TEILE UND HERRSCHE

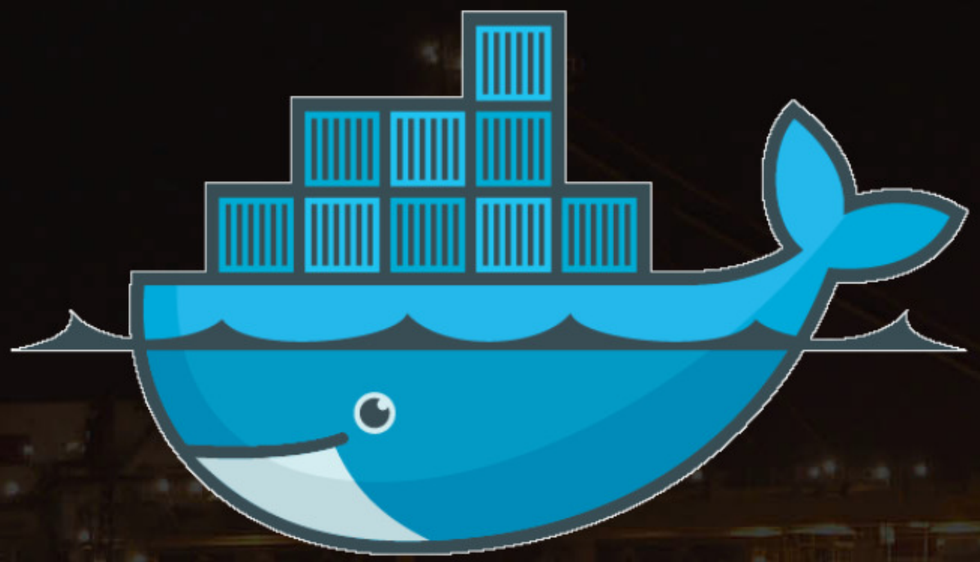
Verteilte Java-Anwendungen mit Docker

Benjamin Schmid
Technology Advisor
@bentolor

Dr. Ralph Guderlei
Technology Advisor
@rguderlei

Teile und Herrsche von Benjamin Schmid, Dr. Ralph Guderlei ist lizenziert unter einer Creative Commons-Lizenz (by-nc-sa). Beruht Reveal.JS.

exXcellent solutions



docker

150 MIO. USD RISIKOKAPITAL

Google

redhat

amazon.com

Microsoft

Alle Warenzeichen und Logos sind Eigentum ihrer jeweiligen Rechteinhaber.

HYPE?

oder

»THE NEXT BIG THING«?

JUMP INTO DOCKER: LIVE DEMO

```
$ mvn -o package
$ java -jar target/rest-microservice-1.0.0.jar server
$ sensible-browser "http://localhost:8080/hello-world"
```

```
$ docker build -t my/microservice .

$ docker run -d --name web1 my/microservice
$ docker run -d --name web2 my/microservice
$ docker run -d --name web3 my/microservice

$ docker pull haproxy:1.5
$ docker run -d --name balancer \
       --link web1:web1 --link web2:web2 --link web3:web3 \
       -p 8080:80 \
       -v (pwd)/etc:/usr/local/etc/haproxy:ro \
       haproxy:1.5
$ sensible-browser "http://localhost:8080/hello-world"
```

Inhaltsverzeichnis

1. Docker in a nutshell
2. Hands on Docker
3. Docker advanced
4. Docker & Java
5. Hype vs. Potential

DOCKER IN A NUTSHELL

- **Isolierte Prozessausführung** (Sandbox) auf Basis von »Linux Containers (LXC)«
- Wesentlich **effizienter** als VMs/Hypervisoren
- **Portables Format** für Container mit **Versionierung**
- **Leichtgewichtige** Laufzeit- und Packaging-Tools
- **Cloud-Repository** für Container-Vorlagen

