Containerisierung einer Webanwendung mit Docker und Automatisierung der Infrastruktur mit Ansible

Simon Lipke

Hochschule der Medien
sl110@hdm-stuttgart.de
Matr. Nr.: 30799

Zusammenfassung

Diese Arbeit wurde für die Vorlesung Ultra Large Scale Systems im Wintersemester 15/16 verfasst. Die Vorlesung handelt von hochskalierbaren modernen Infrastrukturen, die bei hochverfügbaren Diensten, wie beispielsweise Twitter oder Facebook, zum Einsatz kommen. Ein wichtiges neues Konzept in diesem Bereich ist die Containerisierung (engl.: "Containerization"[1]) von Anwendungen. Hierbei werden Anwendungen in viele kleine Dienste zerbrochen (sog. "Microservices"[2]) und in Containern verpackt. Jeder einzelne Container beinhaltet ausschließlich die für den Dienst benötigten Abhängigkeiten. Diese Container können dann in einer Produktionsumgebung auf unterschiedliche Server verteilt werden, somit entsteht eine hohe Flexibilität und Saklierbarkeit der einzelnen Dienste.

I. Ziel dieser Arbeit

Das Ziel dieser Arbeit ist das Containerisieren einer einfachen Webanwendung und der anschließende automatisierte Rollout auf einen oder mehrere Produktionsserver. Hierfür wird ein Werkzeugkasten auf Basis von Ansible und Docker entwickelt, mit welchem eine Entwicklungs- und Produktionsumgebung aufgesetzt werden kann. Die Entwicklungsumgebung stellt grundlegende Werkzeuge bereit, um über mehrere Entwickler hinweg die selben Technologie-Versionen zu verwenden. Hier wird als Grundlage Docker in Kombination mit *docker-compose* verwendet, um eine mehrschichtige Umgebung aufzusetzen. Auf der Produktionsumgebung wird schließlich die fertige Version der Webanwendung mit Hilfe von Ansible ausgerollt.

II. Installation

Die Maßnahmen zu dieser Arbeit werden auf einem MacBook mit dem Betriebssystem Mac OS X: El Capitan in der Version 10.11.3 ausgeführt. Zunächst muss die benötigte Software installiert werden.

1. VirtualBox

Um das Nutzen von virtuellen Maschinen zu ermöglichen, muss VirtualBox in der aktuellen Version 5.0.14 installiert werden. Diese kann von der Webseite `https://www.virtualbox.org/wiki/Downloads` heruntergeladen werden. VirtualBox wird benötigt, um in Kombination mit Vagrant zu Testzwecken einen Produktionsserver auf dem lokalen Rechner zu betreiben. Nach dem Verstehen und Konfigurieren der Prozesse kann diese virtuelle Maschine durch einen richtigen Produktionsserver (beispielsweise einen Cloud-Server) ausgetauscht werden.

2. Vagrant

Vagrant wird in der aktuellen Version 1.8.1 genutzt, um die benötigten virtuellen Maschinen zu verwalten. Mit Hilfe von Vagrant können

vorgefertigte Standard-Maschinen zur Eigennutzung aus der Cloud geladen werden. Vagrant nutzt eine Datei mit dem Namen *Vagrantfile*, um die zu importierenden virtuellen Maschinen zu beschreiben. Das Installationsprogramm kann von der Webseite `https://www.vagrantup.com/downloads.html` heruntergeladen werden.

3. Ansible

Die Ansible-Plattform in der Version 2.0.0.2 wird verwendet, um das Aufsetzen ("Provisionieren") von Servern zu automatisieren. In Ansible wird mit Hilfe von sogenannten Playbooks [3] definiert, welche Software auf welchen Servern installiert werden muss. Als Kommunikations-Technologie wird SSH verwendet, weshalb hier kein zusätzlicher Dienst auf den Servern vorinstalliert sein muss. Über Inventories ([4]) kann in Ansible die vollständige Infrastruktur verwaltet werden und schnell beispielsweise ein neuer Produktionsserver aufgesetzt werden. Ansible auf Mac OS X kann mit Hilfe des Python-Paketmanagers *pip* ("Pip Installs Python") installiert werden. Hierfür sind folgende Kommandos auszuführen:

```
sudo easy_install pip
pip install ansible
```

