



Microservices

Service Oriented Architecture 2.0



Cegos Group

inspire
qualify
change



Inhalts- verzeichnis



Einführung



Software-Entwicklung






Container für Microservices



Einführung



-  Von Monolithen und Services
-  Was sind Microservices
-  Modellierung



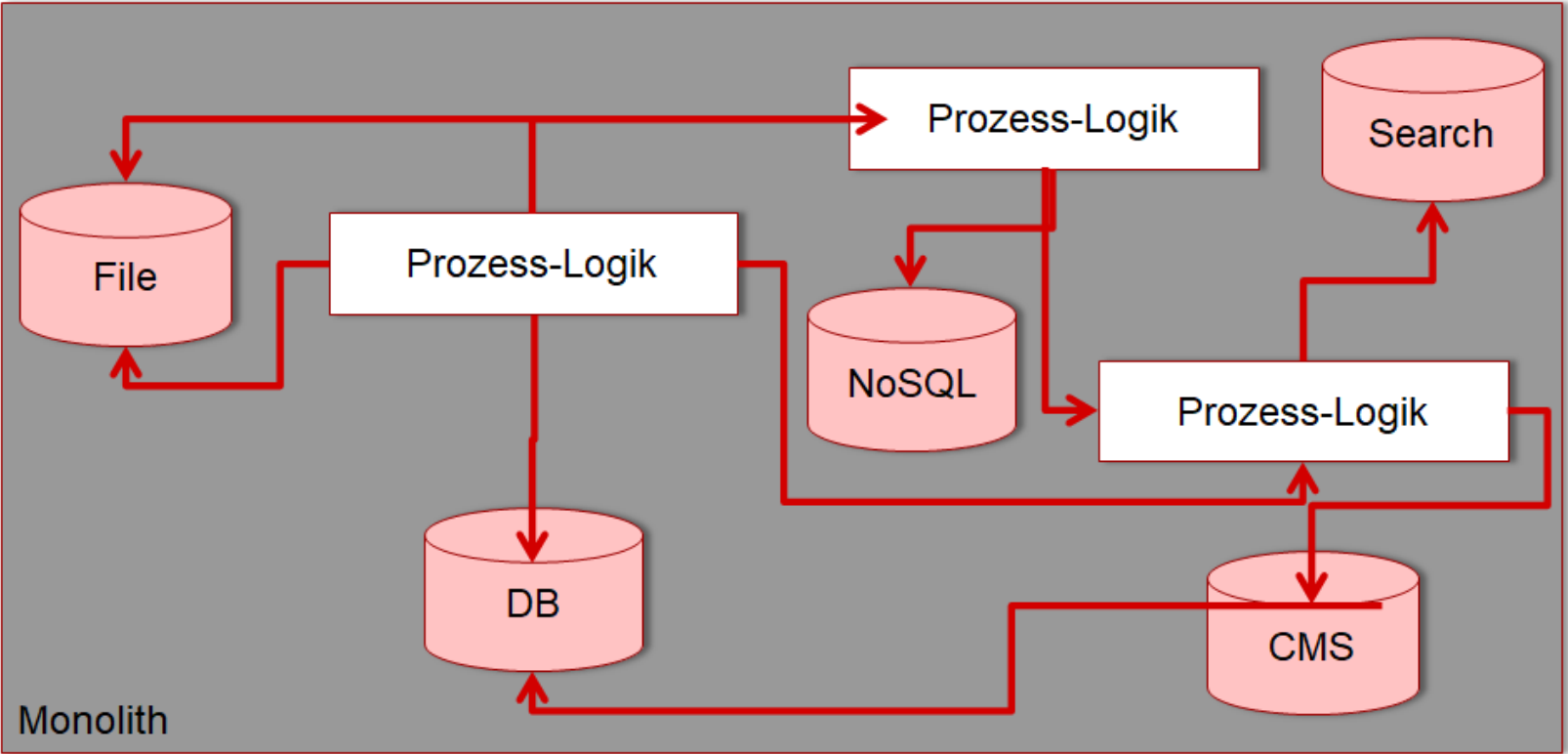
Von Monolithen und Services



Eine Anwendung

- Primärfokus:
 - Anwendung muss fachlich funktionieren
 - Nicht-funktionale Aspekte wie Performance und Ausfallsicherheit
- Eher uninteressant sind:
 - Details der technischen Modellierung
 - Wartbarkeit und Erweiterbarkeit, Wiederverwendung

Monolith

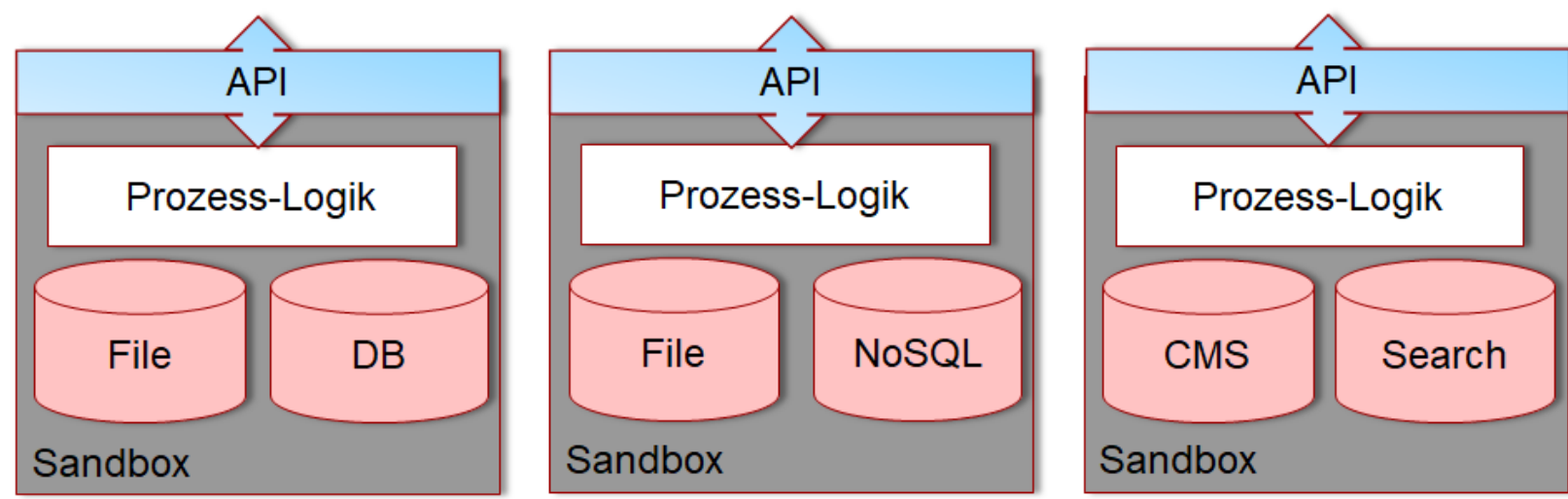




Was wäre "besser"?

- Kapselung
 - Öffentliches „API “
 - Rein interne Implementierung
- Vermeidung von Redundanzen
 - Wartbarkeit
- Dependency Management
 - Statisch determinierte Abhängigkeiten oder
 - lose gekoppelte Systeme
 - Damit auch leichtere Erweiterbarkeit

Gekapselte Services





Das API

- Besteht abstrakt aus
 - Datentypen
 - Operationen
 - Synchron und Asynchron
- Konkretisierung mit weiteren Informationen
 - Authentifizierung
 - Aufruf über Netzwerk
 - Nicht-funktionale Teile
 - Antwortzeiten
 - Skalierbarkeit
 - Fehlertoleranz



Dokumentation ist alles

- Auch eine auf Microservices beruhende Architektur steht und fällt mit der Dokumentation
 - Vollständig
 - so natürlich utopisch
 - "eventual complete"
 - Aktuell
 - Unmissverständlich
 - Prägnant
 - Synchron zur Realisierung



Realisierung der API-Beschreibung

- (Formale) Prosa-Texte
 - Fachdokumentationen
- Unified Modelling Language
- Service-Beschreibung mit
 - Web Service Description Language
 - WSDL ist eine Spezifikation des W3W-Konsortiums
 - Primär nur in Kombination mit SOAP-basierten Web Services
- Swagger
 - Proprietäre Beschreibungssprache
 - swagger.io
 - UI und Code Generatoren für verschiedene Sprachen werden angeboten
 - Primär ausgerichtet auf RESTful Web Services
- Elemente einer Programmiersprache
 - Beispielsweise Java-Interfaces und Annotationen



Stereotypen zur Vermeidung von Redundanzen

- Grundprinzipien der Software-Qualität gelten auch für die Dokumentation
 - Don't repeat yourself!
- UML Stereotypen
 - Prägnanter Name
 - Gültigkeitsbereich
 - Auswirkung
 - Beispiel:
 - Ein <<stateless>> System baut kein Konversationsgedächtnis auf
- Stereotypen sind auch unter dem Namen Annotations bekannt



Definition der Service- Kommunikation

- Etablierte Protokolle stehen zur Verfügung
 - SOAP
 - REST
 - Messaging
 - Java RMI
- Was ist "richtig"?
 - Messaging-Systeme bleiben
 - Trend geht zu REST
 - SOAP wird häufig als unnötig komplex gewertet
 - Java RMI ist eine Nischen-Lösung



