



# Docker for Developers

Dr. Roland Huß, ConSol\* Software GmbH  
JBoss OneDayTalk, 29.9.2014, Germering

# Agenda

- **Introduction into Docker**
  - What is Docker ?
  - Demo
  - Docker Advanced
  - Building Images
- **Docker for Developers**
  - Integration Tests
  - Deployment
  - Build Integration
  - Demo

# Roland Huß

ro14nd @



- Head of Research & Development
- Java Dev and Software Architect
- Open Source
  - [www.jolokia.org](http://www.jolokia.org)
  - [labs.consol.de](http://labs.consol.de) & [ro14nd.de](http://ro14nd.de)
  - <https://github.com/rhuss>
- Conference Speaker
  - JavaZone 2014
  - W-JAX 2014
  - Devovx 2014



**ConSol**   
Consulting & Solutions

# Docker

Docker is an open platform for **developers** and **sysadmins** to **build**, **ship**, and **run** distributed applications.

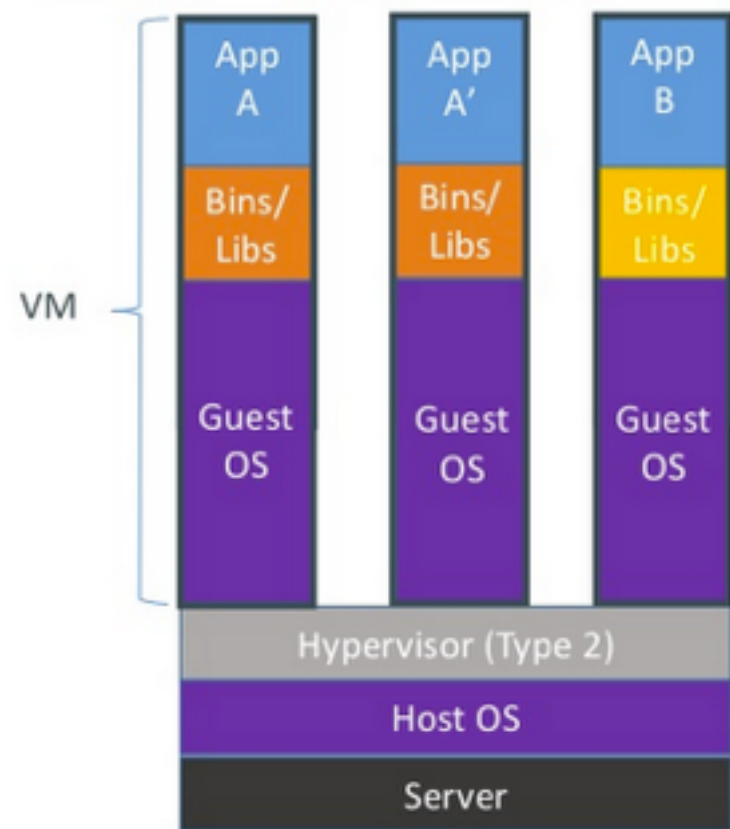
*[docker.io](https://docker.io)*



# Docker is ....

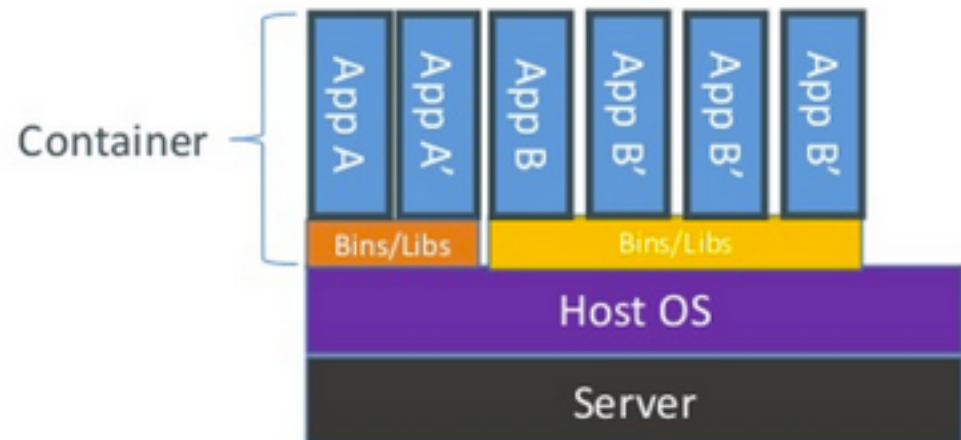
- ... a **lightweight** Linux-Container
- ... **portable**
  - VM, Cloud (Open-Stack, GCE, ...), bare Metal, ...
- ... **very fast** and **scalable**
  - Laptop: 10 - 100, Server: 100 - 1000 Container
- ... **scriptable**
  - via Dockerfiles
- ... „**social imaging**“
  - Image sharing via Registries