DOCKER IN A NUTSHELL

Build Once, **Configure** Once and **Run Anywhere**

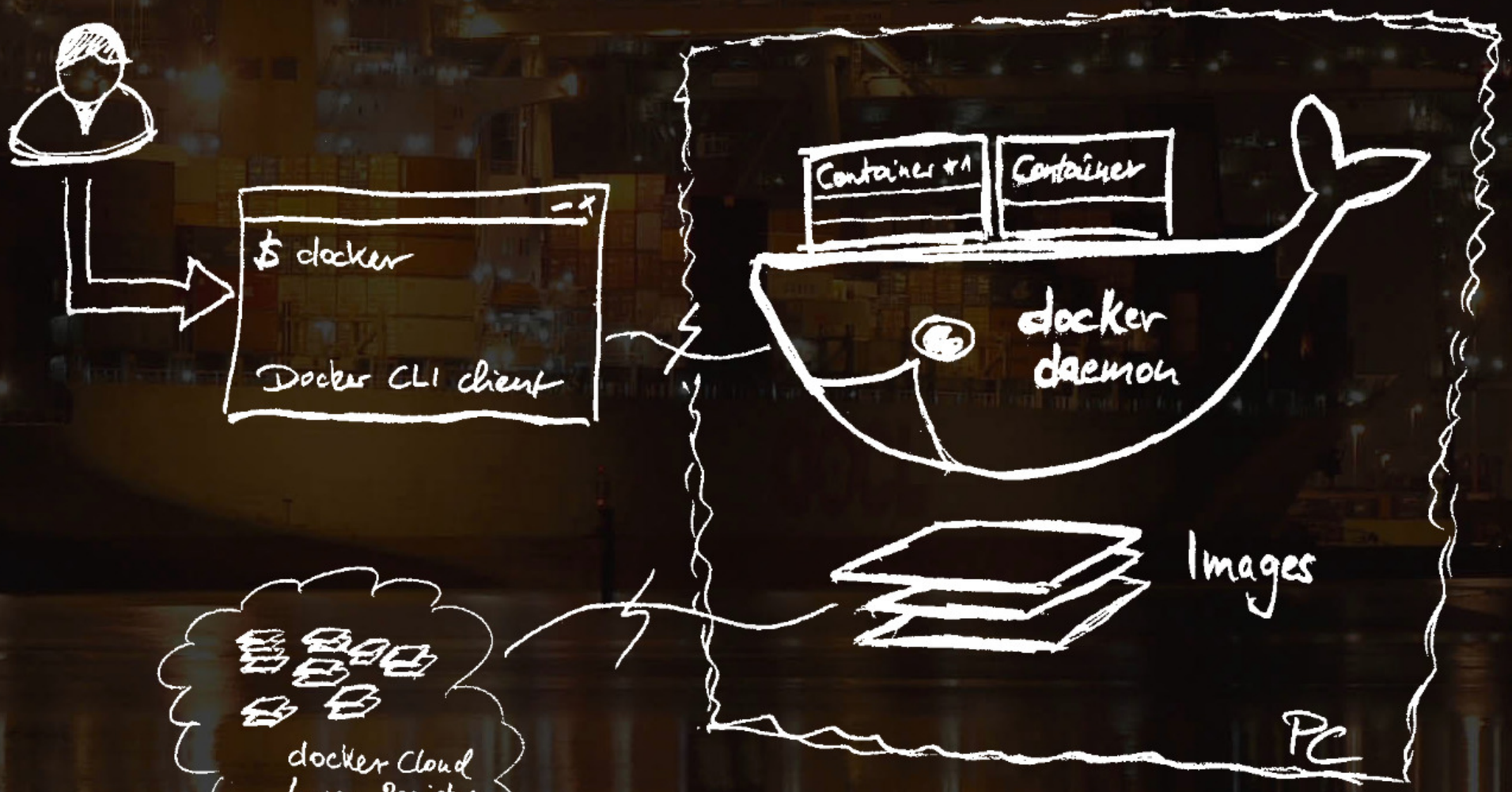
Grundlegende Begriffe