4. Docker

Docker bildet mit seinem Container-Konzept die Grundlage für diese Arbeit. Es wird die aktuelle Version 1.10.2 verwendet. Für die lokale Entwicklung werden die beiden Werkzeuge *docker-compose* in der Version 1.6.0 und *docker-machine* in der Version 0.6.0 verwendet. Zur Installation aller Docker-Werkzeuge in der aktuellen Version (inklusive der *docker-engine*) kann die Docker Toolbox von der Webseite `https://www.docker.com/products/docker-toolbox` heruntergeladen und installiert werden.

5. Symfony2

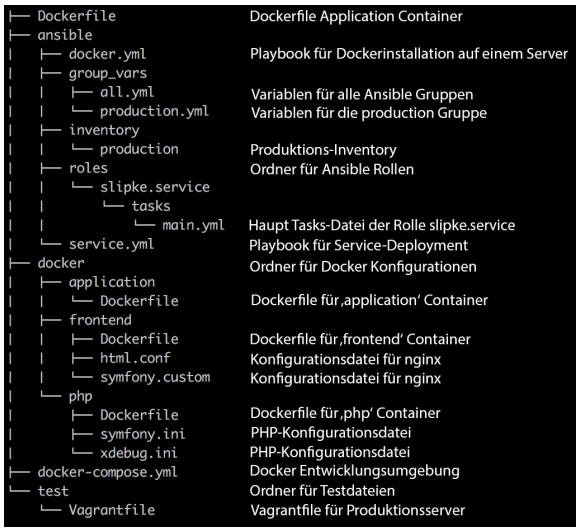
Symfony2 ist ein flexibles und einfach erweiterbares Open-Source PHP-Framework mit einer großen Community. Dieses Framework dient dieser Arbeit als Grundlage für eine einfache 'Hello-World'-Anwendung. Das Framework ist in der aktuellen Version 3.0.2 auf der Webseite `http://symfony.com/download` herunterladbar. Zur Installation der Anwendung wird das *symfony* CLI-Tool genutzt, das wie folgt auf dem lokalen Rechner installiert werden kann:

```
sudo curl -LsS \
  https://symfony.com/installer \
  -o /usr/local/bin/symfony
sudo chmod a+x /usr/local/bin/symfony
```

III. Das Projekt

Der Quellcode für diese Arbeit kann aus dem GitHub-Repository `https://github.com/slipke/ansible-docker-paper.git` ausgecheckt werden. Das Projekt ist wie in Bild 1 beschrieben aufgebaut.

Abbildung 1: *Ordner-Struktur*



Zunächst müssen die Abhängigkeiten für die Webanwendung heruntergeladen werden. Composer liest die Datei *code/composer.json* und

lädt die benötigten Abhängigkeiten in den *code/vendor/* Ordner. Dies geschieht mit folgenden Befehlen:

```
cd code/
./composer.phar install \
  --no-interaction
cd ../
```

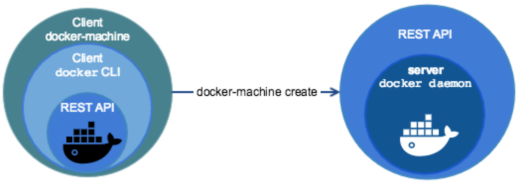
IV. Lokale Entwicklung

Der erste Schritt ist das Aufsetzen der Entwicklungsumgebung, mit welcher an der Webanwendung gearbeitet werden kann. Diese Umgebung basiert auf Docker-Containern. Hierbei ist es wichtig, dass jeder Entwickler exakt dieselbe Umgebung verwendet. Im besten Fall gleicht die Entwicklungsumgebung auch der finalen Produktionsumgebung, so werden später Fehler durch unterschiedliche Technologie-Versionen vermieden. Jeder Entwickler nutzt dieselben einfachen Kommandos, um seine Entwicklungsumgebung aufzusetzen.

