Docker – Automatisches Deployment für Linux-Instanzen

CommitterConf Essen 2014

28. Oktober 2014

Christian Rost
Linux Consultant
B1 Systems GmbH
rost@b1-systems.de

Vorstellung B1 Systems

- gegründet 2004
- primär Linux/Open Source-Themen
- national & international tätig
- über 60 Mitarbeiter
- unabhängig von Soft- und Hardware-Herstellern
- Leistungsangebot:
  - Beratung & Consulting
  - Support
  - Entwicklung
  - Training
  - Betrieb
  - Lösungen
- dezentrale Strukturen

Schwerpunkte

- Virtualisierung (XEN, KVM & RHEV)
- Systemmanagement (Spacewalk, Red Hat Satellite, SUSE Manager)
- Konfigurationsmanagement (Puppet & Chef)
- Monitoring (Nagios & Icinga)
- IaaS Cloud (OpenStack & SUSE Cloud & RDO)
- Hochverfügbarkeit (Pacemaker)
- Shared Storage (GPFS, OCFS2, DRBD & CEPH)
- Dateiaustausch (ownCloud)
- Paketierung (Open Build Service)
- Administratoren oder Entwickler zur Unterstützung des Teams vor Ort

Partner


SUSE
PartnerNet
Platinum
CERTIFIED
SUSE PARTNER


redhat.
PREMIER
BUSINESS
PARTNER


ubuntu
Advantage reseller


puppet
labs


pfSense


Nagios
Authorized
Reseller

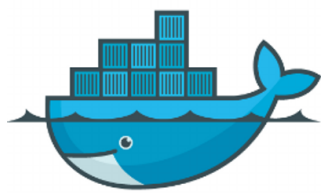

Hortonworks


ARISTA


rackspace
the open cloud company


ownCloud
YOUR CLOUD, YOUR DATA, YOUR WAY!
Gold Partner

Docker – Build, Ship and Run Applications



Kurze Frage vorweg:

- Wer kennt Docker?
- Wieviele haben Docker ausprobiert?
- Setzt jemand Docker produktiv ein?

Was ist Docker?

- Open Source Engine zum Standardisieren von Linux Containern (libcontainer)
- „*build, ship and run (distributed) applications*“
- offene Plattform für Entwickler und Systemadministratoren
- einfaches Erstellen und Teilen von Container Images
- Docker ist *kein* virtueller Server !
  $\Rightarrow$ wenig Overhead

