31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Teile und herrsche

Verteilte Anwendungen mit Docker

Dr. Ralph Guderlei & Benjamin Schmid

eXXcellent solutions GmbH

Die interaktive HTML-Präsentation finden Sie unter
https://bentolor.github.io/docker-talk/

TEILE UND HERRSCHE

Verteilte Java-Anwendungen mit Docker

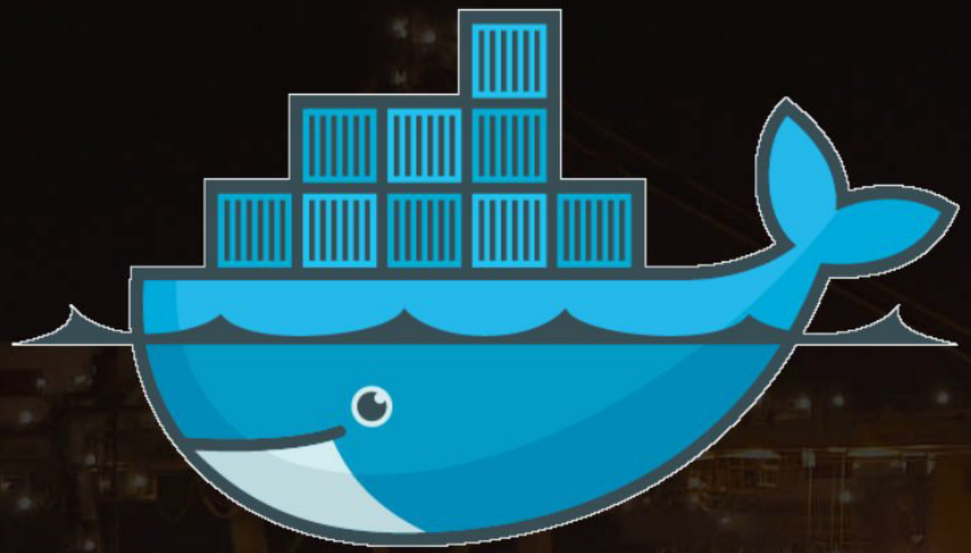
Benjamin Schmid
Technology Advisor
@bentolor

Dr. Ralph Guderlei
Technology Advisor
@rguderlei

? für Hilfe, S für Folien-Notizen

15. Juni 2015

ex|xcellent solutions



docker


150 MIO. USD
RISIKOKAPITAL


HYPE?
oder
»THE NEXT BIG THING«?