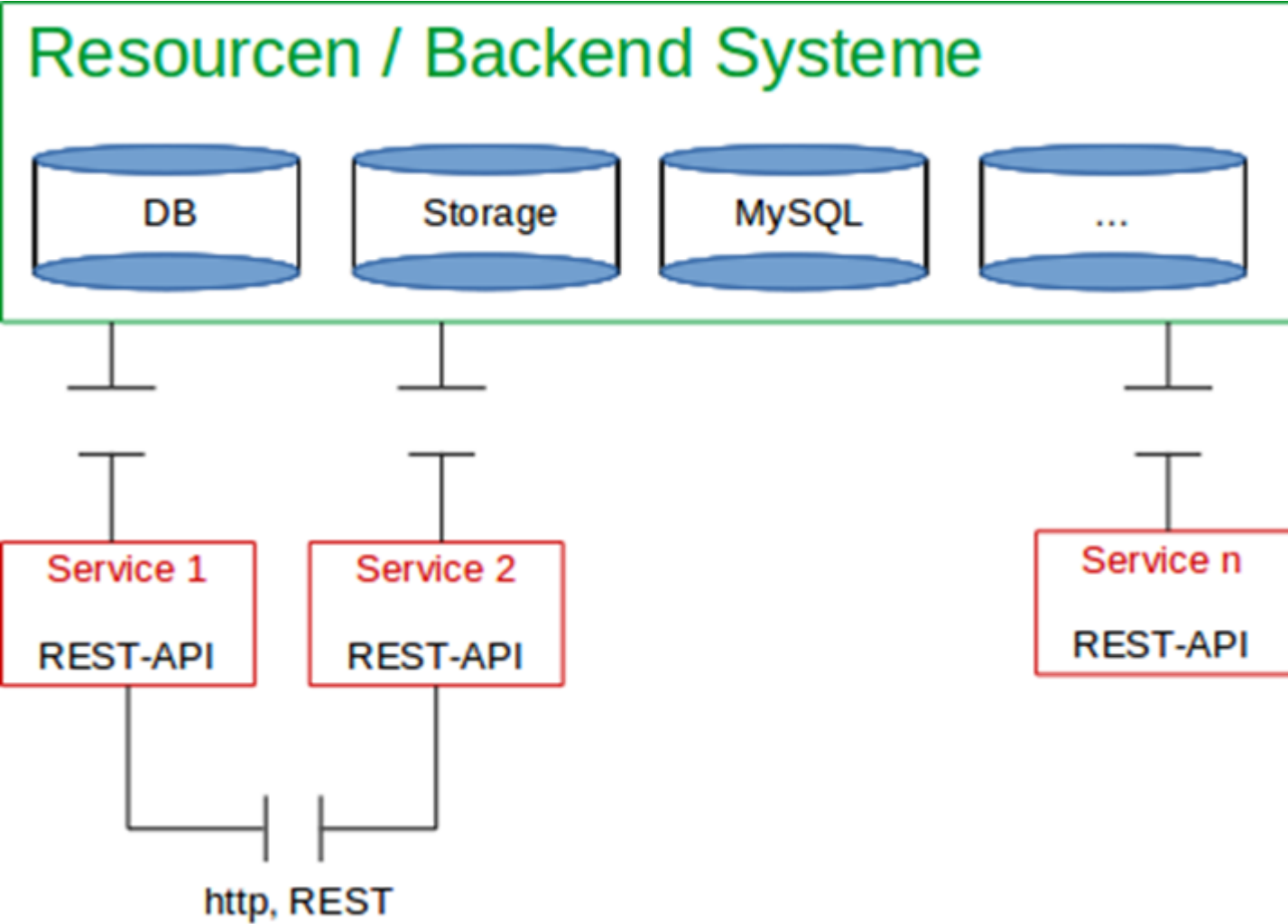
Was sind Microservices



Problemstellung

- Eine vollständige API-Beschreibung für den Aufruf des Services ist leider nicht genügend!
- Services können durch die eigentlich "interne" und gekapselte Implementierung trotzdem gekoppelt bleiben
 - Gemeinsam genutzte Backend-Systeme!
 - Damit hat jeder Service damit zwei APIs, die beschrieben und stabil gehalten werden müssen

Architektur mit gemeinsamen Ressourcen

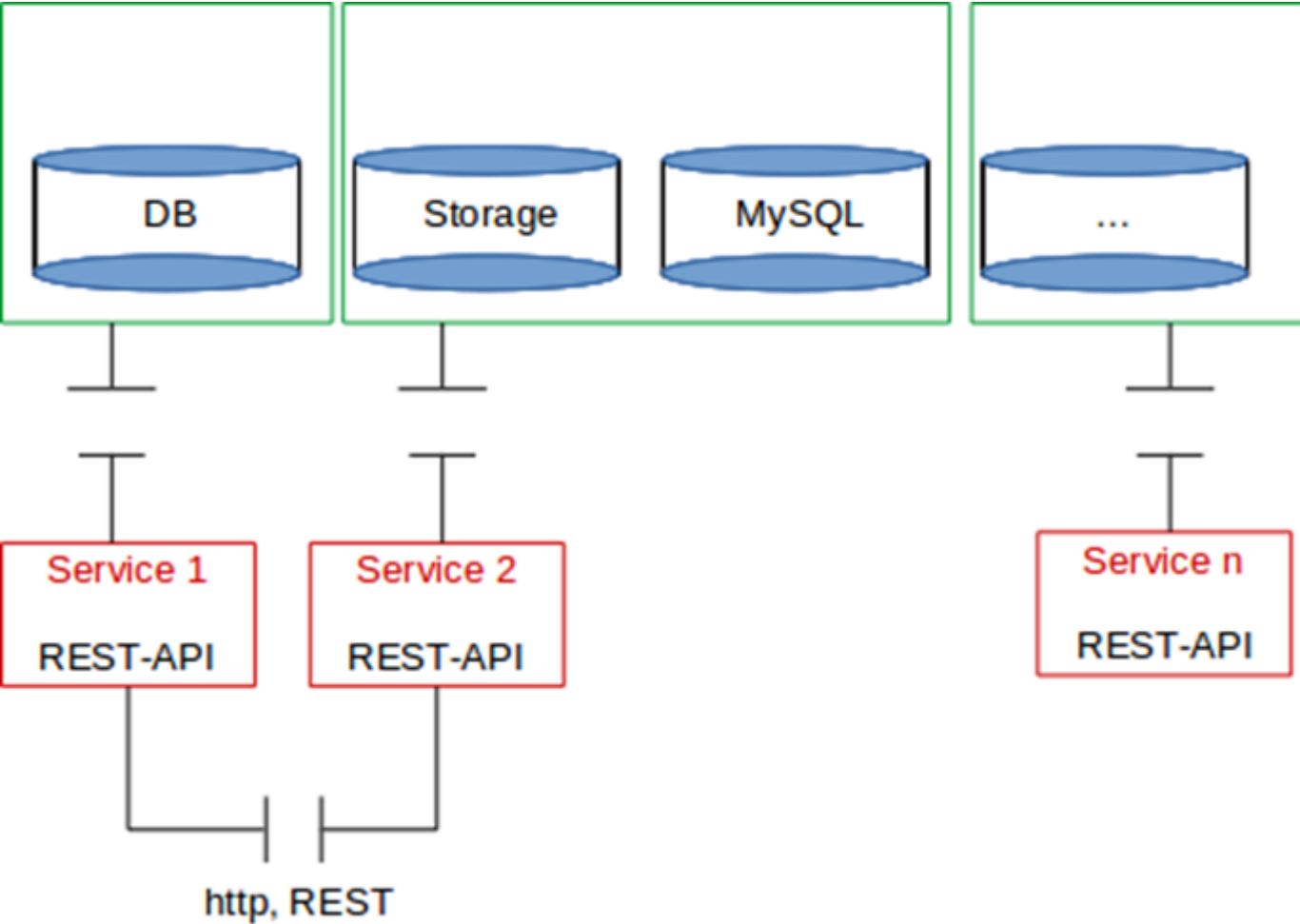




Definitionen

- WIKI
 - <https://de.wikipedia.org/wiki/Microservices>
- Martin Fowler
 - <https://martinfowler.com/articles/microservices.html>
- Microservice.io
 - <http://microservices.io/patterns/microservices.html>
- Alles noch etwas im Fluss...

Microservices mit gekapselten Ressourcen





Microservices kompakt

- Der Microservice wird durch ein versioniertes API vollständig beschrieben
- Die Implementierung ist komplett gekapselt
 - Kriterium: Selbst eine Neuimplementierung ist möglich, ohne andere Services zu ändern
- Jeder Microservice bringt seine Backend-Ressourcen mit
 - Kriterium: Ein Austausch eines Backend-Systems ist möglich



Warum "Micro"?

- Ein Microservice
 - "besteht nur aus einer überschaubaren Anzahl von Programmzeilen"
 - "Kann in zwei Wochen neu entwickelt werden"
 - "Wird von kleinen Teams verantwortet"
- Eine irreführende Bezeichnung
 - Selbst ein "kleiner" Service kann die Anforderungen an Microservices verletzen
 - Auch eine komplexe (vielleicht intern sogar monolithisch realisierte!) Anwendung kann als Microservice betrieben werden



Microservices und Daten-Redundanzen

- Nachdem jeder Microservice seine Ressourcen intern kapselt, müssen häufig Daten-Redundanzen in Kauf genommen werden
 - Häufig werden Redundanzen jedoch auch schon auf Grund von "Big&Fast Data" toleriert
 - Damit haben Microservices einen direkten Bezug zu NoSQL-Themen!