Eigenschaften

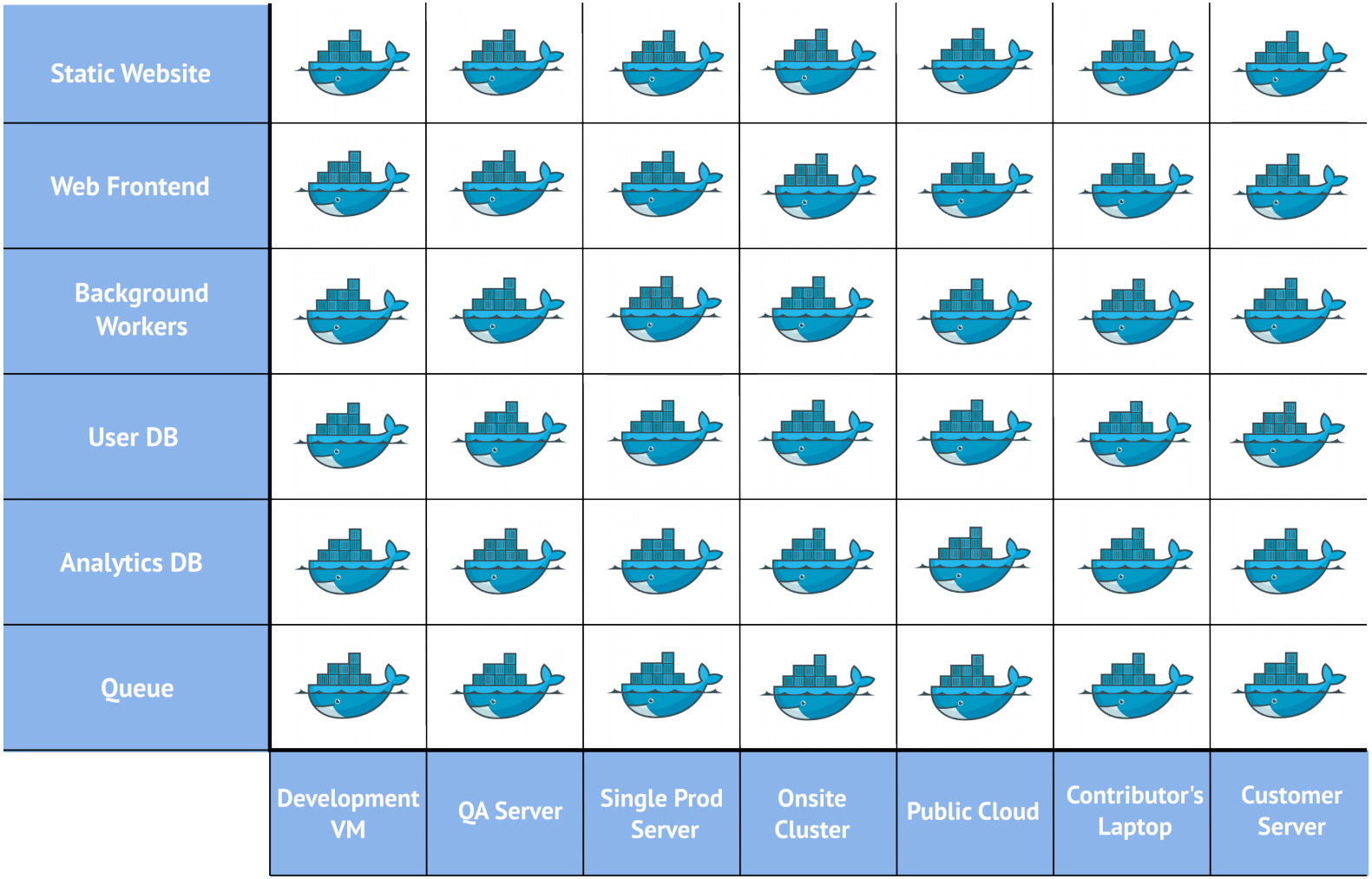
- Docker ermöglicht ein automatisches Deployment einer standardisierten Prozessumgebung für Linux-Anwendungen.
- Alle Linux-Anwendungen laufen in Docker.
  („*If it will run on Linux it will run in Docker.*“)
- Docker läuft auf allen gängigen Linux-Distributionen.
- Alle benötigten Funktionen befinden sich innerhalb des Containers:
  - Bibliotheken
  - Abhängigkeiten
  - Binärdateien
  - ...
- Container sind auf die Architektur der Host-Plattform beschränkt.

Warum Docker?

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's Laptop	Customer Server
Static Website	?	?	?	?	?	?	?
Web Frontend	?	?	?	?	?	?	?
Background Workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Quelle: http://slidedeck.io/pointful/docker-intro

Darum Docker!

