



# Docker und Java

Container für die Java-Entwicklung



Cegos Group

inspire  
qualify  
change



# Inhalts- verzeichnis



Einführung



Docker im Detail



Java und Docker



System-Werkzeuge und Tools








Java Enterprise Anwendungen



# Einführung



-  Ausgangssituation
-  Bestandteile von Docker
-  Installation
-  Ein erstes Beispiel
-  Dokumentation, Community und Ressourcen



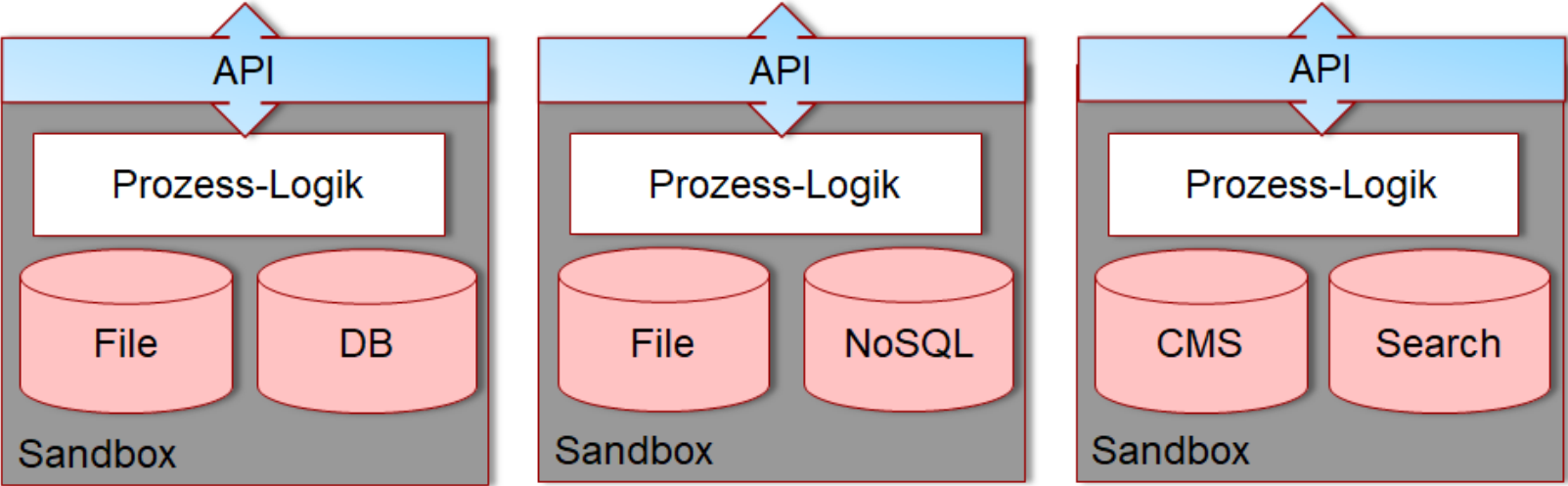
# Ausgangssituation



# Problemstellungen

- Wir benötigen:
  - Kapselung
    - Öffentliche „API “
    - Rein interne Implementierung
  - Vermeidung von Redundanzen
    - Wartbarkeit
  - Dependency Management
    - Statisch determinierte Abhängigkeiten oder
    - lose gekoppelte Systeme
- Diese Aufgabenstellung sieht nach Architektur-Vorgaben für die Programmentwicklung aus!
  - Kurzversion einer Service-Definition

# Gekapselte Services





## Linux-Sandbox (seit 2007)

- PID namespace
  - Process identifiers und Capabilities
- UTS namespace
  - Host und Domain Name
- MNT namespace
  - File System
- IPC namespace
  - Process Communication über Shared Memory
- NET namespace
  - Network Access
- USR namespace
  - User names und Identifiers
- chroot()
  - Lokation des File System Root
- cgroups
  - Schützen von Ressource

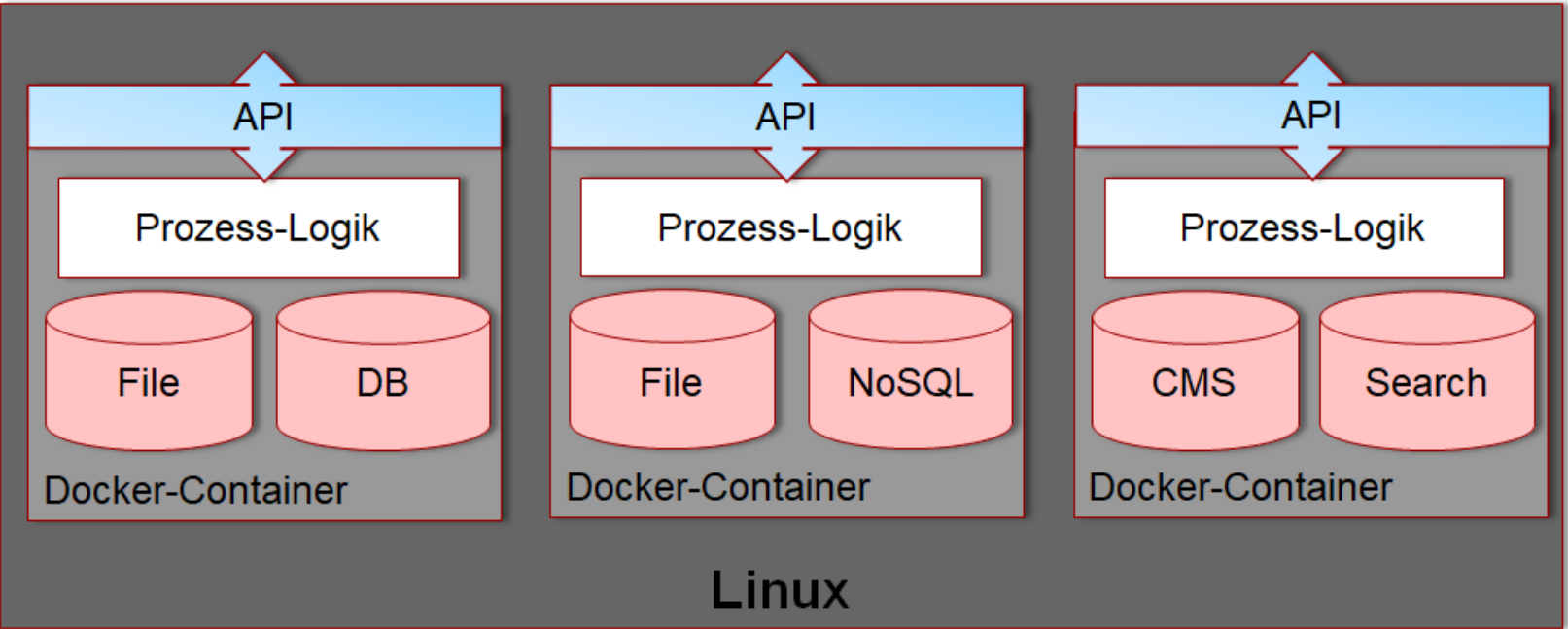


## Docker nutzt Linux

- Die Implementierung einer Linux-Sandbox ist aufwändig und komplex
- Docker ist ein Framework, das eine fertig konfigurierte und implementierte Sandbox-Lösung zur Verfügung stellt.
  - Fehler in der Sandbox: Bug im Docker-Produkt
- Docker ist mittlerweile auch für Windows erhältlich
  - Ab Windows 10 nativ
  - Benutzt Hyper-V



# Gekapselte Services mit Docker





## Definition des APIs

- Der Docker selbst stellt keine hochwertigen Protokolle zur Verfügung
  - SOAP
  - REST
  - Java RMI
- Es stehen zur Verfügung
  - Netzwerk-Sockets
    - Server-Implementierungen
    - Client-Zugriffe aus dem Container heraus
  - Dateisystem

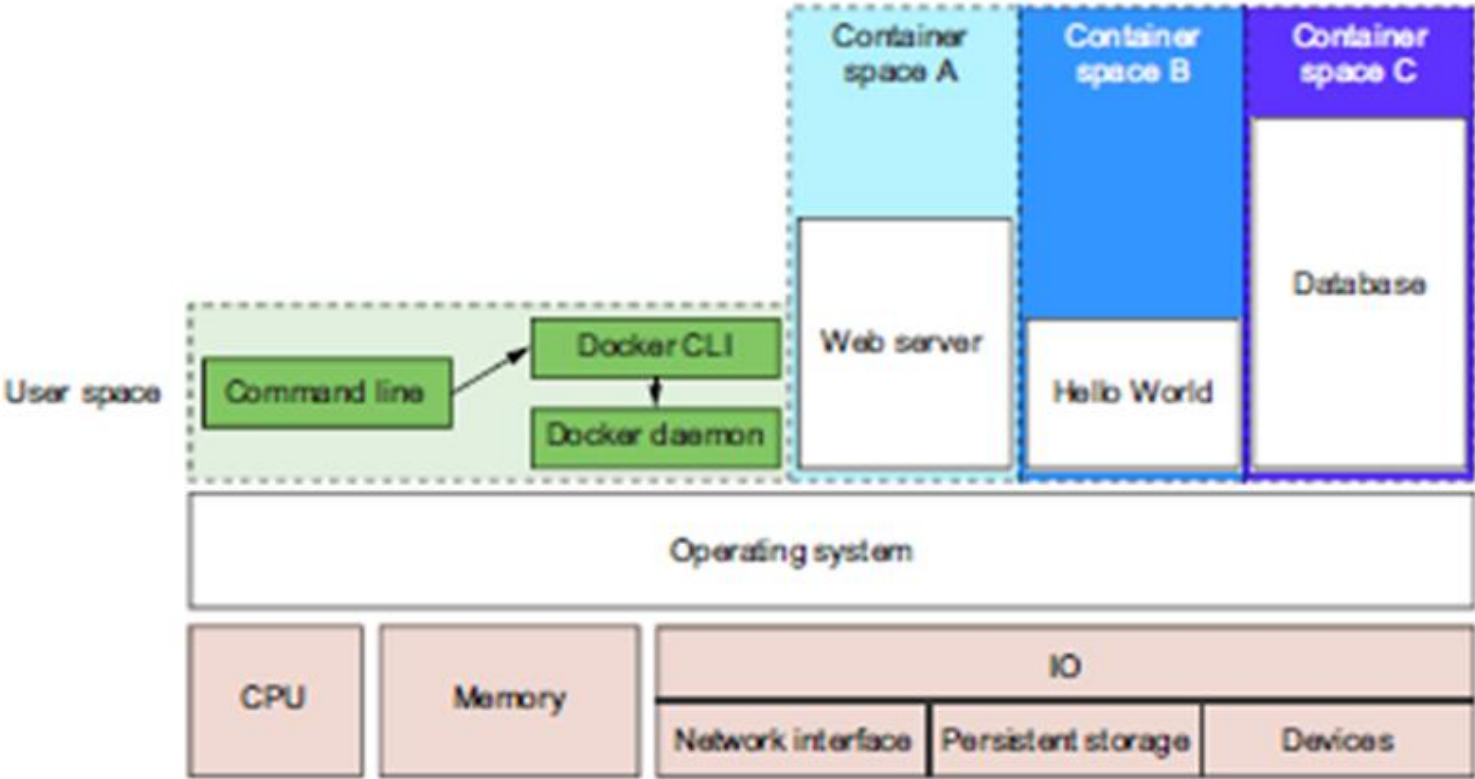


# Was ist ein Container?

- "Ein Container ist eine (modifizierte) Laufzeitumgebung, die der darin laufenden Anwendung den Zugriff auf geschützte Ressourcen unmöglich macht."
  - Unterschied zu Virtualisierung: Hier wird eine komplette Hardware durch eine "Virtuelle Maschine" abstrahiert
- Docker-Container laufen direkt ohne weitere Emulation auf dem Linux-Kernel
  - dabei werden etablierte Linux-Features wie namespaces und cgroups benutzt



# Container-Isolation





# Bestandteile von Docker



# Bestandteile

- Docker Command Line
  - Dieser Befehl wird vom Docker-Anwender direkt benutzt
- Docker Host
  - Eine Maschine mit installierter Docker Umgebung
    - Docker Dämon/Engine
    - Lokale Registry
- Docker Dämon/Engine
  - Über den Dämon werden die Docker-Container gestartet
    - Technisch gesprochen sind die Container Child-Prozesse des Dämons
- Remote Services
  - Registries für Docker Images
    - DockerHub



# OOP versus Container

- In der Objekt-orientierten Programmierung werden Datenstrukturen und Verhalten in Objekten gekapselt
  - Ein Container kapselt einen kompletten Service samt Infrastruktur
- Objekte kommunizieren miteinander über ein Interface
  - Container kommunizieren über Netzwerk und Streams
- Eine OOP-Anwendung wird durch ein Context&Dependency Injection –Framework dynamisch aufgebaut
  - Docker-Container definieren Dependencies
- Eine OOP-Anwendung wird durch einen Build-Prozess erzeugt
  - Docker-Images werden durch eine Konfigurationsdatei definiert und mit Docker-Kommandos erzeugt

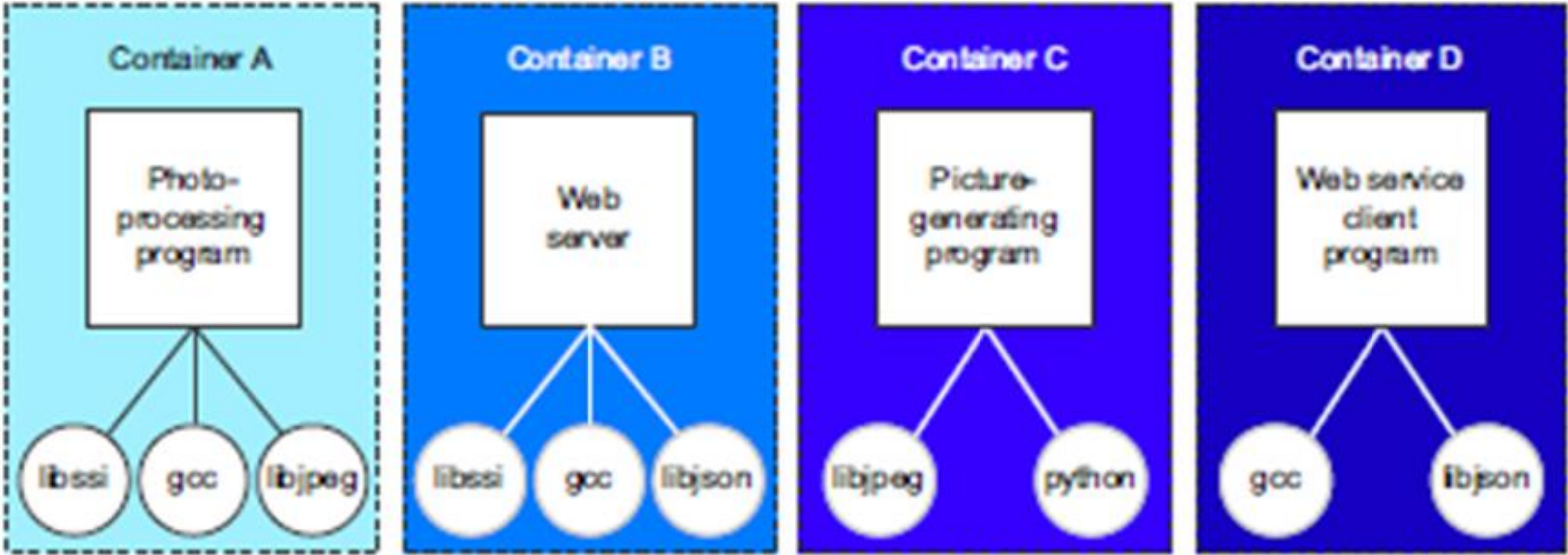


# Docker und (Micro)Services

- Microservices
  - Wohl-definiertes API
    - hier nicht relevant
  - Netzwerk
  - Isolation der internen Implementierung
  - Bereitstellung aller benötigten Ressourcen
    - Interne Ressourcen wie Datenbanken
    - Dependencies auf andere Services
- Docker passt!