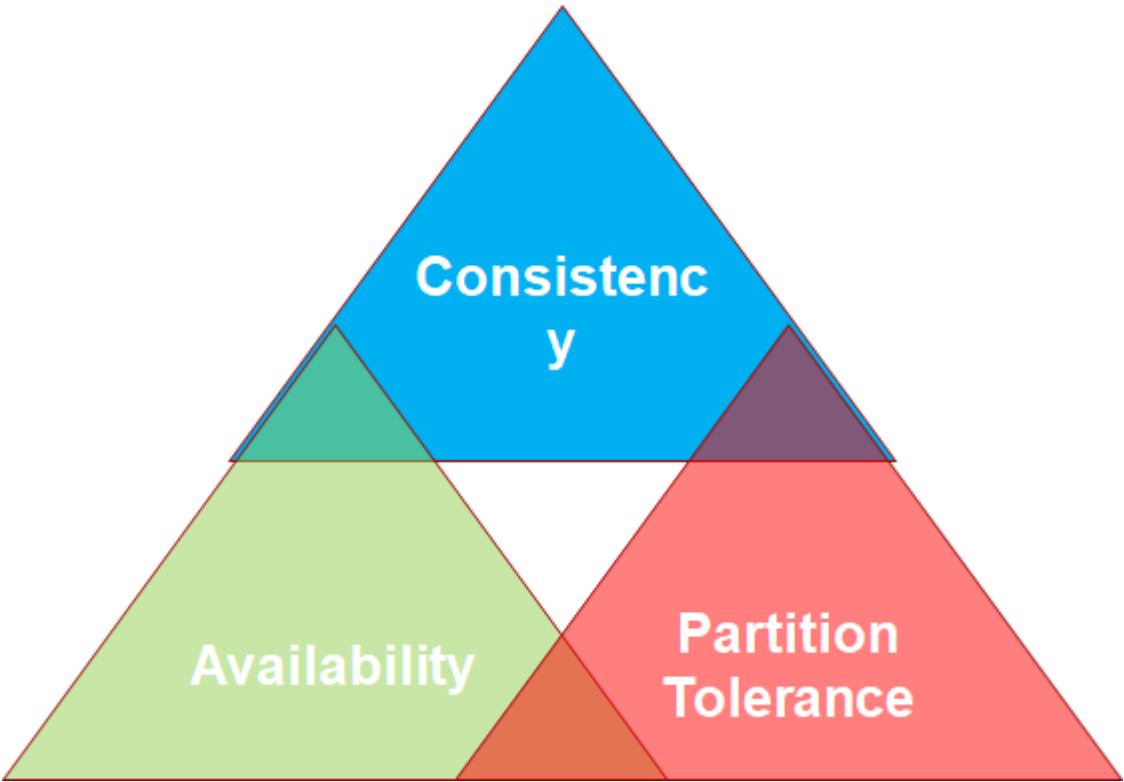


Das CAP-Theorem

- Consistency
 - Alle Knoten haben jederzeit den selben Informationsstand
- Availability
 - Jeder Client bekommt garantiert Antwort
 - Solange zumindest ein Knoten im Cluster läuft
- Partition Tolerance
 - Das System toleriert
 - Den Ausfall von Knoten
 - Den Ausfall des Netzwerks zwischen Knoten



Keine gemeinsame
Schnittmenge!





BASE: Ein Schlupfloch

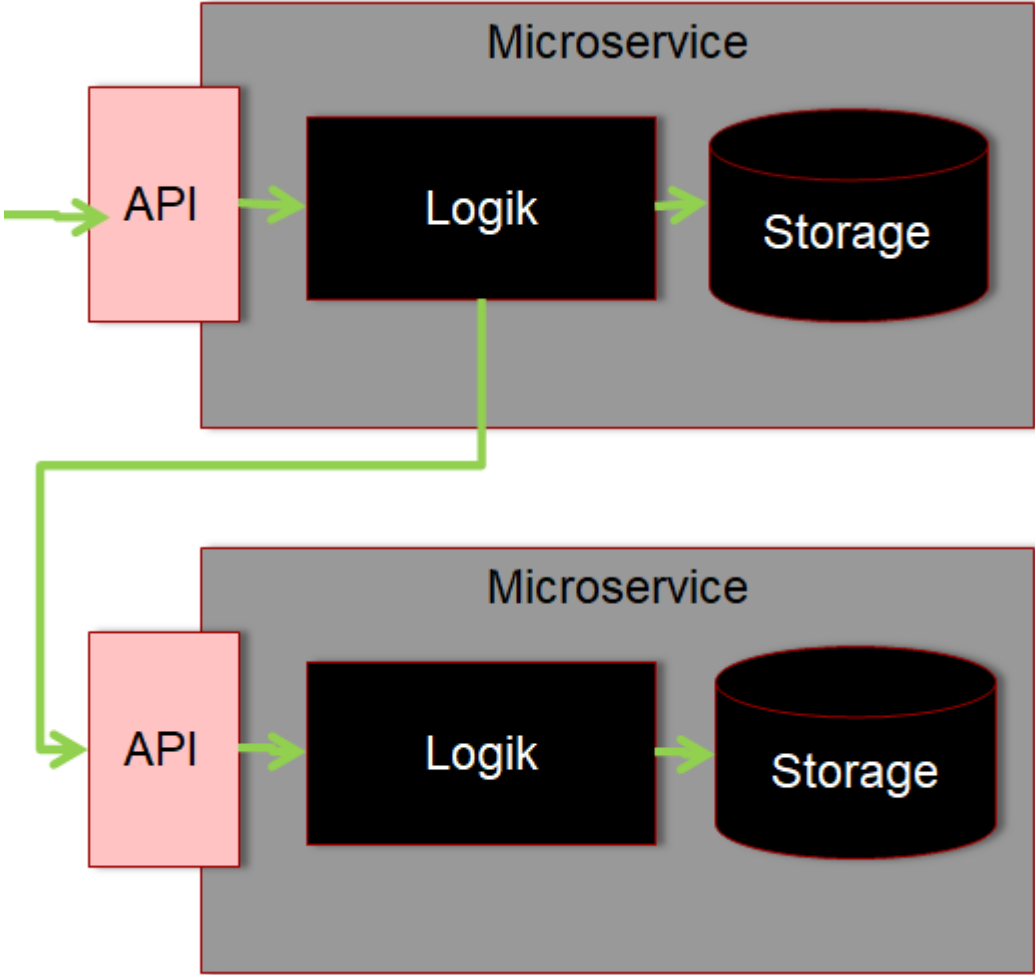
- Consistency
 - Alle Knoten haben **jederzeit** den selben Informationsstand
- Eventually Consistent
 - Änderungen des Datenbestandes werden zeitlich versetzt an die anderen Knoten propagiert
 - Das System wird schlussendlich konsistent
- BASE
 - **B**asically **A**vailable
 - **S**oft state
 - **E**ventual consistency



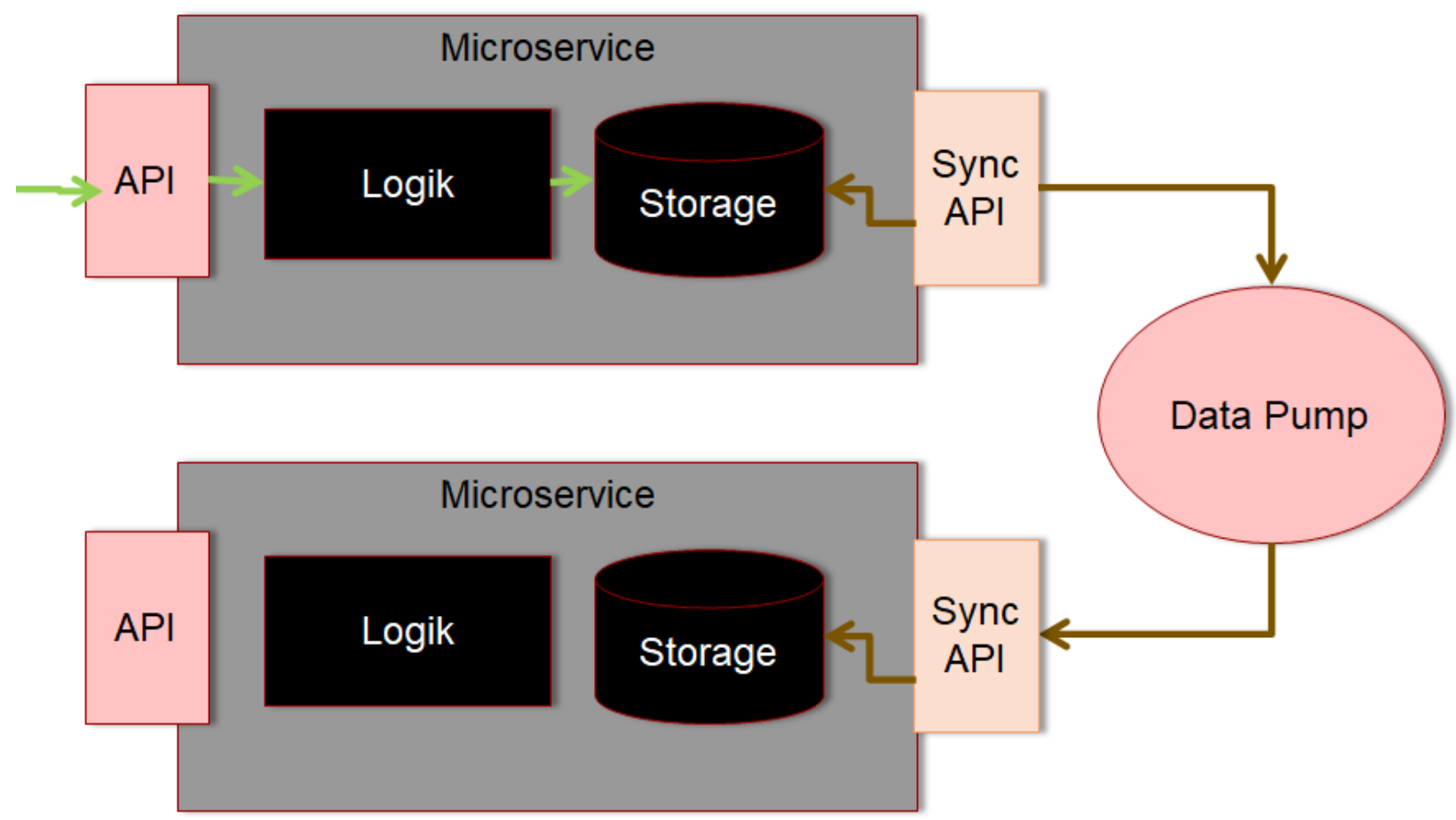
ACID und BASE

- ACID
 - Starke Konsistenz der Daten
 - Transaktionssteuerung
 - Transaction-Isolation
 - Bei Bedarf Zwei-Phasen-Commit
 - Relativ komplexe Entwicklung
- BASE
 - Schwache Konsistenz
 - Hohe Verfügbarkeit
 - Schnelligkeit
 - Bei Bedarf "Fire-and-forget"
 - Leichtere Entwicklung

