andrena OBJECTS

Experts in agile software engineering

Tools in der Testpyramide

Wenn eine Größe nicht allen passt…

ObjektForum München

21. Oktober 2015

Daniel Knapp
Mustafa Yilmaz

Motivation

- Wie kann man beim Testen vorgehen?
- Welche Arten von Tests sind zu berücksichtigen?
- Welche Bestandteile enthält ein Test?
- Wie kann man Tests leichtgewichtig implementieren?
- Welche Werkzeuge eignen sich für die unterschiedlichen Testtypen?

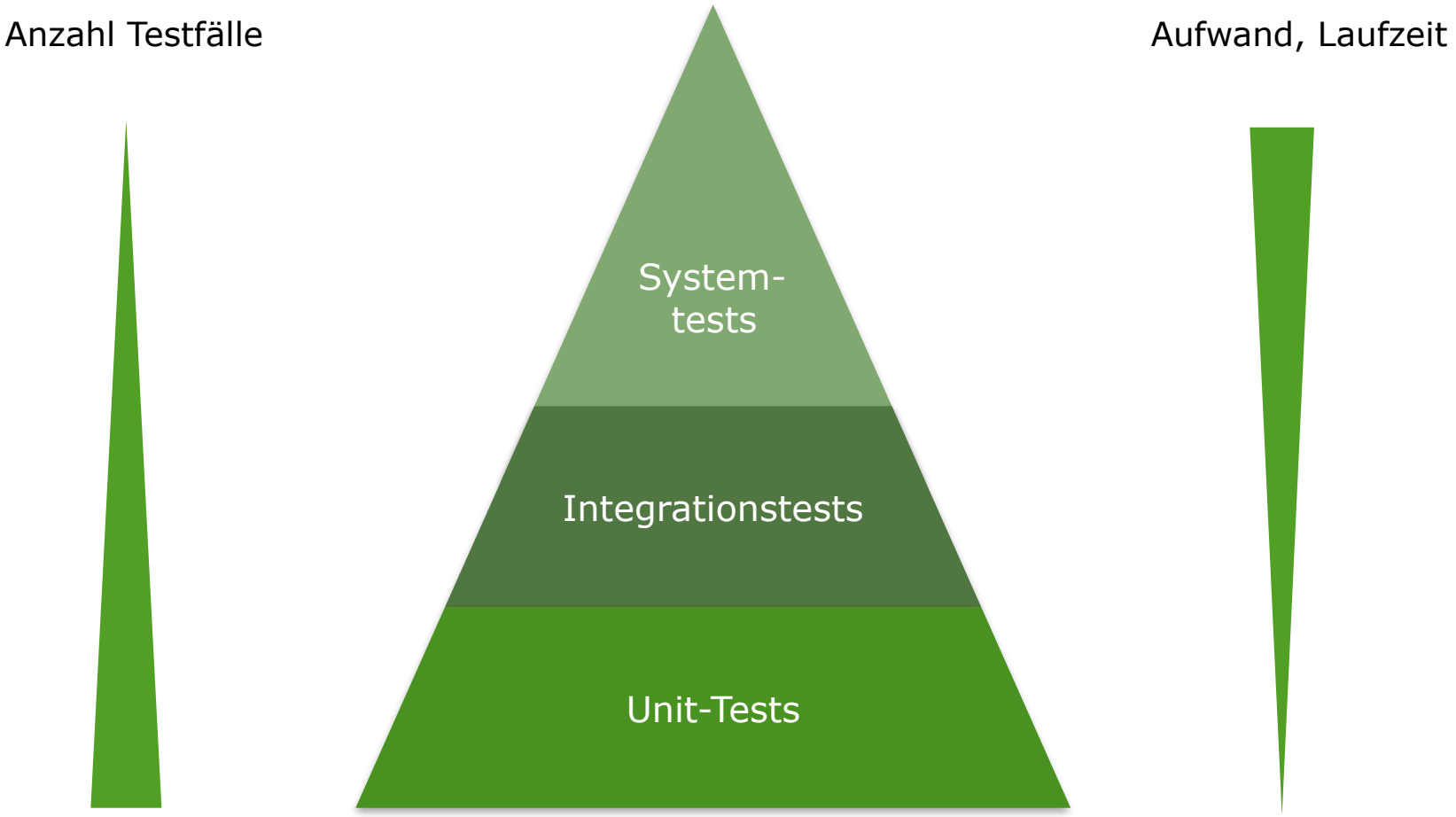
Agenda

- Effiziente Verteilung der Tests: die Testpyramide
- Vorstellung der Beispielapplikation
- Vier Teilschritte eines Integrationstests
- 1. Initialisierung von Testsystemen
- 2. Testdatenbereitstellung
- 3. Abhängigkeiten und Kommunikationsschnittstellen
- 4. Testlauf und Validieren der Ergebnisse