# Lightweight Container vs. VM



Containers are **isolated**,  
but sharing the Kernel and  
(some) Files

→ faster & lighter



# Docker Architecture

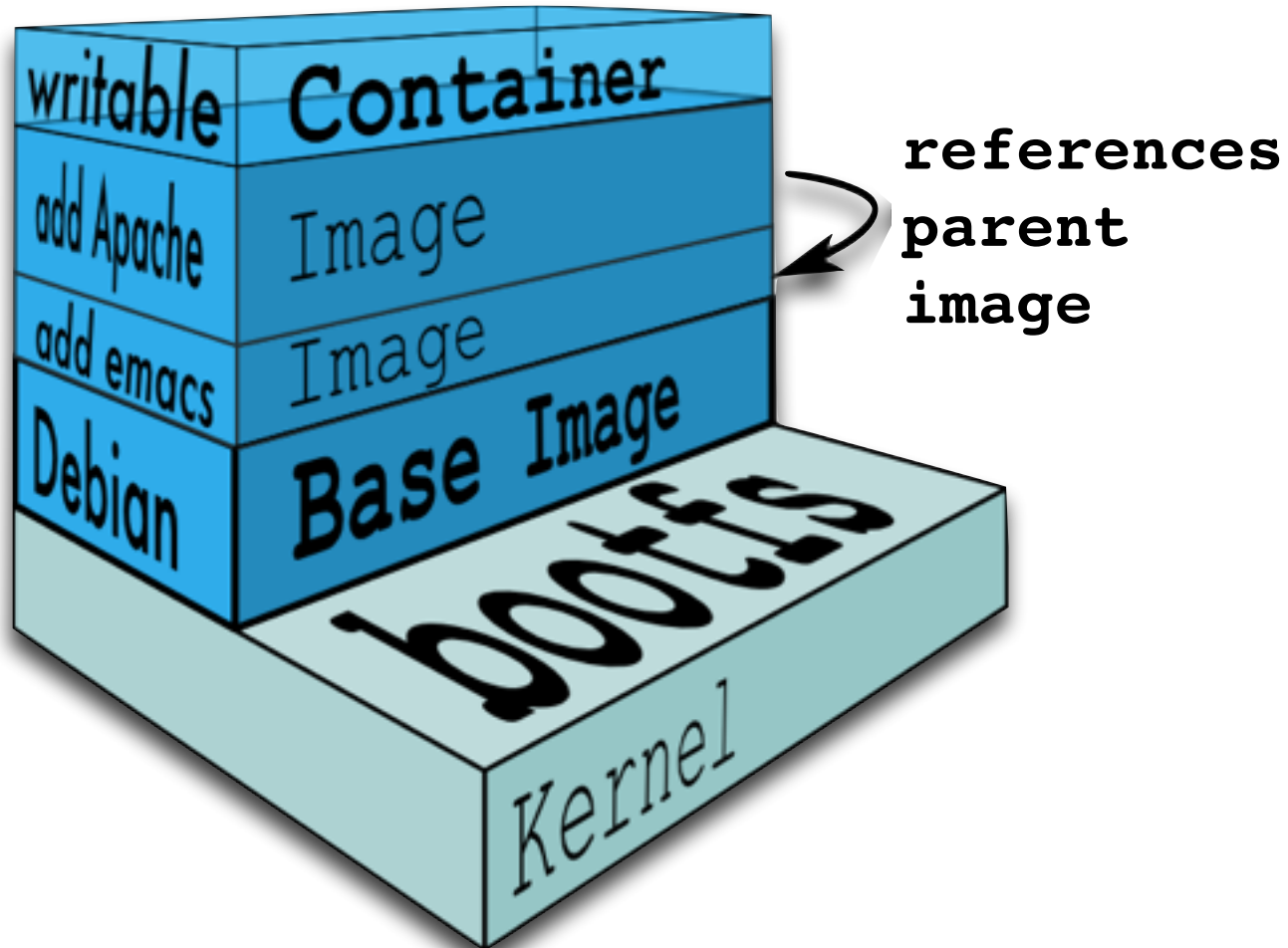
- Written in **Go** (current version: 1.2.0)
- Originally based on **Linux Container (LXC)**
  - since 0.90 : Own abstraction with **libcontainer**
- **Client-Server** Architecture
  - Server communicates via Unix- or INET-Sockets with a REST API
  - Docker Commandos vial CLI (Linux, OS X and Windows)