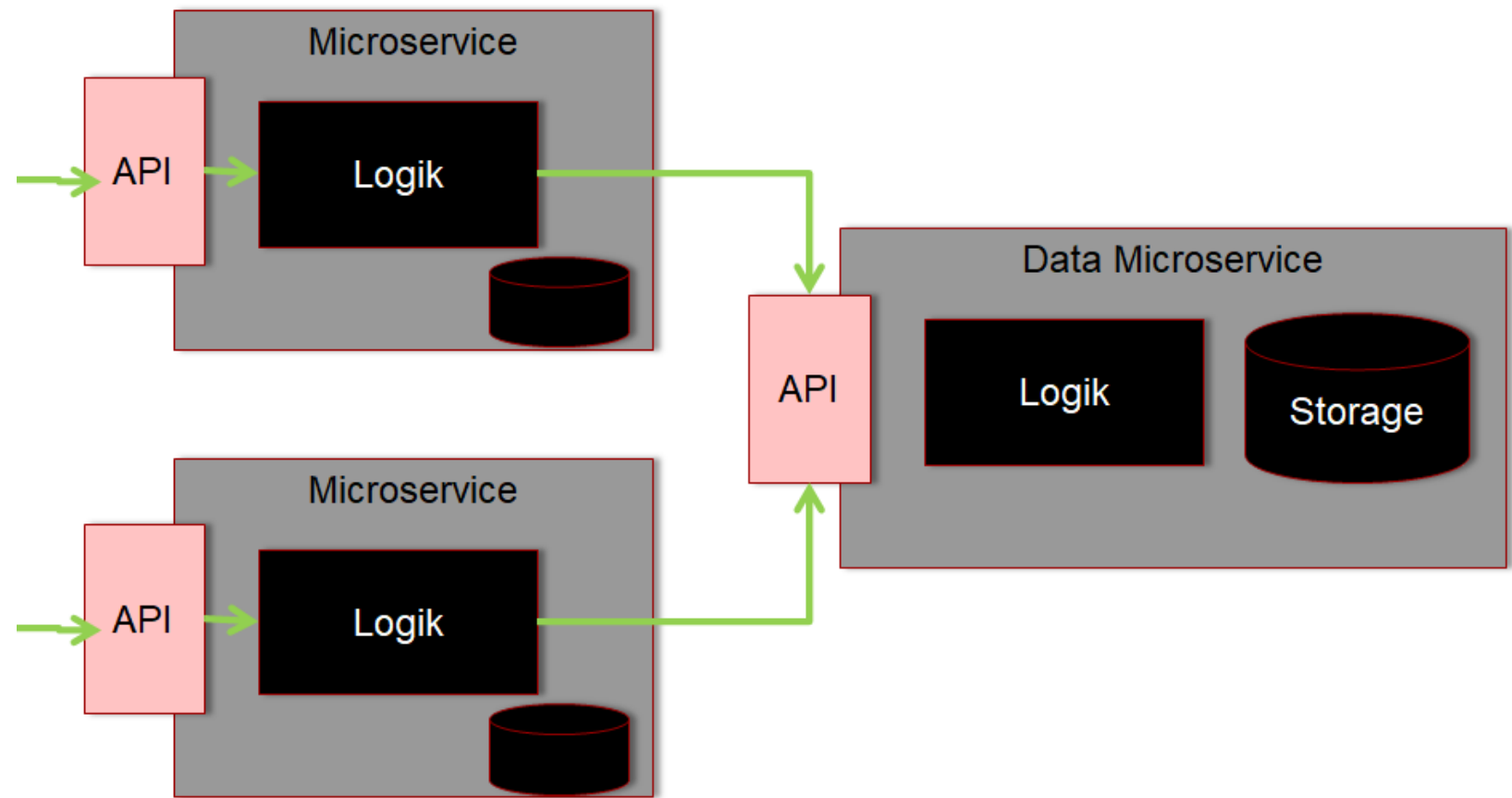
Synchronisation durch Implementierung



Synchronisation durch Data Pump



Synchronisation durch Data Service





Modellierung



Web Services mit SOAP – Eine Lösung?

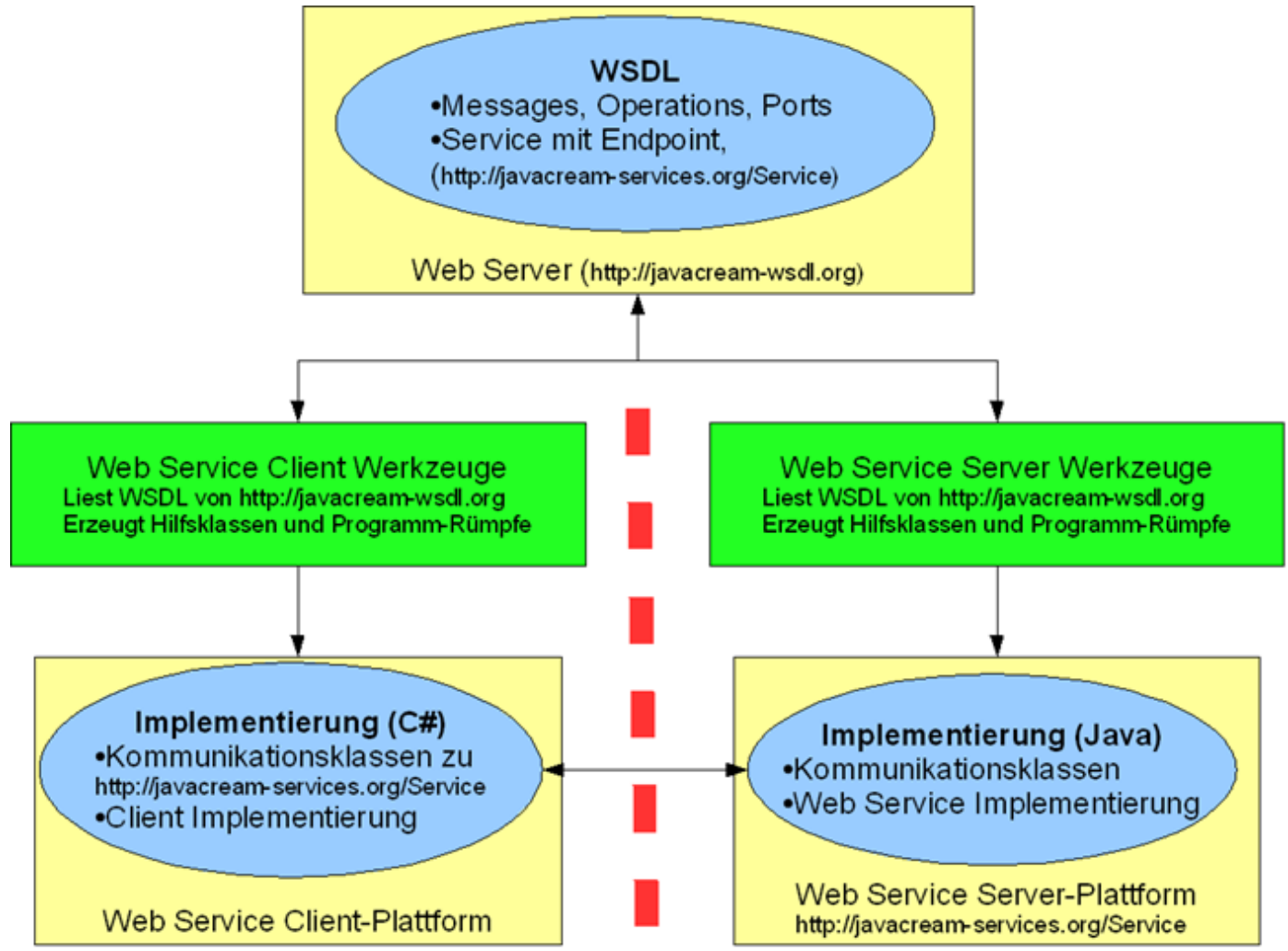
- SOAP-basierte Web Services verlangen eine Schnittstellen-Beschreibung
 - Formulierung in der Web Services Description Language, WSDL
 - WSDL ist ein XML-Schema zur Beschreibung eines Interfaces bestehend aus
 - Typen
 - Operationen
 - Ports
 - Eine WSDL ist damit ein XML-Dokument
- Ein SOAP-Envelope definiert den Aufruf
 - SOAP ist ebenfalls ein XML-Schema
 - Das SOAP-Dokument wird an die Zieladresse des Web Services, den Endpoint gesendet



Benutzung von SOAP-basierten Web Services

- Definiert wird eine Service-Beschreibung in Form einer WSDL
- Realisierung
 - Für die Server-Implementierung werden Code-Rümpfe erzeugt,
 - für den Client fertige Stub-Klassen
- Beim Aufruf werden die SOAP-Envelopes über das Netzwerk ausgetauscht

Beispiel eines SOAP-WebServices





Das Web interessiert sich nicht für SOAP!

- Keine einzige Browser-Implementierung erzeugt beispielsweise beim Formular-Submit einen SOAP-Envelope
- JavaScript könnte dies zwar prinzipiell, es gibt aber hierfür keine einzige ernst zu nehmende Utilities-Bibliothek
- Hardware und Software von Systemen sind auf http-Requests ausgerichtet
 - Firewalls
 - Load Balancer und Router
 - Cache-Lösungen



Der Representational State Transfer

- Doktorarbeit von Roy Fielding, 2000
- Siehe <http://en.wikipedia.org/wiki/REST>



Kern-Konzepte

- Die Arbeitsweise des Internet abstrahiert
 - Was sind Ressourcen?
 - Wie werden Ressourcen identifiziert?
 - Was sind Ressourcen-Operationen?
 - Wie werden Services beschrieben?
 - Was sind Stateless Operationen?
 - Voraussetzungen und Umsetzung von Caching-Mechanismen
 - Optional: Übertragung von Skript-Logik auf den Client
- Grundlegendes Konzept sind die verlinkten HyperText-Dokumente
- Nicht überraschend: „Das Internet ist ein Beispiel für die Implementierung eines REST-basierten Systems“



Das http-Protokoll

- Eine umfassende Spezifikation des w3w-Konsortiums
 - Siehe <http://en.wikipedia.org/wiki/Http>



Elemente der http-Spezifikation

- Definition von URIs
 - Pfad
 - Parameter
- http-Request und http-Response
 - Daten-Container mit Header und Body
 - Encodierung
- Umfassender Satz von Header-Properties
 - Content-Length
 - Accepts
 - Content-Type



Elemente der http-Spezifikation II

- http-Methoden
 - PUT
 - GET
 - POST
 - DELETE
 - OPTIONS
 - HEAD
- Statuscodes für Aufrufe
 - 404: „Not found“
 - 204: „Created“
 - ...