JUMP INTO DOCKER: LIVE DEMO

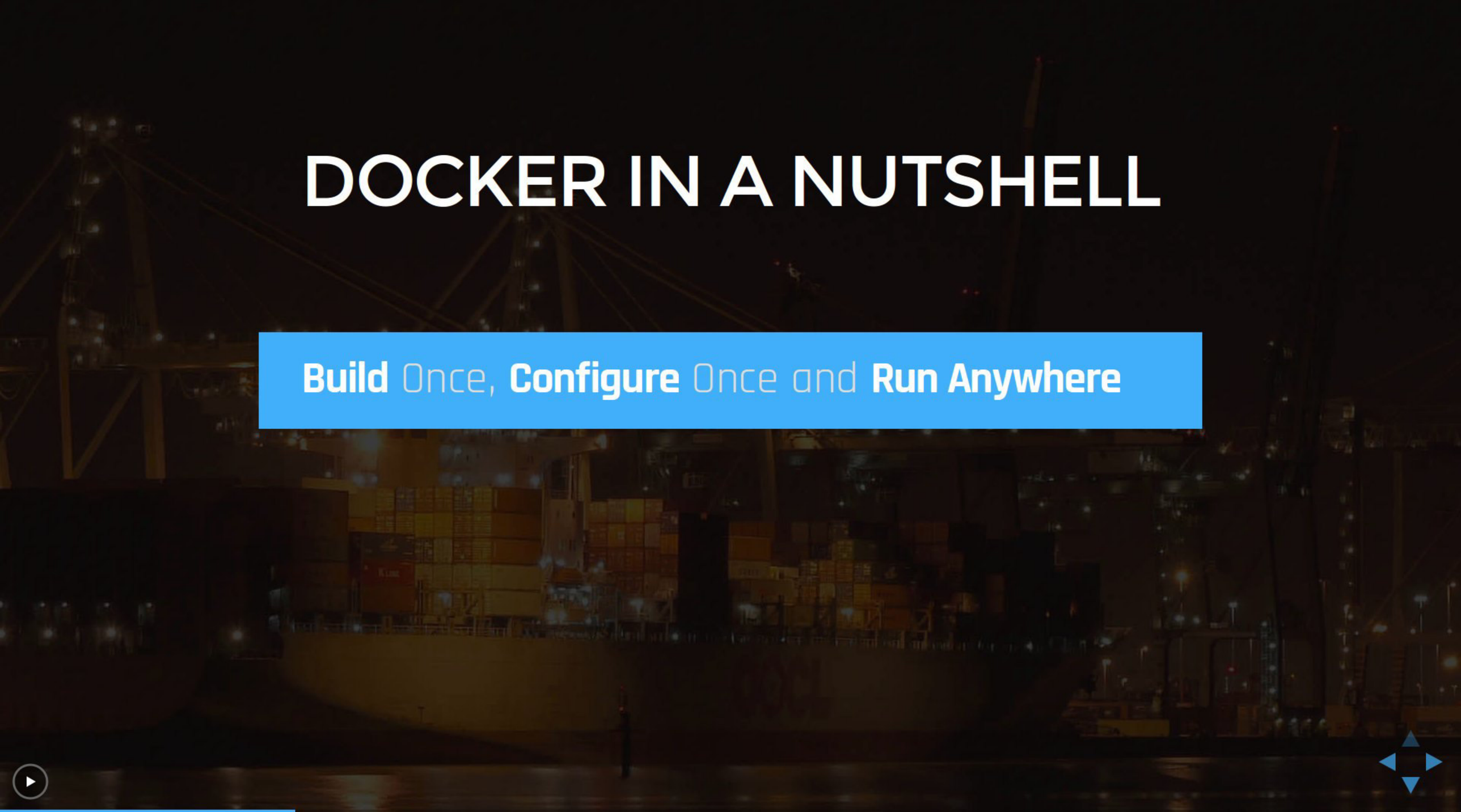
```
$ mvn -o package
$ java -jar target/rest-microservice-1.0.0.jar server
$ sensible-browser "http://localhost:8080/hello-world"
```

```
$ 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

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

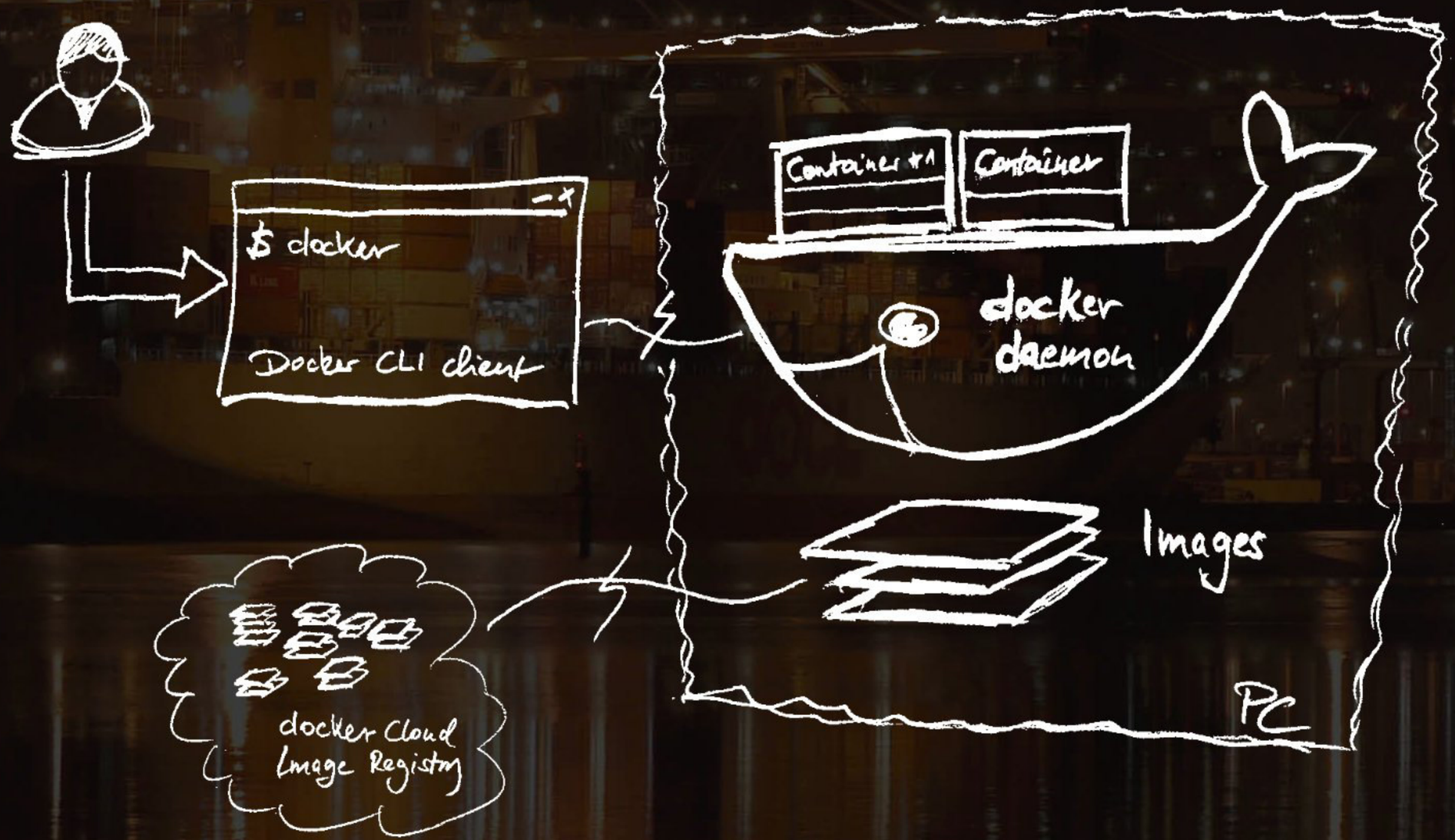