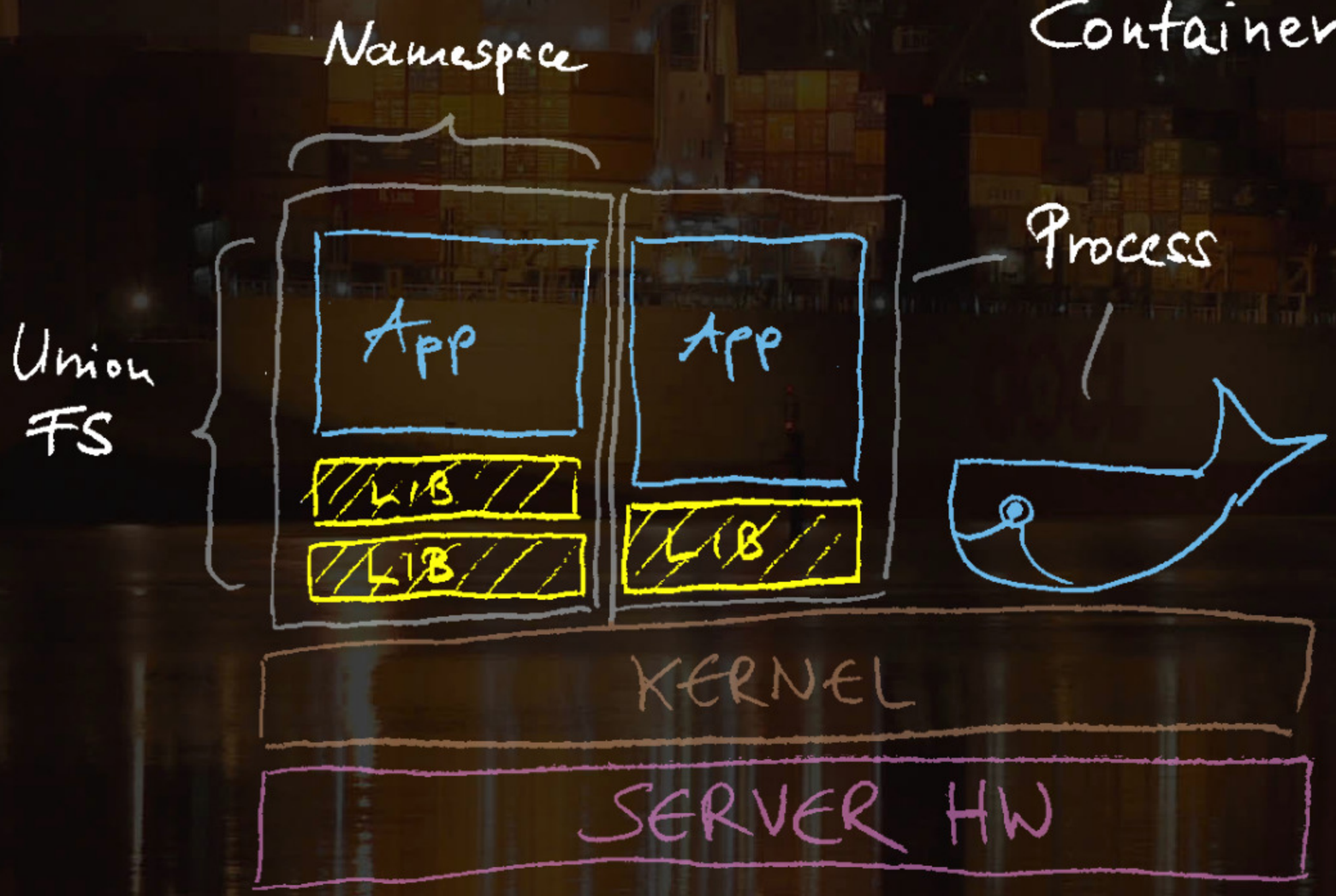
Image: Paketierte Zusammenstellung von Dateien. Damit eine *schreibgeschützte Blaupause* für eine Systemumgebung.

