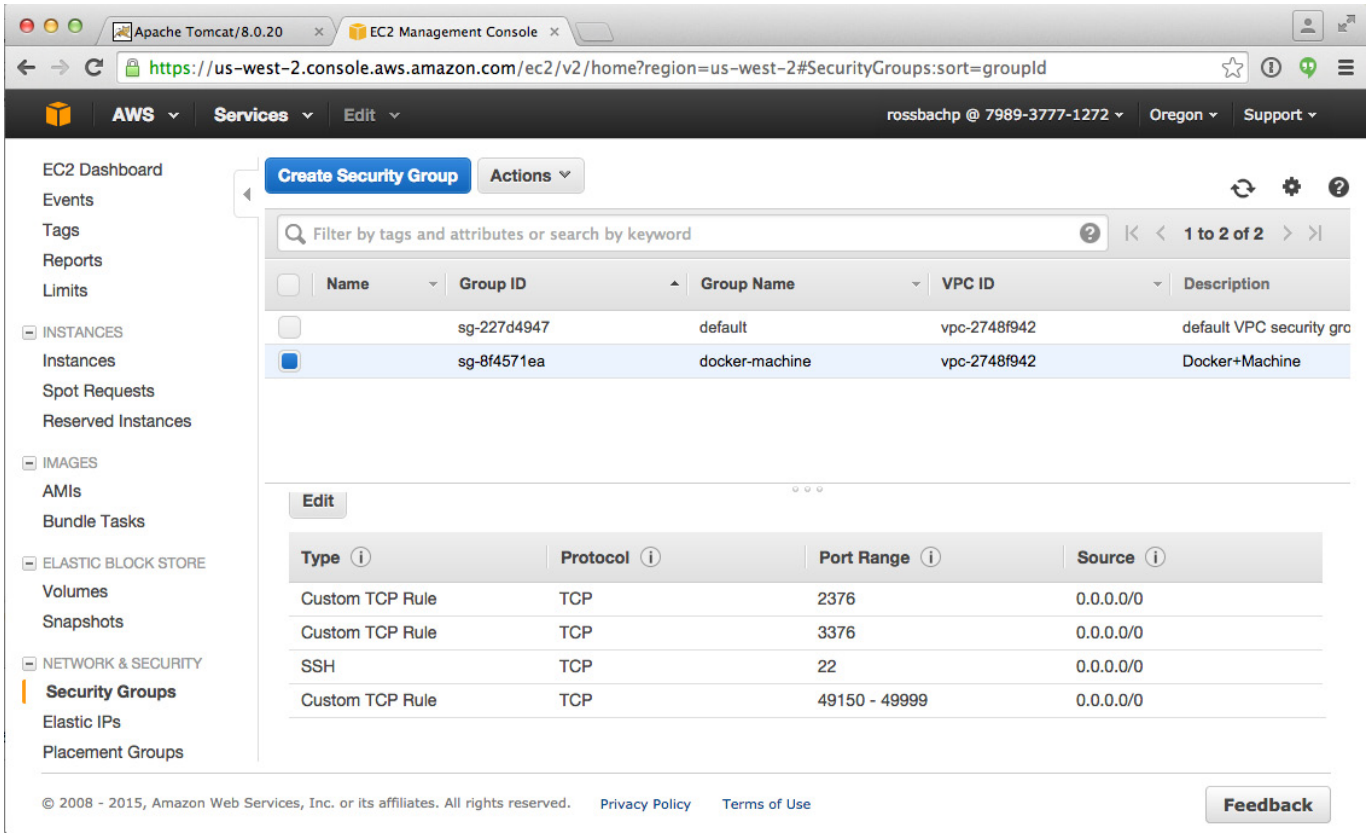

Abb. 5: AWS-Console-Security-Group

teilung einfließen zu lassen. Es ist möglich, einen weiteren Container dort zu installieren, wo ein bestimmtes Image geladen ist oder wo ein anderer Container mit einem bestimmten Namen abläuft.

Damit ist es möglich, Gruppen von Containern auf einer Maschine zu organisieren. Wenn in Docker 1.6 auch die Container Label bekommen, werden noch feinere Steuerungsmöglichkeiten in Swarm ermöglicht [3].