Grundlegende Begriffe

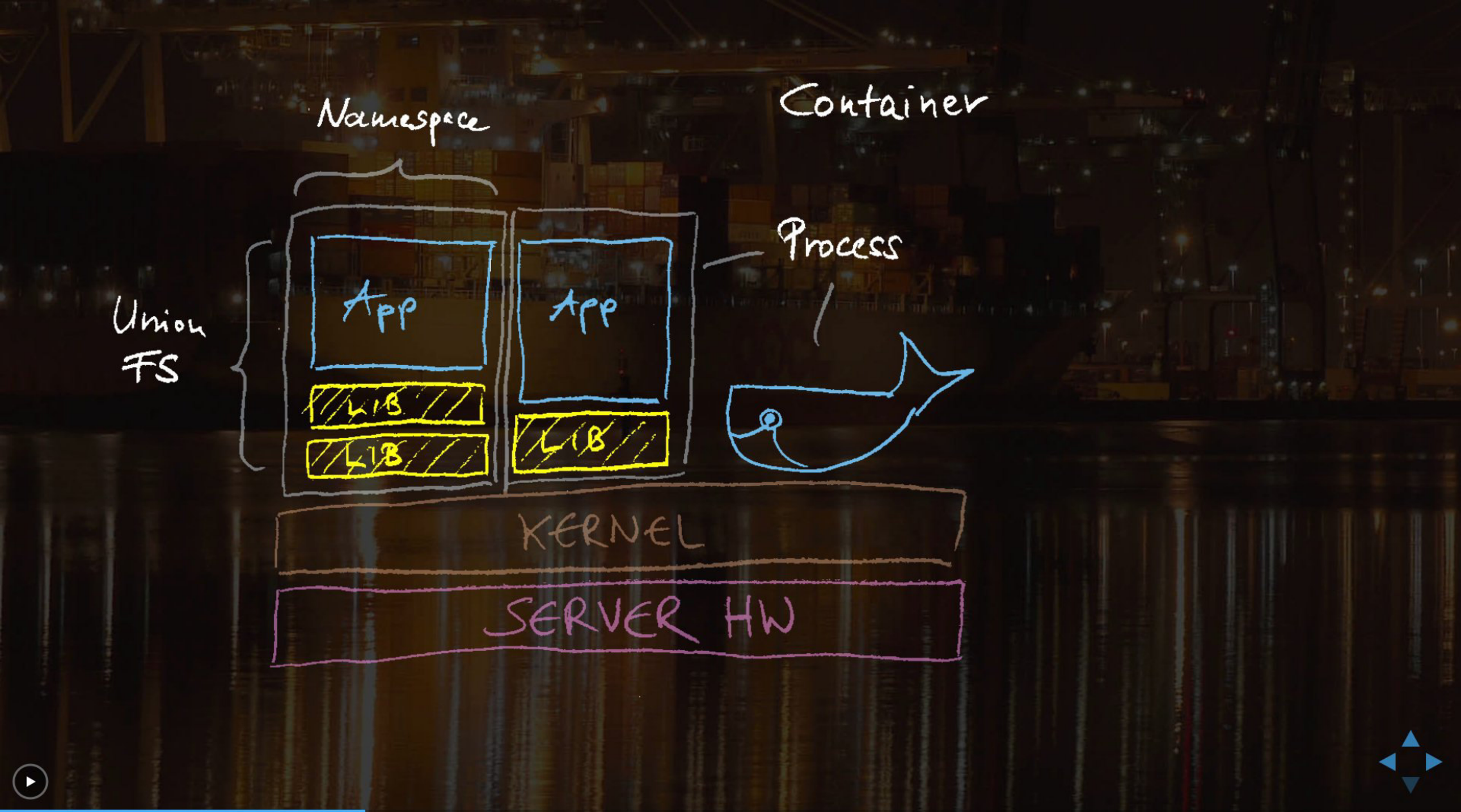
Image: Paketierte Zusammenstellung von Dateien. Damit eine *schreibgeschützte Blaupause* für eine Systemumgebung.

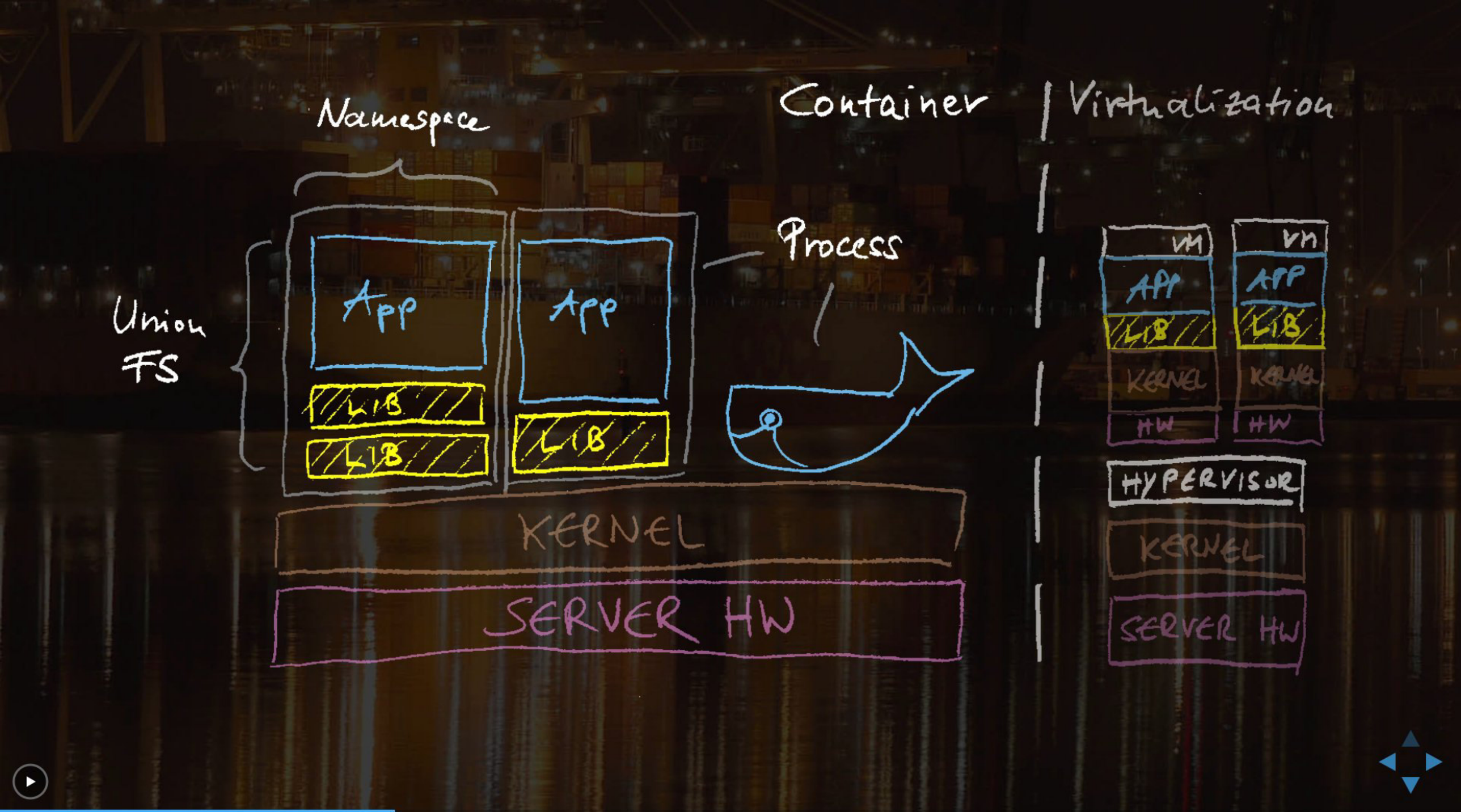
Container: Eine unabhängige Systemumgebung basierend auf einem Image

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

