Docker

-

Einsatz als Entwicklungstool im Projekt eTrading Pro

Evaluation

Author	Roman Würsch, LIFHF
Rolle	Entwicklungsleiter, eTrading Pro
Im Auftrag der	Zürcher Kantonalbank

Juni 2015

Zusammenfassung

Im Rahmen eines viertages Einsatzes ausserhalb der Räumlichkeiten der Zürcher Kantonalbank wird das Docker Ökosystem unter die Lupe genommen.

Es wird Docker als Entwicklungstool angeschaut und anhand eines Proof of Concept auf einen möglichen Einsatz im Projekt eTrading Pro getestet.

Der Proof of Concept beinhaltet einen Jetty WebSocket Server mit *yass* als Service Framework und einem Typescript/HTML/CSS Client. Der Server wird als Docker-Image erstellt und in zwei verschiedenen Docker-Container laufen gelassen. Zum den Jetty WebSocker Server soll soll ein LoadBalancer davor betrieben werden.

Es wird eine Empfehlung für den Einsatz von Docker als Entwicklungstool ausgesprochen.

Durch die Studie des Docker Ökosystems wird eine Empfehlung für den Einsatz von Docker in der Betriebsplattform ausgesprochen.

Inhaltsverzeichnis

1. Vorwort

Dieses Thema ist für mich aktuell und sehr spannend. Bei dem Besuch der JAX-W im Herbst 2014 in München bin ich das erste Mal mit Docker in Berührung gekommen. Ich habe gleich das Potential dieser neuen Technologie erkannt. Bei einem Vortrag wurde gesagt, dass es nicht die Frage ist, ob wir in der Zukunft mit Docker arbeiten, sondern wann. Diese Aussage ist mir geblieben und ich möchte Prüfen was dahinter steckt.

1.1. Rahmenbedingungen

Diese Arbeit wurde im Einverständnis von Andreas Hofstetter, LIFH & Degu Dagne, LIFHF erstellt.

1.2. Urheberrecht

2. Einleitung

Docker ist ein Ökosystem das von der Firma **Docker, Inc.** aufgebaut wird. Die Firma wurde im Jahr 2013 gegründet und ist dabei schon sehr erfolgreich in dem was sie tut.

Docker wurde in drei Runden mit 11 Investoren auf 150 Millionen Doller gefounded.

Die Firma Entwickelt die Software **Docker** und sie betreibt auch einen Service der sich **Docker Hub** nennt. Zudem bietet sie in ihren eigenen Datacenters die möglichkeit Docker Hub zu nutzen.

2.1. Was ist Docker

"*Docker* ist eine Open-Source-Software, die beim Linux-Betriebssystem dazu verwendet werden kann, Anwendungen mithilfe von Betriebssystemvirtualisierung in Containern zu isolieren. Dies vereinfacht einerseits die Bereitstellung von Anwendungen, weil sich Container, die alle nötigen Pakete enthalten, leicht als Dateien transportieren und installieren lassen. Andererseits gewährleisten Container die Trennung der auf einem Rechner genutzten Ressourcen, sodass ein Container keinen Zugriff auf Ressourcen anderer Container hat."[1]

Docker basiert auf einer ähnlichen Technologie wie Linux Containers (LXC).

2.1.1. Was ist ein Docker Image

Ein Docker Image ist mehr oder weniger ein Betriebsystem Image. Mit Docker kann man Docker Images als .iso Dateien exportieren und importieren. Speziell bei Docker Images ist, dass sich ein diese in Layers aufteilen, somit kann ich ein bestehendes Docker Image nehmen und mein eigenes darauf aufbauen. Das ist meines erachtens die grösste Stärke bei Docker, da man z.B. sein RHEL 7 Linux perkeft aufsetzen und härten kann und das als Docker Image speichert. Danach kann man alle weiteren Applikationen darauf aufbauen. Der Aufbau solcher Images kann man mittels einer Scriptspache in Dockerfiles beschreiben und somit automatisieren.