Container: Eine unabhängige Systemumgebung basierend auf einem Image

Repository: Ein Sammlung von Images auf dem lokalen Docker oder zentralen Registry Server

Docker Komponenten




Container
Namespace
Process
Union FS
App
App
LIB
LIB
LIB
KERNEL
SERVER HW

Container | Virtualization

Namespace

Process

Union FS

App

App

LIB

LIB

LIB

KERNEL

SERVER HW

VM

App

LIB

KERNEL

HW

VM

App

LIB

KERNEL

HW

HYPERVISOR

KERNEL

SERVER HW

Erstellen von Images

1. **Manuell:** *Commit* eines Containers
2. **Automatisiert:** via `Dockerfile`

In der Regel: Automatisiert, aufbauend auf einem via `docker pull` bezogenem Base-Image

Repositories (134)

Users (0)

Organizations (1)

library

Show: All Sort by: Relevance Results: Si

ubuntu 10 hours ago
Ubuntu is a Debian-based Linux operating system based on free software.
1810 4773757

node 14 hours ago
Node.js is a JavaScript-based platform for server-side and networking applications.
777 996330

debian 19 hours ago
Debian is a Linux distribution that's composed entirely of free and open-source software.
492 514414

postgres 13 hours ago
The PostgreSQL object-relational database system provides reliability and data integrity.
771 1332802

Base Images

- ~130 offizielle Images
- 100.000+ Images insgesamt
- 200+ Millionen Downloads

Images bauen: `Dockerfile`

```
FROM java:8
MAINTAINER Inspector Gadget

                                # Kommando im Container ausführen
RUN apt-get update && apt-get dist-upgrade -y &&  \
    apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

                                # JAR und Config-File in Image aufnehmen
WORKDIR /opt
ADD target/rest-microservice-1.0.0.jar app.jar
ADD src/main/resources/example.yml     app.yml

                                # announce exported port 8080 and 8081
EXPOSE 8080 8081
                                # Wichtig: Separate Volumes für Daten
VOLUME ["/srv/"]

                                # JAR ausführen beim Start des Containers
ENTRYPOINT java -jar app.jar server
```

HANDS ON DOCKER

Neues Image erstellen

Auf Basis unseres `Dockerfile` erzeugen wir unser Image:

```
docker images
docker build -t my/dockerapp:latest .
docker images
```

Dabei sehen wir wie die Schritte einzeln abgearbeitet werden; bei jedem der Schritte entsteht ein neuer Layer. Mehr dazu später.

Container erstellen & starten

Nun starten einen neuen (anonynmen) Container auf Basis des Images: Einmal interaktiv, einmal als Daemon

```
docker run -it my/dockerapp
[ctrl-c]
docker run -d my/dockerapp
```

Container managen

Nun schauen wir mal was so los ist. Wir die eingangs gestarteten Container laufen noch, welche wir erst mal unsanft killen.

```
docker ps
docker kill balancer web1 web2 web3
docker ps
```

Regulär stoppt man Container aber höflich mit `docker stop`.
Wieder starten über `docker start` ... nicht ~~`docker run`~~ .

```
docker stop name
docker ps
docker ps -as
docker start name
```