# Terms

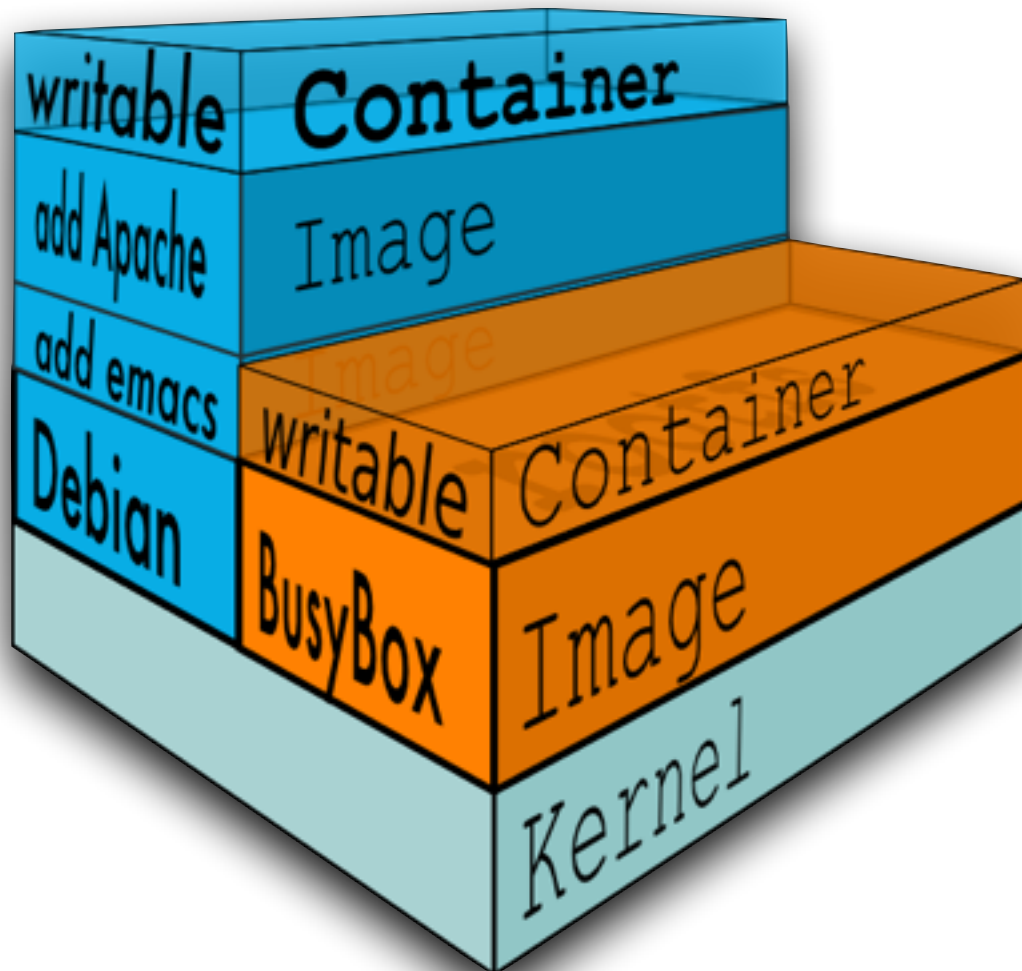
- **Image :**
  - read-only filesystem layer
  - can be deployed and shared
  - "Blueprint for Container"
- **Container:**
  - read-write filesystem layer on top of Image
  - copy on write (stateful)
  - can be started and stopped
  - can be committed to create an Image
  - typically short-lived
  - "Instance of an Image"