# Microservices mit Docker-Containern





# Installation



# Voraussetzungen

- <https://docs.docker.com/engine/installation/>
- Beispiel:
  - SUSE Linux Enterprise
  - Windows 10 Enterprise mit Hyper-V



# Installation unter Linux

- Download der passenden Distribution von [docker.com](https://docker.com)
- Anschließend ist Docker auf dem System aktiv
  - Docker CLI
  - Docker Dämon
    - Dieser muss eventuell noch manuell gestartet werden
- Vorsicht: Der Zugriff auf den Docker-Dämon benötigt Berechtigungen
  - `sudo`
  - Hinzufügen des Benutzers zur `docker`-Gruppe
    - Diese wurde während der Installation automatisch eingerichtet



## Befehle zur Installation

- Hinzufügen des Docker-Repositories
  - `sudo zypper addrepo https://yum.dockerproject.org/repo/main/opensuse/13.2/ docker-main`
- Refresh des SUSE Package Managers
  - `sudo zypper refresh`
- Die eigentliche Installation
  - `sudo zypper install docker-engine`
- Auf Grund eines kleinen Problems mit den Berechtigungen muss der Docker-Service manuell gestartet werden
  - `sudo service docker start`



# Ein erstes Beispiel



# Starten eines ersten Containers

- Nach der Installation kann der erste Container gestartet werden (!)
  - `docker run <docker_image>`
    - z.B. `docker run hello-world`
- Das wars



# Ausgabe

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
535020c3e8ad: Pull complete
af340544ed62: Pull complete
Digest: sha256:a68868bfe696c00866942e8f5ca39e3e31b79c1e50feaaee4ce5e28df2f051d5c
Status: Downloaded newer image for hello-world:latest
Hello from Docker.

This message shows that your installation appears to be working correctly.
To generate this message, Docker took the following steps:
1. The Docker Engine CLI client contacted the Docker Engine daemon.
2. The Docker Engine daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker Engine daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker Engine daemon streamed that output to the Docker Engine CLI client, which
sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
https://hub.docker.com

For more examples and ideas, visit:
https://docs.docker.com/userguide/
```

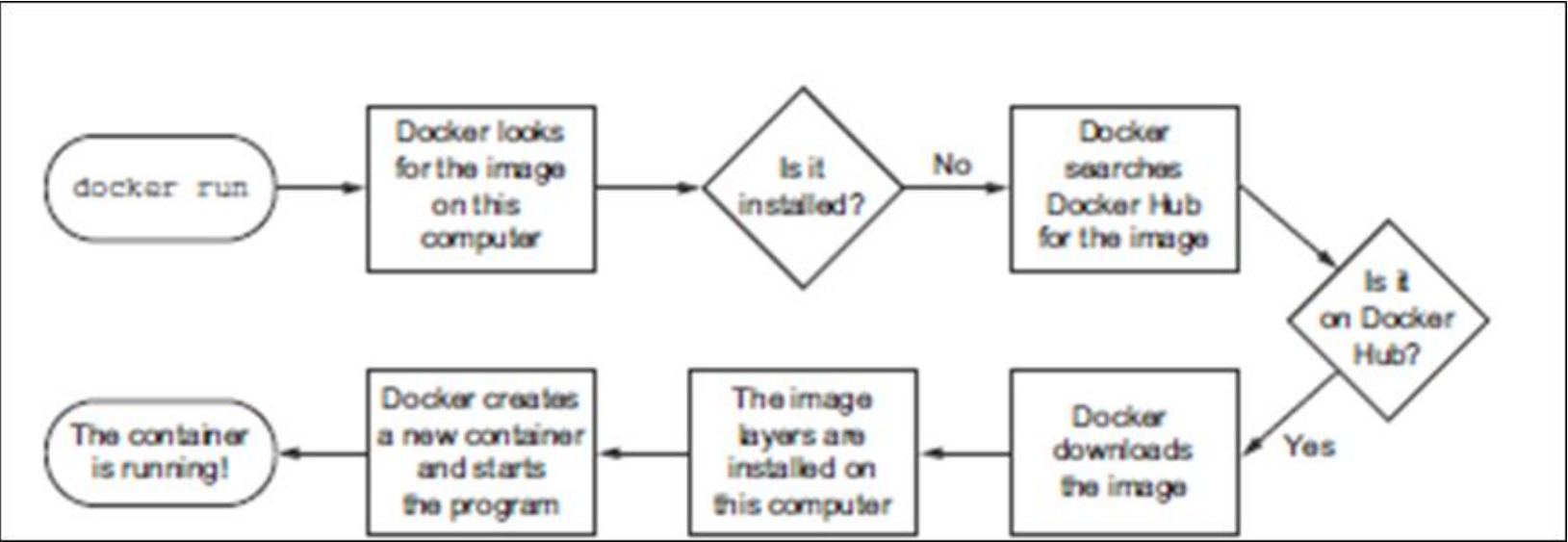




Wie funktioniert das  
denn?



# Docker-Workflow: 1. und 2. Lauf





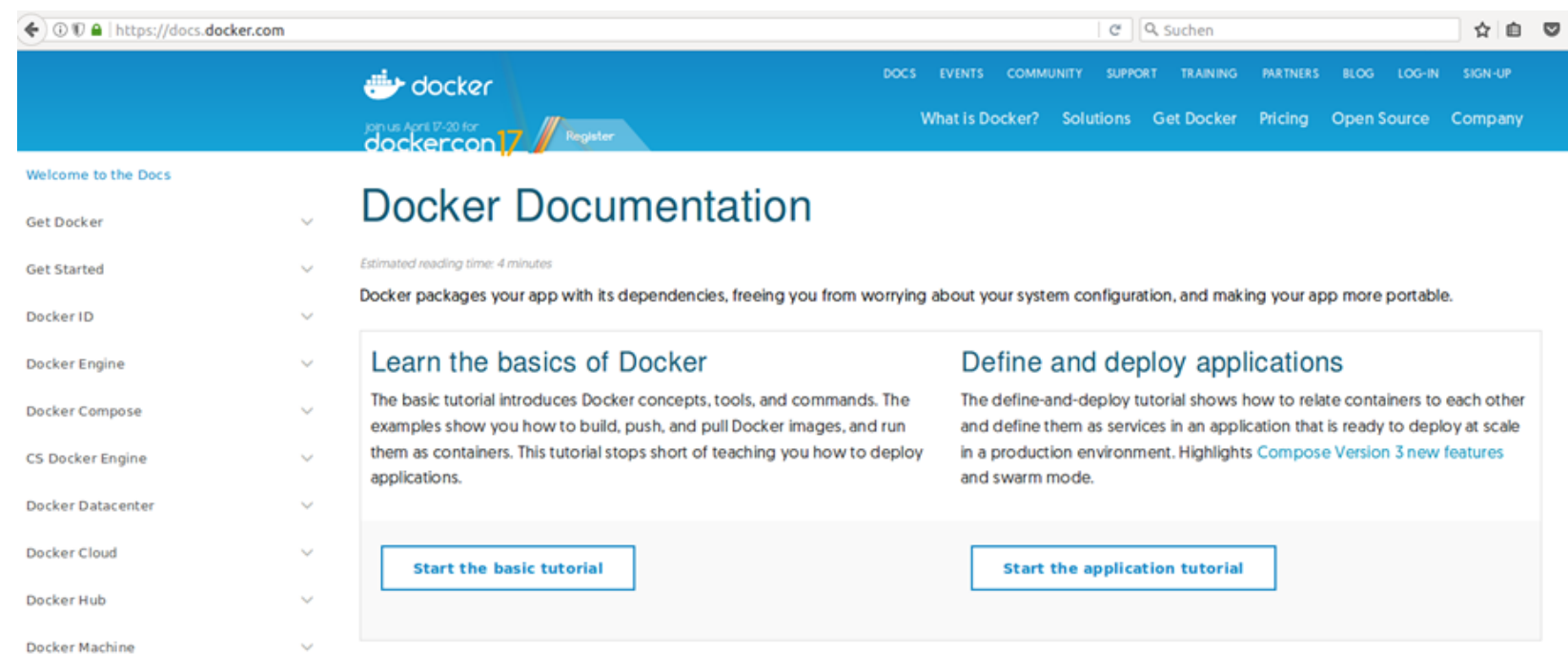
# Lebenszyklus des Containers

- Der Containers wird gestartet
  - Nochmal: Keine VM, sondern ein gekapselter Prozess auf dem Host-System
- Innerhalb des Containers werden Prozesse gestartet
  - Diese werden sind der Definition des Images angegeben
- Der Container läuft, solange ein definierter Prozess läuft
  - Anschließend wird der Container selbst beendet



# Dokumentation, Community und Ressourcen

# Dokumentation





# Foren

Success Center

Knowledge Base

Documentation

Community Forums

Technical Support

Service Status

What Is Docker? Enterprise Get Docker Pricing Open Source Company

Log In

**Welcome to the Docker Community Forums!**

This is a public forum for users to discuss questions and explore current design patterns and best practices about Docker and related projects in the Docker Ecosystem.

To participate, just log in with your Docker Hub account. Make sure to also review our [Community Guidelines](#), [Terms of Service](#), and [Privacy Policy](#).

When posting issues or feedback, make sure to remove any sensitive information and please provide the following:

- Issue type
- OS Version/build
- App version
- Steps to reproduce

all categories

Categories

Latest

Top

Category	Topics	Latest
<b>Announcements</b> Category for announcing new or updates to the Docker community, products, projects, training, etc, including ...	15	<div>T</div> <div>Docker Containers IP externally</div> <div>General Discussions</div> <div>0</div> <div>4m</div>
<b>General Discussions</b> General discussions, feature requests, and training inquiries. General Feature Requests	29 / week	<div></div> <div>Docker on windows fails with //pipe/docker_engine: The system cannot find the file specified</div> <div>Docker for Windows</div> <div>1</div> <div>9m</div>
<b>Docker Data Center</b>	6 / week	<div>C</div> <div>Issues Installing UCP 2.1.0 on Docker 1.13</div> <div>UCP</div> <div>4</div> <div>30m</div>

# Docker Installation

Get Docker

Install Docker Engine

Docker for Mac

Docker for Windows

Docker for Linux

Docker for AWS

Docker for Azure

Docker Toolbox (Legacy)

Get Started

Docker ID

Docker Engine

Docker Compose

CS Docker Engine

Docker Datacenter

Docker Cloud

docker

Join us Apr 17-20 for **dockercon17** [Register](#)

DOCS EVENTS COMMUNITY SUPPORT TRAINING PARTNERS BLOG LOG-IN SIGN-UP

What is Docker? Solutions Get Docker Pricing Open Source Company

Welcome to the Docs

## Install Docker Engine

Estimated reading time: 1 minute

Docker Engine is supported on Linux, Cloud, Windows, and macOS. Installation instructions are available for the following:

### On Linux

- CentOS
- Debian
- Fedora
- Oracle Linux
- Red Hat Enterprise Linux
- openSUSE and SUSE Linux Enterprise
- Ubuntu
- Other Linux distributions

If your linux distribution is not listed above, don't give up yet. To try out Docker on a different Linux distribution, consider [installing from binaries](#).

### On macOS and Windows

- Docker for Mac
- Docker for Windows

### On Cloud

# Docker Hub

← → ↻

Sicher | https://hub.docker.com/explore/

Docker Store is the new place to discover public Docker content. [Check it out →](#)

Dashboard Explore Organizations Create javacream

### Explore Official Repositories

<b>nginx</b> official	5.2K STARS	10M+ PULLS	<a href="#">➤ DETAILS</a>
<b>redis</b> official	3.3K STARS	10M+ PULLS	<a href="#">➤ DETAILS</a>
<b>busybox</b> official	922 STARS	10M+ PULLS	<a href="#">➤ DETAILS</a>
<b>ubuntu</b> official	5.5K STARS	10M+ PULLS	<a href="#">➤ DETAILS</a>
<b>registry</b> official	1.3K STARS	10M+ PULLS	<a href="#">➤ DETAILS</a>
<b>alpine</b> official	1.9K STARS	10M+ PULLS	<a href="#">➤ DETAILS</a>





# Docker im Detail



Docker Images und Container



Docker Command Line

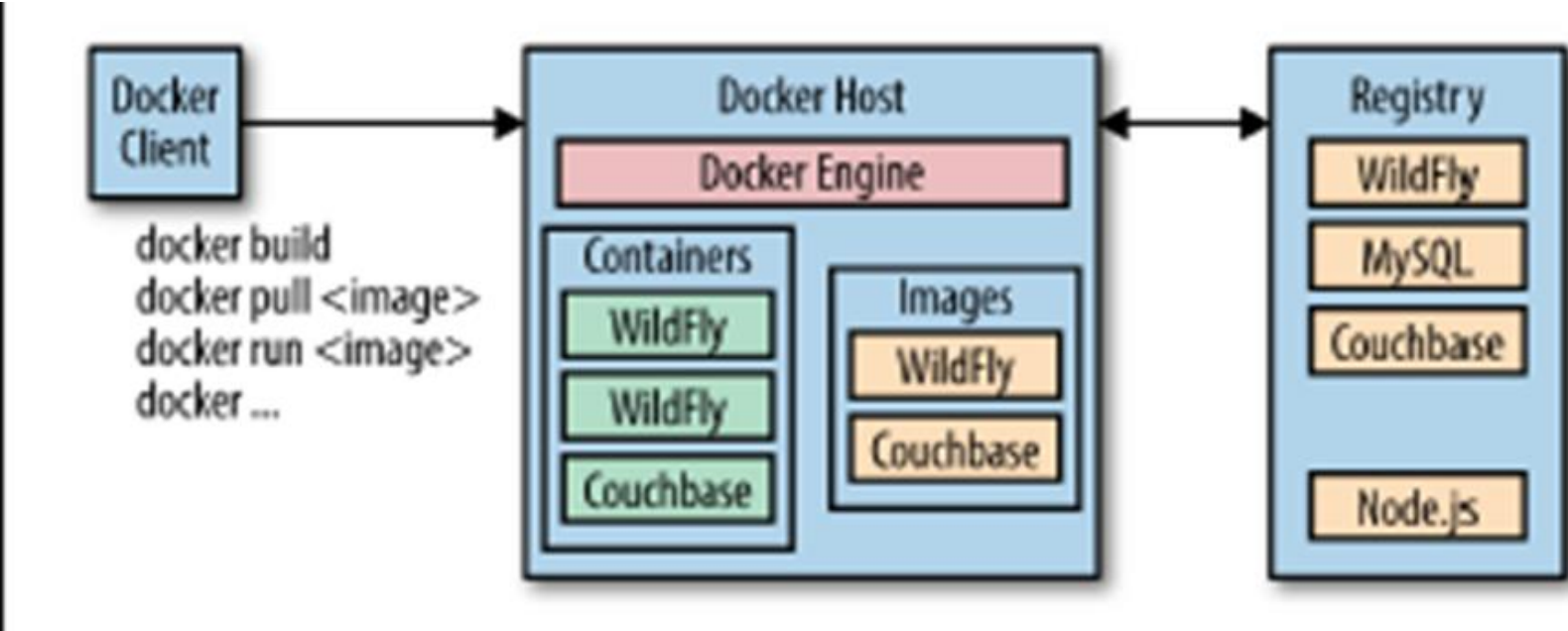


Definition eines Docker Images



# Docker Images und Container

# Docker im Gesamtbild





## Images und Layers

- Stellt eine Read-Only-Umgebung zur Verfügung
  - Analogie OOP: Eine Klassen-Definition
- Im Gegensatz zu VMWare-Images aus Layern aufgebaut
  - Vererbung und Assoziation
- Durch die Layerung sind einzelne Images deshalb relativ schlank
- Images und Layers werden in Repositories verwaltet
  - Inklusive Meta-Informationen
    - Versionierung!
  - Die Docker-Client-Installation verwaltet ein lokales Repository



Beispiel: Ein Java-Image  
(<https://microbadger.com/images/java>)

Tags

openjdk-8u111openjdk-8u111-jdkopenjdk-8openjdk-8-jdklatestjdk8u1118u111-jdk88-jdk

Created

January 17, 2017 at 01:52 AM

ID

d11c3799fa6a

Download Size

232.1 MB

Labels

No labels

Layers

14

107.2 MB

buildpack-deps

scm

jessie-scm

What's this?

49.0 MB

ADD file:89ecb642d662ee7edbb868340551106d51336c7e589...

CMD ["/bin/bash"]

17.7 MB

RUN apt-get update && apt-get install -y --no-instal...

40.5 MB

RUN apt-get update && apt-get install -y --no-instal...

579.2 kB

RUN apt-get update && apt-get install -y --no-install-recommends b...

214 bytes

RUN echo 'deb http://deb.debian.org/debian jessie-backports main' > ...

ENV LANG=C.UTF-8

242 bytes

RUN [ echo '#!/bin/sh'; echo 'set -e'; echo; echo 'dirname "...

ENV JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

ENV JAVA\_VERSION=8u111

ENV JAVA\_DEBIAN\_VERSION=8u111-b14-2~bpo8+1

32 bytes

ENV CA\_CERTIFICATES\_JAVA\_VERSION=20140324

124.1 MB

RUN set -x && apt-get update && apt-get install -y openjdk-8-jdk...

282.3 kB

RUN /var/lib/dpkg/info/ca-certificates-java.postinst configure



# Container

- Eine Instanz eines Images
  - Analogie zu OOP: Ein Objekt
  - Identifikation über einen Namen bzw. über einen technischen Schlüssel
    - Ein Hash
- Ein Container hat einen Lebenszyklus
  - Gesteuert über Docker-Kommandos
    - `create`
    - `run`
    - `start`
    - `stop`
    - `delete`
- Im Gegensatz zu Images können Container einen Zustand aufweisen
  - Container-Environment
  - Interne Ressourcen
    - Dateisystem des Containers



# Inter-Container-Kommunikation

- Container sind komplett gekapselt
  - Interaktion nur über Docker-Kommandos
    - Auslesen des Docker-Logs
    - Inspect
    - Ausführen einer Shell im Container
- Benutzung externer Volumes
  - Ein Teil des Dateisystems des Containers vom Host zur Verfügung gestellt
- Netzwerk-Kommunikation
  - Container können Netzwerk-Sockets bereit stellen
  - Diese werden auf reale Sockets des Hosts gemapped
- Container-Linking
  - Direkte Kommunikation der Container untereinander
  - Verschiedene Abstufungen
    - Gemeinsames Netzwerk
    - Gemeinsames File-Layer
    - Shared Memory



# Docker Command Line





# Kommandoreferenz

- <https://docs.docker.com/engine/reference/commandline/docker/#/related-commands>
- Elementare Befehle
  - Container anlegen
    - `sudo docker create <<Image-Name>>:<<Versionsnummer>>`
  - Container löschen
    - `sudo docker rm <<containerId>>`
  - Docker-Container automatisch nach Ausführung löschen
    - `sudo docker run -rm <<Image-Name>>`
  - Alle Container anzeigen lassen
    - `sudo docker ps -a`
  - Container starten
    - `sudo docker start <<containerId>>`
  - Container anlegen, starten und assoziiert eigene Ausgabe-Konsole (nicht in der Terminal-Session, in der gearbeitet wird)
    - `sudo docker run --detach <<Image-Name>>`
  - Images anzeigen lassen
    - `sudo docker images`



# Definition eines Docker Images



# Workflow

- Anlegen eines leeren Verzeichnisses
  - Alle Verzeichnisse und Unterverzeichnisse werden Bestandteil des Images
  - `excludes` in `.dockerignore`
- Erstellen eines Docker-Files
  - Bestehend aus Kommandos
    - FROM
    - COPY
    - ENV
    - RUN
    - EXPOSE



# Dockerfile für eine Java-Umgebung

- Das Dockerfile selbst ist ein simples Text-Dokument
  - Referenz unter <https://docs.docker.com/engine/reference/builder/>
  - Beispiel

