



Business  
Technology|Days

BIG  
DATA  
CON

Dr. Roland Huß, ConSol\* (@ro14nd)

Docker für Java-Entwickler

# Agenda

- **Docker Crash Intro**
- **Docker für Java Entwickler**
  - Integrationstests
  - Anwendungs-Paketierung
  - docker-maven-plugin
  - Demo

# Roland Huß

- Java Entwickler @ ConSol\*
- Open Source
  - [www.jolokia.org](http://www.jolokia.org)
  - [labs.consol.de](http://labs.consol.de)
  - <https://github.com/rhuss>
- Konferenzen
  - JavaZone 2014
  - W-JAX 2014
  - Devovx 2014
  - JavaLand 2015

rol4nd



ConSol   
Consulting & Solutions

# Docker

Docker ist eine offene Plattform für **Entwickler** und **Administratoren**, um verteilte Applikationen zu **bauen**, **auszuliefern** und zu **betreiben**.

*docker.io*

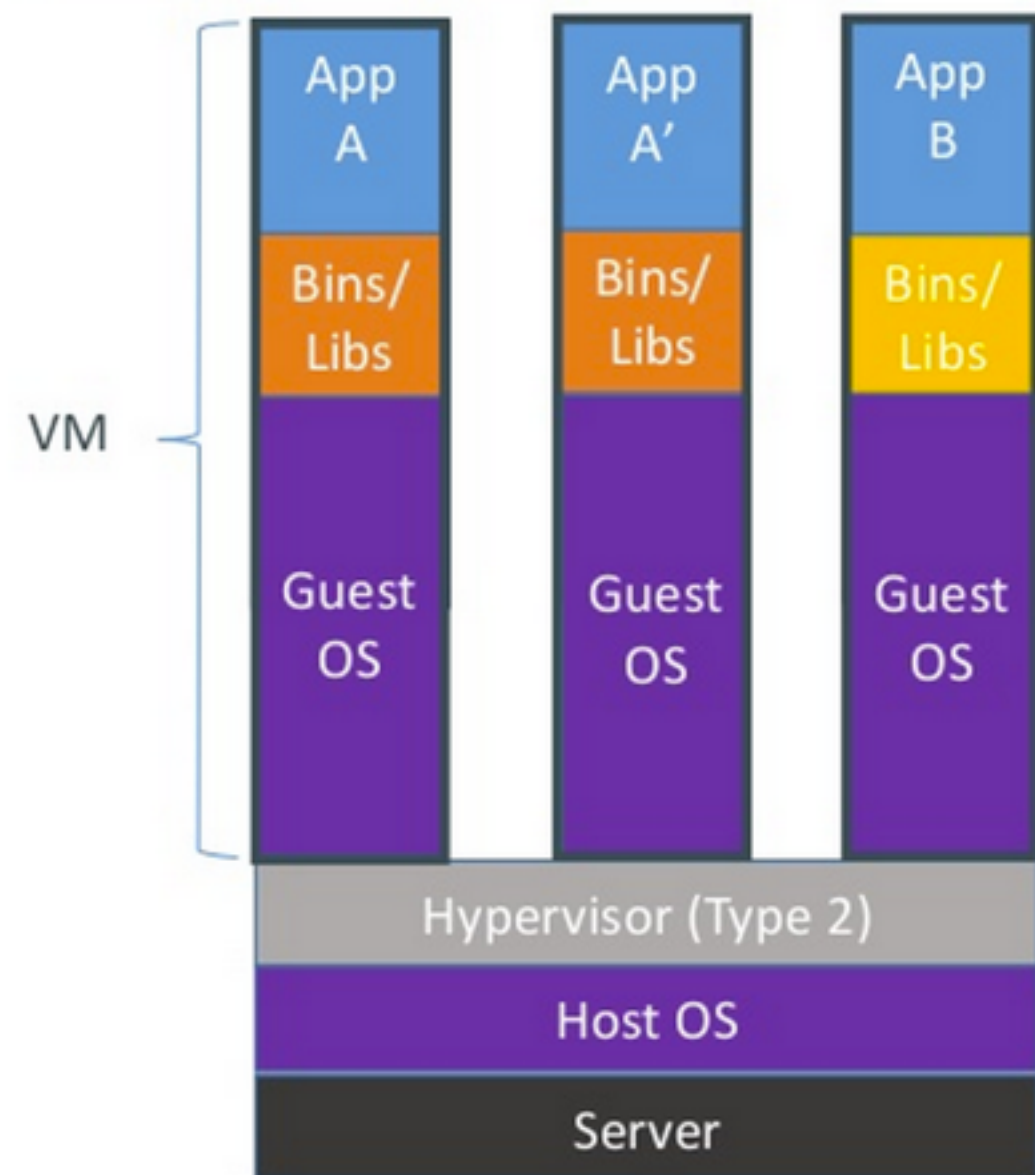