1. Docker Machine

Bevor die Entwicklungsumgebung gestartet werden kann, muss das Tool *docker-machine* ([5]) konfiguriert werden. Die Docker Engine unterstützt den Kernel von Mac OS X nicht nativ, weshalb *docker-machine* genutzt wird, um ein sehr kleines Linux-basiertes Image in VirtualBox hochzufahren, auf welchem die Docker-Container laufen werden. Bild 2 zeigt einen einfachen Überblick.

Abbildung 2: *docker-machine, Quelle: [5]*



Um das benötigte neue Image hochzufahren und die Umgebungsvariablen anzupassen, müssen folgende Befehle ausgeführt werden:

```
docker-machine create paper \
  --driver virtualbox
eval $(docker-machine env paper)
```

2. Starten der Umgebung

Um die Umgebung zu starten, muss der Entwickler den nachfolgenden Befehl im Hauptverzeichnis ausführen:

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

Dieser Befehl liest die Datei *docker-compose.yml* und baut die, in der Datei beschriebene, Infrastruktur auf. Er startet die benötigten Container mit den Namen *application*, *php* und *frontend* und verbindet diese innerhalb eines eigenen Netzwerks. Die einzelnen Dockerfiles liegen im jeweiligen *docker* Unterordner. Die 'Hello-World'-Anwendung kann nun unter `http://192.168.99.100` im Browser aufgerufen werden.

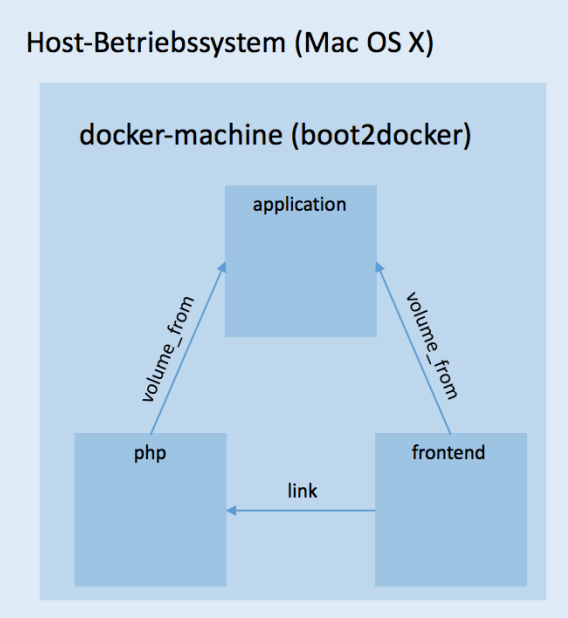
3. Aufbau

Der Container *application* (definiert in *docker/application/Dockerfile*) nutzt das Basis-Image *ubuntu:14.04* vom Docker Hub und installiert einige zusätzliche Pakete aus dem Ubuntu-Repository, wie beispielsweise *uglifyjs* und den SASS-Compiler. Dann wird in der *docker-compose.yml* die Webanwendung im Ordner *code* in den Docker-Container eingehängt, mit Hilfe der *VOLUME*-Anweisung. So werden die getätigten lokalen Änderungen an den einzelnen Dateien direkt in den Container synchronisiert, und sie müssen nicht bei jeder Änderung händisch hochgeladen werden.

Der *php*-Container nutzt das Basis-Image *php:5.5-fpm* vom Docker Hub. Im Dockerfile (*docker/php/Dockerfile*) werden einige zusätzliche PHP-Erweiterungen installiert, um Dinge wie beispielsweise Debuggen zu ermöglichen. In der Datei *docker-compose.yml* werden dann alle eingehängten Pfade aus dem Container *application* ebenfalls in den Container *php* mit eingehängt, insbesondere der Pfad */var/www/symfony*, in welchem die Webanwendung liegt.

