ConSol Consulting & Solutions

Wir unternehmen IT.

Docker für Java Entwickler

Dr. Roland Huß, ConSol* Software GmbH

JavaLand, 24.3.2015

Agenda

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

Roland Huß

ro14nd @

- Java Developer
- Open Source
  - www.jolokia.org
  - labs.consol.de & ro14nd.de
  - https://github.com/rhuss
- Konferenzsprecher
  - JavaZone 2014
  - W-JAX 2014
  - Devoxx 2014

ConSol

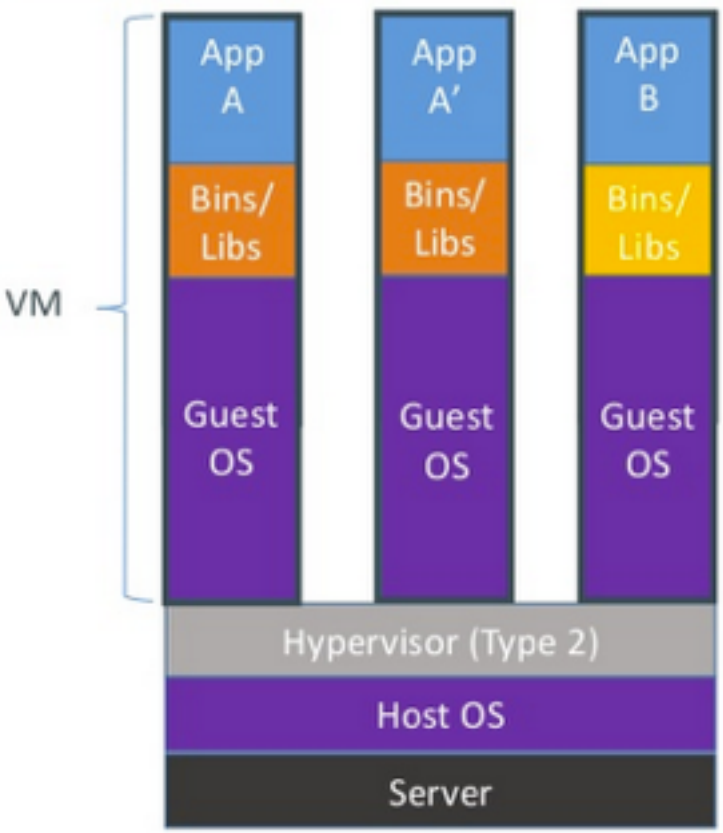
Consulting & Solutions

Docker

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

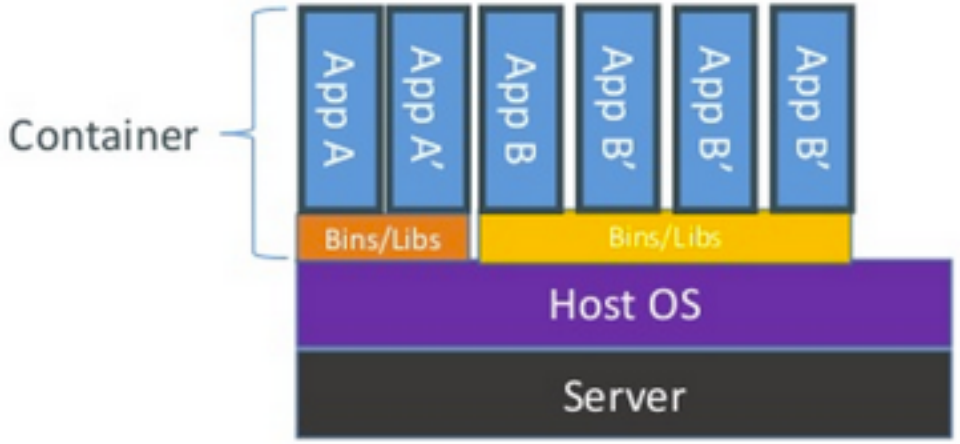
docker.io

Container versus VM



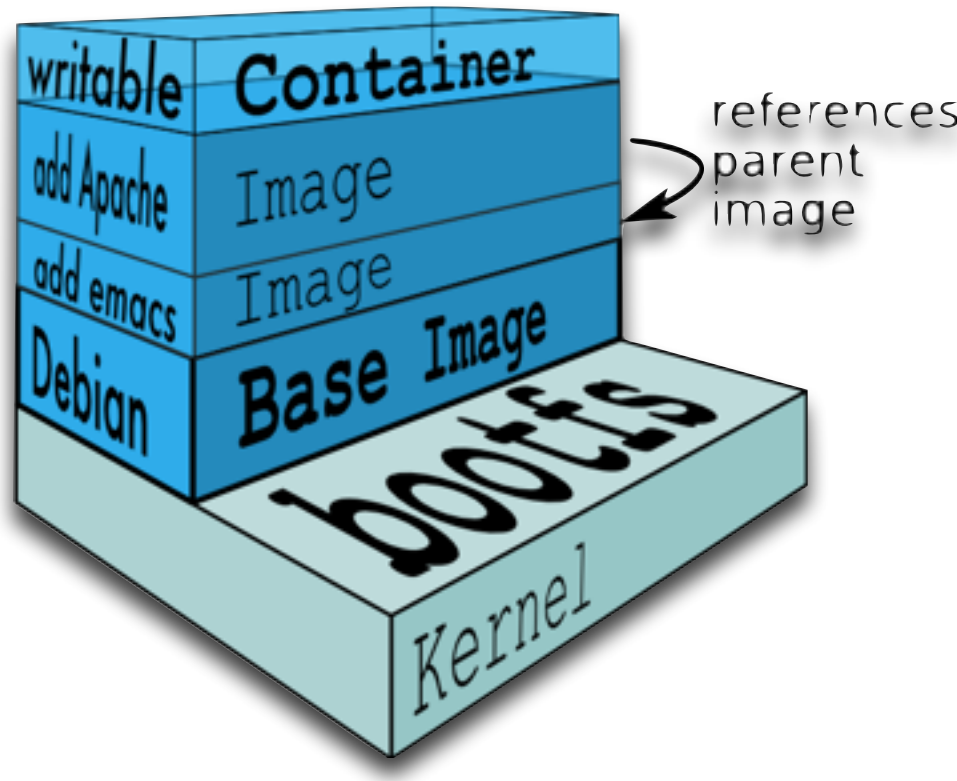