docker

Business  
Technology|Days

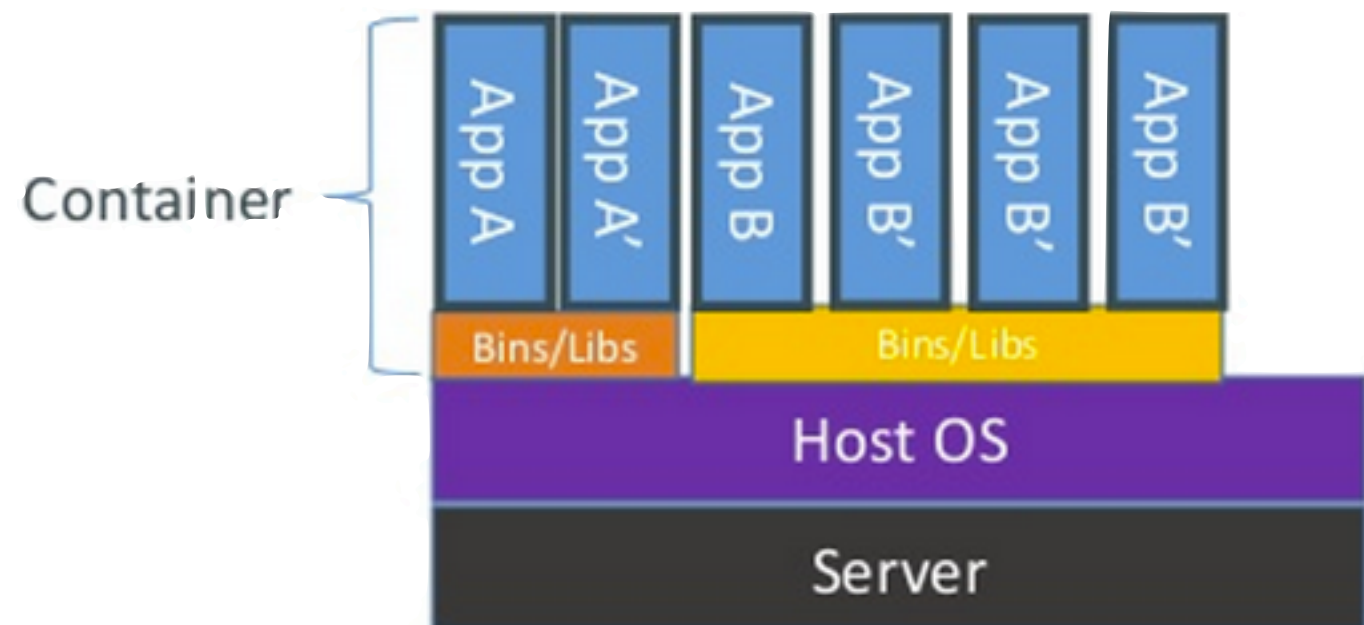
BIG  
DATA  
CON

# Container versus VM



Container sind isoliert, teilen sich aber den Kernel und (einige) Dateien

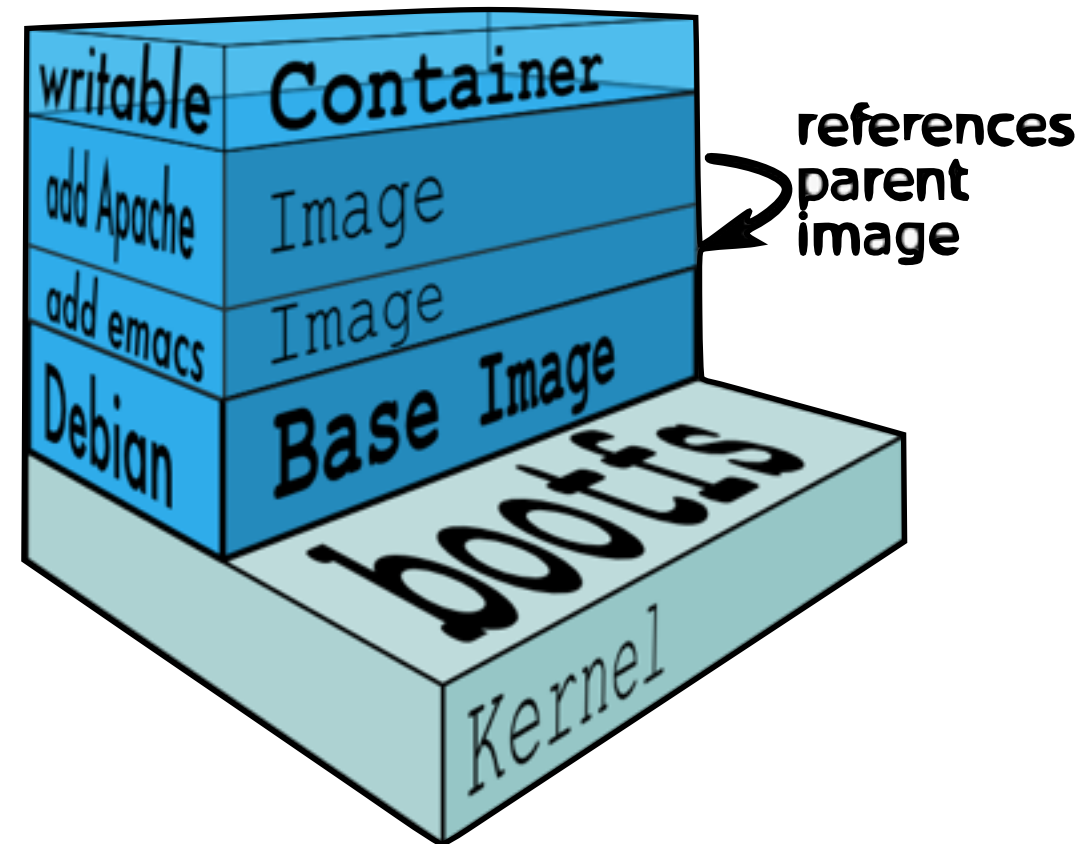
→ schneller und sparsamer





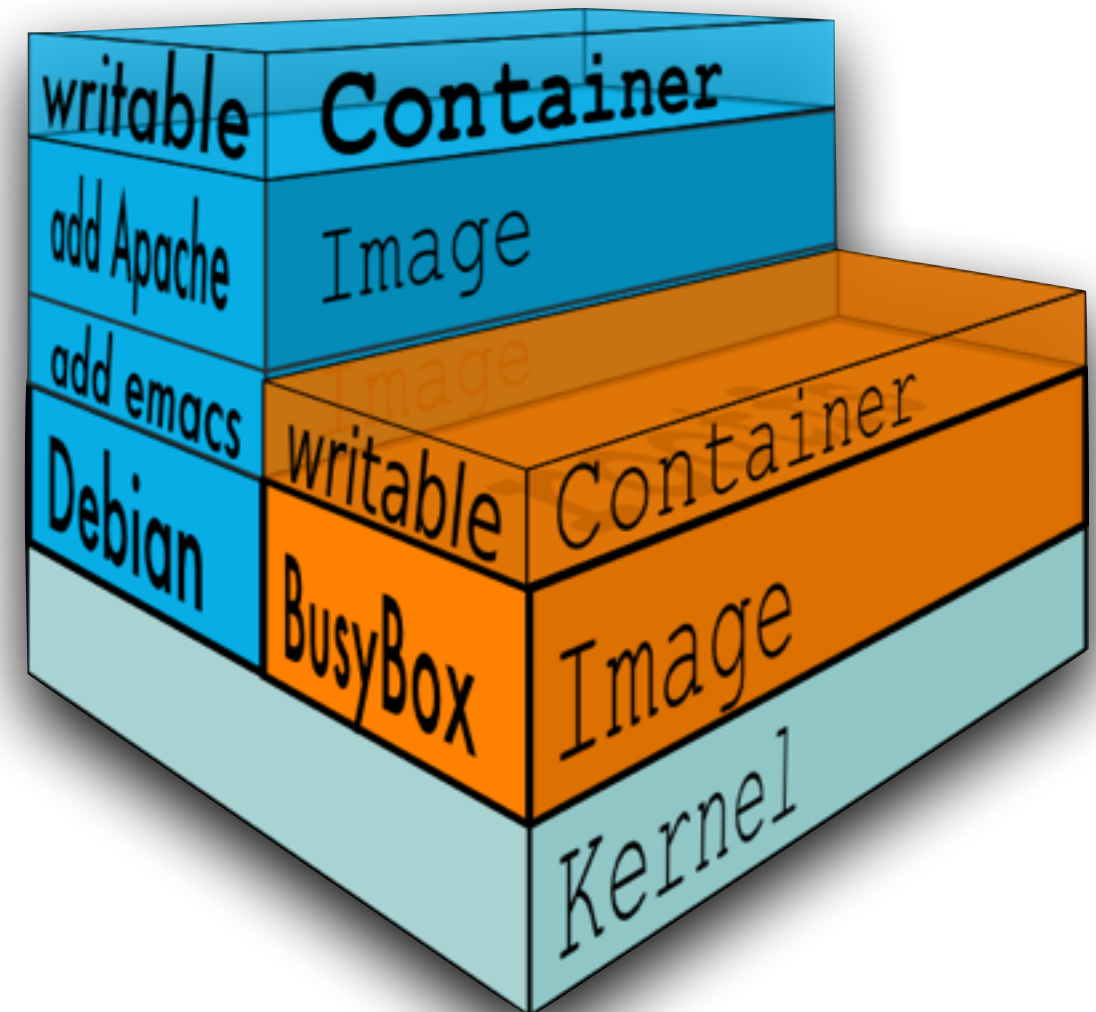
# Image & Repository

- Image: *read-only* Dateisystem Schicht
- Repository: Sammlung von Images
- Repository kann installiert und verteilt werden (z.B. über eine Registry)
- "Blaupause für einen Container"



# Container

- *read-write* Schicht über den Images
- Copy-On-Write
- kann gestartet und gestoppt werden
- "Instanz eines Repositories"



# Docker CLI Kommandos

<b>ps</b>	Anzeigen erzeugter Container
<b>images</b>	Anzeigen lokaler Images