MimeTypes



REST und http

- REST hat mit http prinzipiell nichts zu tun
 - REST ist eine abstrakte Architektur
 - http ist ein konkretes Kommunikationsprotokoll
- Aber
 - http passt als Kommunikations-Protokoll der „Referenz-Implementierung“ Internet natürlich perfekt zum REST-Stil



Mapping REST - http

- http Methoden und Ressourcen-Operationen
 - PUT
 - Neu-Anlegen einer Ressource
 - Aktualisierung
 - GET
 - Lesen einer Ressource
 - POST
 - Aktualisierung
 - Neuanlage
 - DELETE
 - Löschen
- Idempotenz
 - Idempotente Operationen dürfen beliebig oft aufgerufen werden und verursachen keine Nebeneffekte
 - REST verlangt eine idempotente Implementierung für PUT und DELETE



Konzeption eines RESTful Services: Neuanlage

- Mit PUT
 - Der Client muss die Ressourcen-ID mit angeben
 - Rückgabe ist ein Statuscode „201: Created“
- Mit POST
 - Der Server entscheidet, ob er eine neue Ressource anlegen muss
 - Falls ja:
 - Statuscode „201: Created“
 - Gesetzter `Location`-Header mit URI der eben angelegten Ressource
 - Optional: Body enthält die angelegte Ressource



Konzeption eines RESTful Services: Update

- Mit PUT
 - Statuscode „200: OK“ oder „204: No content“
 - PUT ist idempotent (!)
- Mit POST
 - POST wird für nicht-idempotente Updates benutzt



Konzeption eines RESTful Services: Delete

- Mit DELETE
 - Statuscode „200: OK“ oder „204: No content
 - PUT ist idempotent (!)
- Konzeptionell muss unterschieden werden:
 - Ein „echtes“ DELETE löscht die Ressource
 - Ein fachliches Löschen (z.B. Storno) ist eigentlich ein Update der Ressource
 - Ein überladen des http-DELETE ist für diese Zwecke jedoch durchaus legitim
 - `DELETE order/ISBN42?cancel=true`



Service-Beschreibung

- Ein RESTful Web Service benötigt in erster Näherung keine gesonderte Beschreibung
 - PUT, GET, POST, DELETE decken alle relevanten Ressourcen-Operationen ab
 - Datentypen werden durch MIME-Types bestimmt
- Die einzige notwendige Information ist die Endpoint-Adresse
- In der Praxis werden jedoch in den meisten Fällen zusätzliche Informationen benötigt
 - Ein GET auf den Endpoint liefert beispielsweise eine Liste aller Ressourcen
 - Aber:
 - Liste aller Informationen oder Referenzen?
 - Bestimmung der Sortierreihenfolge?
 - ...



REST Services

- Analog zur WSDL wird eine detailliertere Service-Beschreibung geliefert
 - Web Application Description Language (WADL)
 - http://en.wikipedia.org/wiki/Web_Application_Description_Language
 - Rest Service Description Language (RSDL)
 - <http://en.wikipedia.org/wiki/RSDL>
 - Swagger
 - <http://swagger.io>



Ein komplett anderer Ansatz: HATEOAS

- Hypertext as the Engine of Application State
 - <http://en.wikipedia.org/wiki/HATEOAS>
- Grundidee: Jeder Response enthält eine Menge von Verlinkungen, die die weiteren sinnvollen Aktionen definieren
 - „Hört sich an, wie eine Web Anwendung!“
 - Aber nun konsequent mit PUT, DELETE



Wie groß ist "Micro"

- In der Literatur zu finden: "So klein wie möglich"
 - So natürlich unsinnig
 - Der kleinstmögliche Microservice
 - enthält exakt eine Operation
 - liefert ein primitives Ergebnis
- Besser: "So groß wie nötig"
 - Kriterien:
 - Aufwand der Daten-Replikation über verschiedene Microservices hinweg
 - Transaktions-Kontexte
 - Performance-Aspekte



Software-Entwicklung



Java, JEE und Spring Boot



Build-Prozess



Integration und Überwachung



Java, JEE und Spring Boot



Java

- Die Java Virtual Machine ist eine gekapselte Laufzeitumgebung für JAva-Anwendungen
- Parallele Ausführung auf einer Hardware problemlos möglich
- Anbindung an System-Überwachung ist gegeben



JEE

- Die Java Enterprise Edition (JEE) erweitert die Java Virtual Machine um weitere Infrastruktur
 - Applikationsserver
- Der Applikationsserver führt JEE-Anwendungen aus
 - Kapselung innerhalb des Applikationsservers
 - Hot Deployment
 - Inter-App-Kommunikation
 - In Memory durch Verlinkung
 - Internes Bus-Systems
 - Insbesondere Messaging
 - Unterstützung aller geläufiger Kommunikationsprotokolle



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



Build-Prozess



Kriterien für den Build-Prozess

- Reproduzierbar
- Plattform-übergreifend
 - Developer-Rechner
 - Build-Machine
- Erzeugt "Artefakte"



Bestandteile

- Source Code Management
 - Enthält die Quellcodes, Konfigurationen der Anwendung
 - die so genannten "Werke"
 - Auch die Build-Prozesse sind Werke
- Build-Maschine
 - Enthält die Build-Jobs und führt diese aus
- Artefakt-Repository
 - Die vom Build-Prozess erzeugten Ergebnisse werden Artefakte genannt
 - Diese werde in versionierter Struktur in einem Repository gehalten



Beispiel: Apache Maven

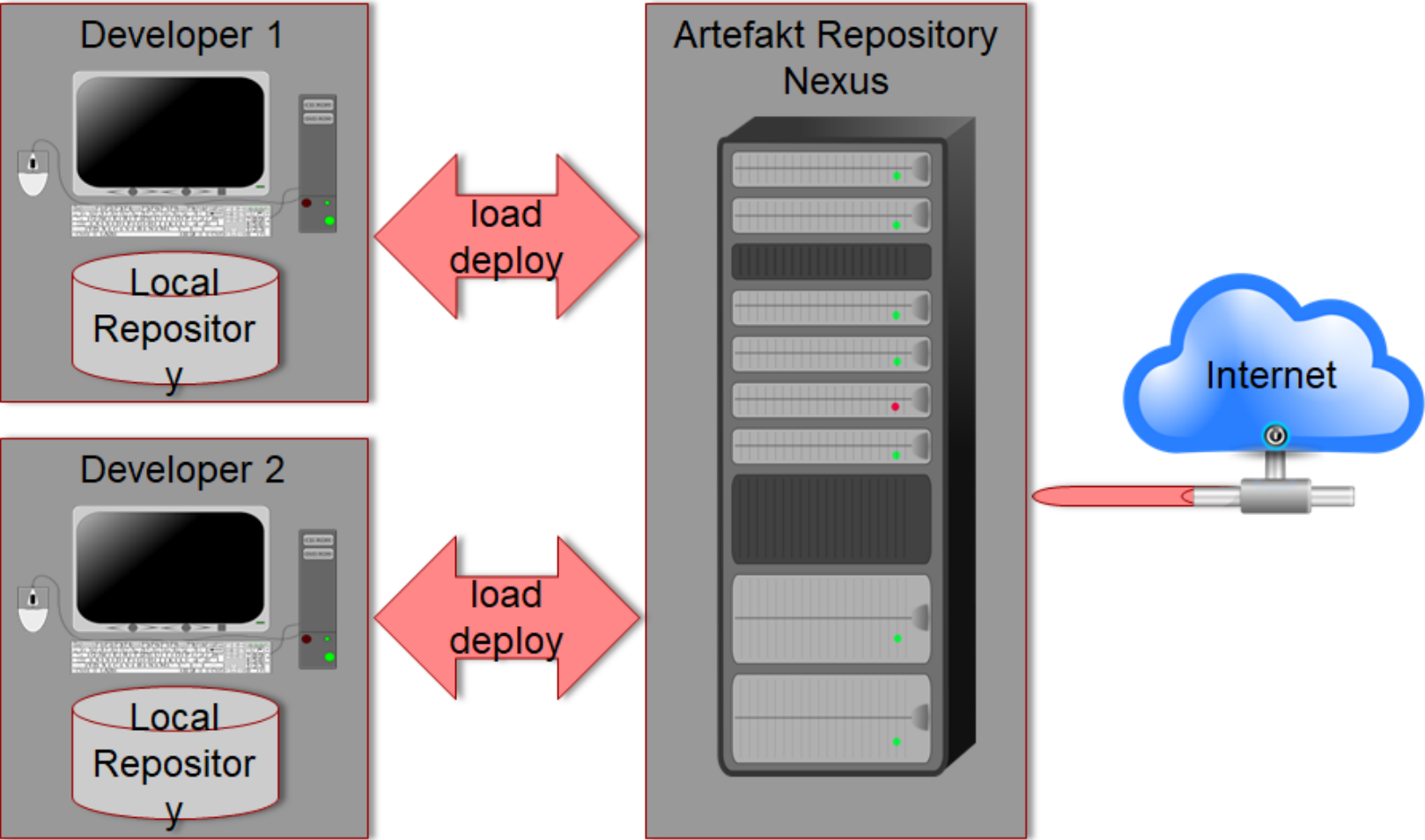
- In der Java-basierten Software-Entwicklung weit verbreitet
- Bestandteile
 - Fester Build-Prozess
 - Aufruf mit `mvn <command>`
 - `compile`
 - `package`
 - `install`
 - `deploy`
- Lokaler Cache für gebildete Artefakte
- Connectivity zu einem Remote-Repository
 - Standard: Internet-Repository `repo1.maven.org`
 - Im Unternehmen: Eigener Repository-Server
 - Nexus
 - Artefactory
 - ...