Container sind **isoliert**, teilen sich aber den Kernel und (einige) Dateien

➜ schneller und sparsamer



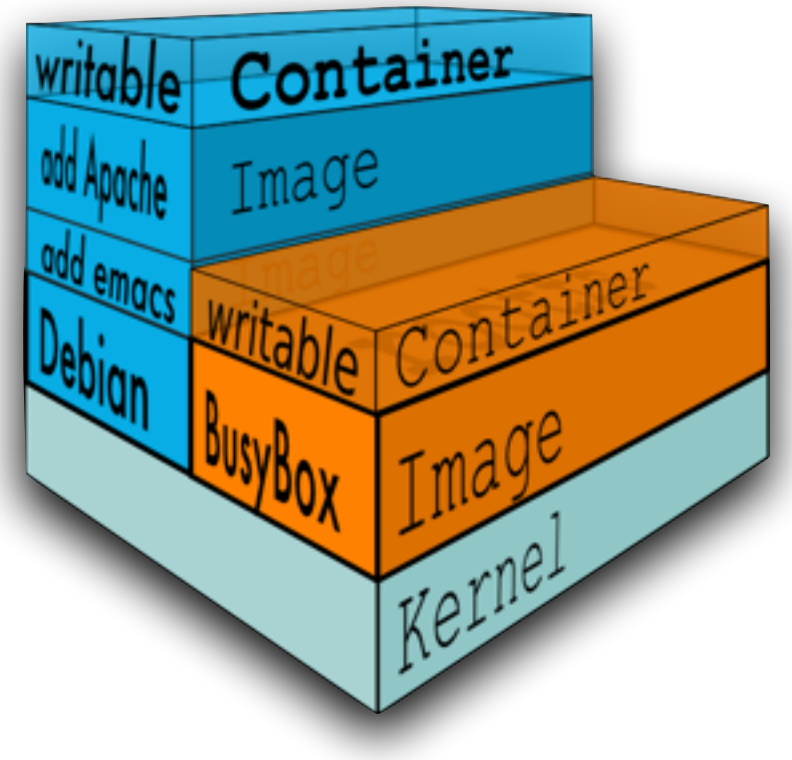
Image

- *read-only*
  Dateisystem Schicht
- kann installiert und verteilt werden
- "Blaupause für einen Container"



Container

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



Docker CLI Kommandos

`ps`	Anzeigen erzeugter Containern
`images`	Anzeigen lokaler Images
`run`	Erzeugen und starten von Containern
`search`	Suchen von Images in einer Registry
`pull`	Download von Images
`rm`	Entfernen eines Container
`rmi`	Entfernen eines Images
`exec`	Ausführen eines Kommandos im Container