2.1.2. Was ist ein Docker Container

Ein Docker Container ist eine Instanz eines Docker Images. Der Docker Container läuft auf dem Hostbetriebsystem, in dem auch die Docker Software läuft. Aus einem Docker Image können beliebig viele Docker Container gestartet werden.

Die Dockersoftware kann beim starten eines Containers Umgebungsvariablen und Netzerkports von aussen her setzten und somit den Kontext eines jeweiligen Containers bestimmen.

[1] Siehe: http://de.wikipedia.org/w/index.php?title=Docker_(Software)&oldid=142961470

2.2. Was ist Docker Hub

"*Docker Hub* ist ein Repository für Docker-Images. Es teilt sich in einen öffentlichen und einen privaten Teil auf. Im öffentlichen Teil kann jeder Nutzer seine selbst erstellten Images hochladen und damit anderen Nutzern zur Verfügung stellen. Außerdem gibt es mittlerweile offizielle Images, z. B. von Linux-Distributoren. Im privaten Teil des Docker Hubs können Benutzer ihre Docker Images hochladen und dadurch einfach z. B. firmenintern verteilen, ohne dass diese damit sofort öffentlich auffindbar sind.

Die Hub Software wurde von Docker Inc. als Open-Source-Software veröffentlicht, sodass man die Vorteile des Hubs nun auch nutzen kann, ohne die eigenen Images auf die Server von Docker laden zu müssen."[2]

Analog zu Nexus für Java Artefakte kann man also auch Docker Hub einsetzen um Docker-Images zu verwalten.

2.3. Linux Container vs. Virtual Machines

Jede virtuelle Maschine hat nicht nur seine Applikations Binaries die ca. 100 MB gross ist, sondern auch die Binaries des gesammten Betriebsystems, das ca. 10 GB gross ist. Zudem läuft das Betriebsystem komplett in seiner eigenen Instanz.

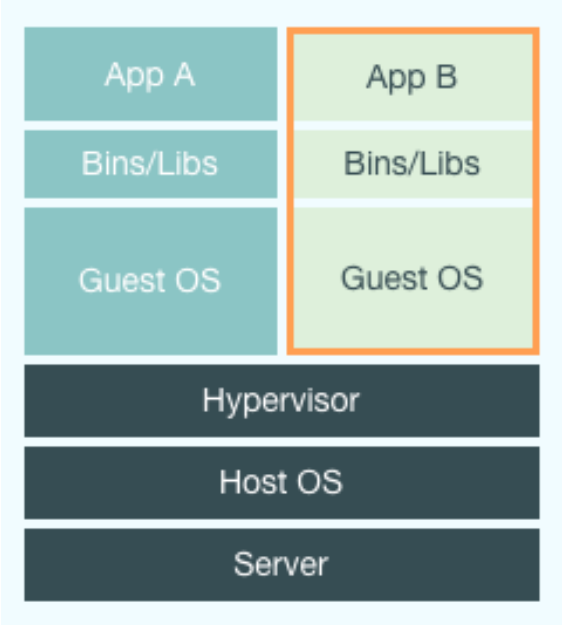

Abbildung 2.1.: Klassiche Virtualisierung mit Host OS and Guest OS

Ein Linux Container läuft in einem eigenen Prozess im Host Betriebsystem. Der Linux Kernel isoliert den Prozess von allen anderen Prozessen und mittels Namespaces. Das gibt die vorteile einer virtualisierung, ist aber viel leichtgewichtiger und Ressourcen schonender, da der Betriebsystem overhead nur einmal anfällt.