Damit die Services auch extern erreichbar werden, muss in der AWS-Console in der docker machine Security-Group der entsprechende Port-Range eingetragen werden (vgl. Abbildung 5). Normalerweise werden aber die einzelnen Services eher Bestandteil eines elastischen Loadbalancers, der den Zugang auf die einzelnen Container verteilt.

Einen Planer, der Failover oder die Skalierung weiterer Maschinen steuert, gibt es im Docker Orchestration-Toolset noch nicht. Auf der Roadmap steht, dass Integrationen des Mesos Schedulers-Marathon, Kubernetes und CoreOS-Fleet geplant sind. Wer nicht warten will, erzeugt sich vermutlich lieber eine eigene Lösung [9] oder integriert den entsprechenden Load Balancing-Service des jeweiligen Cloud-Providers.

Docker Compose

Aus dem Docker-Werkzeug fig.sh ist nun Docker Compose entstanden [4]. Die Aufgabe von Compose ist das Management einer Gruppe von Docker-Containern. Die Deklaration einer Gruppe wird in der Datei docker-compose.yml abgelegt. Mit dem Werkzeug können nun einzelne Container oder Gruppen erzeugt, gestartet, verwaltet und wie im Beispiel skaliert werden. Die Berücksichtigung von Links zwischen Containern und die Integration von Volumes sind natürlich gegeben.

```
$ mkdir status
$ cat >status/index.jsp <<EOF
hello dockers
EOF
$ cat >Dockerfile <<EOF
FROM busybox
ADD status/index.jsp /usr/local/tomcat/
webapps/status/index.jsp
VOLUME [ „/usr/local/tomcat/webapps/
status“ ]
CMD true
EOF
$ cat >docker-compose.yml <<EOF
status:
  build: .
  environment:
    constraint: rule==tomcat
tomcat:
  image: tomcat:8
  ports:
    - „8080“
  volumes_form:
    - status
  environment:
    constraint: rule==tomcat
    affinity: container==status
EOF
$ Docker Compose up -d
$ Docker Compose scale status=4 tomcat=4
```

Dieses Beispiel klappt leider noch nicht ganz, da Docker Compose noch nicht in der Lage ist, den affinity-Docker Swarm-Filter auf den richtigen Namen des status-Containers zu setzen [19]. Es fehlt noch ein Konzept, Docker-Links über Host-Grenzen hinweg bereitzustellen. Docker Compose kennt noch kein wirkliches Konzept von Container-Gruppen für Swarm.

Eindeutige Namen im gesamten Cluster zu vergeben, ist eine globale Einschränkung und sicherlich keine Eigenschaft, die ein skalierbarer dynamischer Service haben sollte. Hier können die Label, die in Docker 1.6 implementiert sind, helfen, die Gruppen von Containern zu markieren.

Docker Compose ist im produktiven Einsatz und heute schon ein unverzichtbarer Bestandteil vieler Docker-Entwicklungen. Die Komplexität eines Deployments muss unbedingt durch eine verbesserte Konfiguration abgebildet werden. Diskussionen über ein Docker Machine-Konfigurationsformat und die Nutzung von externen Variablensets in Compose werden geführt und erste Vorschläge und Implementierungen existieren.

Mehr Orchestration wagen

Mit den vorgestellten Werkzeugen ist ein erster Schritt zur einfachen Kontrolle von Maschinen und deren vereinfachter Provisionierung umgesetzt. Weitere Schritte sind zur Vereinheitlichung der Netzwerk-Konfiguration von Docker geplant.

Im Docker-Ökosystem sind verschiedene Vorschläge und Lösungen im Einsatz (vgl. Abbildung 6). Die bekanntesten sind sicherlich Pipework, Weave, Flannel, Socketplane und Calico. Docker Inc. hat gerade das Team von Socketplane übernommen und versucht mit dieser Verstärkung das Network-Plugin Proposal umzusetzen. Mit dem Netzwerk-Plugin wird der Zugriff verschiedener Docker-Container auf viele Maschinen ermöglicht und kontrollierbar.

Weitere Herausforderungen sind die Integration von verbesserten Algorithmen zur dynamischen Steuerung der Bereitstellung von Containern und vereinfachten Betriebssystem-Distributionen [5, 6, 16, 17, 18]. Zur Erreichung dieser Ziele wird Docker erweiterbar werden [12, 13, 14].