Erweiterte Konzepte

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

Demo

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 mit dem gleichen Test fehl
  - **Autark**
    Minimale externe Abhängigkeiten, eigenständig
  - **Isoliert**
    Parallele Ausführung ähnlicher Tests
  - **Schnell**
    Kurze Turnaround-Zeiten

Externe Testsysteme

~~Robust~~	Test Systeme werden extern verwaltet und konfiguriert.
~~Autark~~	Test Systems müssen installiert und verfügbar sein.
~~Isoliert~~	Parallele Tests greifen auf das gleiche System zu und stören sich gegenseitig.
~~Schnell~~	Wegen paralleler Nutzung und Netzwerklatenz oft langsam.

aber sind ***nahe*** an der Realität !

Simulierte (Mock) Testsysteme

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

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

Integrationstests mit Docker

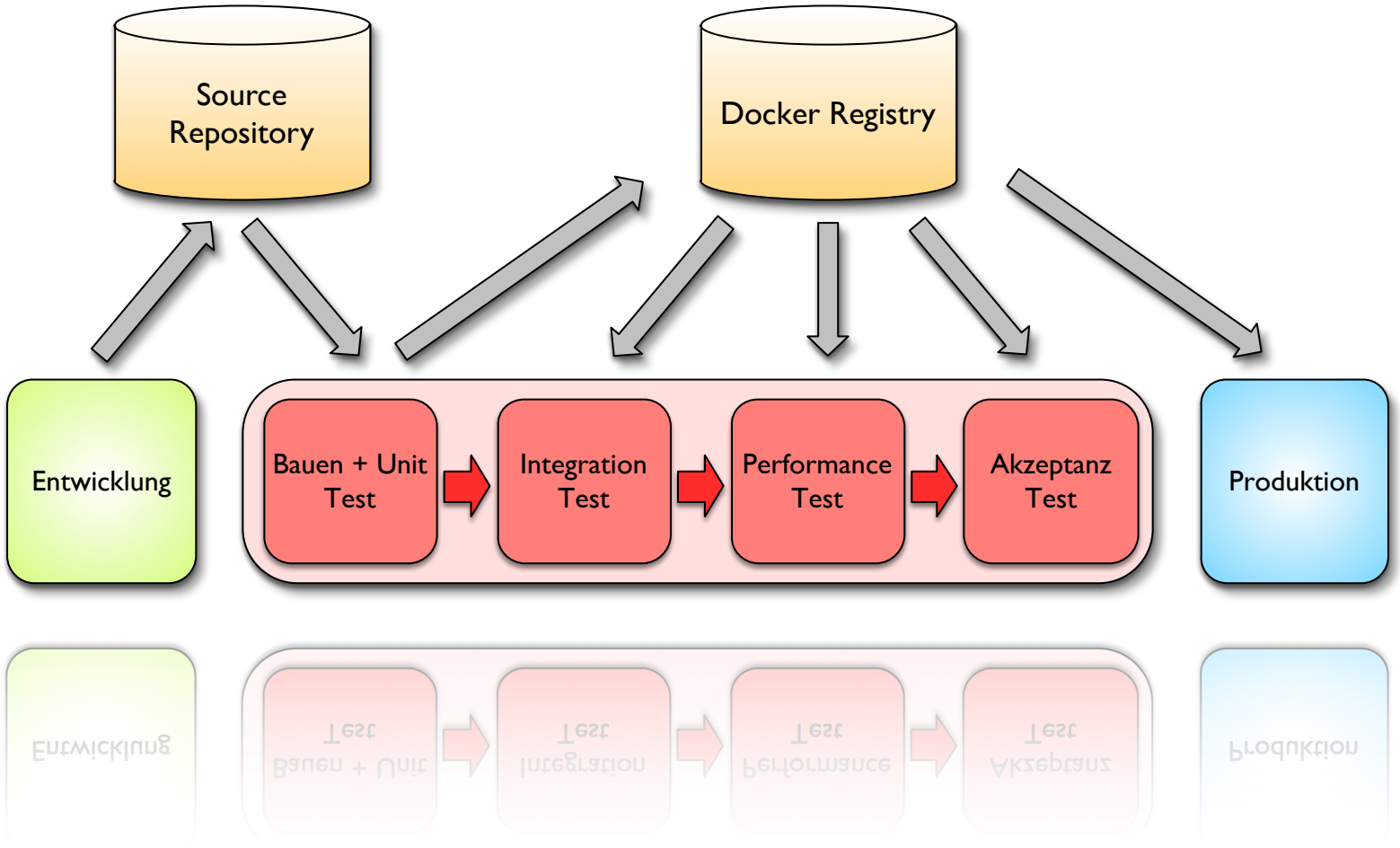
Robust	Jeder Testlauf hat einen eigenen Container und Ausführungskontext.
Autark	Ausser einer Docker Installation keine externen Abhängigkeiten.
Isoliert	Perfekte Isolation der Container möglich.
Schnell	Schneller Container Start dank der Systemlevel Virtualisierung.