<b>run</b>	Erzeugen und Starten von Containern
<b>search</b>	Suchen von Images in einer Registry
<b>pull</b>	Download von Images
<b>rm</b>	Entfernen eines Containers
<b>rmi</b>	Entfernen eines Images
<b>exec</b>	Ausführen eines Kommandos im Container



# Erweiterte Konzepte

- **Port Mapping**
  - Container Ports können flexibel exportiert werden
- **Container Linking**
  - Netzwerkverknüpfung zwischen lokalen Containern
- **Volumes**
  - Mounten lokaler Verzeichnisse
  - Mounten von Verzeichnissen zwischen Containern
- **Eigene Images**
  - Skriptbar via Dockerfiles

# Docker für Java Entwickler ?

Integrationstests

Applikations Deployment

# Integrationstests

Integrationstests prüfen Applikationen in einem **realistischen Kontext**, der der **Produktionsumgebung** so nahe wie möglich kommt.

# Integrationstests

- Gute Integrationstests sind ...
  - **Robust** (aka wiederholbar)  
Laufen entweder immer durch oder schlagen immer mit dem gleichen Tests fehl
  - **Autark**  
Minimale externe Abhängigkeiten, eigenständig
  - **Isoliert**  
Parallele Ausführung ähnlicher Tests
  - **Schnell**  
Kurze Turnaround-Zeiten

# Externe Testsysteme

<del>Robust</del>	Test Systeme werden extern verwaltet und konfiguriert.
<del>Autark</del>	Test Systeme müssen installiert werden und verfügbar sein.
<del>Isoliert</del>	Parallele Tests greifen auf das gleiche System zu und stören sich gegenseitig.
<del>Schnell</del>	Test Systeme sind aufgrund paralleler Nutzung und Netzwerklatenz oft langsam.

aber sind *nahe* an der Realität !