# Image



# Container



# Terms

- **Repository** : Collection of layered Images
  - Image name + Tag

```
[user_name]/repository_name[:version_tag]
```

- **Registry**: Storage of Repositories
  - Default: **index.docker.io:80**
  - Own private Registry can be easily used
    - <https://github.com/docker/docker-registry>

# Docker Commands

<b>ps</b>	Show all containers
<b>images</b>	Show all images
<b>run</b>	Create and run a container
<b>search</b>	Search for images on a registry
<b>pull</b>	Download of images
<b>rm</b>	Remove container
<b>rmi</b>	Remove image

Demo

# Port Mapping

- Containers can **expose** Ports
  - specified during build time
- Exported ports can be mapped to host ports.

<b>docker run -P</b>	Maps all exposed container ports dynamically to host ports in 49000 ... 49900
<b>docker run -p 8080:8080 -p 2200:22</b>	Maps container ports 8080 and 22 to host ports 8080 and 2200
<b>docker run -p 8080 -p 22</b>	Maps container ports 8080 and 22 dynamically to host ports from 49000 ... 49900

# Volumes

- Sharing of file data between ...
  - ... Container and Container
  - ... Container and Docker Host

```
docker run -v /var/volume1 \  
          -v /var/volume2 \  
          --name DATA busybox true  
docker run -t -i --rm \  
          --volumes-from DATA \  
          --name client1 ubuntu bash
```

# Container Linking

- Naming a container during startup:

```
docker run -d --name redis crosby/redis
```

- Reference container via name:

```
docker run -t -i --link redis:db ubuntu bash
```

- Connection information to referenced container

- via `/etc/hosts`

```
172.17.0.3      db
```

- via environment variables

```
DB_PORT_5432_TCP=tcp://172.17.0.3:5432
```

```
DB_ENV_PG_VERSION=9.3.5-1.pgdg70+1
```

```
.....
```



# Building Images - Run & Commit

- Select a base image
  - **`docker run -t -i ubuntu bash`**
- Installation of software, etc within container
- Stop container
  - **`docker commit <container-id> <image>`**
  - **`docker tag <image> <repository>`**
  - **`docker push <user-name>`**

# Building Images - Dockerfile

```
FROM dockerfile/java
```

```
MAINTAINER roland@jolokia.org
```

```
EXPOSE 8080
```

```
RUN wget http://archive.apache.org/tomcat-7/.. -O /tmp/c.tgz
```

```
RUN tar xzf /tmp/c.tgz -C /opt
```

```
RUN rm /tmp/c.tgz
```

```
CMD [ "/opt/apache-tomcat-7/bin/catalina.sh", "run" ]
```

```
docker build -t jolokia/tomcat-7 .
```

Demo

# Docker for Java Developers ?

Boost your Integration Tests

Ship your Applications

# Integration Tests

Integration tests exercise applications within a **realistic context** that is as close as possible to the **production** environment.