```
FROM openjdk
CMD ["java", "-version"]
```
- Erzeugen des Images
  - `docker build -tag <name>`





# Java und Docker



Entwicklungsumgebungen



Build-Prozess



Der Repository-Server



# Entwicklungsumgebungen



# Entwicklungsumgebungen

- Produktauswahl
  - Eclipse
  - IntelliJ
  - ...
- Docker wird durch Plugins unterstützt!



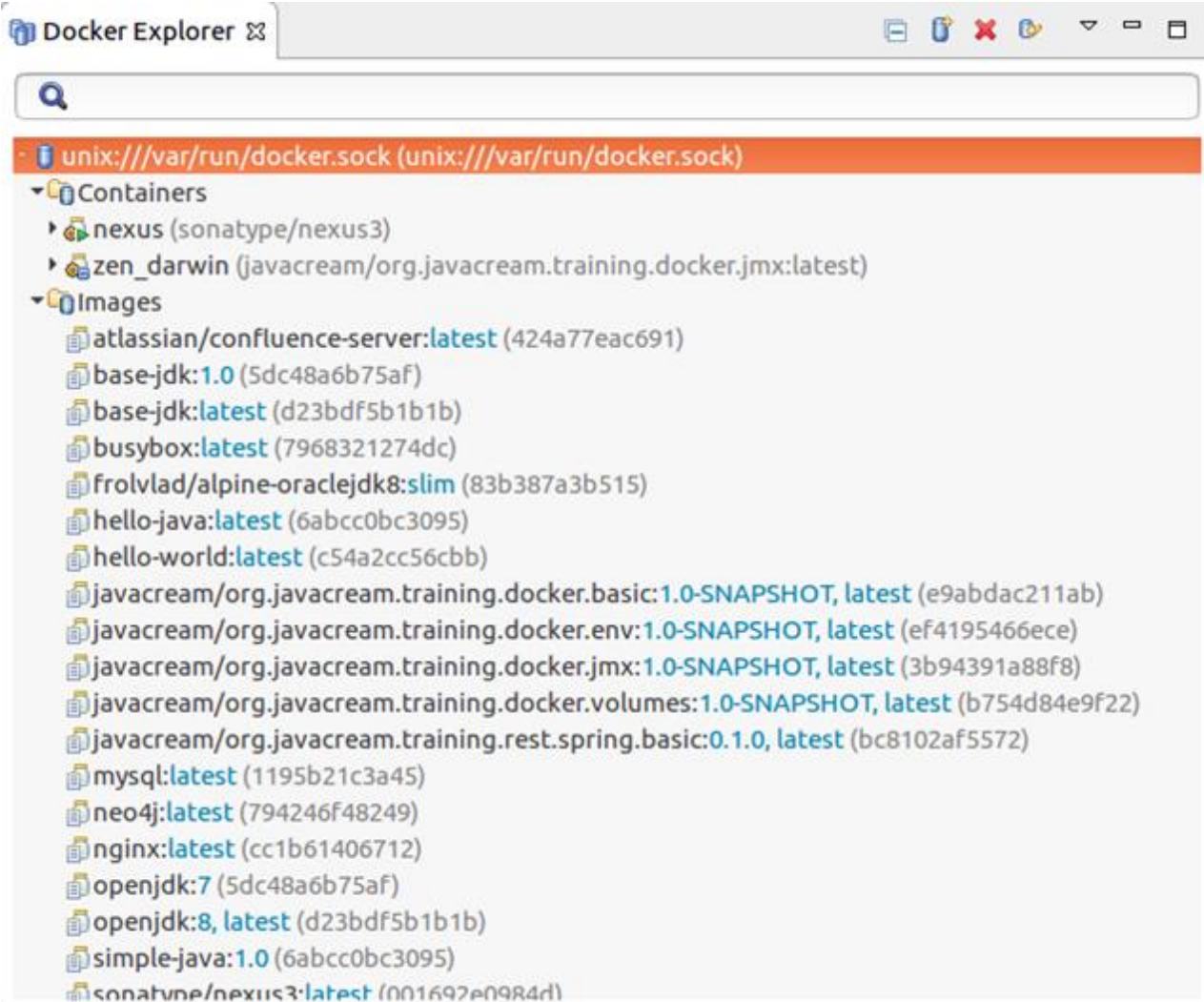


## Beispiel: Das Docker-Plugin für Eclipse

- Perspektive
  - Docker Explorer
  - Docker Images
  - Docker Containers
  - Docker Console
- Wizards
  - Build
  - Run Container




# Docker Explorer






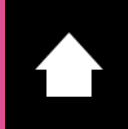
# Docker Container

Docker Containers 

unix:///var/run/docker.sock (2 Containers)



Name	Image	Created	Command	Ports	Status
zen_darwin	javacream/org.javacream.training	2017-02-23	/bin/sh -c 'java -javaagent:jolokia-jv		Exited (137) 3 weeks ago
nexus	sonatype/nexus3	2017-02-22	bin/nexus run	0.0.0.0:8082->8082/tcp, 0.0.0.0:808	Up About an hour



# Docker Images

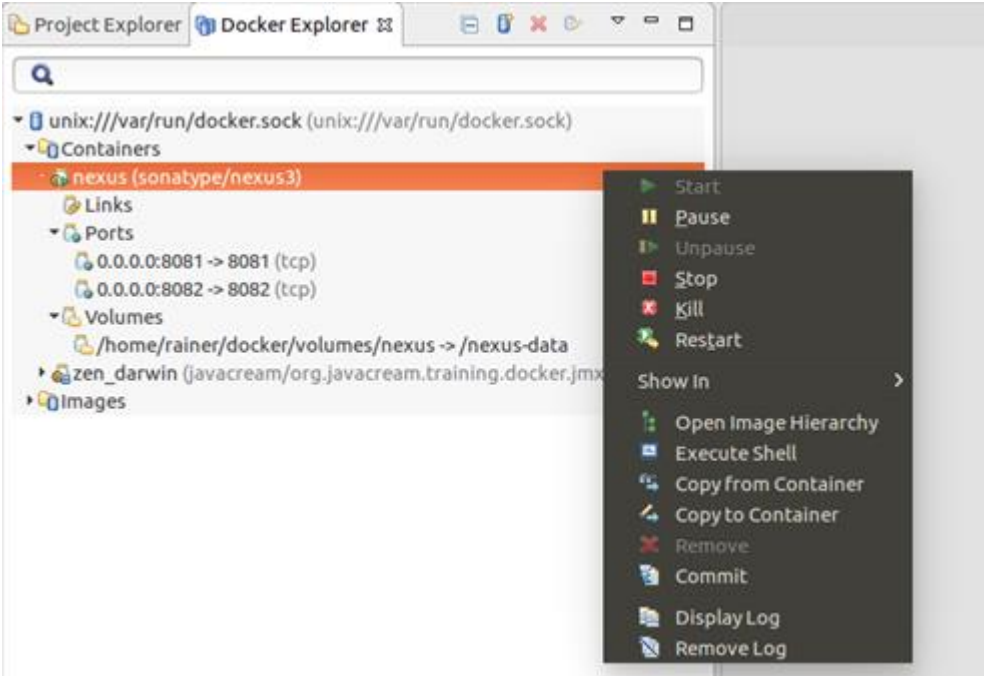
Docker Images 18/36 Images

unix:///var/run/docker.sock (18/36 Images)

Id	Repo Tags	Created	Virtual Size
b754d84e9f22	javacream/org.javacream.training.docker.volumes:1	2017-02-25	643,2 MB
ef4195466ece	javacream/org.javacream.training.docker.volumes:1	2017-02-25	643,2 MB
e9abdac211ab	javacream/org.javacream.training.docker.env:1.0-SNAPSHOT	2017-02-25	643,2 MB
3b94391a88f8	javacream/org.javacream.training.docker.env:latest	2017-02-25	643,2 MB
bc8102af5572	javacream/org.javacream.training.docker.basic:1.0-SNAPSHOT	2017-02-23	643,6 MB
001692e0984d	javacream/org.javacream.training.docker.basic:latest	2017-02-22	698,8 MB
2dc5069d7582	javacream/org.javacream.training.rest.spring.basic:1.0-SNAPSHOT	2017-02-16	460,4 MB
aa26cdcc81ea	sonatype/nexus3:latest	2017-02-10	400,2 MB
6abcc0bc3095	wordpress:latest	2017-02-07	222,1 MB
cc1b61406712	springio/gs-spring-boot-docker:latest	2017-02-06	643,2 MB
83b387a3b515	hello-java:latest	2017-01-24	181,8 MB
d23bdf5b1b1b	simple-java:1.0	2017-01-22	166,5 MB
5dc48a6b75af	nginx:latest	2017-01-17	643,2 MB
7968321274dc	frolvlad/alpine-oraclejdk8:slim	2017-01-17	584,5 MB
794246f48249	base-jdk:latest	2017-01-13	1,1 MB
424a77eac691	openjdk:8	2016-09-23	377,4 MB
c54a2cc56cbb	openjdk:latest	2016-09-09	817,2 MB
1195b21c3a45	base-jdk:1.0	2016-07-01	1,8 kB
	openjdk:7	2016-06-10	380,2 MB
	busybox:latest		
	neo4j:latest		
	atlassian/confluence-server:latest		
	hello-world:latest		
	mysql:latest		

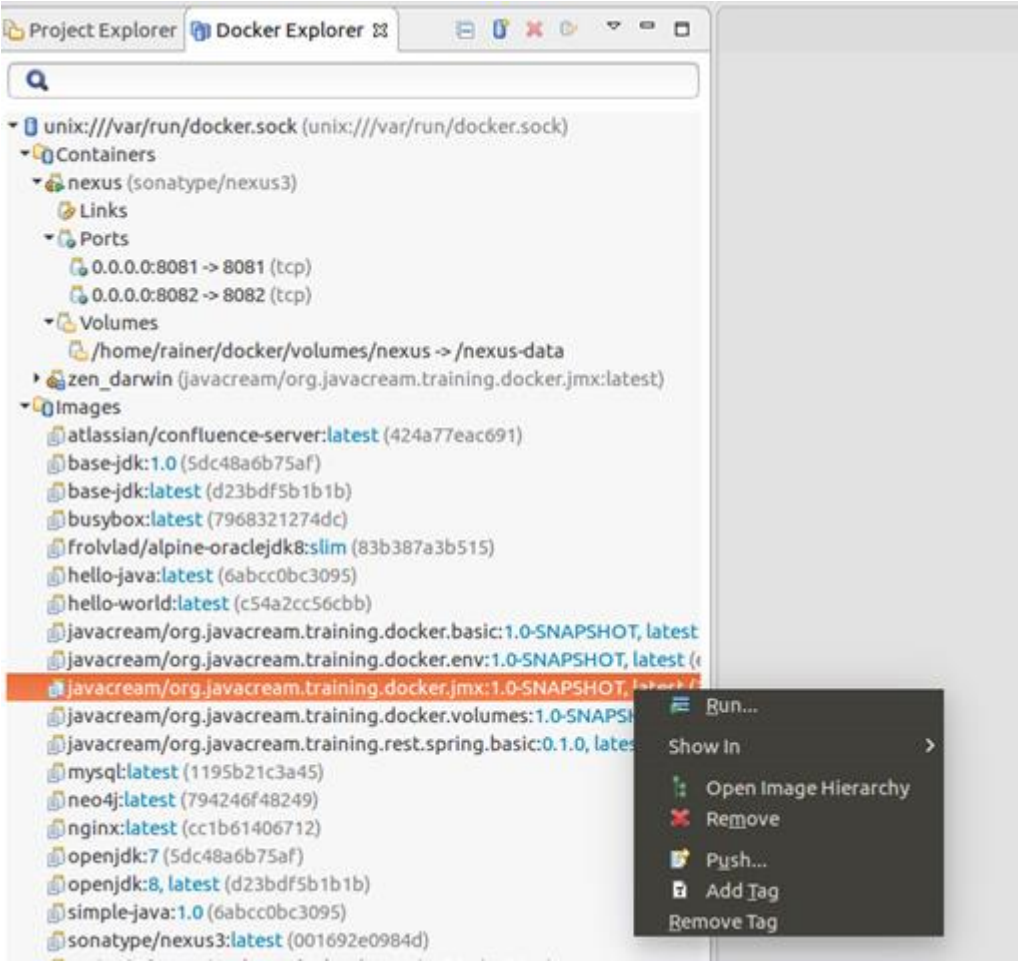


# Docker Context Menü für Container





# Docker Context Menü für Images





# Build-Prozess



# Apache Maven

- Bestandteile
  - Fester Build-Prozess
    - Aufruf mit mvn <command>
      - compile
      - package
      - install
      - deploy
- Lokaler Cache für gebildete Artefakte
- Connectivity zu einem Remote-Repository
  - Standard: Internet-Repository [repo1.maven.org](https://repo1.maven.org)
  - Im Unternehmen: Eigener Repository-Server
    - Nexus
    - Artefactory
    - ...



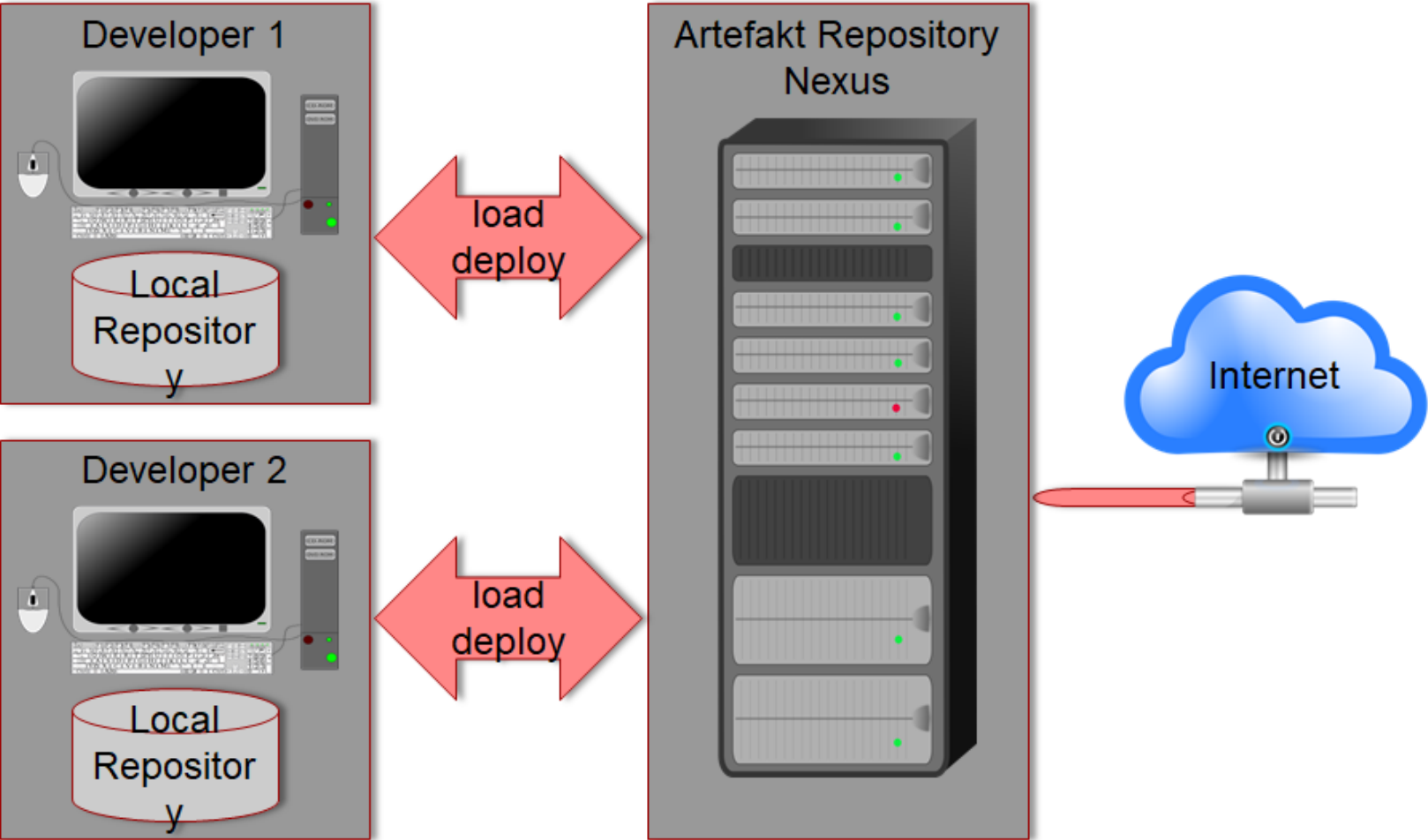


# Build-Prozess für Java-Projekte

- Unterscheiden „Werke “ (Java-Code) von „Artefakten “ (Java-Archive)
  - Der Build-Prozess wandelt Werke in Artefakte um
- Werke liegen klassischerweise in der Versionsverwaltungssysteme
  - Branches
  - Tags (Versionsnummern)
- Artefakte werden von einem Artefakt-Repository verwaltet
  - Damit können Dependencies ohne Zugriff auf das Versionsverwaltungssystem aufgelöst werden
  - Identifikation von Artefakten: "Artefakt-Koordinate"
- Build-Sequenz
  - Compiler-Aufruf
  - JAR-Archivierer
  - + x, z. B. Auschecken aus Versionsverwaltung, Testen, Ausbringen aus Servern, etc.
- Dependency Management
  - Andere interne Software-Projekte
  - Open Source-Projekte
  - Produkt-Bibliotheken



# Apache Maven





# Maven- Konfiguration: Das interne Repository

- settings.xml, allgemein gültig