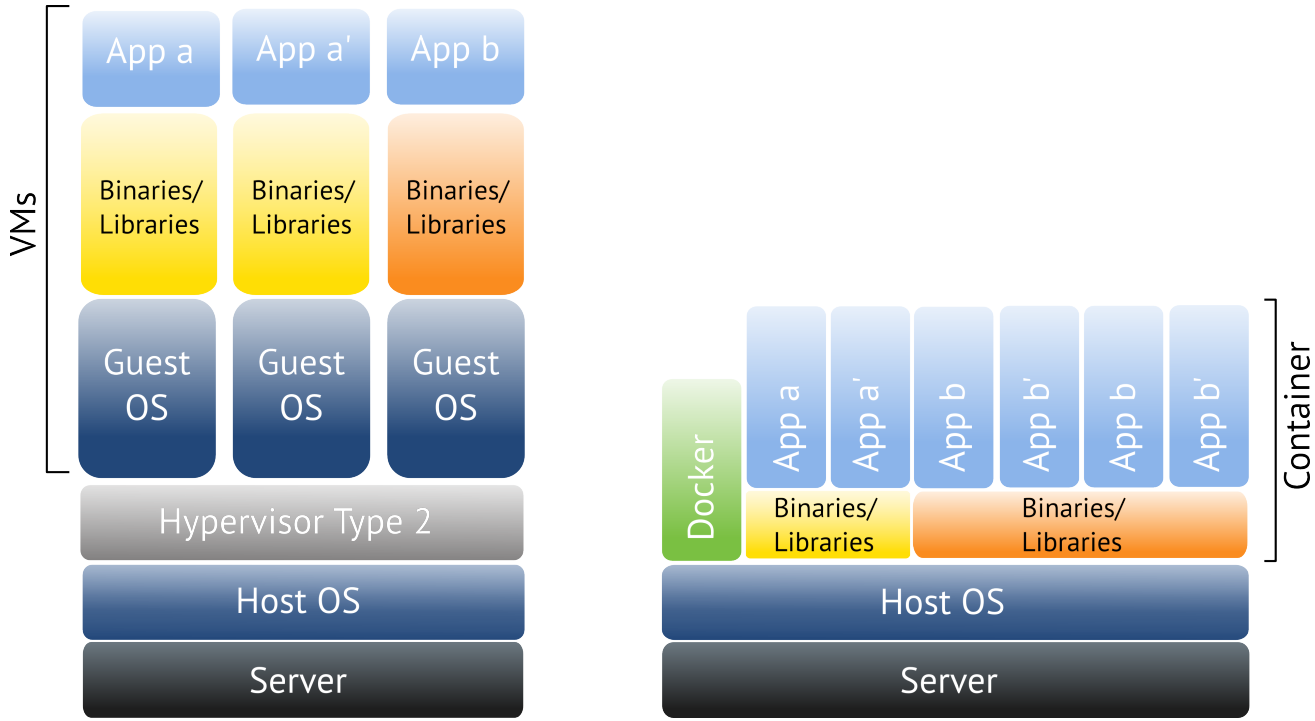


Quelle: http://slidedeck.io/pointful/docker-intro

Einsatzbereiche

- einfaches Deployment auch verteilter Anwendungen:
  - Evaluationssysteme
  - POC-Systeme
- schnelles Bereitstellen reproduzierbarer (Laufzeit-)Umgebungen:
  - für Entwickler
  - in Test-/QA-/Live-Umgebungen
- Konfigurationstests
- Kapselung kritischer Dienste in eigenen Containern
  $\Rightarrow$ erhöhte Ausfallsicherheit
- ...

VMs vs. Container



VMs komplette Maschine einschl. Kernel wird mit Hilfe eines Hypervisors virtualisiert.

Container nur Prozesse werden virtualisiert, das OS und Binaries/Libraries gemeinsam genutzt.

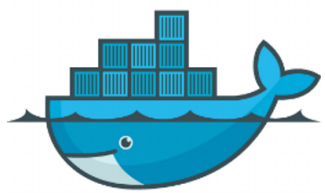
Technologien hinter Docker 1/2

- chroot on steroids
- Go Programming Language
- Linux Kernel Feature
  - Namespaces
  - Control groups (cgroups)
  - SELinux
  - capabilities
  - netlink
  - netfilter
  - ...
- Union file system/AuFS

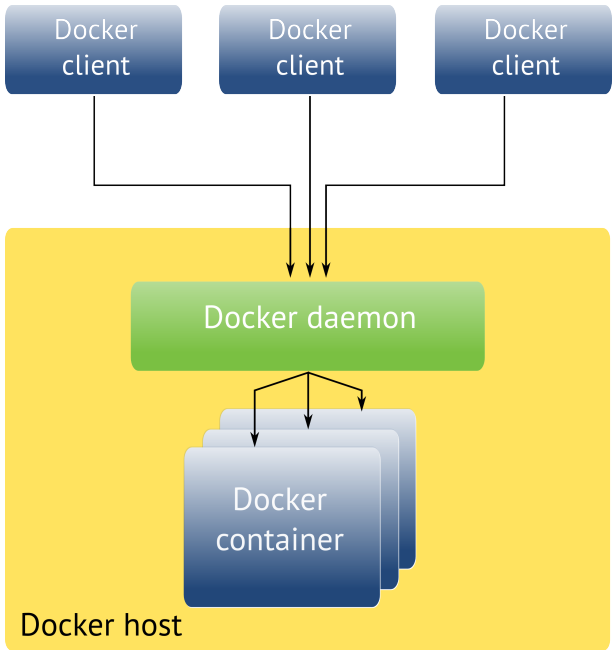
Technologien hinter Docker 2/2

- Containerformat
  - libcontainer
  - LXC
  - ...