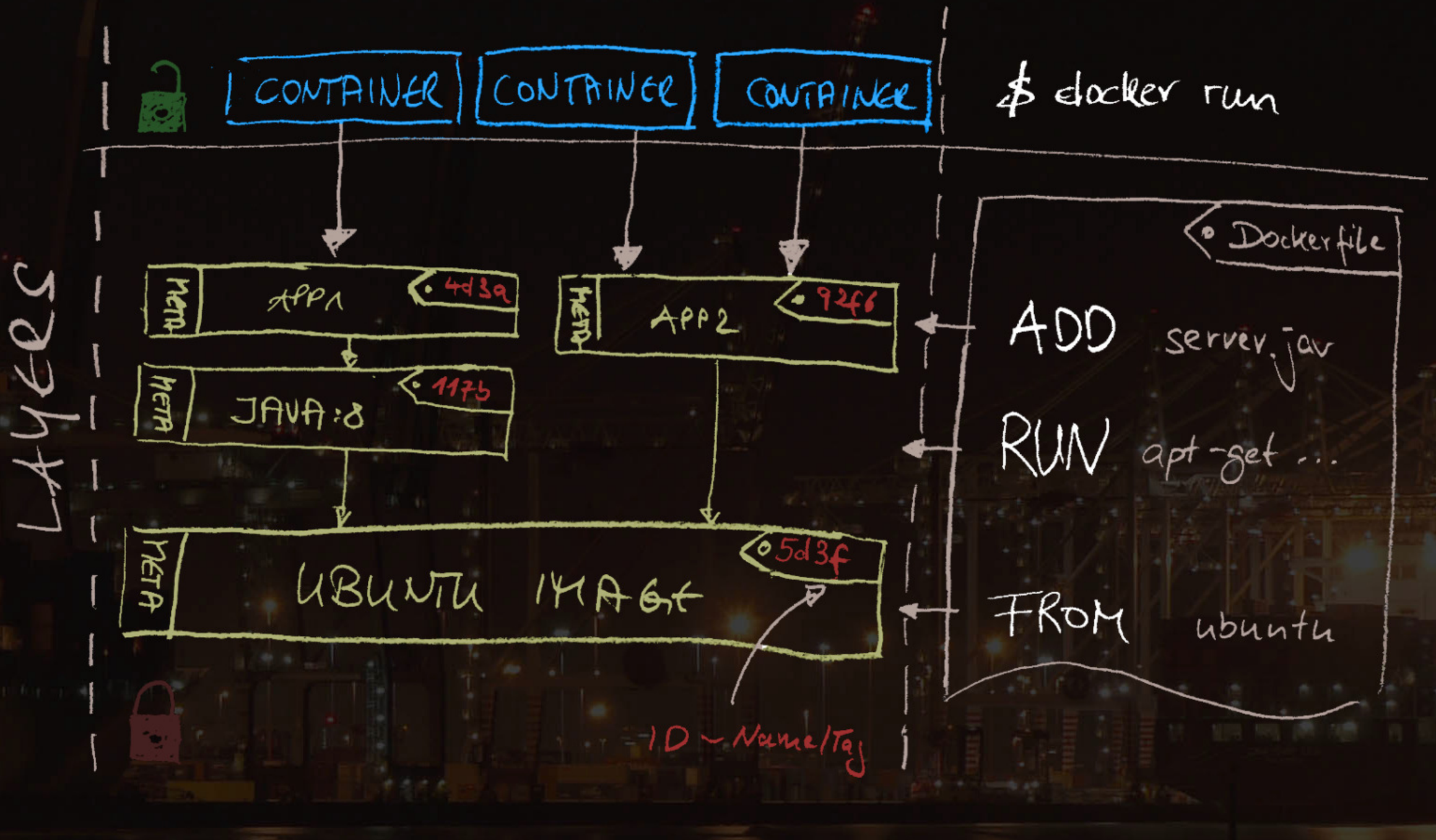
Diagnostik von Container

Unsere Container laufen im Hintergrund. Wie kommen wir an Infos, wenn mal etwas nicht funktioniert?

```
docker inspect name | grep IP
sensible-browser "http://ip:8080/hello-world"

docker logs name
docker exec -it name /bin/bash
```

DOCKER ADVANCED


CONTAINER
CONTAINER
CONTAINER
$ docker run
Layers
META
APP1
4d39
META
APP2
9246
META
JAVA:8
1175
META
UBUNTU IMAGE
5d3f
ID ~ Name/Tag
Dockerfile
ADD server.jar
RUN apt-get ...
FROM ubuntu

LAYERS & UNION FILE SYSTEM

- Filesystem aus gestapelten **Layern** (Union File System)
- **Images** sind schreibgeschützte Layers & Meta
- Schreibbarer Layer dazu ⇨ **Container**.
- Dockerfile: Jeder Schritt = **neuer Layer** ⇨ `docker history`
- **Versionierung:** Layer-ID = Container/Image ID ⇨ `ac408c338`

DATA VOLUMES

Datenhaltung unabhängig vom Container-Lebenszyklus.
Überlebt: Löschen & Updates. Erlaubt: Sharing.