```
<settings>
  <mirrors>
    <mirror>
      Hier wird die Adresse des internen Servers
      eingetragen
    </mirror>
  </mirrors>
  <servers>
    <server>
      Authentifizierungs-Informationen für den
      internen Server
    </server>
  </servers>
</settings>
```



# Maven- Konfiguration: Ausbringen eigener Artefakte

- Das übernimmt eine so genannte Parent-POM
  - Eine Art Superklasse für alle Maven-Buildprozesse
  - Diese wird als eigenes Artefakt im Repository abgelegt

- Beispiel:

- Ausschnitt des Distribution-Managements

```
<distributionManagement>
  <repository>
    <uniqueVersion>false</uniqueVersion>
    <id>nexus</id>
    <name>Corporate Repository</name>
    <url>http://...</url>
  </repository>
  <snapshotRepository>
    <uniqueVersion>true</uniqueVersion>
    <id>nexus</id>
    <name>Corporate Snapshots</name>
    <url>http://...</url>
  </snapshotRepository>
</distributionManagement>
```



## Weitere Elemente der Parent-POM

- Konfiguration des Java-Compilers
  - Java-Version
- Reporting
- Allgemein zu benutzende Dependencies zu anderen Artefakten
  - JUnit
  - Datenbank-Treiber
  - ...
- Konfiguration weiterer Plugins
  - Spezielle Reports
  - DOCKER!



## Maven-PlugIns für Docker

- `fabric8io/docker-maven-plugin`
- `spotify/docker-maven-plugin`
- `wouterd/docker-maven-plugin`
- `alexec/docker-maven-plugin`



# Beispiel: Das Spotify-Plugin

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.4.11</version>
  configuration>
    <imageName>${docker.namespace.prefix}/${project.artifactId}</imageName>
    <dockerDirectory>src/main/docker</dockerDirectory>
    <resources>
      <resource>
        <targetPath></targetPath>
        <directory>${project.build.directory}</directory>
        <include>${project.build.finalName}.jar</include>
      </resource>
      <resource>
        <targetPath></targetPath>
        <directory>${project.build.directory}</directory>
        <include>libs/*.jar</include>
      </resource>
    </resources>
    <imageTags>
      <imageTag>${project.version}</imageTag>
      <imageTag>latest</imageTag>
      <imageTag>localhost:5000/${project.build.finalName}</imageTag>
    </imageTags>
  </configuration>
</plugin>
```



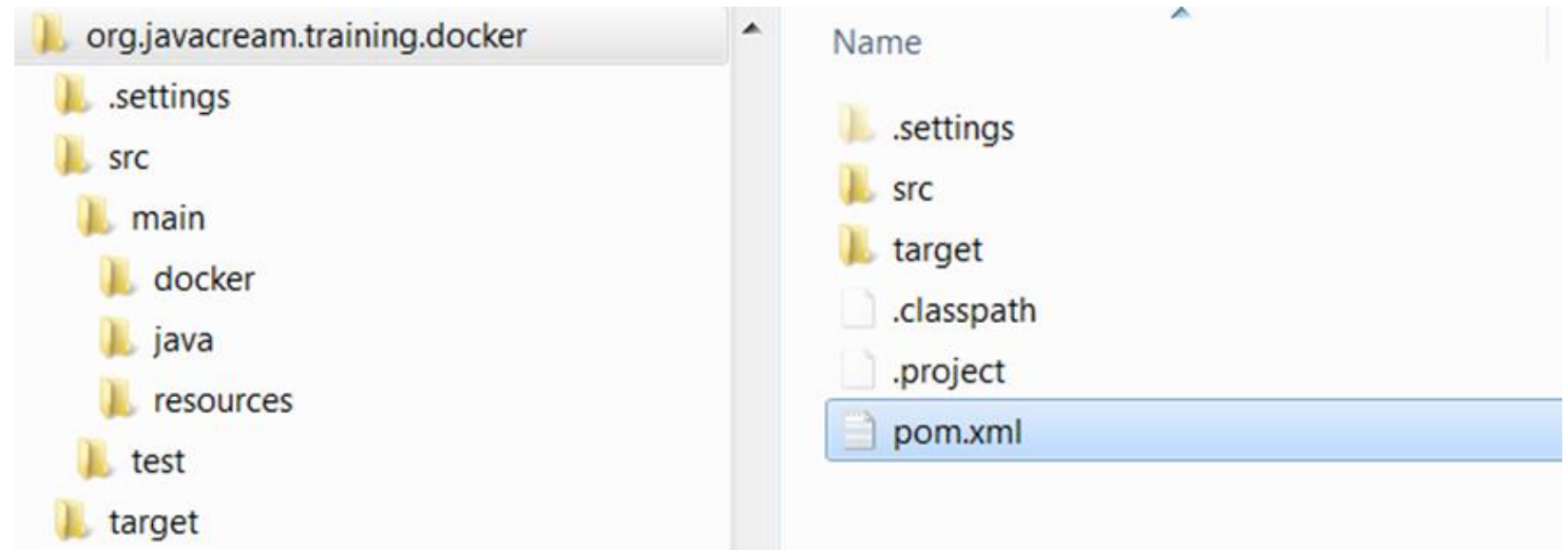
## Was macht das Docker-Plugin?