Build-Prozess für Java-Projekte

- 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

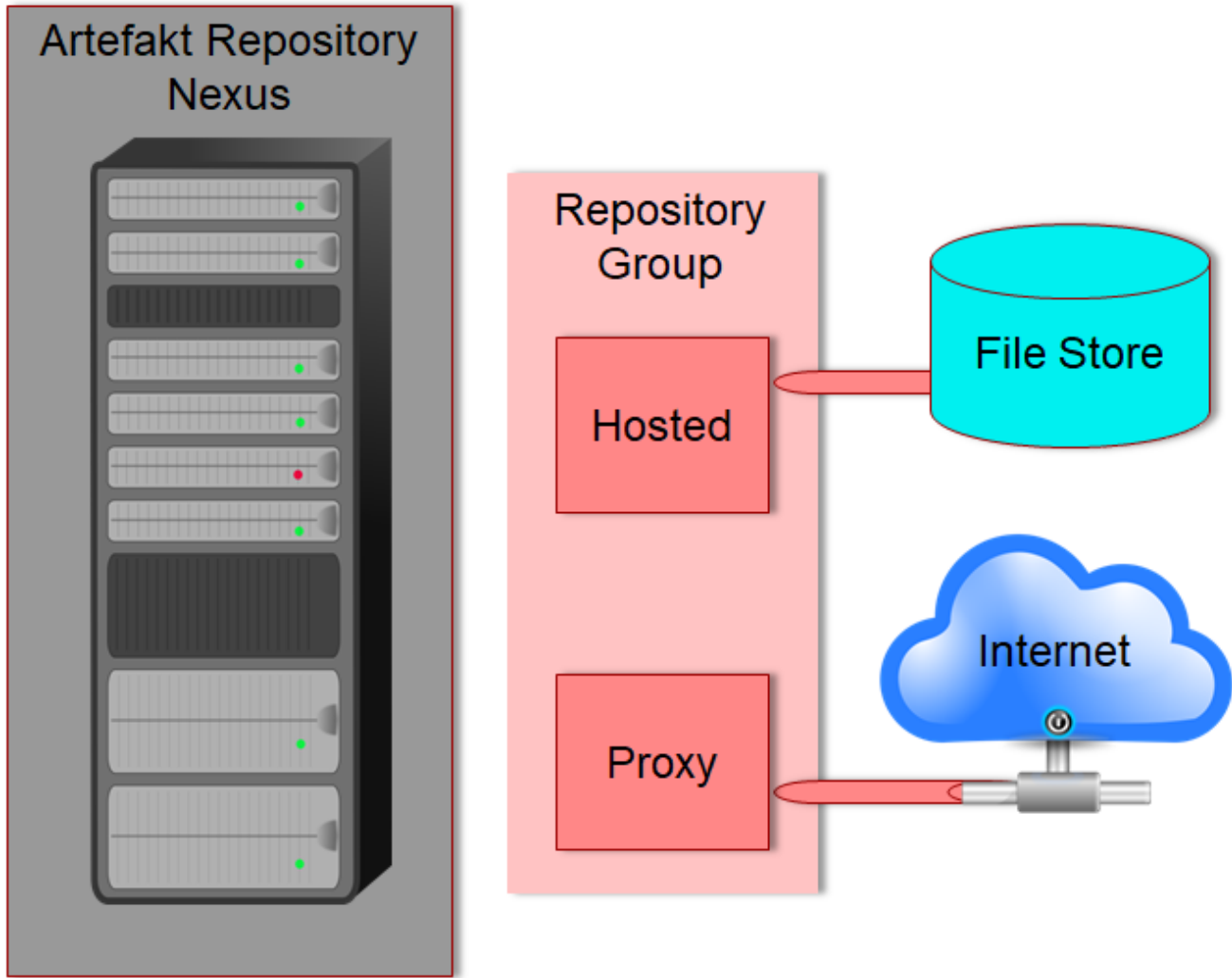
- Konfiguration des Java-Compilers
 - Java-Version
- Reporting
- Allgemein zu benutzende Dependencies zu anderen Artefakten
 - JUnit
 - Datenbank-Treiber
 - ...
- Konfiguration weiterer Plugins
 - Spezielle Reports



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

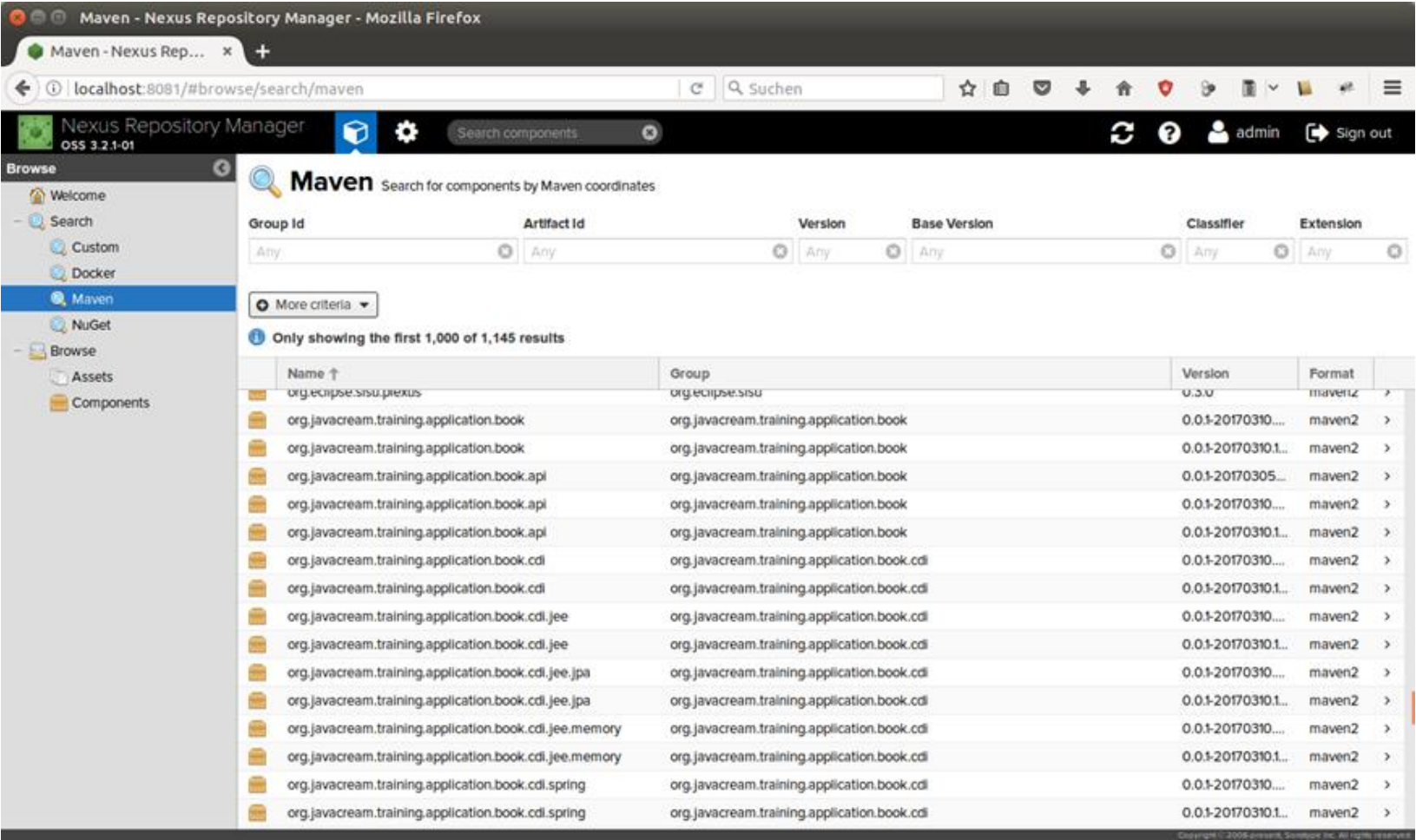




Konfiguration

- 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





Integration und Überwachung




Problemstellung

- Die Betriebssystem-nahe Überwachung ist 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



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 Screencasts](#)