Docker Komponenten



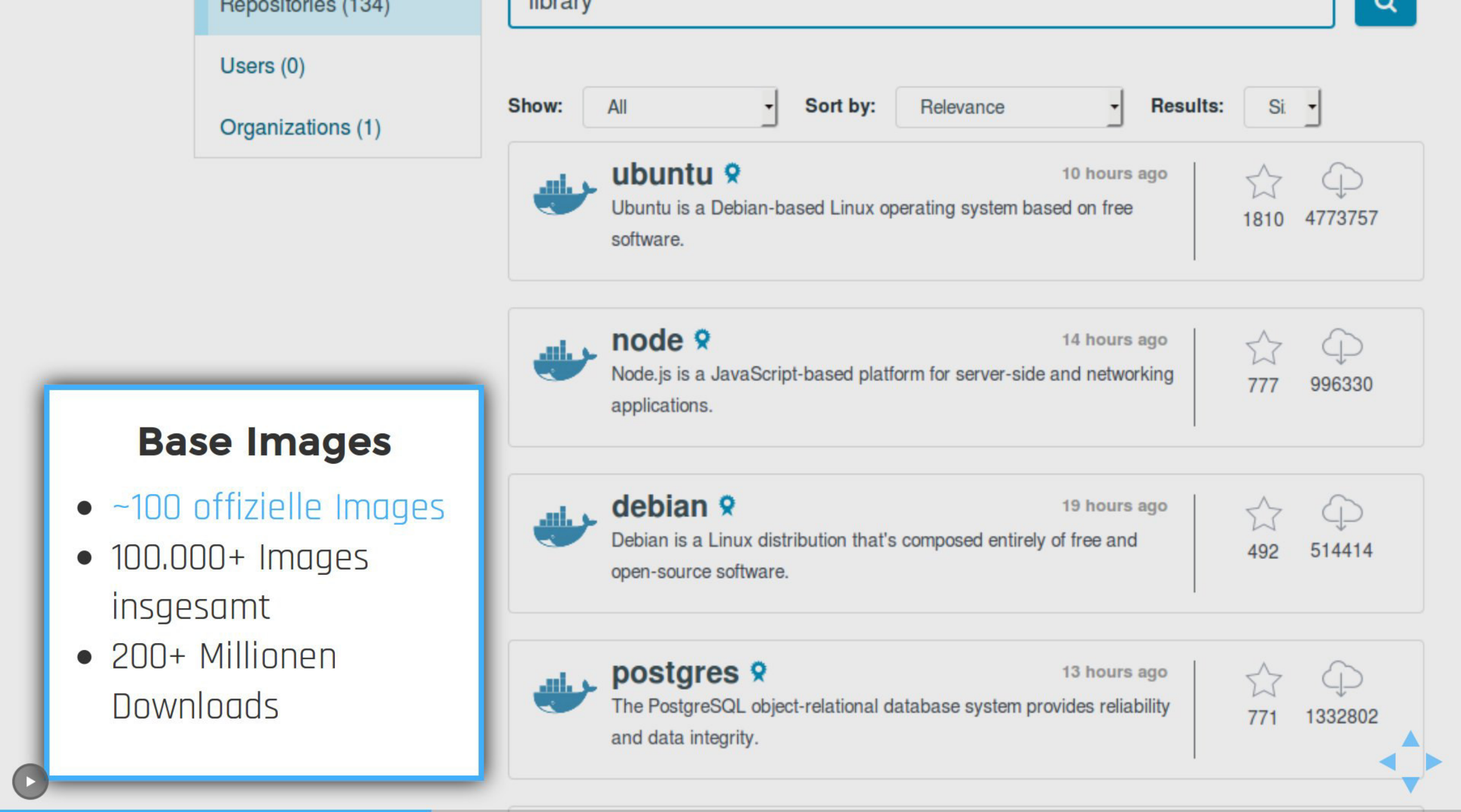

Namespace
Container
Process
Union FS
App
App
LIB
LIB
LIB
KERNEL
SERVER HW


Namespace
Container
Virtualization
Union FS
App
App
LIB
LIB
LIB
Process
KERNEL
SERVER HW
VM
VM
App
App
LIB
LIB
KERNEL
KERNEL
HW
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
~100 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`~~ .