Option 1: *Container Volumes*

```
docker create -v /mydata --name mydata java:8 /bin/true
docker run -d --volumes-from mydata my/dockerapp
```

Option 2: *Lokale Verzeichnisse*

```
docker run -v $(pwd)/etc:/usr/local/etc:ro haproxy:1.5
```

EINSTIEGS-DEMO REVISITED

```
docker build -t my/microservice

docker run -d --name web1 my/microservice
docker run -d --name web2 my/microservice
docker run -d --name web3 my/microservice

docker run -d --name balancer \
    --link web1:web1 --link web2:web2 --link web3:web3 \
    -p 8080:80 \
    -v (pwd)/etc:/usr/local/etc/haproxy:ro \
    haproxy:1.5
```

DOCKER KOMMANDOS

Images managen

Kommando	Beschreibung
`docker images`	Liste der lokal vorhandenen Images
`docker pull`	Image-Download aus dem Repository
`docker build`	Docker Image via `Dockerfile` erstellen
`docker commit`	Containers als neues Image comitten

Container kontrollieren

Kommando	Beschreibung
`docker run`	Erzeugen eines neuen Containers
`docker start`	Starten eines bestehenden Containers
`docker stop`	Stoppen eines Containers
`docker ps -a`	Liste *aller* Container
`docker kill`	Stoppen mit Nachdruck

Diagnose & Tools

Kommando	Beschreibung
`docker inspect`	Low-Level Informationen über Container/Images
`docker log`	Terminal-Ausgaben des Containers
`docker exec`	Prozess im Container starten, z.B. interaktive Shell
`docker diff`	Was wurde im Container geändert?
`docker history`	Image: Welche Schichten & wie entstanden?

DOCKER & JAVA

Anwendungsfälle

- **Entwicklung**:
  - Lokale, isolierte & portable Umgebung
  - Infrastruktur analog zu Produktion
  - fixe Versionen für Tools & Laufzeitumgebungen
- **CI-Server:** Tests in *isolierten* Referenzumgebungen
- **Betrieb**:
  - einfaches Deployment
  - Resourcenschonender Parallel-Betrieb vieler Instanzen
- **Architektur**: Microservice als Uberjar statt Appserver

Microservice Frameworks

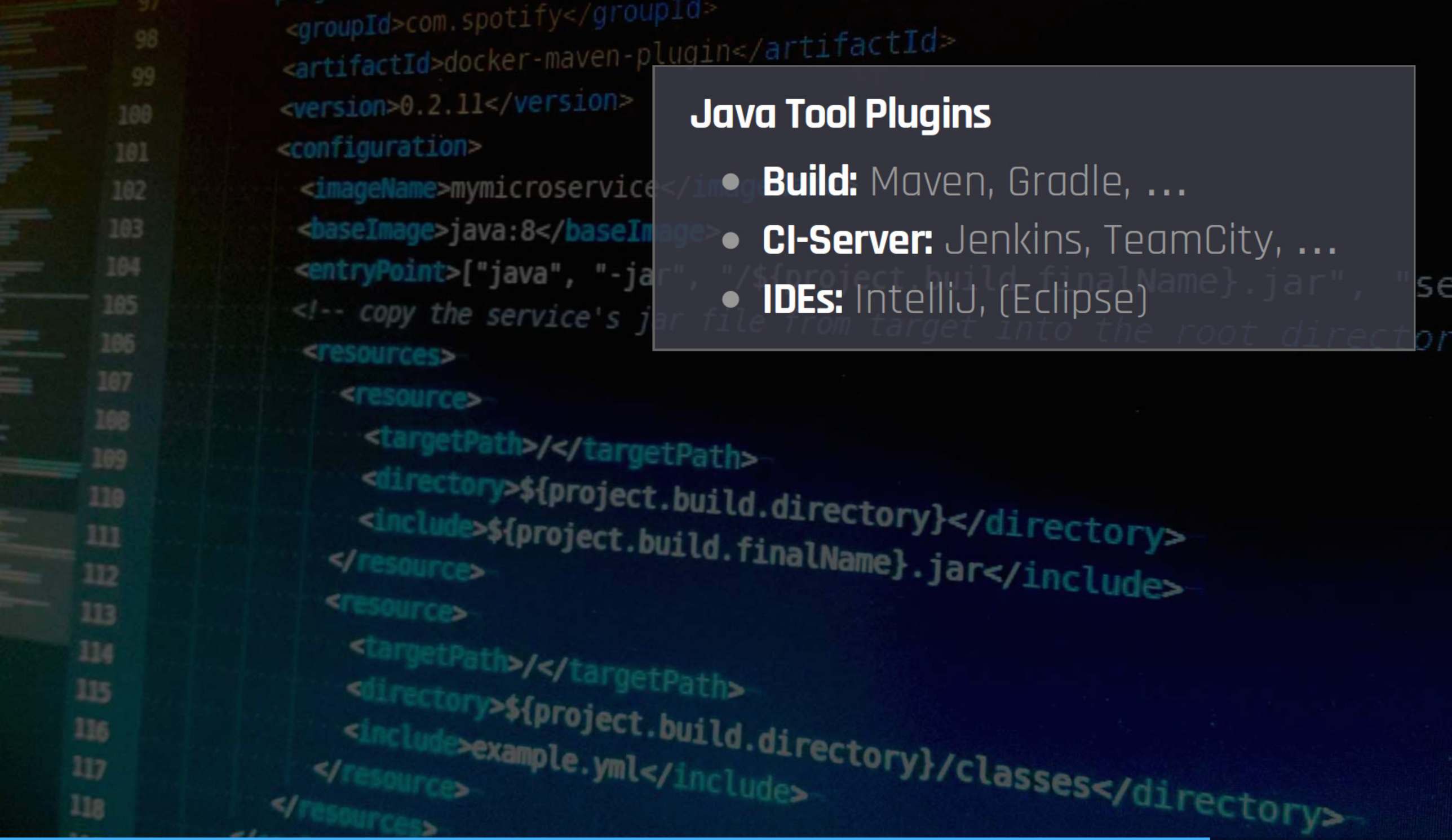
- Spark Framework
- Dropwizard
- Spring Boot
- Vert.x