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



Container für Microservices



Applikationsserver



Docker



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

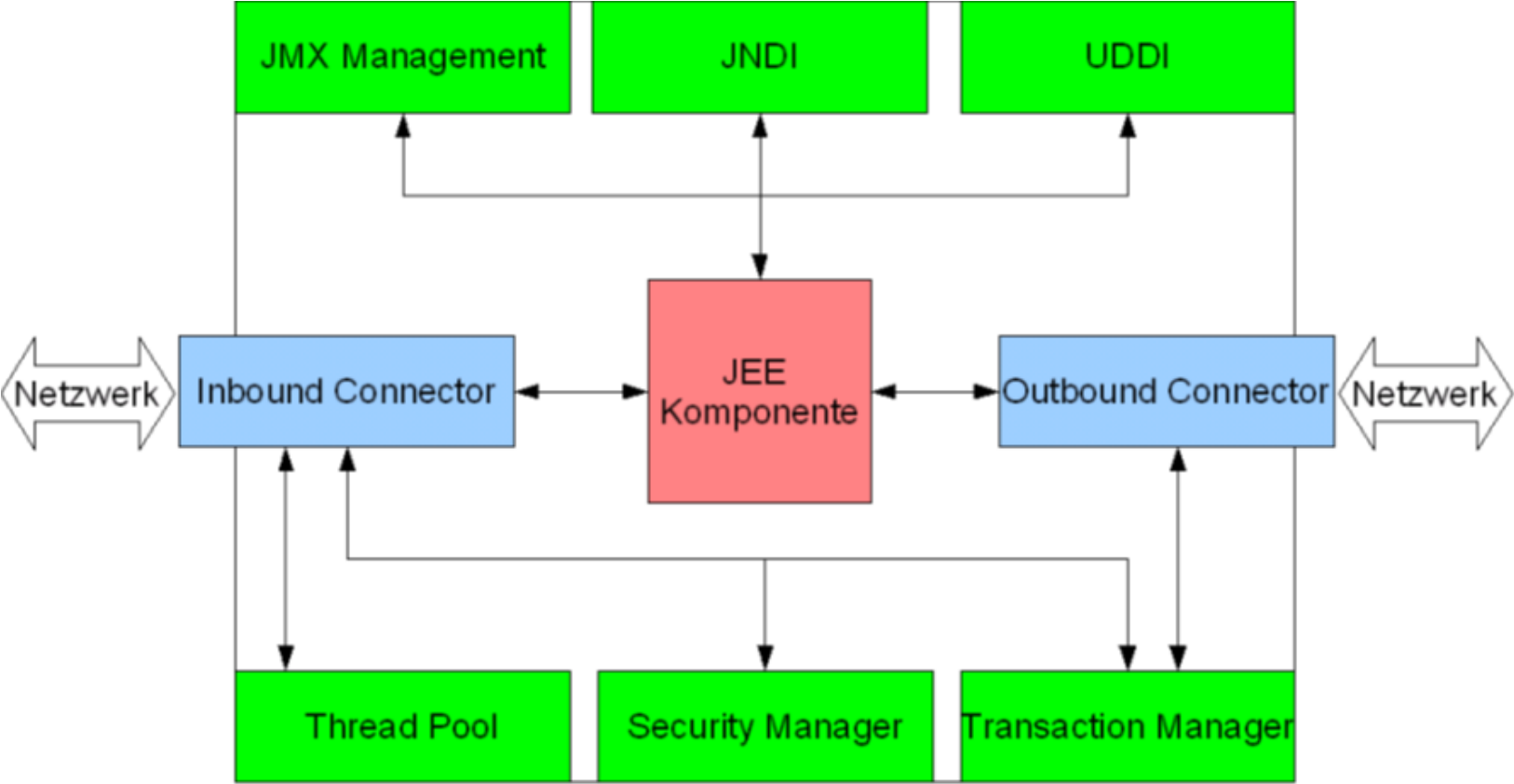


Überwachung

- 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!



Der Applikationsserver





Inbound Connectors

- Inbound Connectors ermöglichen die Kommunikation mit dem Applikationsserver
 - http/https
 - Java Remote Method Invocation (RMI) bzw. die interoperable Variante RMI/IIOP
 - Empfangen von Messages mit dem Java Messaging Service (JMS)
 - SOAP
- Für Protokolle, die nicht im Standard enthalten sind, können spezielle Implementierungen für Inbound Connectors als Plug-In in den Applikationsserver installiert werden
 - Dafür existiert auch ein breiter Markt von Herstellern und Produkten



Aufgaben des Inbound Connectors

- Die Inbound Connectors enthalten selber keine Geschäftsprozesse
- Laut Konzeption wandeln sie eine einkommende Netzwerkverbindung um in einen Request der, wie oben bereits angesprochen, von einem Thread aus dem Thread Pool ausgeführt wird
- Geschäftslogik wird in einer JEE-Komponente definiert
- Ein Inbound Connector kann im Rahmen einer Verteilten Transaktion aufgerufen werden



JEE Komponenten

- JEE Komponenten bestehen aus zwei Komponenten:
 - Einem Container, der vom Applikationsserver zur Verfügung gestellt wird
 - und der eigentlichen Komponente, die als Bestandteil eines Anwendungsprogramms aufzufassen ist



JEE Komponenten

- Servlets werden vom http Inbound Connector aufgerufen
 - Die Entscheidung, welches Servlet (und damit welche Anwendungslogik) aufgerufen werden soll erfolgt über die URL des http-Aufrufs
 - Ein spezieller Typ von Servlet kann auch über SOAP aufgerufen werden
 - In modernen Anwendungen existiert wahrscheinlich nur ein einziges zentrales Servlet
- Enterprise JavaBeans dienen primär zur Transaktionssteuerung
 - Dem Anwendungsprogrammierer werden verschiedene Sub-Typen angeboten, die sich bezüglich Lebenszyklus und Aufruf-Protokoll unterscheiden
 - Es gibt „Stateless“ und „Stateful SessionBeans“, „Message Driven Beans“ sowie „Singletons“
 - Enterprise JavaBeans sind in modernen Anwendungen häufig nicht mehr anzutreffen



Outbound Connectors

- Outbound Connectors werden von JEE-Komponenten lokal benutzt
- Ihre Aufgabe ist es, einen Geschäftsprozess innerhalb eines Backend-Systems aufzurufen
 - Dazu hält der Outbound Connector einen Connection Pool
- Die Outbound Connectors werden auch vom Transaktionsmanager (für die Koordinierung der Verteilten Transaktionen) und vom Security Manager (Propagierung des angemeldeten Benutzers) benutzt



Beispiele für Outbound Connectors

- Die JEE-Spezifikation verlangt die folgenden Outbound Connectors:
- Die Data Sources verbinden zu SQL-Datenbanksystemen
- Versenden von JMS Messages
- Anbinden an ein Email-System
- Alle weiteren Systeme sind wiederum durch Plug-Ins installierbar, für die ein breiter Markt existiert



Docker

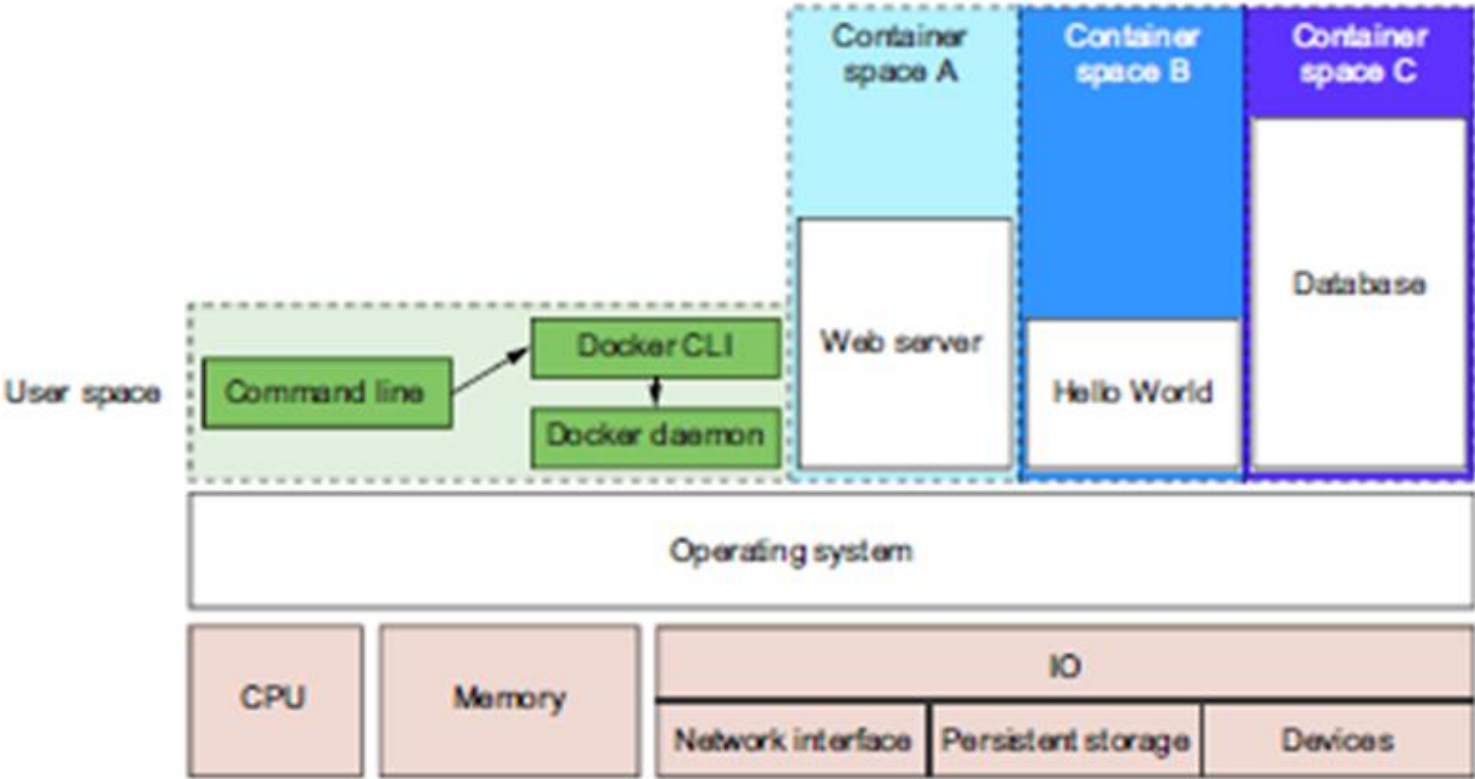


Was ist ein Container?