Der Container *frontend* basiert auf dem Image *nginx* und teilt ebenfalls alle eingehängten Volumes aus dem *application*-Container. Im Dockerfile (*docker/frontend/Dockerfile*) werden lediglich einige kleinere Konfigurationen für den Webserver nginx vorgenommen. Zusätzlich werden die Ports 80 und 443 nach außen hin freigegeben, um den Zugriff auf die Webanwendung zu ermöglichen.

Abbildung 3: *Aufbau*



4. Performance

Ein Problem bei der Entwicklung mit Docker, Docker Machine und VirtualBox unter Mac OS X ist die Performance. Bei der lokalen Entwicklung ist im Normalfall aufgrund der physikalischen Nähe der Geräte die Reaktionszeit der Webanwendung sehr niedrig. VirtualBox nutzt zum Teilen von Ordnern zwischen der Gast-Maschine und dem Host-Maschine den Treiber *vboxsf* ("Virtual Box Shared Folders"), welcher sehr große Performance-Einbußen mit sich bringt, wie in Artikel [6] beschrieben.

Eine mögliche Lösung für dieses Problem unter Mac OS X ist die Nutzung des Netzwerk-Protokolls *NFS* anstelle von *vboxfs*. Hierzu muss das Docker Machine Image angepasst werden, was zum Beispiel mit Hilfe des Bash-Skriptes aus dem Github-Repository `https://github.com/adlogix/docker-machine-nfs` erledigt werden kann.

V. Produktionsumgebung

Um die Produktionsumgebung aufzusetzen kommen Ansible und Vagrant ins Spiel. Zum Test wird als Produtkionsserver eine virtuelle Maschine mit dem Betriebssystem Ubuntu 14.04 verwendet. Das Herunterladen und Importieren des Servers erfolgt mit Hilfe von Vagrant und der Datei *test/Vagrantfile*. Um die Maschine zu starten werden folgende Befehle nacheinander ausgeführt:

```
cd test/
vagrant up
cd ../
```

Vagrant liest die Datei *Vagrantfile* aus, lädt das entsprechende Image herunter und importiert selbiges in VirtualBox. Nach erfolgreichem Kopieren und Hochfahren ist die Maschine unter der IP-Adresse 192.168.33.201 erreichbar. Um auf dieser virtuellen Maschine die zum Ausführen von Docker-Containern benötigte Software zu installieren, muss zunächst die entsprechende Ansible Rolle installiert werden. Hierzu wird im Root-Ordner folgender Befehl ausgeführt:

```
ansible-galaxy install \
  angstwad.docker_ubuntu \
  -p ansible/roles/
```

Dieser Befehl installiert die Rolle *angstwad.docker_ubuntu* in den Ordner *ansible/roles*. Die Rolle kann nun von den Playbooks verwendet werden. Als nächstes wird mit der Kommandozeilenanwendung *ansible-playbook* das Playbook *ansible/docker.yml* ausgeführt:

```
ssh-keyscan -t rsa 192.168.33.201 \
  >> ~/.ssh/known_hosts
ansible-playbook ansible/docker.yml \
  -i ansible/inventory/production
```

Der Befehl liest das Ansible Inventory unter dem Pfad *ansible/inventory/production* aus, in dem die IP-Adresse des Produktionsservers hinterlegt ist. Daraufhin wird das Playbook *ansible/docker.yml* ausgeführt, welches die zuvor installierte Rolle *angstwad.docker_ubuntu* ausführt und das benötigte Paket *docker-engine* mit seinen Abhängigkeiten installiert. Die Rolle kann bei Bedarf über Variablen in der Datei *ansible/group_vars/all.yml* konfiguriert werden ([7]).