und es ***kann*** 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

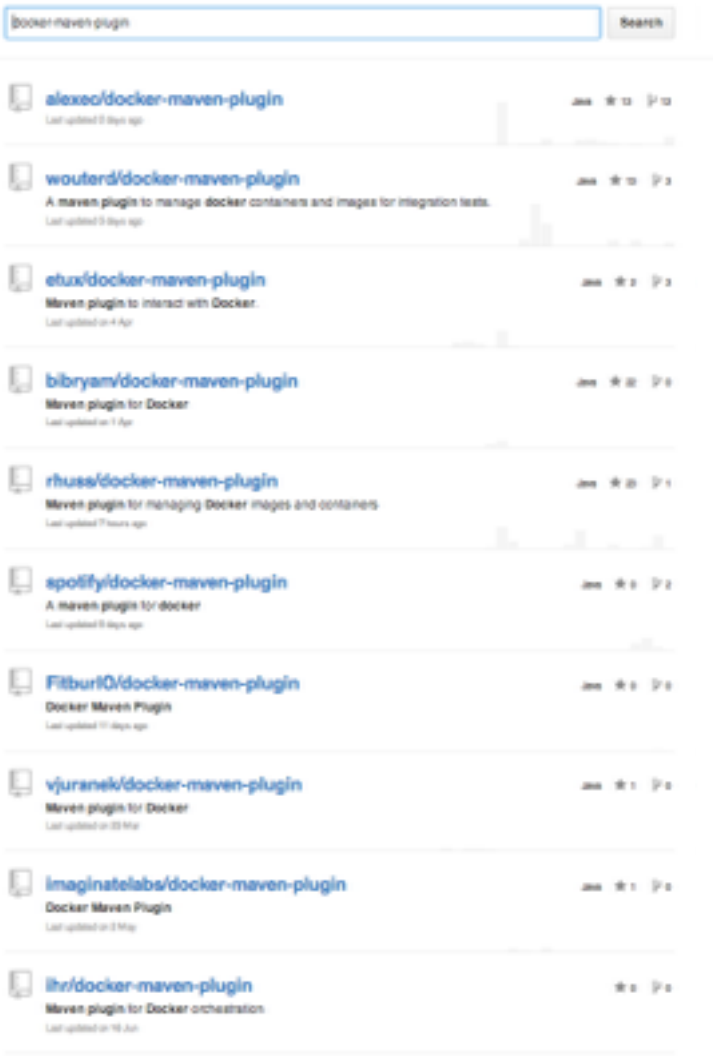
- **Datencontainer**:
  - Artefakte werden in einen **Datencontainer** verpackt.
  - Datencontainer wird mit einem **Plattformcontainer** verknüpft.
  - Applikation wird beim Start des Plattformcontainers deployed.

- **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



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

docker-maven-plugin

	wouterd	*alexec*	*spotify*	*rhuss*
API	jaxrs	docker-java	spotify/docker-client	Apache HC
Start/Stop	✔	✔	✘	✔
Build/Push	✔	✔	✔	✔
Image	Dockerfile + Maven Config	Dockerfile + custom YML	Maven config + Dockerfile	Maven config + Assembly + (Dockerfile)

docker-maven-plugin

	wouterd	*alexec*	*spotify*	*rhuss*
Logs	✘	✔	✘	✔
Security	Plain	Plain	✘	Encrypted/ Plain
URL Wait	✘	✔	✘	✔
★	36	57	73	90
Size LOC	2300	1600	600	4200

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

docker:start	Starten von Container
docker:stop	Stoppen von Container
docker:build	Bauen von Images
docker:push	Upload zu einer Registry
docker:remove	Entfernen von Images
docker:logs	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

Zusammenfassung

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

Referenzen

- index.docker.io - Public Docker Registry
- Entwickler Magazin Spezial
  Vol.2: Docker
  - http://entwickler.de/docker_spezial
- "The Docker Book"
  - sehr zu empfehlen !
  - http://www.dockerbook.com/

Danke !

```
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
www.consol.de