[2] Siehe: http://de.wikipedia.org/w/index.php?title=Docker_(Software)&oldid=142961470

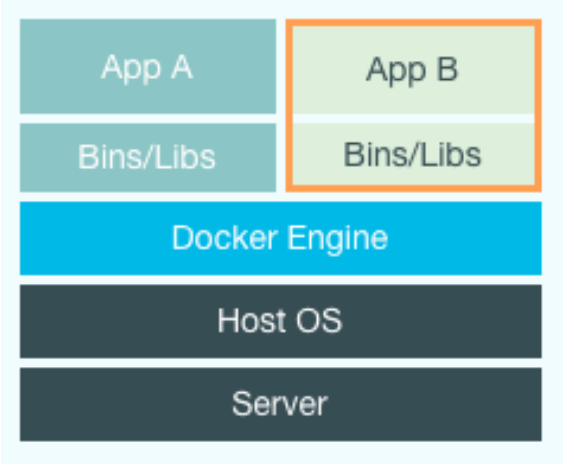

Abbildung 2.2.: Linux Container Virtualisierung

3. Docker Workflow

Docker hat den Grunsätzlichen Workflow von Build, Ship, Run. Wie ich das im Proof of Concept gemacht habe, werde ich hier veranschaulichen.

3.1. Build

Um ein Image zu bauen kann man ein Dockerfile erstellen. meines sieht so aus:

3.1.1. Dockerfile

```
1  FROM java:8
2  MAINTAINER Roman Wuersch
3
4  COPY . /usr/src/yass
5  WORKDIR /usr/src/yass
6
7  EXPOSE 9090
8
9  CMD ["./start-in-docker.sh"]
```

Listing 3.1: Dockerfile

Erklärung der Zeilen:

1) Das Image leitet vom offiziellen OpenJDK 8 Image ab und baut darauf auf.

4) Kopiert alle Dateien im aktuellen Pfad in das Image in den Pfad /urs/src/yass

5) Setzt das Startverzeichnis

7) Gibt an, dass der Port 9090 von einem Prozess verwendet wird und zukünftig angebunden werden kann

9) Der Befehl, welcher ausgeführt wird, wenn ein Container aus dem Image gestartet wird.

3.1.2. Bauen via Shellscript

Um das Image zu bauen habe ich ein Shell-Script gemacht:

```
1  #!/usr/bin/env bash
2  docker rmi -f sushicutta/docker-test
3  docker build -t sushicutta/docker-test .
```

Listing 3.2: Bauen eines Docker Images

Dieses Shell-Script sucht im aktuellen Verzeichnis ein Dockerfile und baut mit dessen Angaben das Image.

Erklärung der Zeilen:

2) Löscht das bestehende Image sushicutta/docker-test aus dem Docker Hafen

3) erstellt das Image neu unter dem selben Namen.

3.2. Ship

Um das Image zu liefern, kann man nun Docker Hub verwenden.

```
1 docker push sushicutta/docker-test
```

Listing 3.3: Push to Docker Hub

Oder man kann das Image als .tar Datei speichern.

```
1 docker save -o docker-test.tar sushicutta/docker-test
```

Listing 3.4: Docker Image exportieren

Das gelieferte Image kann man dann irgendwo wieder importieren.

```
1 docker load -i docker-test.tar
```

Listing 3.5: Docker Image importieren

Oder gleich das Image per SSH transportieren zippen und unzippen on the fly, und mit pv den Pipe traffic darstellen:

```
1 docker save <image> | bzip2 | pv | \
2      ssh user@host 'bunzip2 | docker load'
```

Listing 3.6: SSH Transport mit zippen und network traffic Visualisierung

3.3. Run

Das Image ist jetzt gebaut worden und im Docker Hafen angekommen, und es trägt den Namen sushicutta/docker-test.

Dieses Image kann nun beliebig viele Male gestartet werden.

Um das Image zweimal zu starten, benötigt man folgende Commands:

```
1 docker run -d -p 9091:9090 --name yass-node-1 sushicutta/docker-test
2 docker run -d -p 9092:9090 --name yass-node-2 sushicutta/docker-test
```

Listing 3.7: Image zwei mal starten auf den Ports 9091 und 9092

Das identische Image läuft jetzt zweimal, der Inhalt ist zu 100% identisch. Der unterschied besteht darin, dass Docker nun die Ports 9091 des Host Betriebsystems auf den Port 9090 des ersten Containers mapped und das selbe für den Port 9092 des zweiten Containers.

Abbildung 3.1.: Zwei identische Container auf verschiedenen Ports

A. Abbildungsverzeichnis

B. Listingverzeichnis

C. Literaturverzeichnis