Docker – das Ökosystem



Docker Engine



Die Docker Engine besteht aus zwei Teilen:

Server Daemon für den Serverprozess zur Verwaltung der Container.

Client Client zur Fernsteuerung des Daemons.

Inside Docker

- Docker Images
  - read only Template
  - Basis für Container
- Docker Registries
  - halten Images
  - privat und öffentlich
  - Docker Hub
- Docker Container
  - wird auf Basis eines Images und Dockerfiles erzeugt
  - hält Änderungen vor – read/write Layer
  - kann zu einem neuen Image „committed“ werden

Docker – Client-Befehle (Auszug)

`attach`	in laufenden Container hängen
`build`	Container aus Docker-File bauen
`commit`	neues Images aus den Änderungen eines bestehenden Containers erzeugen
`images`	Images auflisten
`kill`	Container beenden
`login`	am Docker Registry Server anmelden
`ps`	Container auflisten
`rm`	Container löschen
`run`	Kommando in einem neuen Container ausführen
`start`	beendeten Container starten
`stop`	gestarteten Container stoppen

Docker – Container (Beispiele)

Download des „offiziellen“ openSUSE-Containers:

```
$ docker pull opensuse:13.1
```

Programm innerhalb des Containers ausführen (allgemein):

```
$ docker run opensuse:13.1 <command> <command params>
```

Programm innerhalb des Containers ausführen (Beispiel: *Bash*):

```
$ docker run -t -i opensuse:13.1 /bin/bash
```

Dockerfile

- Bauanleitung für ein Image
- enthält die notwendigen Anweisungen Erstellen eines Image
- stellt so die Reproduzierbarkeit eines Image bei jeder neuen Erstellung sicher
- Anweisungen sind einzeilig und beginnen mit einem Schlüsselwort.
- Anweisungen sind nicht case-sensitive.
  (Schlüsselwörter werden gewöhnlich groß geschrieben.)
- Kommentare werden durch ein #-Zeichen eingeleitet.

Dockerfile – Beispiel

Dockerfile für MySQL (Beispiel):

```
FROM opensuse:13.1
MAINTAINER Tux <info@b1-systems.de>
RUN zypper install -y mysql mysql-server
ADD start.sh /start
RUN chmod +x /start
EXPOSE 3306
CMD ["/start"]
```

Der Docker-Kosmos

Docker ist kollaborativ, modular und erweiterbar:

Atomic/CoreOS spezialisierte Linux-Distributionen zur Ausführung von Containern

Gitlab Collaboration on Code

Jenkins Continuous Integration System für Servlet Container (z.B. Apache Tomcat)

Puppet Konfigurationsmanagement

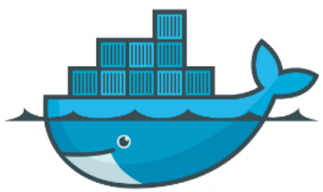
Fig Orchestration

...

Docker Orchestration – Fig

- Orchestration Tool für Docker
- erlaubt das gemeinsame Deployment mehrerer Container (*Multi-Container Service*)
- einfach in der Benutzung (Beispiel in der Live-Demo)

Docker – Live Demo



Docker – Live Demo

- Installation
- Container starten
- Handling von Docker
- Verknüpfung von Containern mit Fig
- ...