- Ab jetzt kann automatisiert ein Docker-Image erstellt werden
  - `mvn docker:build`
- Voraussetzung ist
  - ein vorhandenes Dockerfile
  - Eine zusätzliche Konfiguration des Plugins





# Ein fertiges Java-Projekt mit Docker-Unterstützung





# Der Repository- Server

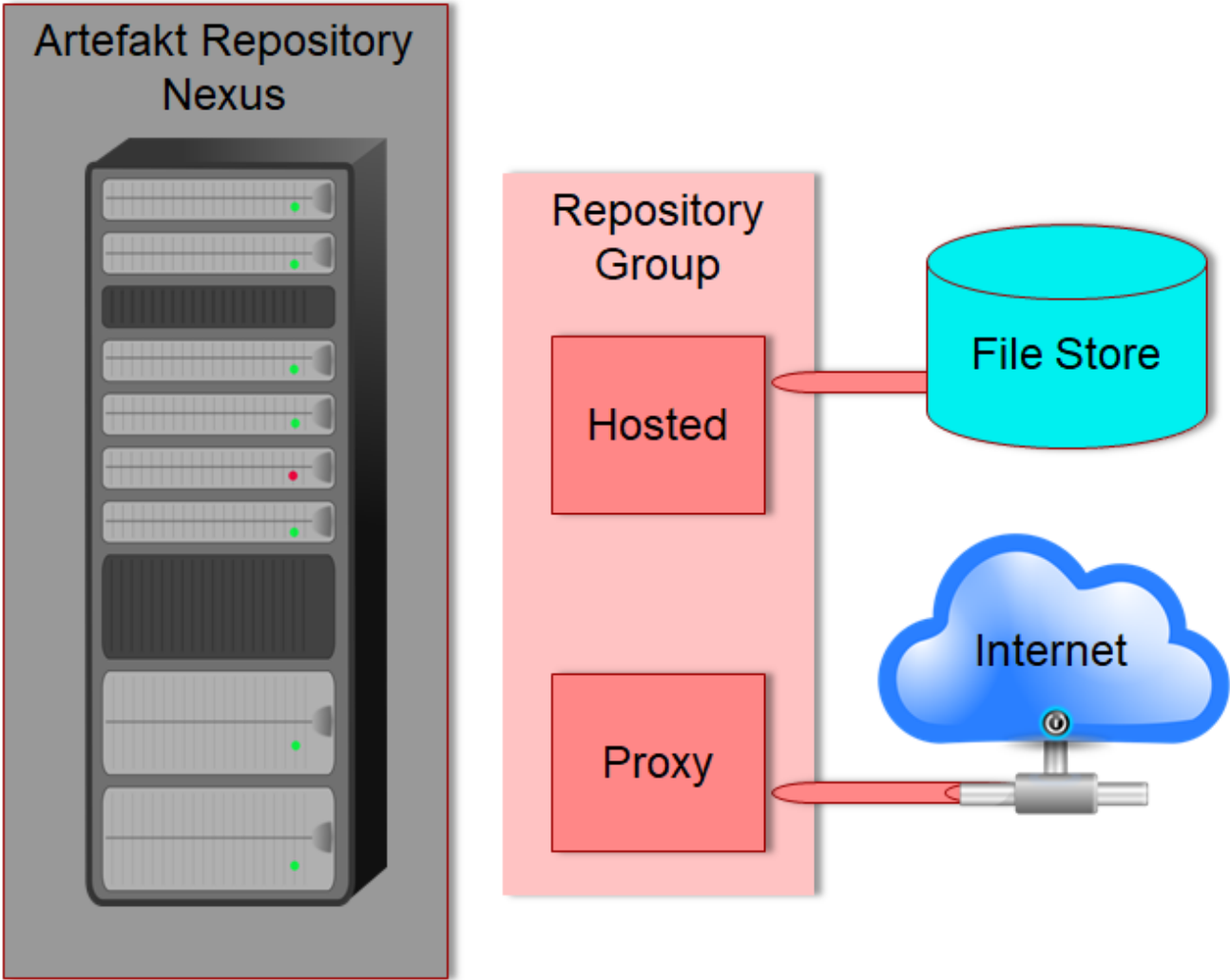


## Beispiel: Sonatype Nexus

- Fertige Produktlösung von Sonatype
  - Community
  - Kommerzieller Support
- Repository Server für
  - Java Artefakte
  - Aber auch Docker-Images
- Alternative Produkte
  - Apache Archiva
  - JFrog Artefactory
  - Atlassian Bitbucket



# Grundsätzliche Organisation

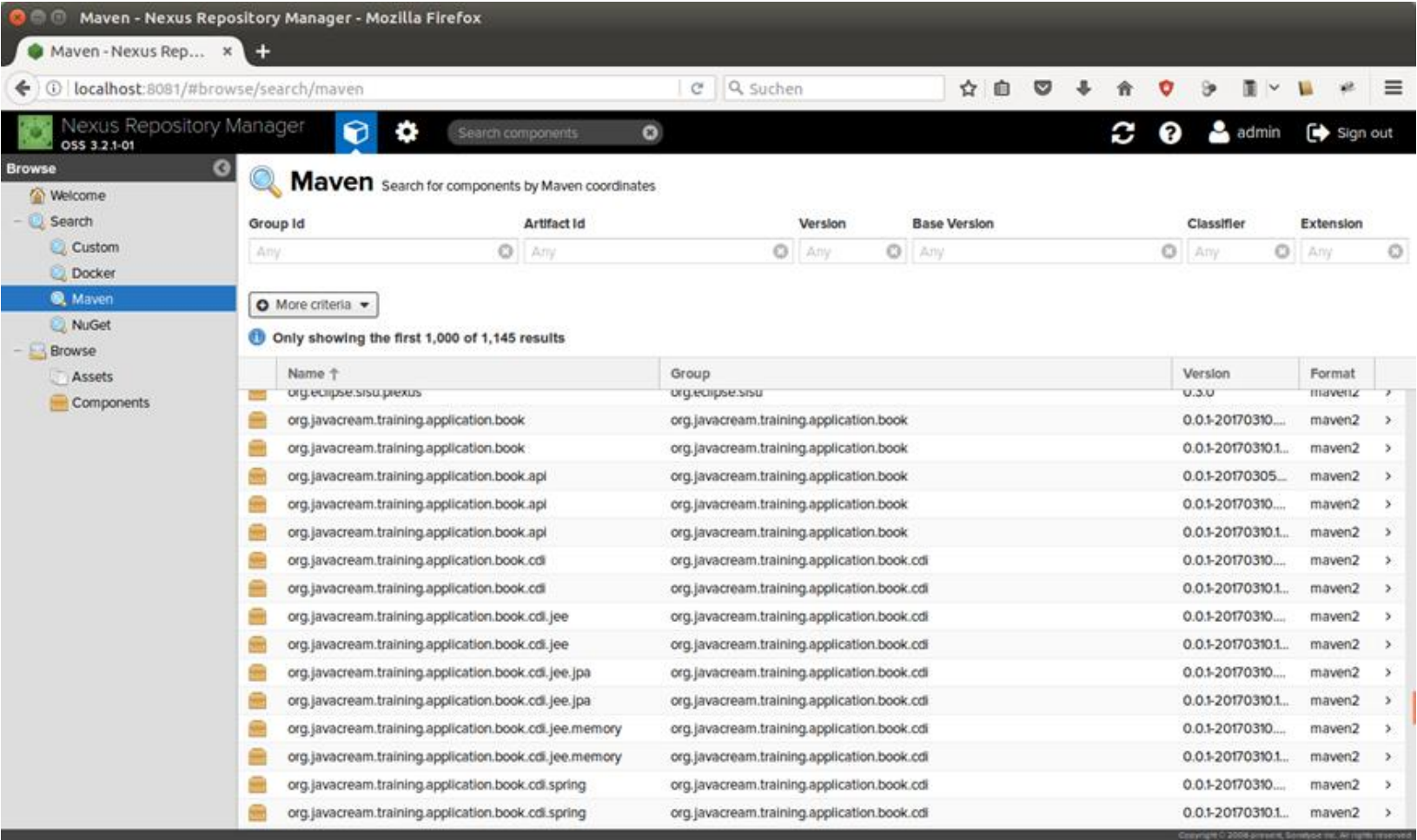




# Konfiguration

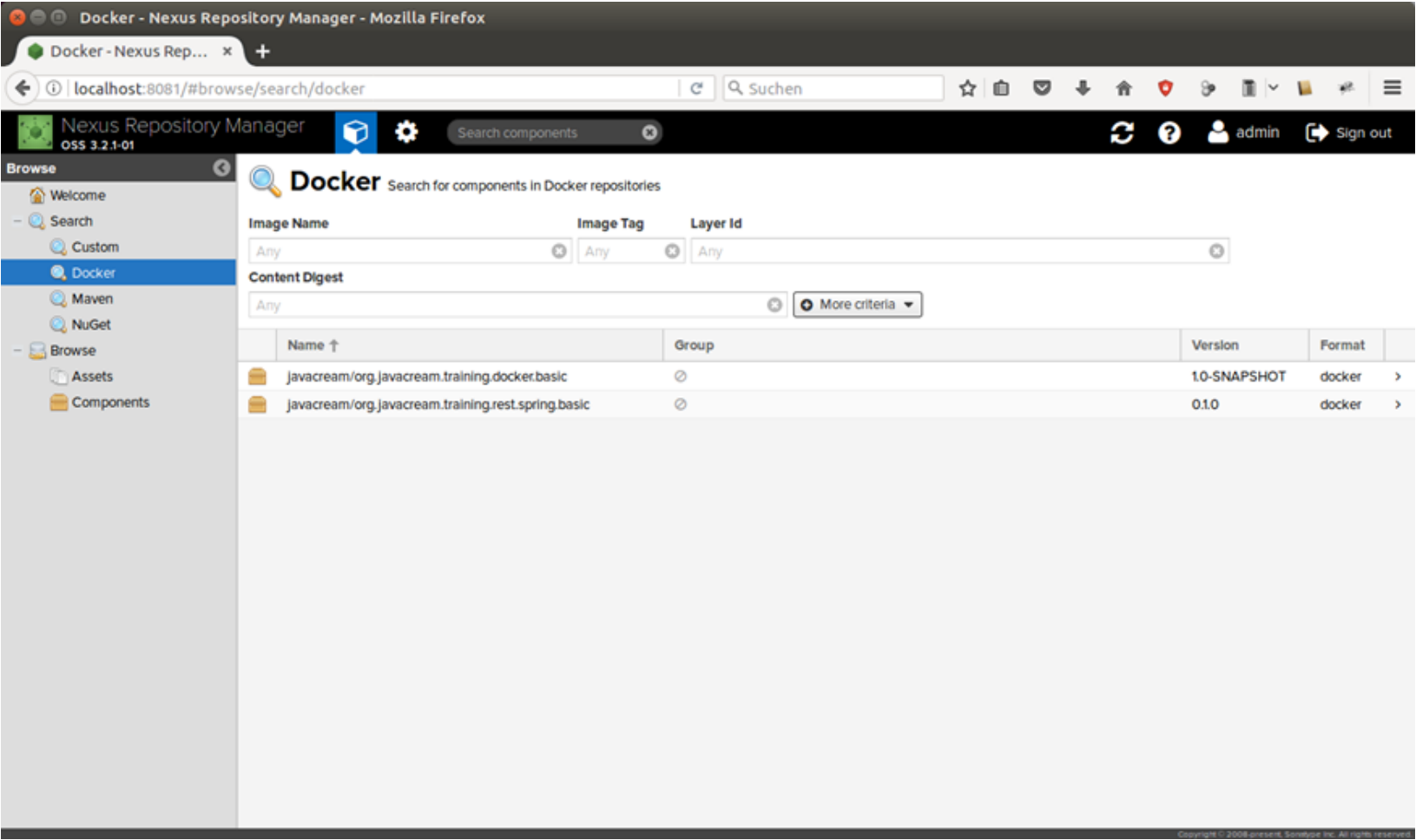
- Die Repository -Group wird für Lese-Vorgänge genutzt
  - Nicht vorhandene Artefakte oder Images werden über den Internet-Proxy nachgeladen
- Das Hosted-Repository wird für die Ablage eigener Artefakte und Images genutzt