- "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

- 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

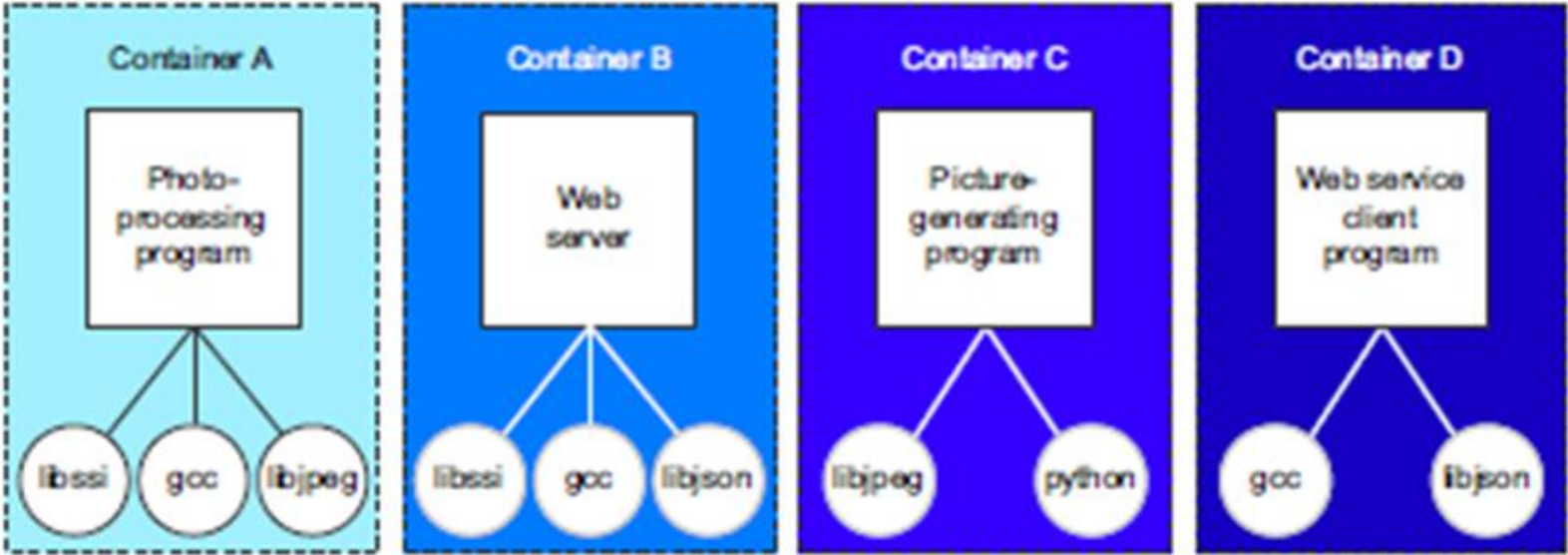


Docker und (Micro)Services

- 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





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:a68868bfe696c00866942e8f5ca39e3e31b79c1e50feaaa4ce5e28df2f051d5c
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.

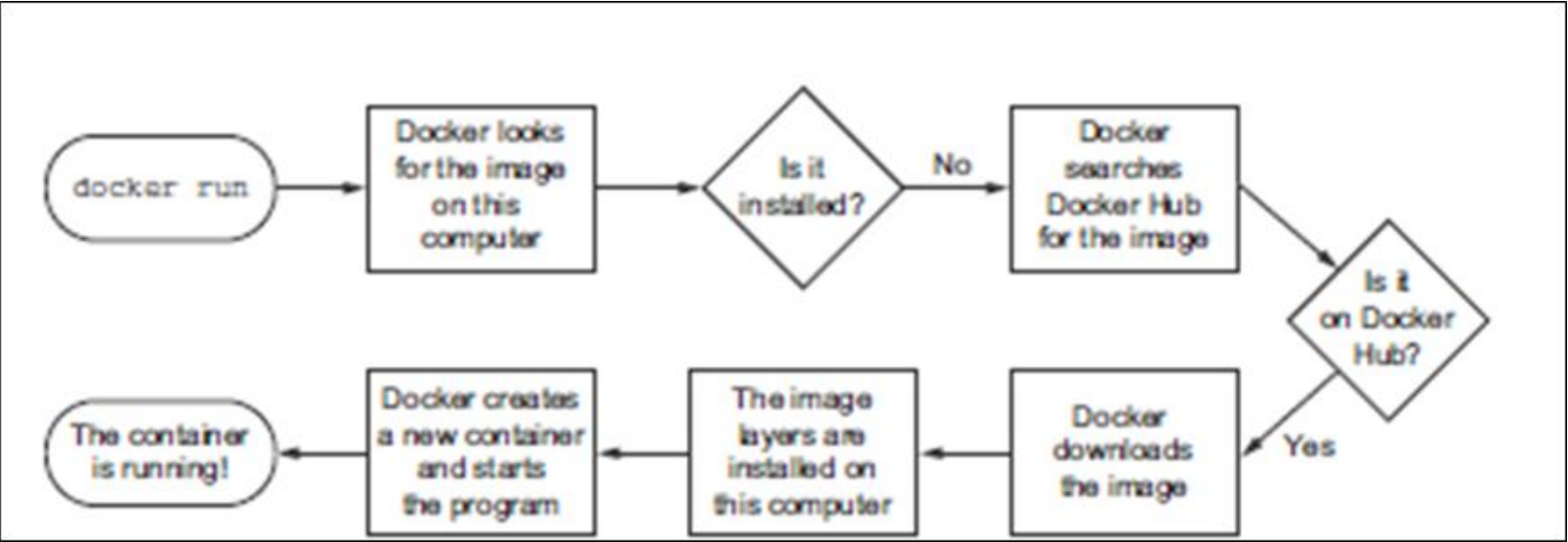
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

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/
```



Docker-Workflow: 1. und 2. Lauf



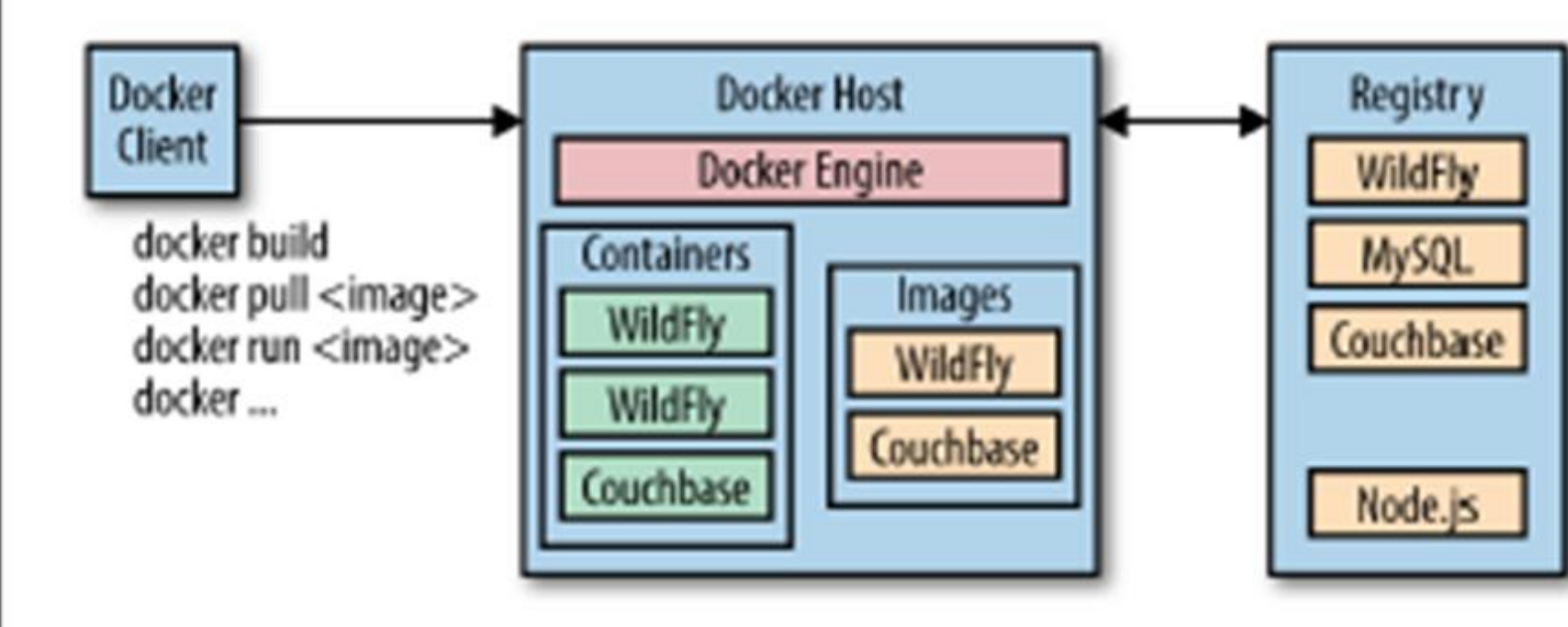


Lebenszyklus des Containers

- 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



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-8u111 openjdk-8u111-jdk openjdk-8 openjdk-8-jdk latest jdk 8u111 8u111-jdk 8 8-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



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`



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>`



Dockerfile Referenz

docker docs

Search the docs

Guides

Product Manuals

Glossary

Reference

Samples

Dockerfile reference

Cloud API

Cloud stack file reference

Compose file reference

Compose (docker-compose) CLI

Daemon CLI reference (dockerd)

Docker ID Accounts API

Engine API

Engine (docker) CLI

Machine (docker-machine) CLI

Registry API

Trusted Registry API

Trusted Registry CLI

← → ↺

Sicher | https://docs.docker.com/engine/reference/builder/

Dockerfile reference

Estimated reading time: 60 minutes

Dockerfile reference

Docker can build images automatically by reading the instructions from a `Dockerfile`. A `Dockerfile` is a text document that contains all the commands a user could call on the command line to assemble an image. Using `docker build` users can create an automated build that executes several command-line instructions in succession.

This page describes the commands you can use in a `Dockerfile`. When you are done reading this page, refer to the [Dockerfile Best Practices](#) for a tip-oriented guide.

Usage

The `docker build` command builds an image from a `Dockerfile` and a context. The build's context is the files at a specified location `PATH` or `URL`. The `PATH` is a directory on your local filesystem. The `URL` is a Git repository location.

A context is processed recursively. So, a `PATH` includes any subdirectories and the `URL` includes the repository and its submodules. A simple build command that uses the current directory as context:

```
$ docker build .  
Sending build context to Docker daemon 6.51 MB  
...
```

The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon. In most cases, it's best to start with an empty directory as context and keep your Dockerfile in that directory. Add only the

2.0.0820 © Javacream

Microservices

101



Gekapselte Services mit Docker