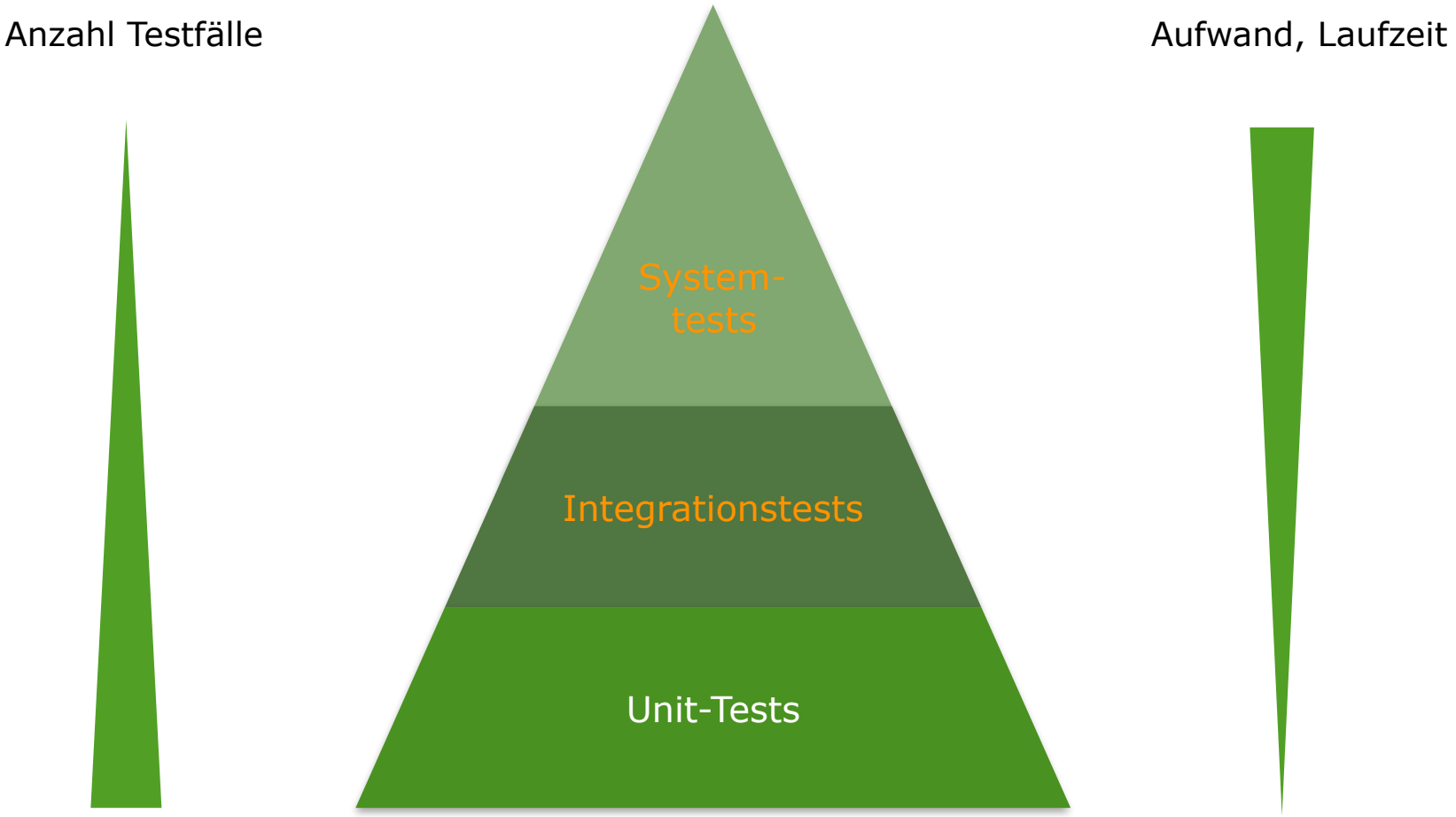
Agenda

- **Effiziente Verteilung der Tests: die Testpyramide**
- Vorstellung der Beispielapplikation
- Vier Teilschritte eines Integrationstests
- 1. Initialisierung von Testsystemen
- 2. Testdatenbereitstellung
- 3. Abhängigkeiten und Kommunikationsschnittstellen
- 4. Testlauf und Validieren der Ergebnisse