# Simuliertes (Mock) Testsystem

<b>Robust</b>	Kann während des Testlaufs gestartet werden.
<b>Autark</b>	Kann deklarativ konfiguriert werden (z.B. Citrus).
<b>Isoliert</b>	Verschiedene Ports pro Testlauf können konfiguriert werden.
<b>Schnell</b>	Mock Systeme sind aufgrund begrenzter Funktionalität oft schneller.

aber entsprechen ***nicht*** der Wirklichkeit !

# Integrationstests mit Docker

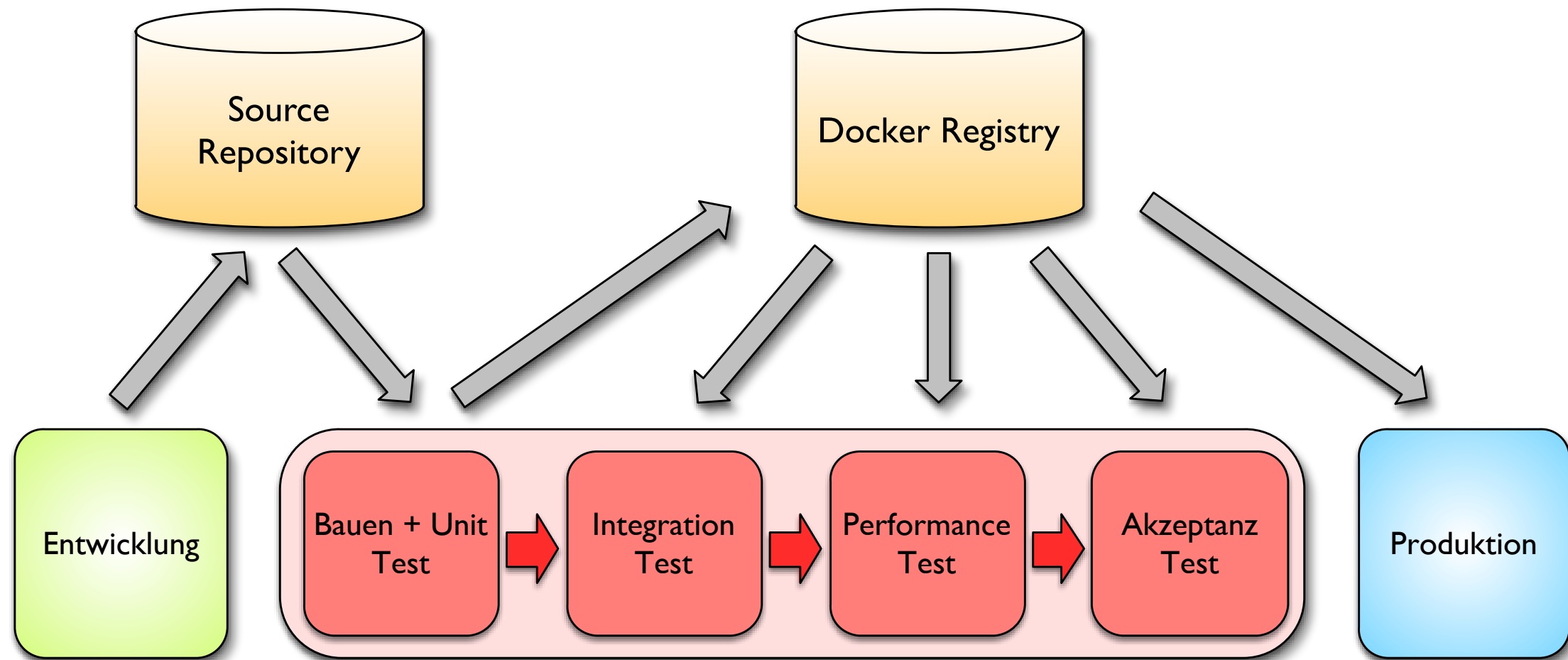
<b>Robust</b>	Jeder Testlauf hat einen eigenen Container und Ausführungskontext.
<b>Autark</b>	Außer einer Docker Installation keine externen Abhängigkeiten.
<b>Isoliert</b>	Perfekte Isolation der Container möglich.
<b>Schnell</b>	Schneller Container Start dank der Systemlevel Virtualisierung.

und ***können*** die Realität abbilden !

# Applikations Deployment

- Standard "Container" Formate für Java Enterprise Anwendungen:
  - Web-Archive (**WAR**)
  - Enterprise-Archive (**EAR**)
- Mit Docker wird der **Ausführungskontext** (Server) mit in den Container gepackt.
- **Immutable Server Pattern**
  - Bei einer neuen Version der Applikation wird auch ein neuer Server deployed.