Fazit

Die Magie des Docker-Ökosystems ist die enorm produktive Community, die fast täglich Verbesserungen erzeugt. Die Docker-Orchestration-Tools sind noch in einem frühen Stadium der Entwicklung, zeigen aber schon erste brauchbare Ergebnisse.

Viele sind sich einig, dass die Vision von Solomon Hykes in der Bereitstellung einheitlicher API's hilfreich ist. Seine Prinzipien für Docker, „API first" und „Batteries included but swappable" haben für die Entwickler von Infrastruktur einen großen Antrieb gezündet. Der Schub neuer Projekte ist ungebrochen.

Die Idee, Software als Container zu bauen, zu verteilen und zu betreiben, ist schon heute erfolgreich und von entscheidender Bedeutung, um schneller Software an die Marktbedürfnisse anzupassen. Die Wahrheit ist aber auch, dass Organisationen, die diese flexibel verteilten Software-Welten nutzen, sich darauf vollständig einlassen müssen. ■

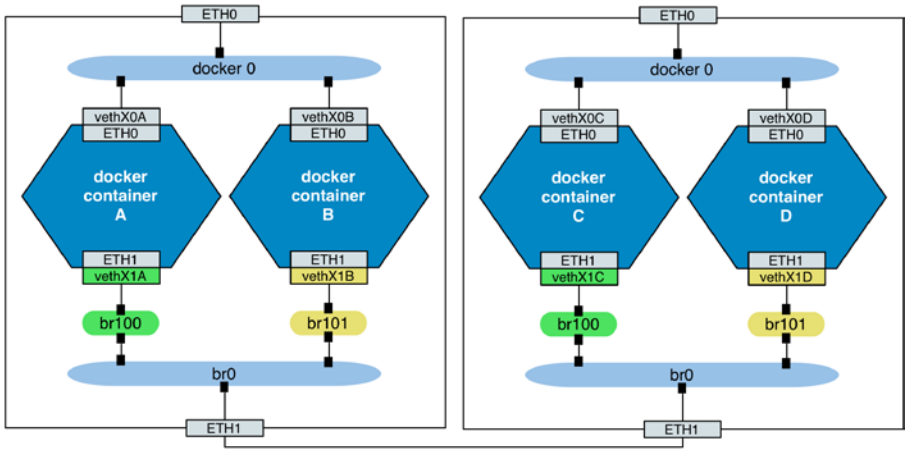

Abb. 6: Docker Network

Aktuelle Einschränkungen der Docker Orchestrierungs-Tools

- Erst im nächsten Docker Machine Release >0.1.0 wird das start/stop mit wechselnden IP's der meisten Cloud-Provider unterstützt [7,8].
- Neue Container-Images können mit Swarm nicht gebaut oder erzeugt werden.
- Attach zu einem Container wird nicht von Swarm unterstützt.
- Die Container Label von Docker 1.6 werden noch nicht als Filter-Kriterien genutzt.
- Aktuell werden von Docker Machine nur OS-Images mit Ubuntu oder Boot2docker unterstützt.
- Verbindung von Swarm und Compose ist noch sehr rudimentär.
- Es fehlt die einfache und unabhängige Erweiterbarkeit der Tools. Aktuell ist die einzige Möglichkeit für eine Ergänzung, einen entsprechend akzeptierte Pull-Requests zu formulieren.

Links

[1] https://github.com/docker/docker
[2] https://github.com/docker/machine
[3] https://github.com/docker/swarm
[4] https://github.com/docker/compose
[5] http://rancher.com/rancher-os/
[6] https://coreos.com/
[7] https://github.com/docker/machine/pull/770
[8] https://github.com/docker/machine/issues/806
[9] https://speakerdeck.com/rossbachp/microxchg2015-docker-tomcat-and-Micro-Services
[10] https://registry.hub.docker.com/u/library/tomcat/
[11] „Amazon ECS Container-Service", Sascha Möllering & Peter Rossbach, Docker-Kolumne 6/15 JavaMagazin http://www.javamagazin.com
[12] https://github.com/ClusterHQ/powerstrip
[13] https://github.com/binocarlos/powerstrip-weave
[14] https://github.com/ClusterHQ/flocker
[15] https://github.com/Metaswitch/calico-docker
[16] http://kubernetes.io/
[17] http://www.projectatomic.io/
[18] http://mesos.apache.org/
[19] https://docs.docker.com/swarm/scheduler/filter/