# Integration Tests

- Good Integration Tests are ...
  - **Robust** (aka Repeatable)  
Work always or fail always with the same error
  - **Independent**  
Minimal external requirements, self-contained
  - **Isolated**  
Parallel executions of similar tests
  - **Fast**  
Tight feedback loop

# External Testsystems

<del>Robust</del>	Test system are externally managed and configured.
<del>Independent</del>	Test systems must be installed, available and running.
<del>Isolated</del>	Parallel tests access the same test systems and might interfere.
<del>Fast</del>	Often slow because of network latency and parallel usage.

but **close** to the real thing !

# Simulated (Mock) Testsystems

<b>Robust</b>	Can be started during the test run
<b>Independent</b>	Can be configured declaratively (e.g. Citrus)
<b>Isolated</b>	Different ports can be selected via configuration
<b>Fast</b>	Speed depends on framework and setup

but ***not*** the real thing !



# Docker to the rescue

<b>Robust</b>	Each build has its dedicated container and hence its own distinguished execution context.
<b>Independent</b>	No build external requirements except a Docker installation required.
<b>Isolated</b>	Perfect isolation for the System-Under-Test.
<b>Fast</b>	Fast container startup because of OS level virtualization.

and it ***can*** be the real thing !

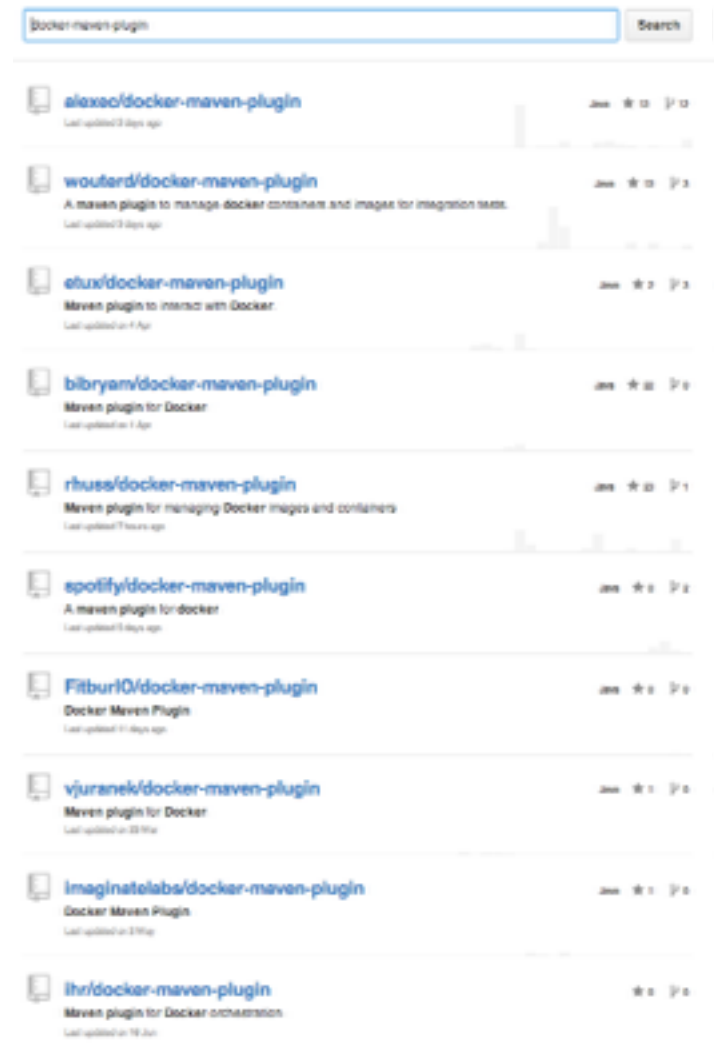
# Shipping Applications

- Data Container:
  - Application artifacts are stored in **data containers**
  - Data containers are linked to **platform container**
  - Application gets deployed during startup
- Service Container:
  - Artifacts and application server are stored in the **same container**.
  - Ideal for Microservices.