# Docker Delivery Pipeline



# Container Patterns











- **Datencontainer:**
  - Artefakte werden in einen **Datencontainer** verpackt.
  - Der Datencontainer wird mit einem **Plattformcontainer** verknüpft.
  - Die Applikation wird beim Start des Plattformcontainers deployed.
- **Servicecontainer:**
  - Artefakte und Laufzeitumgebung (z.B. Applikationsserver) werden in den **gleichen Container** gepackt.
  - Ideal für Microservices.



# Build Integration

- CI Server
  - Pre- und Post-Hooks zum Starten und Stoppen von Docker Container.
- Aufruf der Docker CLI aus dem Build heraus:
  - `exec` Ant-Task
  - `exec-maven-plugin` für Maven
  - mit Groovy aus Gradle heraus
- Dedizierte Maven und Gradle Plugins

# docker-maven-plugin

	<a href="#">alexec/docker-maven-plugin</a>	Java	★ 13	🔗 12
Last updated 2 days ago				
	<a href="#">wouterd/docker-maven-plugin</a>	Java	★ 13	🔗 3
A maven plugin to manage docker containers and images for integration tests. Last updated 3 days ago				
	<a href="#">etux/docker-maven-plugin</a>	Java	★ 2	🔗 3
Maven plugin to interact with Docker. Last updated on 4 Apr				
	<a href="#">bibryam/docker-maven-plugin</a>	Java	★ 22	🔗 0
Maven plugin for Docker Last updated on 1 Apr				
	<a href="#">rhuss/docker-maven-plugin</a>	Java	★ 23	🔗 1
Maven plugin for managing Docker images and containers Last updated 7 hours ago				
	<a href="#">spotify/docker-maven-plugin</a>	Java	★ 0	🔗 2
A maven plugin for docker Last updated 9 days ago				
	<a href="#">FitburlO/docker-maven-plugin</a>	Java	★ 0	🔗 0
Docker Maven Plugin Last updated 11 days ago				
	<a href="#">vjuranek/docker-maven-plugin</a>	Java	★ 1	🔗 0
Maven plugin for Docker Last updated on 23 Mar				
	<a href="#">imaginatelabs/docker-maven-plugin</a>	Java	★ 1	🔗 0
Docker Maven Plugin Last updated on 2 May				
	<a href="#">lhr/docker-maven-plugin</a>		★ 0	🔗 0
Maven plugin for Docker orchestration Last updated on 16 Jun				

WTF or  
FTW ?

# Die 4 Überlebenden

- **wouterd/docker-maven-plugin**
  - Wouter Danes, ING
- **alexec/docker-maven-plugin**
  - Alex Collins
- **spotify/docker-maven-plugin**
  - Spotify
- **rhuss/docker-maven-plugin**
  - Roland Huß, ConSol



<https://github.com/rhuss/shootout-docker-maven>

# rhuss/docker-maven-plugin

- Einfache Konfiguration
- Dynamisches Portmapping
- **Assembly**, um Artefakte und deren Abhängigkeiten in den Container einzubinden
- Upload von Containern zu einer Registry
- Automatischer **Download** von Images
- *"Doing it the Maven way"*

# Maven Goals

<b>docker:start</b>	Starten von Containern
<b>docker:stop</b>	Stoppen von Containern
<b>docker:build</b>	Bauen von Images
<b>docker:push</b>	Upload zu einer Registry
<b>docker:remove</b>	Entfernen von Images
<b>docker:logs</b>	Anzeigen der Container Logs



# Beispiel Konfiguration

```
<images>
  <image>
    <name>jolokia/jolokia-itest</name>
    <build>
      <from>consol/tomcat-7.0</from>
      <assemblyDescriptor>assembly.xml</assemblyDescriptor>
    </build>
    <run>
      <ports>
        <port>jolokia.port:8080</port>
      </ports>
    </run>
  </image>
</images>
```

# Assembly Deskriptor

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

# Artefakte im Container

- Assembly Deskriptor des **maven-assembly-plugin**
  - Build Artefakte
  - Abhängigkeiten
  - Beliebige Dateien
- Vordefinierte Deskriptoren
- Daten stehen im Container unter **/maven** zur Verfügung.