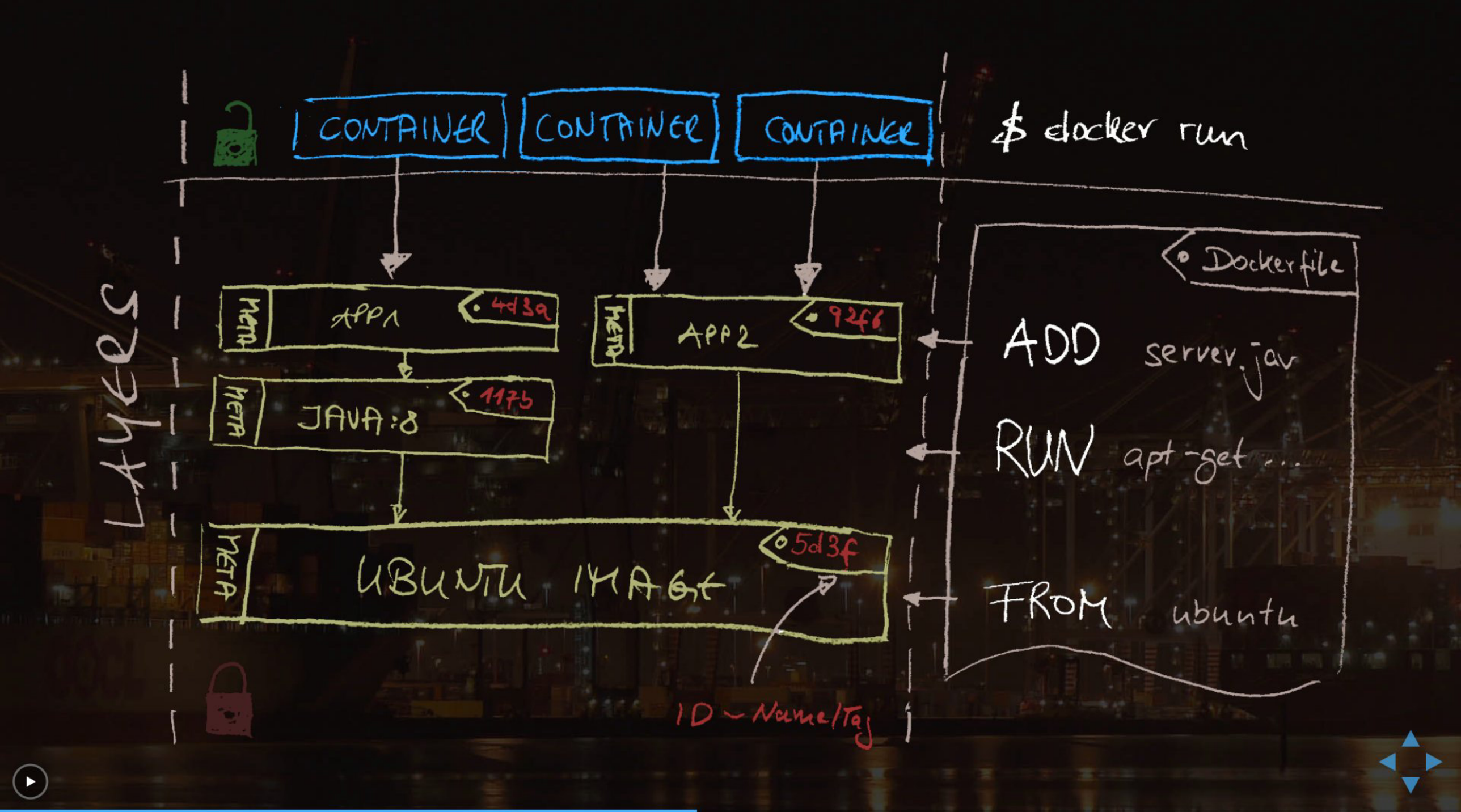
```
docker stop name
docker ps
docker ps -as
docker start name
```

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
4d3a
APP2
92f6
JAVA:8
1175
UBUNTU IMAGE
5d3f
Dockerfile
ADD server.jar
RUN apt-get ...
FROM ubuntu
ID ~ Name/Tag

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
```