# Docker Build Integration

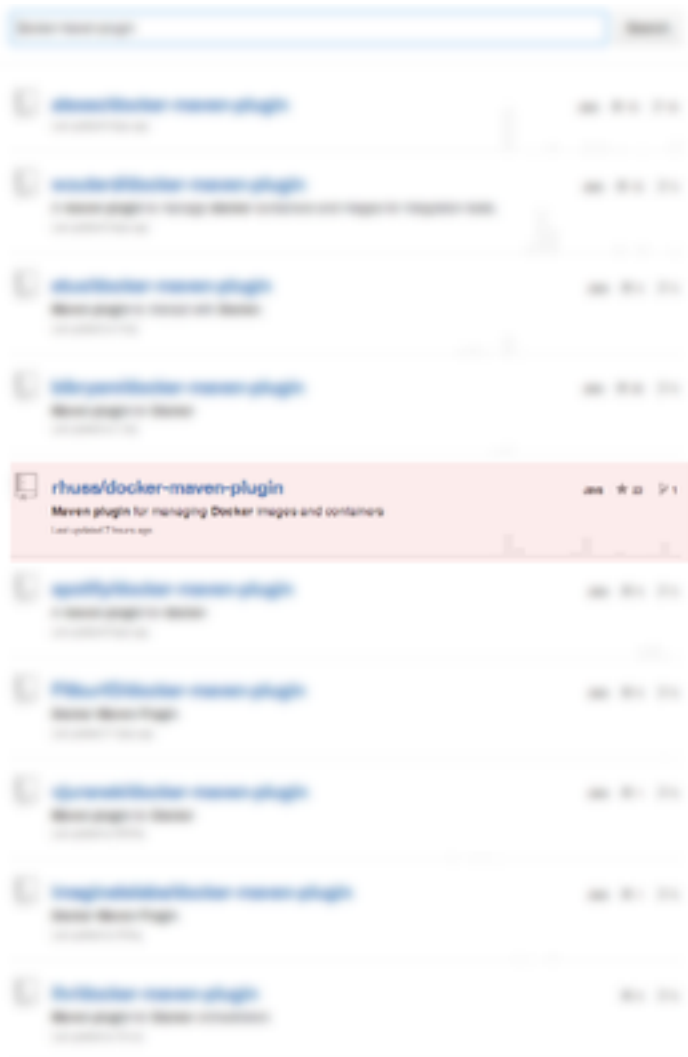
- CI Server
  - Pre and Post-Hooks for starting and stopping Docker containers
- Calling Docker CLI from build
  - `exec` Task in Ant
  - `exec-maven-plugin` for Maven
  - via Groovy in Gradle
- Dedicated Maven and Gradle Plugins

# docker-maven-plugin



WTF or  
FTW ?

# docker-maven-plugin



rhuss/docker-maven-plugin

# rhuss/docker-maven-plugin

- **Simple** configuration
- Automatic **pull** of required images
- Dynamic **portmapping**
- Maven **artifacts** and their dependencies should be available within the container
- **Pushing** containers to a registry
- Doing it the **Maven** way

# Supported goals

<b>docker:start</b>	Start a container (pre-integration-test)
<b>docker:stop</b>	Stop a container (post-integration-test)
<b>docker:build</b>	Build a data image
<b>docker:push</b>	Push data image to registry

# Configuration

```
<configuration>

  <image>consol/tomcat-7.0</image>

  <ports>
    <port>jolokia.port:8080</port>
  </ports>

  <waitHttp>http://localhost:${jolokia.port}/jolokia</waitHttp>
  <wait>10000</wait>

  <assemblyDescriptor>src/main/assembly.xml</assemblyDescriptor>
  <dataImage>jolokia/agents</dataImage>
  <mergeData>false</mergeData>
</configuration>
```



# Data Image

```
<assembly>
  <dependencySets>
    <dependencySet>
      <includes>
        <include>org.jolokia:jolokia-war</include>
      </includes>
      <outputDirectory>.</outputDirectory>
      <outputFileNameMapping>jolokia.war</outputFileNameMapping>
    </dependencySet>
  </dependencySets>
</assembly>
```