# Nexus Maven-Repository





# Nexus Docker-Repository





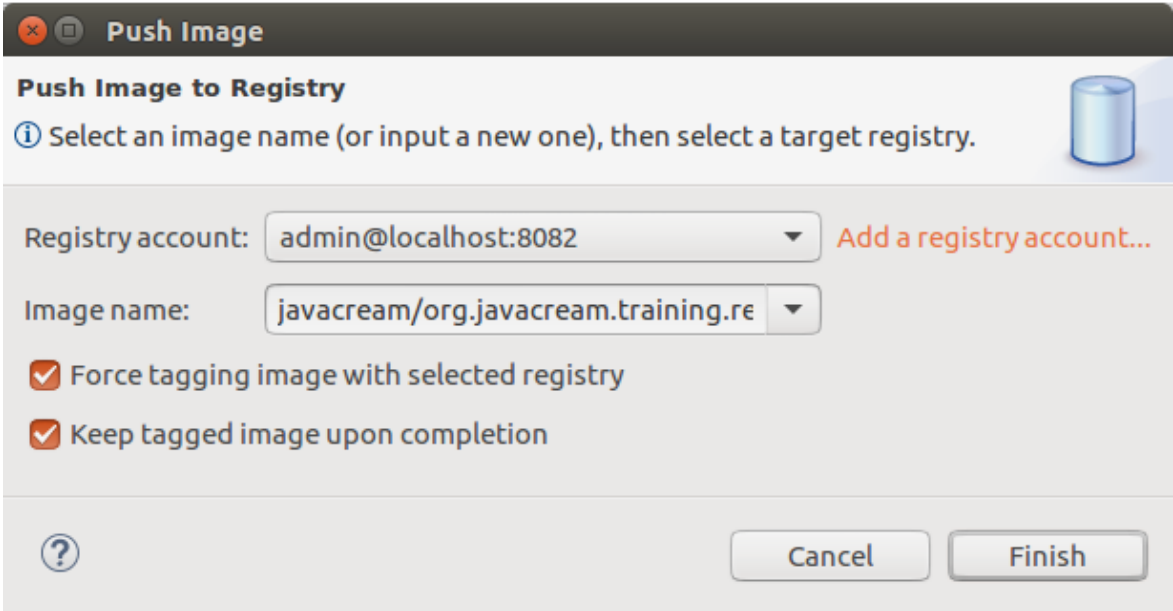
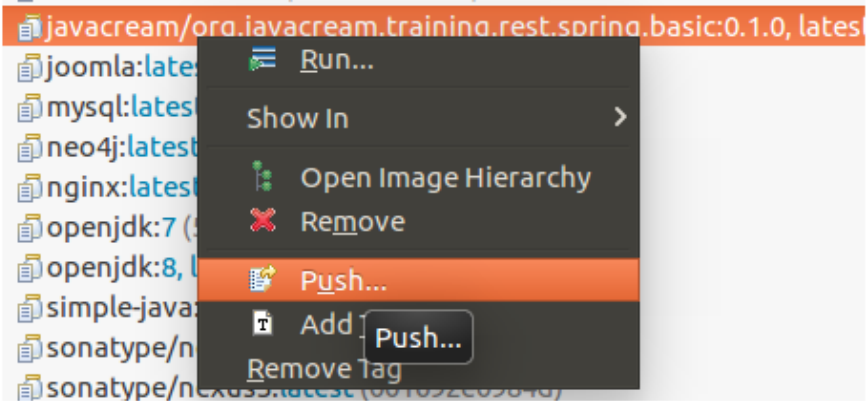
## Docker-Push

- Ablage der Credentials im Docker-System
  - `docker login -u user -p pwd`
- Images benötigen für die Ablage in eigenem Repository die Host-Information als spezielles Tag
  - `docker tag host:port/original:tag original:tag`
- Anschließend wird das Image gepushed
  - `docker push host:port/original:tag`
- Natürlich kann der Push auch über das Docker-Plugin und Maven erfolgen





# Docker-Push mit Eclipse





# System-Werkzeuge und Tools



Docker Compose



Docker Swarm



Kubernetes



OpenShift



# Docker Compose



# Docker Compose

- Mit Hilfe von Compose werden Container-Definitionen vereinfacht
  - Konfiguration
    - Ports
    - Volumes
    - Environment
    - Links auf andere Container
- Verwendet wird das YAML-Format
  - Docker Compose als Werkzeug ist keine echte Erweiterung von Docker, sondern ein Parser, der mit Hilfe eines Templates aus der YAML-Datei die korrespondierenden Aufrufe erzeugt



# Das Wordpress- Beispiel der Docker- Community

```
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
```



## Einige Befehle

- `up`
  - Erzeugt alle notwendigen Container und Volumes
  - Anschließend werden die Container gestartet
- `down`
  - Stoppt und löscht alle Container
- `start`
- `stop`



# Docker Swarm



# Konfiguration

- Basiert auf Docker Compose
- Die Container-Konfiguration wird um die Anzahl der jeweils notwendigen Knoten ergänzt





## Arbeitsweise

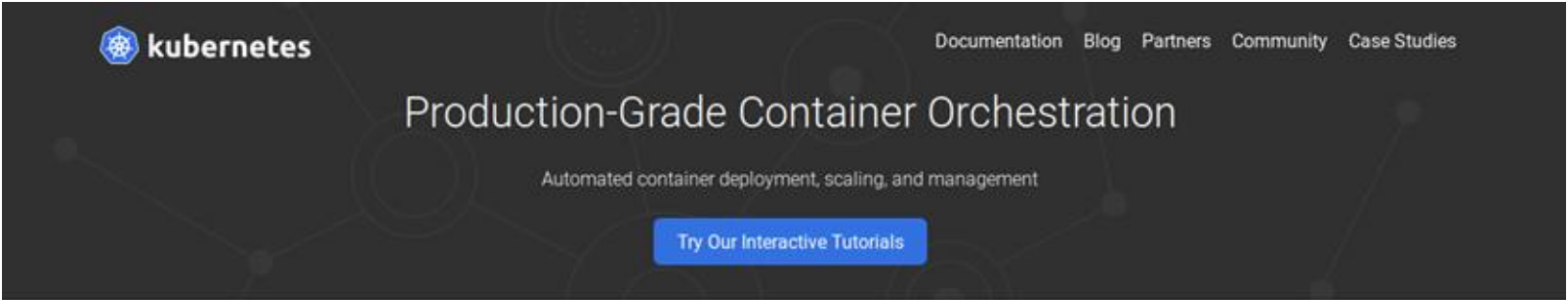
- Docker Swarm benutzt einen Master
- Dieser nimmt alle Anfragen entgegen und delegiert auf einen Container, der auf einem Host läuft
- Zum Starten und Stoppen der Container muss auf jedem Host ein Swarm Agent installiert sein
- Beispiel und Tutorial Bestandteil der Docker-Dokumentation



# Kubernetes



# kubernetes.io



Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.



## Planet Scale

Designed on the same principles that allows Google to run billions of containers a week, Kubernetes can scale without increasing your ops team.

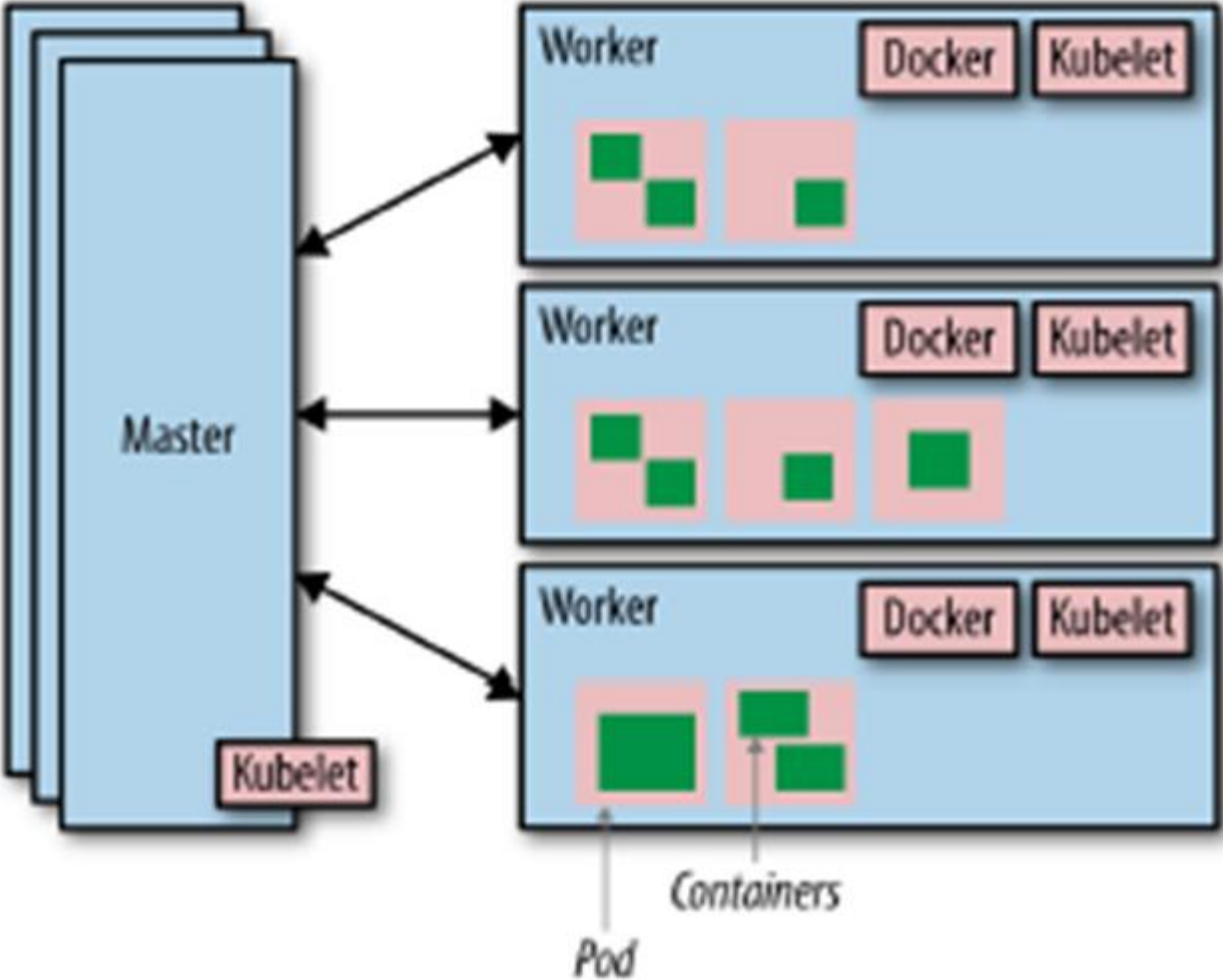
## Never Outgrow

Whether testing locally or running a global enterprise, Kubernetes flexibility grows with you to deliver your applications consistently and easily no matter how complex your need is.





# Architektur

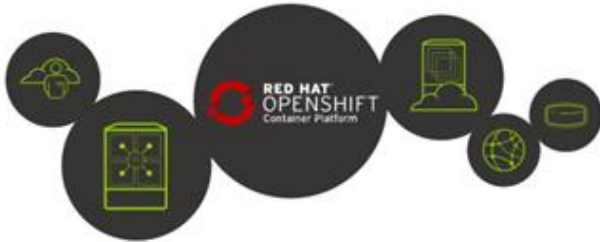
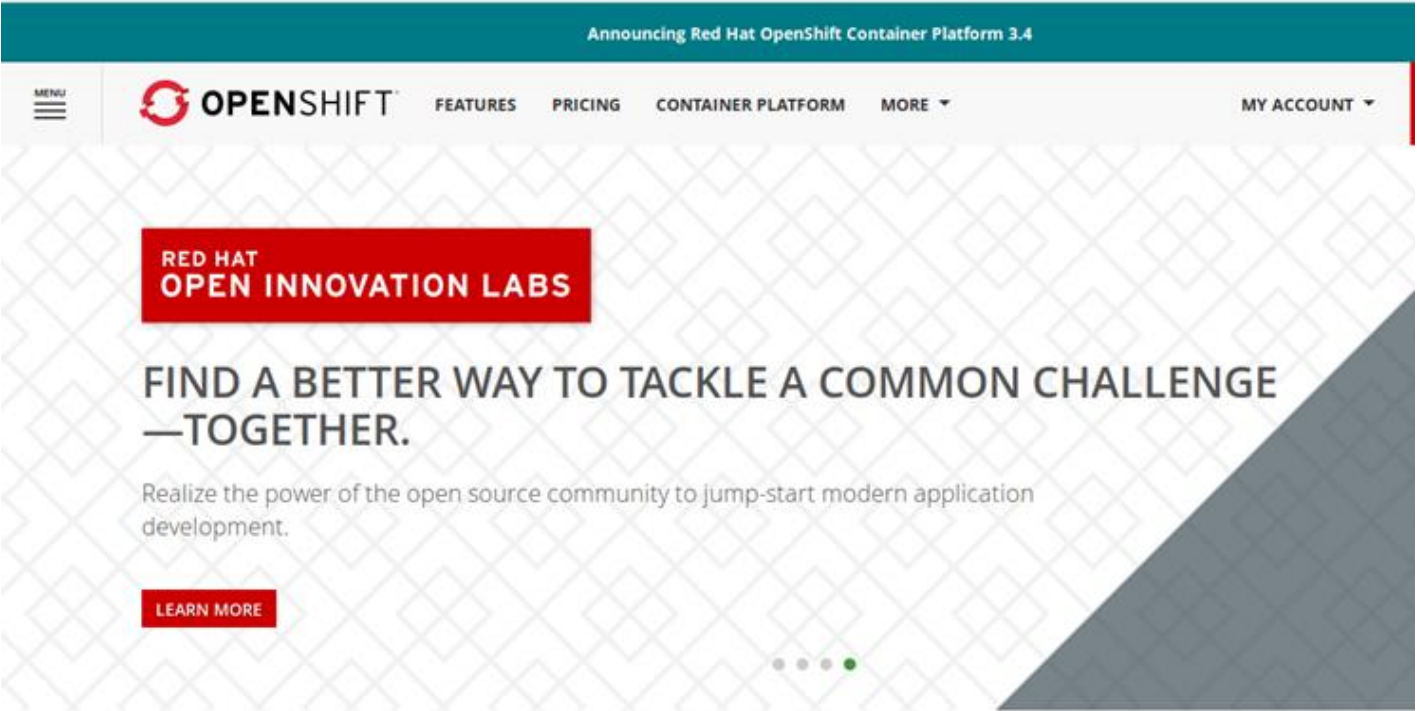