1. Erstellen von Container Images

Um die Webanwendung in die Liveumgebung auszurollen, müssen die Container zunächst auf dem lokalen Rechner erstellt werden. Danach werden sie in eine Docker Registry ([11]) hochgeladen. Für die Produktionsumgebung wird derselbe Aufbau wie in der Testumgebung verwendet:

- Ein Container *application*, in welchem der Code Webanwendung liegt
- Ein *php* Container, welcher die Ausführung von PHP übernimmt
- Ein *nginx* Container, welcher den Webserver darstellt und die Ports 80 und 443 nach außen hin öffnet

Der Container *php* ([12]) kann direkt aus dem Docker Hub verwendet werden. Der Container *nginx* ([13]) benötigt einige kleine Anpassungen. Die wichtigste Anpassung ist die Konfigurationsdatei für den Webserver. Da die Änderung der Entwicklungsumgebung ähnelt, wird für diese Arbeit beispielhaft derselbe Container wie in der Entwicklungsumgebung verwendet. Dieser wird zunächst erstellt und dann in ein Repository ([14]) im öffentlichen Docker Hub hochgeladen. Bevor der Container erstellt und hochgeladen werden kann, müssen einmalig die Login-Daten über den Befehl *docker login* angegeben werden. Es werden folgende Befehle ausgeführt:

```
docker login

cd docker/frontend/
docker build -t \
  slipke/ansible-docker-paper-frontend \
  .
docker push \
  slipke/ansible-docker-paper-frontend
cd ../../
```

Durch das Ausführen der Befehle wurde der *frontend* Container erstellt und mit dem Tag *slipke/ansible-docker-paper-frontend* versehen. Anschließend wurde dieser in die Docker Hub Registry hochgeladen.

Um den Anwendungs-Container zu erstellen, müssen folgende Befehle im Root-Verzeichnis ausgeführt werden:

```
docker build -t \
  slipke/ansible-docker-paper-application \
  .
docker push \
  slipke/ansible-docker-paper-application
```

Da der zu deployende *application* Container sich vom Entwicklungscontainer unterscheidet, wurde die Datei *Dockerfile* im Root-Verzeichnis erstellt. Dieses verwendet Ubuntu 14.04 als Basis-Image, kopiert den Quellcode der Webanwendung in den Container und setzt die für den Webserver benötigten Dateirechte. Anschließend wird dieser Container ebenfalls in ein Repository hochgeladen ([15]).

2. Deployment via Ansible

Um die zuvor erstellten Container auf die Produktionsumgebung auszurollen, kann das Playbook *ansible/services.yml* verwendet werden. Folgender Befehl führt das Playbook aus:

```
ansible-playbook ansible/service.yml \
  -i ansible/inventory/production
```

Dieses wiederum ruft die Rolle *slipke.services* auf, welche den *application*, *php* und *nginx* Container herunterlädt und ausführt. Die Anwendung läuft jetzt auf dem Produktionsserver und kann unter der URL `http://192.168.33.201` aufgerufen werden.

3. Nächste Schritte

Der nächste Schritt beim Automatisieren ist das Skalieren der Anwendung. Hierfür kann

das Docker-eigene Tool *docker-swarm* ([8]) genutzt werden, welches mehrere Produktionsserver über eine einheitliche Schnittstelle auf einen einzelnen Server abbildet und die Container verteilt. Zusätzlich wird dann eine Service-Registry (beispielsweise Consul [9]) benötigt, das die verteilten Container miteinander verknüpft.

VI. Fazit

Im Laufe dieser Arbeit wurde eine Entwicklungsumgebung erstellt, die mit dem Befehl *docker-compose up -d* automatisiert gestartet wird. Außerdem wurde ein Produktionsserver mit Hilfe von Vagrant hochgefahren, auf den die Webanwendung deployed wurde. Ansible wurde genutzt, um den Produktionsserver automatisch zu provisionieren und die Webanwendung auf denselbigen Server zu deployen. Möchte man nun in Zukunft weitere Produktionsserver hochfahren und provisionieren, funktioniert die Provisionierung unkompliziert auf dieselbe Art und Weise. Möchte man allerdings die vorher erstellten Container dynamisch auf viele Produktionsserver verteilen, werden zusätzliche Dienste benötigt, wie beispielsweise *docker-swarm* und Consul.