CONTAINER VERKNÜPFEN

```
docker run -d --name web1        my/dockerapp
docker run -d --link web1:web1   haproxy
```

web1 wird damit in *haproxy* verfügbar via

- Umgebungsvariablen: `WEB1_PORT`
- IP & Hostnamen: `/etc/hosts`

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

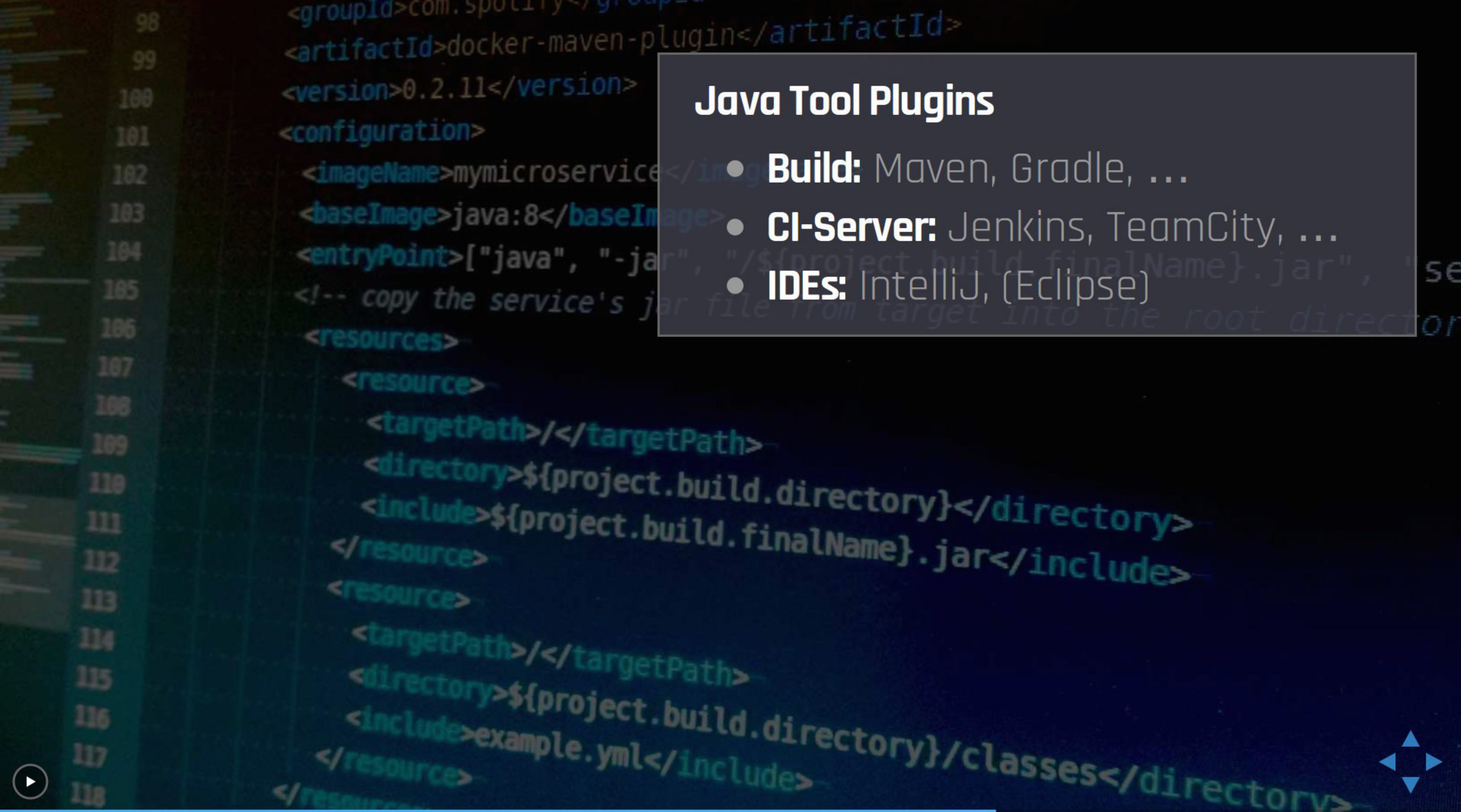
Spark - A tiny Sinatra inspired framework for creating web applications in Java 8 with minimal effort

Quick start

```
import static spark.Spark.*;

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

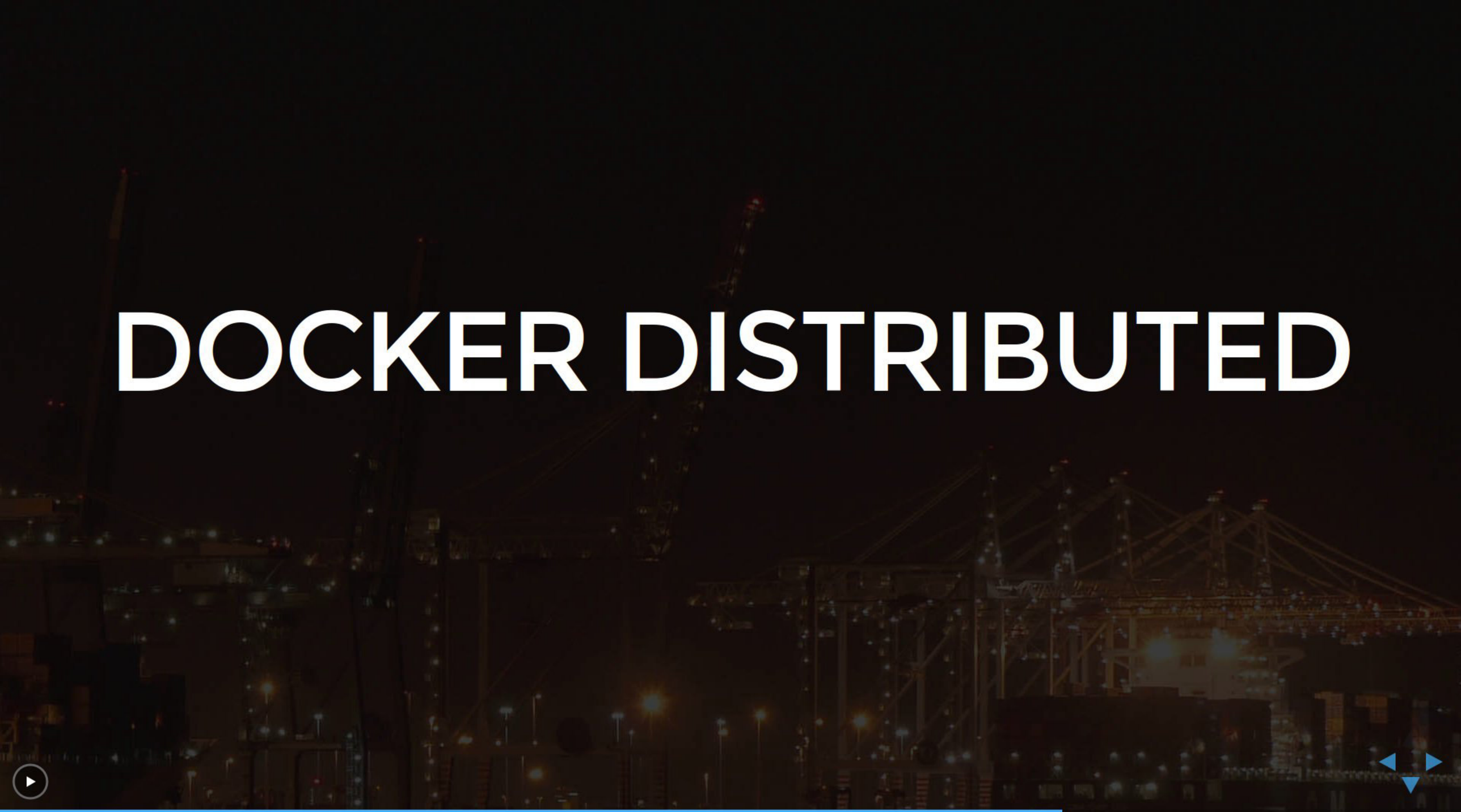
Java Tool Plugins

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



BEST PRACTICES
für Docker Images

- **Immutable infrastructure**
- Konfiguration extern halten
- Kein blindes `docker pull`
- Ein Prozess pro Image
- `docker exec` statt ssh-Server
- Logging nach `STDOUT` / `STDERR`


DOCKER DISTRIBUTED

ORCHESTRIERUNG - DOCKER COMPOSE