# OpenShift

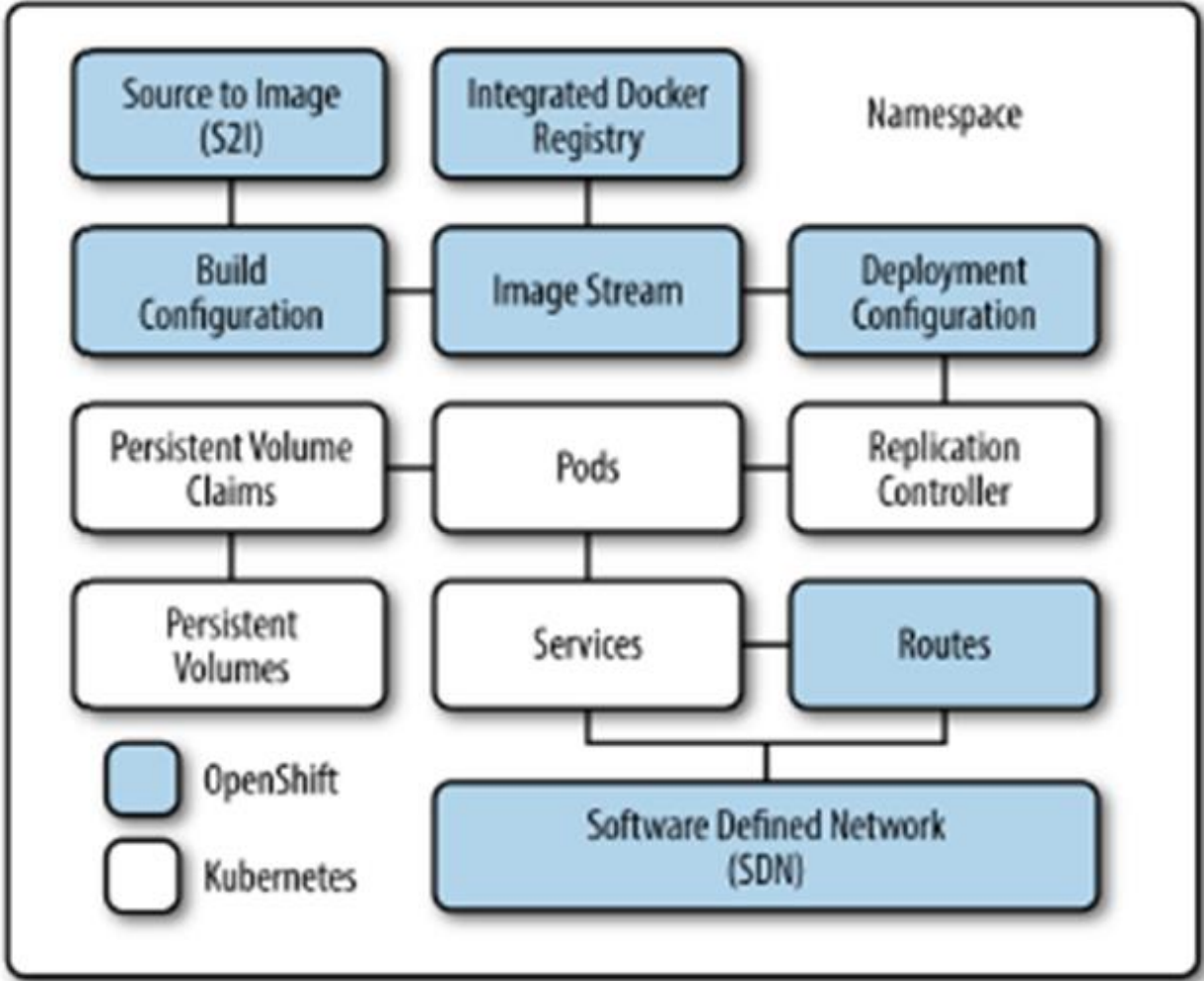


# OpenShift



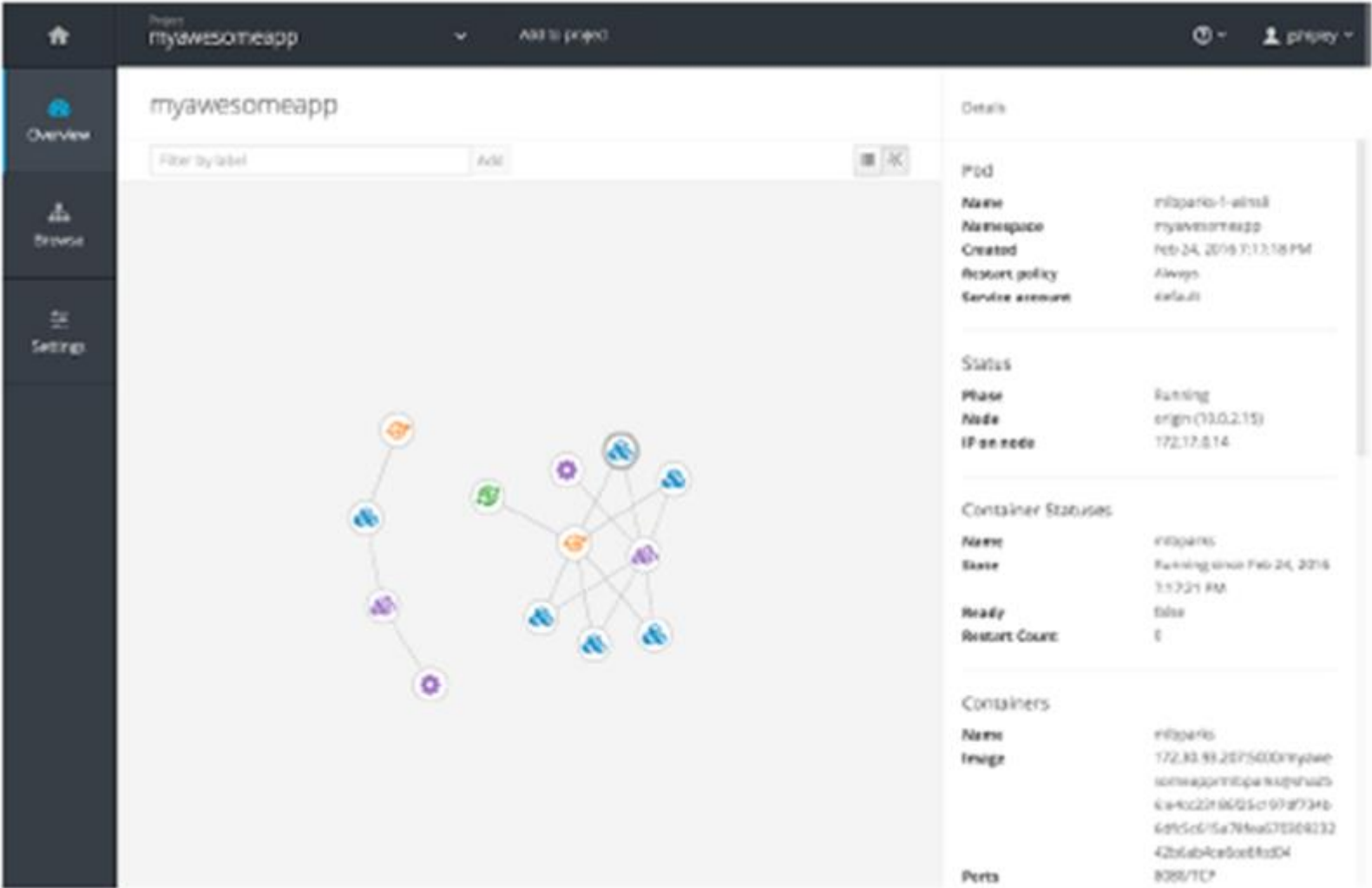


# Architektur





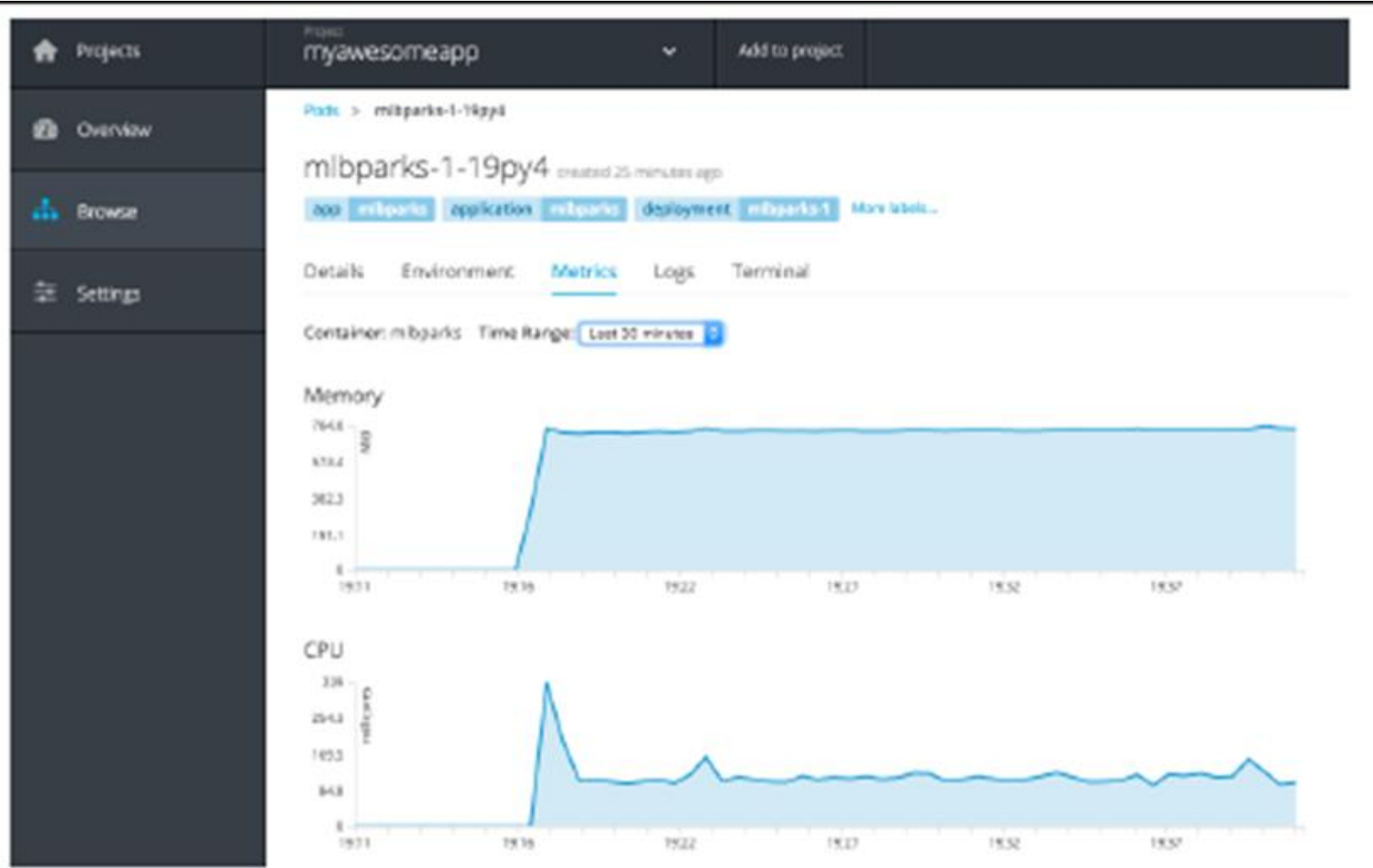
# Anwendungs- Visualisierung







# System-Überwachung





# Java Enterprise Anwendungen



- Überwachung von Java-Prozessen mit JMX
- Enterprise Java
- Applikationsserver
- Spring



# Überwachung von Java-Prozessen mit JMX



## Problemstellung

- Docker liefert eine Überwachungsmöglichkeit auf Ebene der Container
- Diese Überwachung ist jedoch für Java-Prozesse nicht sonderlich aufschlussreich
  - Metriken der Java-Virtual Machine
    - Heap-Speicher
    - Garbage Collections
- Hierfür wird besser JMX benutzt
  - Idee: Auf einem (im Unternehmen standardisierten) Port wird JMX beispielsweise mit Jolokia bereitgestellt
    - Hierfür existieren aber auch andere Lösungen



# Jolokia



The screenshot shows the Jolokia website. At the top is the Jolokia logo with the tagline "JMX on Capsaicin". Below the logo is a navigation bar with links: Home, Download, Features, Documentation, Support, Blog, and About. The main content area is divided into a left sidebar and a right main section. The sidebar contains three sections: "Jolokia" with links to Download, Features, Support, Forum, IRC, and License; "Documentation" with links to Overview, Tutorial, Reference Manual, and Talks and Screenshots; and "Agents" with links to Overview, Web Archive (war), Osgi, JVM, and Mule. The main section features a large red banner with the text "Jolokia is remote JMX with JSON over HTTP. It is fast, simple, polyglot and has unique features. It's JMX on Capsaicin." Below the banner is a paragraph describing Jolokia as a JMX-HTTP bridge. Under the heading "Starting points", there is a bulleted list of key features and resources. At the bottom of the main section is a "News" section.

**Jolokia**  
JMX on Capsaicin

Home Download Features Documentation Support Blog About

**Jolokia**

Download  
Features  
Support  
Forum  
IRC  
License

**Documentation**

Overview  
Tutorial  
Reference Manual  
Talks and Screenshots

**Agents**

Overview  
Web Archive (war)  
Osgi  
JVM  
Mule

**Jolokia is remote JMX with JSON over HTTP.**  
It is fast, simple, polyglot and has unique features. It's JMX on Capsaicin.

Jolokia is a JMX-HTTP bridge giving an alternative to JSR-160 connectors. It is an agent based approach with support for many platforms. In addition to basic JMX operations it enhances JMX remoting with unique features like bulk requests and fine grained security policies.

**Starting points**

- Overview of **features** which make Jolokia unique for JMX remoting.
- The **documentation** includes a **tutorial** and a **reference manual**.
- Agents** exist for many platforms (JEE, OSGi, Mule, JVM).
- Support** is available through various channels.
- Contributions** are highly appreciated, too.

