

Ziel

- Wir lernen die Programmierung von Microservices
 - Programmierung
 - Java
 - viele Bibliotheken: Spring-Boot, Lombok, Jackson, ...
 - Test
 - Unit-Tests (JUnit)
 - Integrationstests (Cucumber)
 - Performancetests (Gatling)
 - Build
 - Maven
 - Betrieb
 - Docker

Warum Microservices?

- Welt der Unternehmensanwendungen (enterprise applications)
 - Software, die Unternehmen für ihren Betrieb brauchen
 - Banken: Verwaltung der Konten, Buchungen, Wertpapier-Handel, ...
 - Versicherungen: Angebote, Verträge, Schaden/Leistung, ...
 - Handel: Lagerverwaltung, Logistik, Beschaffung, ...
 - Produzierendes Gewerbe: Aufträge, Planung, Steuerung, Logistik, ...
 - Behörden
 - ...
 - **Alle** Unternehmen: Buchhaltung, interne Prozesse (z. B. HR), ...

Welt der Unternehmensanwendungen

- Typische Eigenschaften (1)
 - Viele Nutzer, die gleichzeitig arbeiten wollen
 - Interne Mitarbeiter
 - Mitarbeiter anderer Firmen
 - Kunden
 - Daten
 - Zentrale Sicht erwünscht
 - Große Mengen
 - Viele Datenbewegungen

=> hohe Last!

Welt der Unternehmensanwendungen

- Typische Eigenschaften (2)
 - Transaktionale Sicherheit
 - Geschäftsvorfall: ganz oder gar nicht
 - Hohe Sicherheitsanforderungen
 - Schutz vor fremdem Zugriff (lesend/schreibend)
 - Schutz vor Datenverlust
 - Lange Lebensdauer
 - Erfolgreiche Systeme leben Jahrzehnte!
 - Anpassbarkeit
 - Neue Produkte / Prozesse
 - Gesetzesänderungen

Welt der Unternehmensanwendungen

- Wie setzt man solche Systeme um?
 - Alt: Mach es groß!
 - Große Programme (Monolithen)
 - Große Rechner (Mainframe)
 - Neu: Teile und herrsche!
 - Z. B. Microservices

Microservices

- Prinzipien
 - Vollständige Unabhängigkeit
 - Kein shared code
 - Keine gemeinsam genutzte Datenbank
 - Unabhängiges Build/Deployment
 - Technik
 - Jeder Microservice: eigenständige Anwendung
 - Schnittstellen
 - REST/JSON
 - Messages

Microservices

- Prinzipien
 - Jeder Microservice: eigenständig laufende Anwendung
 - Vollständige Unabhängigkeit
 - Kein shared code
 - Keine gemeinsam genutzte Datenbank
 - Unabhängiges Build/Deployment

Microservices

- Technik
 - Schnittstellen
 - REST/JSON
 - Messages
 - Frameworks
 - Java: Spring Boot, Quarkus, ...
- Kann auch in beliebigen anderen Programmiersprachen realisiert werden
 - C#, JavaScript, Python, ...

Microservices

- Praxis-Beispiel
 - Neue Vertriebsplattform der Deutschen Bahn
 - Ca. 300 Mitarbeiter
 - >100 Microservices

Step 10

- Thema: Java
 - Objektorientierte Programmiersprache
 - Erscheinungsjahr 1996
 - Angelehnt an C++
- Inzwischen: eine der am weitesten verbreiteten Programmiersprachen (zumindest für Unternehmensanwendungen)
 - Plattformunabhängigkeit
 - Große Zahl an Bibliotheken

Step 10

- Thema: Java

- Compiler: `javac HelloWorld.java`
 - Übersetzt Quelltext (*.java) in Java-Bytecode (*.class)
- Ausführung: `java HelloWorld`
 - Führt Java-Bytecode aus
 - Laufzeitumgebung:
 - Interpreter
 - JIT-Compiler
 - Standard-Bibliotheken

Step 15

- Thema: Build-Management mit Maven
 - „Maven ist ein auf Java basierendes Build-Management-Tool der Apache Software Foundation“ (Wikipedia)
 - Gibt Standard-Verzeichnisstruktur und Lebenszyklen (Standard-Ablauf) vor
 - Verwaltet Abhängigkeiten (innerhalb des Projekts und zu benutzten Bibliotheken)
 - Ist erweiterbar durch Plugins
 - Deklarative Definition des Projekts in `pom.xml` (Project Object Model)

Step 15

- Thema: Build-Management mit Maven
 - Identifikation eines Build-Ergebnis
(<https://maven.apache.org/guides/mini/guide-naming-conventions.html>)
 - groupId
 - Identifiziert das Projekt
 - Sollte Java package name rules folgen, z. B. de.conciso.starter
 - artifactid
 - Name des jar files (Build-Ergebnis)
 - Lowercase letters, no strange symbols
 - version
 - Typisch: Zahlen und Punkte, z. B. 1.0, 1.1, 1.0.1, ...
 - Beispiel: Semantic Versioning (<https://semver.org/>)

Step 15

- Thema: Build-Management mit Maven
 - Identifikation eines Build-Ergebnis
 - Identifikation unseres eigenen Produkts
 - Identifikation fremder Build-Ergebnisse, die wir nutzen wollen!
 - Woher kommen die?
 - <https://mvnrepository.com/>
 - Beispiel: <https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core>

Step 15

- Thema: Build-Management mit Maven
 - Kommando: `mvn clean package`
 - `clean` und `package` sind Lebenszyklen

Step 20

- Thema: Nutzung eines Maven-Plugins
 - Das `exec`-Plugin ermöglicht es, Java-Programme aus Maven heraus auszuführen
 - <http://www.mojohaus.org/exec-maven-plugin/>
 - <https://mvnrepository.com/artifact/org.codehaus.mojo/exec-maven-plugin>
 - Kommando: `mvn exec:java`
 - `exec:java` ist ein Goal

Step 30

- Thema: Logging mit `log4j`
 - Weit verbreitete Bibliothek für Logging
 - <https://logging.apache.org/log4j/2.x/>
 - <https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core>

Step 30

- Thema: Logging mit `log4j`
 - Ausgaben sollten in produktivem Code nie mit `System.out.println` erfolgen!
 - Konzepte von `log4j`:
 - Logger
 - Appender
 - Loglevel