Ein weiterer Automatisierungsschritt wäre die volle Integration eines Continuous Integration/Continuous Deployment Prozesses. Hierbei wird der Code direkt nach einem Push in das Code-Repository von einem Build-Server heruntergeladen, getestet und der Anwendungs-Container selbstständig gebaut und direkt in die Container-Registry gepusht. Nachdem der Container hochgeladen wurde, wird er automatisiert über ein Deployment-Skript (beispielsweise mit Ansible) auf die Produktionsserver ausgerollt. Hierfür sollte ein Deployment-Prozess verwendet werden, der zunächst die überarbeiteten Container hochfährt, diese in der Service-Registry anmeldet, dann die alten Container herunterfährt, um zu garantieren, dass der Dienst zu keiner Zeit unerreichbar ist.

Um den bisherigen Produktionsaufbau weiter zu verbessern kann noch ein zusätzlicher Reverse-Proxy vor den Web-Server geschalten werden, welcher ebenfalls über einen Docker-Container installiert wird. Wird nun die Anzahl der Webserver-Instanzen erhöht, können diese automatisch in die Konfigurationsdatei des Reverse-Proxy geschrieben werden.

Abschließend gibt es noch das Tool Rancher ([10]), das über eine Web-Oberfläche eine Übersicht und Statistiken über alle laufenden Container bietet. Rancher kann ebenfalls über einen Docker-Container installiert werden. Wird nun ein neuer Produktionsserver provisioniert, kann dieser automatisch in die Konfiguration von Rancher eingetragen werden.

Als Gesamtfazit lässt sich sagen, dass die Kombination von Docker und Ansible einen sehr zukunftsträchtigen Ansatz darstellt. In Zukunft werden aufgrund der steigenden Nutzerzahlen Systeme wesentlich komplexer, weshalb der Bedarf an Automatisierung stark steigen wird. Dieser Trend ist beispielsweise auch im Netzwerk-Bereich mit dem Ansatz *Software defined Networking* ([16]) zu sehen. Docker bietet mit seinem Container-Konzept eine Möglichkeit, Anwendungen in viele Microservices aufzuspalten und diese flexibel zu skalieren. Ansible ermöglicht das Automatisieren der benötigten Prozesse, wie beispielsweise Provisionierung der Server oder dem Deployment. Durch die Kombination beider Plattformen wird die Arbeit für beispielsweise System-Administratoren und Entwickler deutlich vereinfacht.

Literatur

[1] Containerization. `http://www.webopedia.com/TERM/C/containerization.html`. *Stand: 23.02.2016*

[2] Microservices. `http://martinfowler.com/articles/microservices.html`. *Stand: 23.02.2016*

[3] Ansible Playbooks. `http://docs.ansible.com/ansible/playbooks.html`. *Stand: 23.02.2016*

[4] Ansible Inventory. http://docs.ansible.com/ansible/intro_inventory.html. *Stand: 23.02.2016*

[5] Docker Machine Overview. https://docs.docker.com/machine/overview/. *Stand: 23.02.2016*

[6] Comparing Filesystem Performance. http://mitchellh.com/comparing-filesystem-performance-in-virtual-machines. *Stand: 23.02.2016*

[7] GitHub angstwad.docker_ubuntu. https://github.com/angstwad/docker.ubuntu. *Stand: 23.02.2016*

[8] Docker Swarm overview. https://docs.docker.com/swarm/overview/. *Stand: 23.02.2016*

[9] Consul. https://www.consul.io. *Stand: 23.02.2016*

[10] Rancher. http://rancher.com. *Stand: 23.02.2016*

[11] Docker Registry. https://docs.docker.com/registry/. *Stand: 23.02.2016*

[12] Docker Hub PHP Container. https://hub.docker.com/_/php/. *Stand: 23.02.2016*

[13] Docker Hub nginx Container. https://hub.docker.com/_/nginx/. *Stand: 23.02.2016*

[14] Docker Hub ansible-docker-paper-frontend Container. https://hub.docker.com/r/slipke/ansible-docker-paper-frontend/. *Stand: 23.02.2016*

[15] Docker Hub ansible-docker-paper-application Container. https://hub.docker.com/r/slipke/ansible-docker-paper-application/. *Stand: 23.02.2016*

[16] Software Defined Networking. https://www.opennetworking.org/sdn-resources/sdn-definition. *Stand: 23.02.2016*