**News**



## Dockerfile mit Jolokia- Unterstützung

```
FROM openjdk:latest
RUN mkdir /in-dir
ADD org.javacream.training.docker.jmx-1.0-
    SNAPSHOT.jar app.jar
ADD libs/jolokia-jvm-1.3.5-agent.jar jolokia.jar
VOLUME /in-dir
EXPOSE 7777
ENTRYPOINT java
    -
    javaagent:jolokia.jar=port=7777,host=0.0.0.0
    -cp app.jar
    Application
```



# Enterprise Java

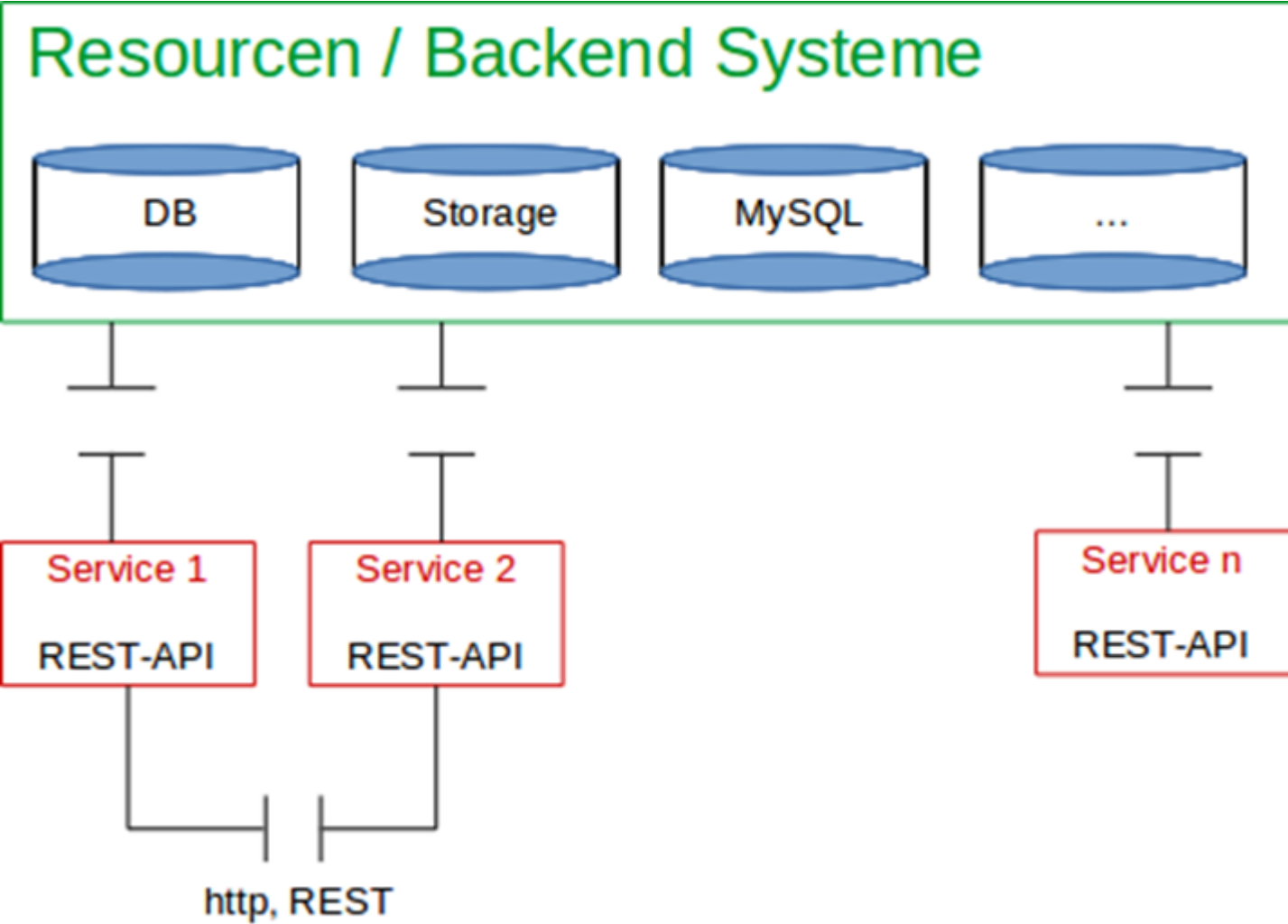


# Allgemeines

- Enterprise-Applikationen sind meistens mehrschichtig gestaltet
  - Web Frontend
  - Service Frontend
    - z.B. RESTful Web Services
  - Transaktionelle Datenzugriffe
  - Backend-Ressourcen

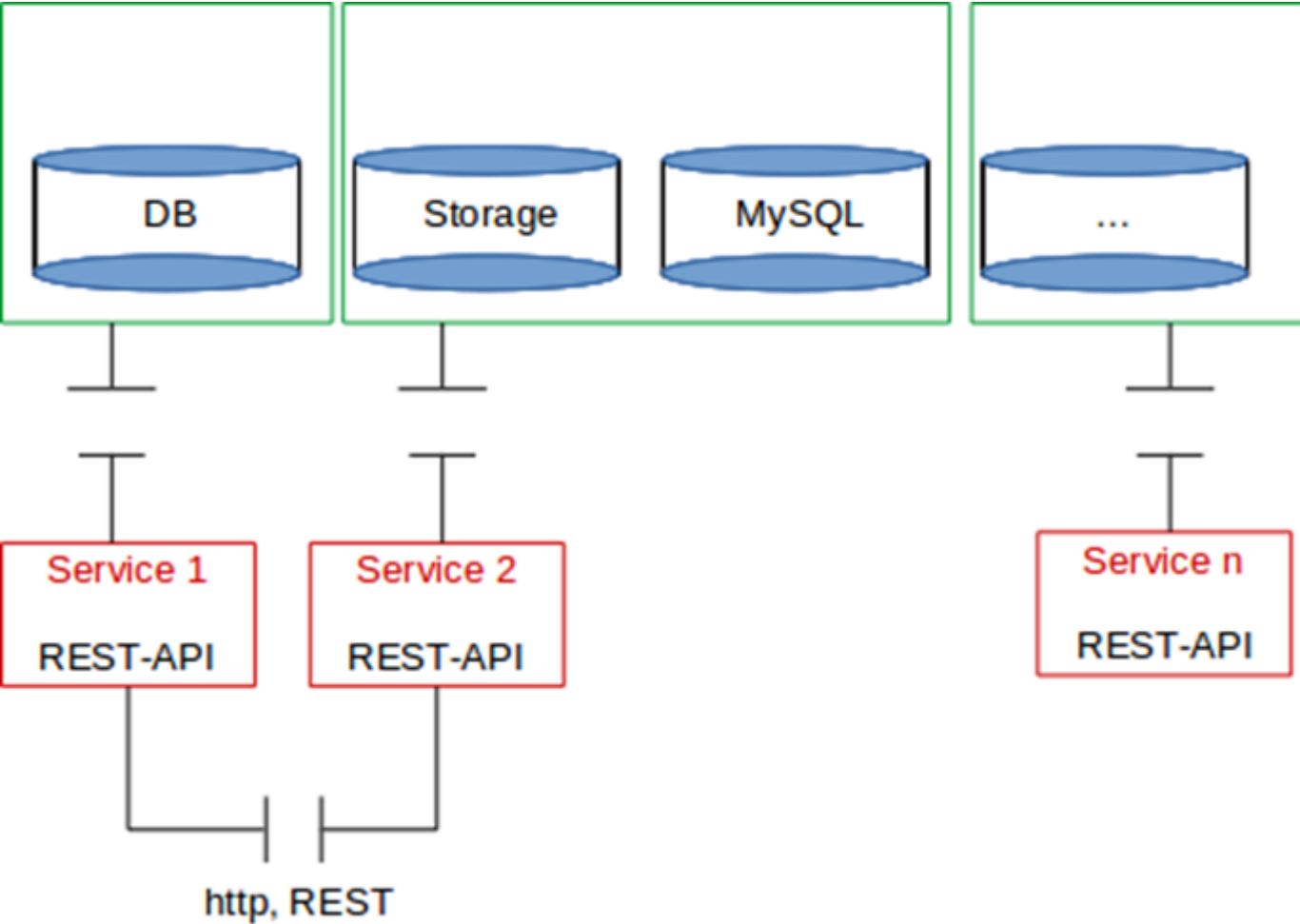


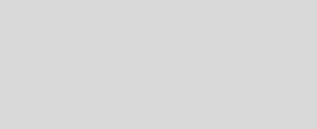
# Architektur mit gemeinsamen Ressourcen



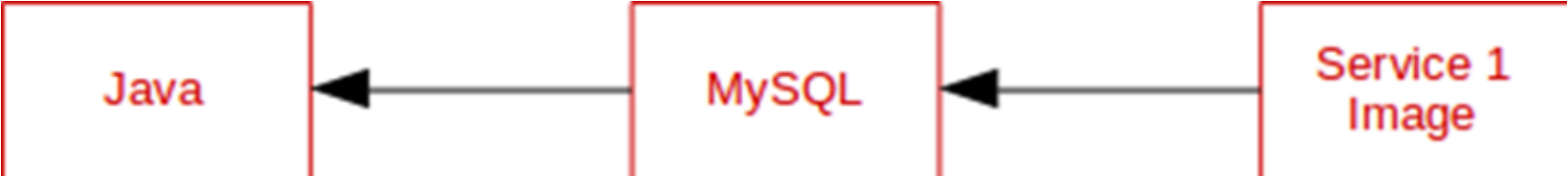


# Microservices mit gekapselten Ressourcen



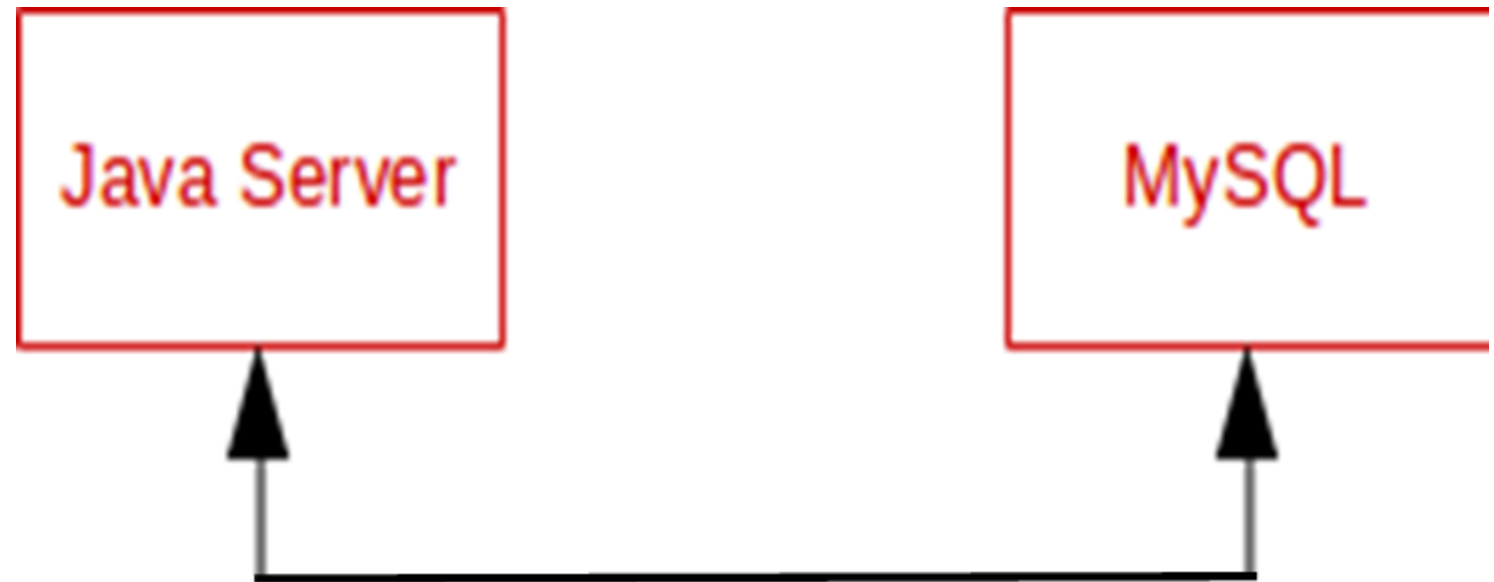


# Realisierung mit Docker Image- Hierarchie





## Realisierung mit verlinkten Containern





# Applikationsserver



# Allgemeines

- Applikationsserver sind Bestandteil der Java Enterprise Edition
  - Eine Spezifikation
- Ausprägungen:
  - Web Profile
    - Web Anwendungen
    - Web Services
  - Full Profile
    - Transaktionale Ressourcen-Zugriffe
    - Messaging
  - Weitere inoffizielle Profile
    - Reine Messaging Systeme
    - Web Profile mit einfachem Transaktionsmanagement



# Deployment

- Ausbringen von Anwendungen in den laufenden Server
  - Hot Deployment
    - File-Transfer in ein überwachtes Verzeichnis
  - Administrativer Vorgang
    - Web Console
    - Admin-Skripte
  - Konfigurativ
    - mit Neustart des Servers



# Überwachung

- Der Server ist über JMX zugreifbar
- Web Konsolen
- Log-Dateien
- Betriebssystem-Überwachung
  - Speicher
  - CPU
  - IO
- Bei Verwendung des Docker-Containers übernimmt dieser die Rolle des Betriebssystems!





## Realisierung mit Docker

- Es existieren fertige Images für gängige Open Source Applikationsserver
  - Apache Tomcat
  - JBoss/Wildfly
  - Glassfish
  - Active MQ
  - ...
- Deployment durch
  - Erzeugen eines neuen Containers
  - Verwendung von Docker-Volumes
- Überwachung
  - Mappen der Ports und somit Netzwerk-Zugriff
  - Verzeichnis der Log-Dateien wird extern gemapped



# Spring



# Spring Source

- Ein weit verbreitetes Open Source-Framework als Alternative zum JEE-Applikationsserver
- Spring-Core-Komponente und viele assoziierte Projekte
  - Spring Data
  - Spring MVC
  - Spring Integration
  - ...



# Spring Boot

- Eine Spring-Boot-Applikation wird als ein einziges Deployment-Archiv ausgeliefert
  - und über einen simplen Java-Aufruf gestartet
- Damit passt Spring Boot hervorragend zu Docker
  - Als Grundlage dient ein simples Java Image
  - Die zu installierende Applikation wird als eine einzige Datei hinzugefügt
  - und über einen Standard-Java-Aufruf gestartet