# Data Image

- Assembly Descriptor from **maven-assembly-plugin**
- Predefined descriptors
- Data image exports **/maven** as Docker volume
- **mergeData**
  - **false**: Assembly gets own container
  - **true**: Assembly is merged into given image

Demo

# docker-maven-plugin

	<i>wouterd</i>	<i>alexec</i>	<i>spotify</i>	<i>rhuss</i>
<i>API</i>	jaxrs	docker-java (forked)	spotify/docker- client	UniREST
<i>Start/Stop</i>	✓	✓	✗	✓
<i>Building</i>	✓	✓	✓	✓
<i>Data Image</i>	Dockerfile + Maven Config	Dockerfile + customYML	Maven config	Maven config + Assembly
<i>Push</i>	✓	✓	✓	✓

# docker-maven-plugin

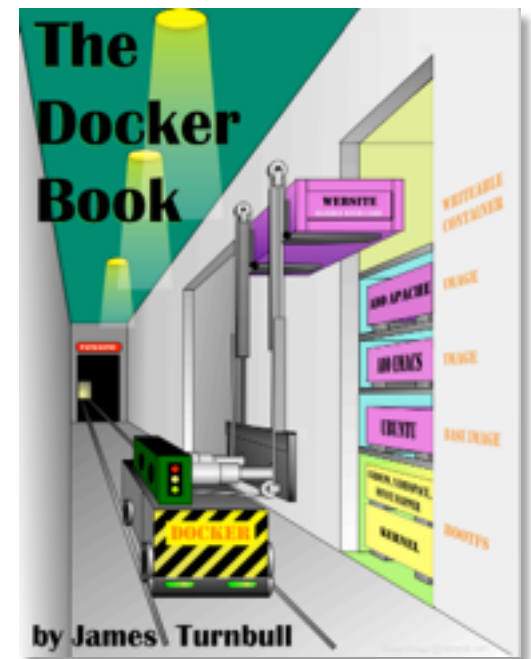
	<i>wouterd</i>	<i>alexec</i>	<i>spotify</i>	<i>rhuss</i>
<i>Cleanup</i>	✓	✓	✗	✓
<i>Security</i>	Plain	Plain	✗	Encrypted/ Plain
<i>URL Wait</i>	✗	✓	✗	✓
<i>Version</i>	2.1	2.0.0	0.0.21	0.9.9
<i>Size LOC</i>	2100	1000	600	1500

# Wrap up

- Docker is a lightweight virtualization technology which can improve the development process:
  - It can help building **good integration tests**
  - It introduces a new paradigm for **shipping applications**.
- Build support is growing but still in the beginning.

# Resources

- [index.docker.io](http://index.docker.io) - Public Docker Registry
- „Docker Introduction“ by Palo Alto
  - 91 pages full of technical Details
  - <http://bit.ly/RlrznC>
- „Are VM Passé?“ by Ben Golub
  - Management Overview „Why Docker ?“
  - <http://bit.ly/1kWxJaL>
- Slidedeck
  - <http://ro14nd.de/talks/2014/docker-onedaytalk.pdf>
- "The Docker Book"
  - highly recommended !
  - <http://www.dockerbook.com/>



# Thank you!

```
docker_nuke() {  
    docker ps -q | xargs docker stop  
    docker ps -q -a | xargs docker rm  
}  
  
docker_rmi_none() {  
    docker images | grep '<none>' | \  
    awk '{ print $3 }' | \  
    xargs docker rmi  
}  
  
docker_go() {  
    docker run --rm -t -i $@  
}
```



# ConSol\* Software GmbH

Franziskanerstraße 38  
D-81669 München

Tel: +49-89-45841-100

Fax: +49-89-45841-111

[info@consol.de](mailto:info@consol.de)

[www.consol.de](http://www.consol.de)

**WORKED FINE IN DEV**



**OPS PROBLEM NOW**