```
haproxy:
  image: haproxy:1.5
  ports:
    - 8080:80
  volumes:
    - ./etc/haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
  links:
    - web1
    - web2
    - web3

web1:
  image: exxcellent/docker_talk
```

CLUSTER-BETRIEB

- Verteilung & Scheduling
- Service Discovery
- Load Balancing

DOCKER OUT OF THE BOX

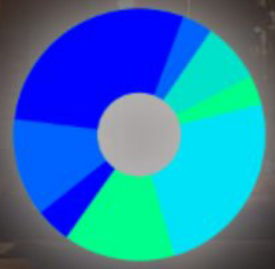
Pures `localhost`: (bis Docker 1.8)

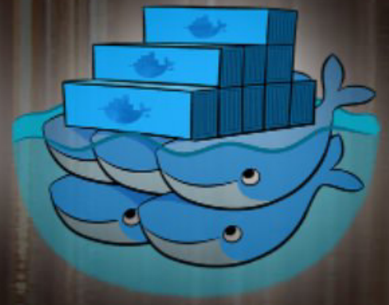
- `--link` und Volumes nur auf lokalem Host
- Kein cross-host Netzwerk

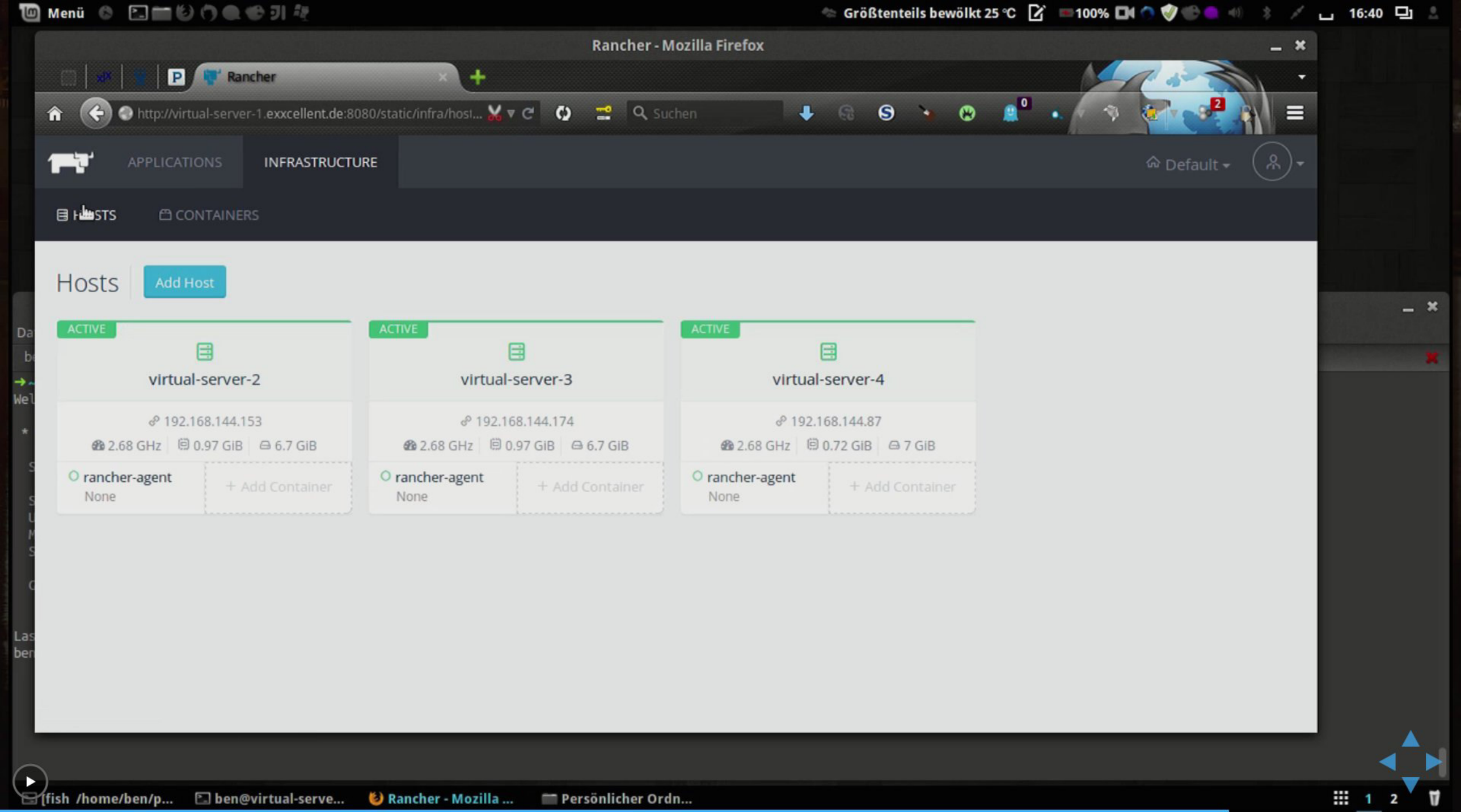
Add-on Tools

- Unübersichtliches, dynamisches Ökosystem
- Sehr oft nur Einzelteile einer Gesamtlösung
- Eher fragil; volles Verständnis notwendig

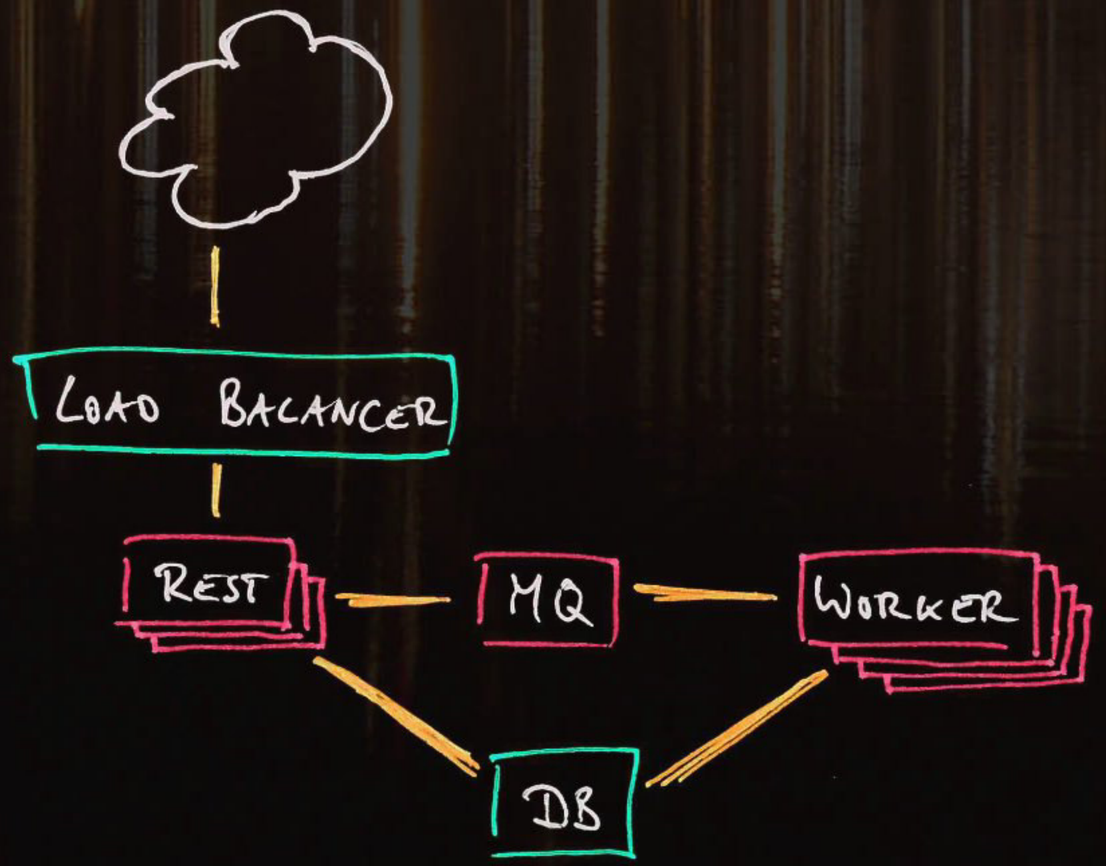
TOOLS






Menü
Größtenteils bewölkt 25 °C
100%
16:40
Rancher - Mozilla Firefox
Rancher
http://virtual-server-1.exxcellent.de:8080/static/infra/hosl...
Suchen
APPLICATIONS
INFRASTRUCTURE
Default
HOSTS
CONTAINERS
Hosts
Add Host
ACTIVE
virtual-server-2