Docker – Installation openSUSE 13.1 (1/2)

Repository einrichten:

```
$ sudo zypper ar -f
http://download.opensuse.org/repositories/Virtualization/
openSUSE_13.1/ Virtualization
$ sudo rpm -import
http://download.opensuse.org/repositories/Virtualization/
openSUSE_13.1/repodata/repomd.xml.key
```

Installation:

```
$ sudo zypper in docker
```

Start des Daemons:

```
$ sudo systemctl start docker
$ sudo systemctl enable docker
```

Docker – Installation openSUSE 13.1 (2/2)

Benutzerrechte durch Gruppenmitgliedschaft:

```
$ sudo /usr/sbin/usermod -a -G docker <benutzer>
```

Start eines Test-Containers:

```
$ sudo docker run --rm -i -t opensuse /bin/bash
```

⇒ Bash des Test-Containers:

```
bash-4.2# uname -a
Linux 9a90ca38ebcf 3.11.10-21-desktop
#1 SMP PREEMPT Mon Jul 21 15:28:46 UTC 2014 (9a9565d) x86_64
x86_64 x86_64 GNU/Linux
bash-4.2#
```

Demo

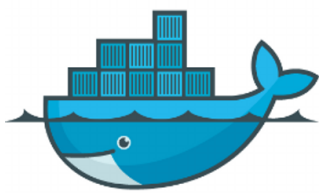


Fig – Beispiel

- Automatisches Starten einer WordPress-Instanz
- Webserver und Datenbank getrennt
- Wordpress über Port 80 erreichbar
- Datenbank mit Webserver verlinkt
- Horizontales Scaling der Webserver

Fig – WordPress

Erstellen der Wordpress Instanz:

```
$ wget http://wordpress.org/wordpress-3.8.1.tar.gz && tar
xvfzp wordpress-3.8.1.tar.gz $ cd wordpress
```

Dockerfile

```
FROM orchardup/php5
ADD . /code
```

Quelle: `http://www.fig.sh/wordpress.html`

Fig – YAML-Datei

fig.yml:

```
web:
 build: .
 command: php -S 0.0.0.0:80 -t /code
 ports:
 - "80:80"
 links:
 - db
 volumes:
 - .:/code
db:
 image: orchardup/mysql
 environment:
 MYSQL_DATABASE: wordpress
```

Quelle: `http://www.fig.sh/wordpress.html`

Fig – WordPress

wp-config.php – Teil I

```
 <?php
define('DB_NAME', 'wordpress');
define('DB_USER', 'root');
define('DB_PASSWORD', '');
define('DB_HOST', "db:3306");
define('DB_CHARSET', 'utf8');
define('DB_COLLATE', '');
define('AUTH_KEY', 'b1-systems-docker-demo');
define('SECURE_AUTH_KEY', 'b1-systems-docker-demo');
define('LOGGED_IN_KEY', 'b1-systems-docker-demo');
define('NONCE_KEY', 'b1-systems-docker-demo');
define('AUTH_SALT', 'b1-systems-docker-demo');
define('SECURE_AUTH_SALT', 'b1-systems-docker-demo');
```

Quelle: http://www.fig.sh/wordpress.html

Fig – WordPress

wp-config.php – Teil II

```
define('LOGGED_IN_SALT', 'b1-systems-docker-demo');
define('NONCE_SALT', 'b1-systems-docker-demo');
$table_prefix = 'wp_';
define('WPLANG', '');
define('WP_DEBUG', false);
if ( defined('ABSPATH') )
 define('ABSPATH', dirname(__FILE__) . '');
require_once(ABSPATH . 'wpsettings.php');
```

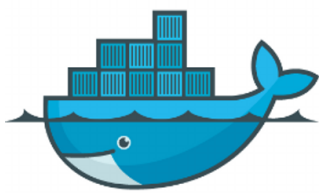
Quelle: http://www.fig.sh/wordpress.html

Fig – WordPress

Fig die Arbeit machen lassen :)

```
$ fig up
$ fig ps
curl <IP>
```

Demo



Links und Quellen

Homepage des Docker-Projekts `http://www.docker.com/`

Homepage von Fig `http://www.fig.sh/`

Docker on GitHub `https://github.com/docker/docker`

Fig on Github `https://github.com/docker/fig`

Vielen Dank für Ihre Aufmerksamkeit!

Bei weiteren Fragen wenden Sie sich bitte an info@b1-systems.de
oder +49 (0)8457 - 931096