# Beispiel Projekt

- Docker Demo Projekt
  - Vanilla PostgreSQL 9 Image
  - HTTP Request Logging Service
    - MicroService mit embedded Tomcat
    - DB Schema wird via Flyway während des Starts gebaut
    - PostgreSQL Container wird über Link angebunden
  - Einfacher Integrationstest, der den Service nutzt
    - REST-assured zum Testen des Service Aufrufes
- Aufruf: `mvn clean install`
- <https://github.com/rhuss/docker-maven-sample>

Demo



# Wormhole Pattern

- Autarke Integrationstest mit Docker möglich
- Idee: Testaufruf selbst in einen Docker Container packen
- **Wurmloch** : `docker-maven-plugin` greift aus dem Container heraus auf den ihn verwaltenden Docker-Daemon zu

# Dockerfile

```
FROM java:7u75

ENV MAVEN_VERSION 3.3.1

ADD cloneAndBuild /cloneAndBuild

RUN wget http://www.eu.apache.org/dist/maven/maven-3/${MAVEN_VERSION}/binaries/apache-maven-${MAVEN_VERSION}-bin.tar.gz -O maven.tgz

RUN tar zxvf maven.tgz
ADD apache-maven-${MAVEN_VERSION} maven

CMD ["clean", "install"]
ENTRYPOINT [ "/cloneAndBuild" ]
```

# cloneAndBuild

```
#!/bin/bash
# Extract Docker host IP from routing table
host=$(ip route show 0.0.0.0/0 | \
    grep -Eo 'via \S+' | \
    awk '{print $2}');
export DOCKER_HOST=tcp://${host}:2375

# Checkout project afresh
git clone https://github.com/rhuss/docker-maven-sample.git

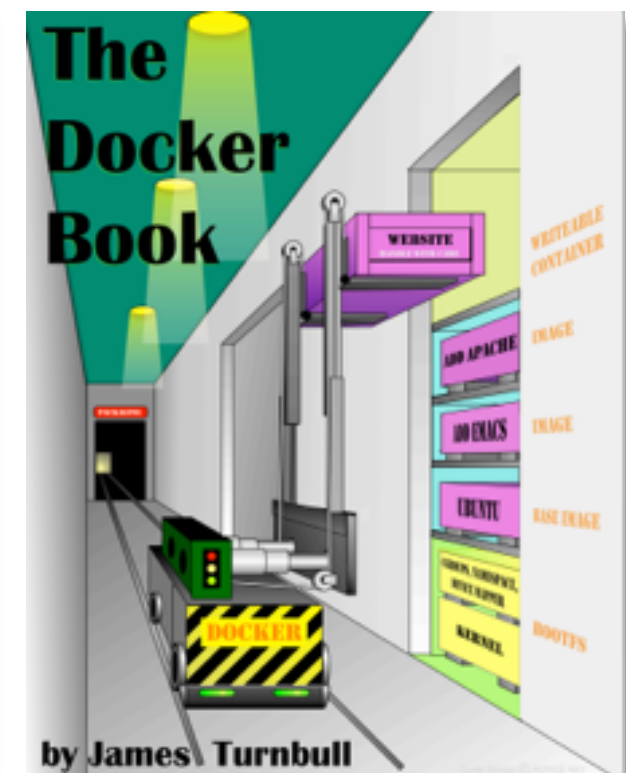
# Call Maven from image with all arguments
/maven/bin/mvn $*
```

# Zusammenfassung

- Docker ist eine leichtgewichtige Virtualisierungstechnik, mit der man:
  - robuste, autarke, isolierte und schnelle **Integrationstests** entwickeln kann.
  - ein neues Paradigma für die **Auslieferung von Applikationen** umsetzen kann.
- Ein komfortabler Weg, Docker in den Java Build Prozess zu integrieren, ist die Verwendung eines **docker-maven-plugins**.

# Referenzen

- index.docker.io - Public Docker Registry
- Entwickler Magazin Spezial Vol.2: Docker
  - [http://entwickler.de/docker\\_spezial](http://entwickler.de/docker_spezial)
- "The Docker Book"
  - sehr zu empfehlen !
  - <http://www.dockerbook.com/>



**Danke !**



# 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)



Business  
Technology|Days

BIG  
DATA  
CON