192.168.144.153
2.68 GHz
0.97 GiB
6.7 GiB
rancher-agent
None
+ Add Container
ACTIVE
virtual-server-3
192.168.144.174
2.68 GHz
0.97 GiB
6.7 GiB
rancher-agent
None
+ Add Container
ACTIVE
virtual-server-4
192.168.144.87
2.68 GHz
0.72 GiB
7 GiB
rancher-agent
None
+ Add Container
[fish /home/ben/p...
ben@virtual-serve...
Rancher - Mozilla ...
Persönlicher Ordn...
1
2

BEISPIEL - VERTEILTES SYSTEM




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

GUIs

- Docker Kitematic
- Panamax, ...

Spezielle Linux-Distros

- RancherOS
- CoreOS, ...

Private Image Registry

- Docker Distribution
- Artifactory, ...

Networking

- Weaveworks
- CoreOS Flannel, ...

Service discovery

- CoreOS etcd
- Apache Zookeeper
- Consul

Cluster / Orchestrierung

- Rancher
- Kubernetes
- Shipyard
- Docker Compose / Swarm
- Apache Mesos / Marathon
- MaestroNG
- CoreOS fleet, ...

Config Mgmt.

- Ansible
- Puppet
- Chef
- Capistrano

Interessante Java-Tools

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