Spark - A tiny Sinatra inspired framework for creating web applications in Java 8 with minimal effort

Quick start

```
import static spark.Spark.*;

public class HelloWorld {
    public static void main(String[] args) {
        get("/hello", (req, res) -> "Hello World");
```


Java Tool Plugins
Build: Maven, Gradle, ...
CI-Server: Jenkins, TeamCity, ...
IDEs: IntelliJ, (Eclipse)

HYPE VS. POTENTIAL

DOCKER & WINDOWS

Heute: (Linux container)

- boot2docker
- docker auf Microsoft Azure

Zukunft: (Windows container)

- Windows Server Containers
- Hyper-V Container

STATUS
von Docker

- nur Teil einer Gesamtlösung
- Dynamisches Umfeld
- 1.x-Technologie & Folgeprobleme
- Unklare Ausrichtung
  - ⇨ Rocket, Open Container, LXD

FRAGEN,

die sich durch Docker ergeben

- Nachvollziehbarkeit & Delivery Model
- Datenhaltung
- Zuständigkeiten
- Security (root, Patches)

POTENTIAL DOCKER

Automatisierung

Provisioning, Config, Delivery

Immutability

Umgebung = Code, Disposable

Serverkonsolidierung

System-agnostisch

Deployment

Einfach; Container als Einheit

POTENTIAL DOCKER

Automatisierung

Provisioning, Config, Delivery

Serverkonsolidierung

System-agnostisch

Immutability

Umgebung = Code, Disposable

Deployment

Einfach; Container als Einheit

Brauche ich docker?

- Was sind signifikante Vorteile?
- Ressourcen für komplexe, ungelöste Probleme?

»Let's Dockerize everything« ⇨ ziemlich wild & übermütig!

TOOLS & REFERENZEN

GUIs

- dockerUI
- shipyard
- panamax
- docker Kitematic

Spezielle Linux-Distros

- Rancher & RancherOS
- CoreOS

Orchestrierung

- Kubernetes
- docker Compose & docker Swarm
- Apache Mesos mit Marathon

Interessante Java-Tools

- Dropwizard
- Spark
- Vert.x
- Spring Boot
- Docker Maven-Plugins

Image Credits: *Southampton Container Port* Geni / CC-BY-SA-4.0