Effiziente Verteilung der Tests: Die Testpyramide



Effiziente Verteilung der Tests: Die Testpyramide



Agenda

- Effiziente Verteilung der Tests: die Testpyramide
- **Vorstellung der Beispielapplikation**
- Vier Teilschritte eines Integrationstests
- 1. Initialisierung von Testsystemen
- 2. Testdatenbereitstellung
- 3. Abhängigkeiten und Kommunikationsschnittstellen
- 4. Testlauf und Validieren der Ergebnisse

Beispielapplikation

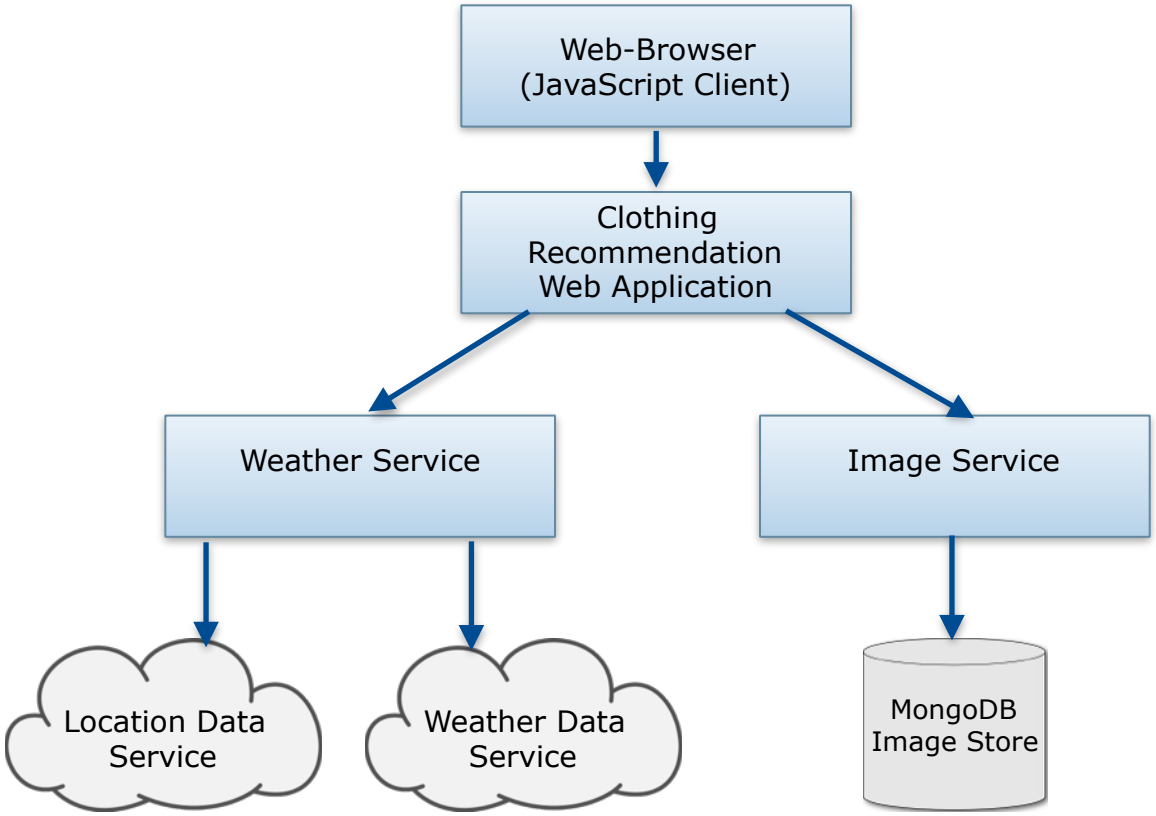
Java/Javascript-Webapplikation zur Ermittlung eines Kleidungsvorschlags in Form eines Bildes anhand der aktuellen Koordinaten und des aktuellen Wetters.

Eingesetzte Technologien:

- AngularJS
- RESTful Web Services
- Spring Boot
- Mongo DB
- Maven

Beispielapplikation

Architektur

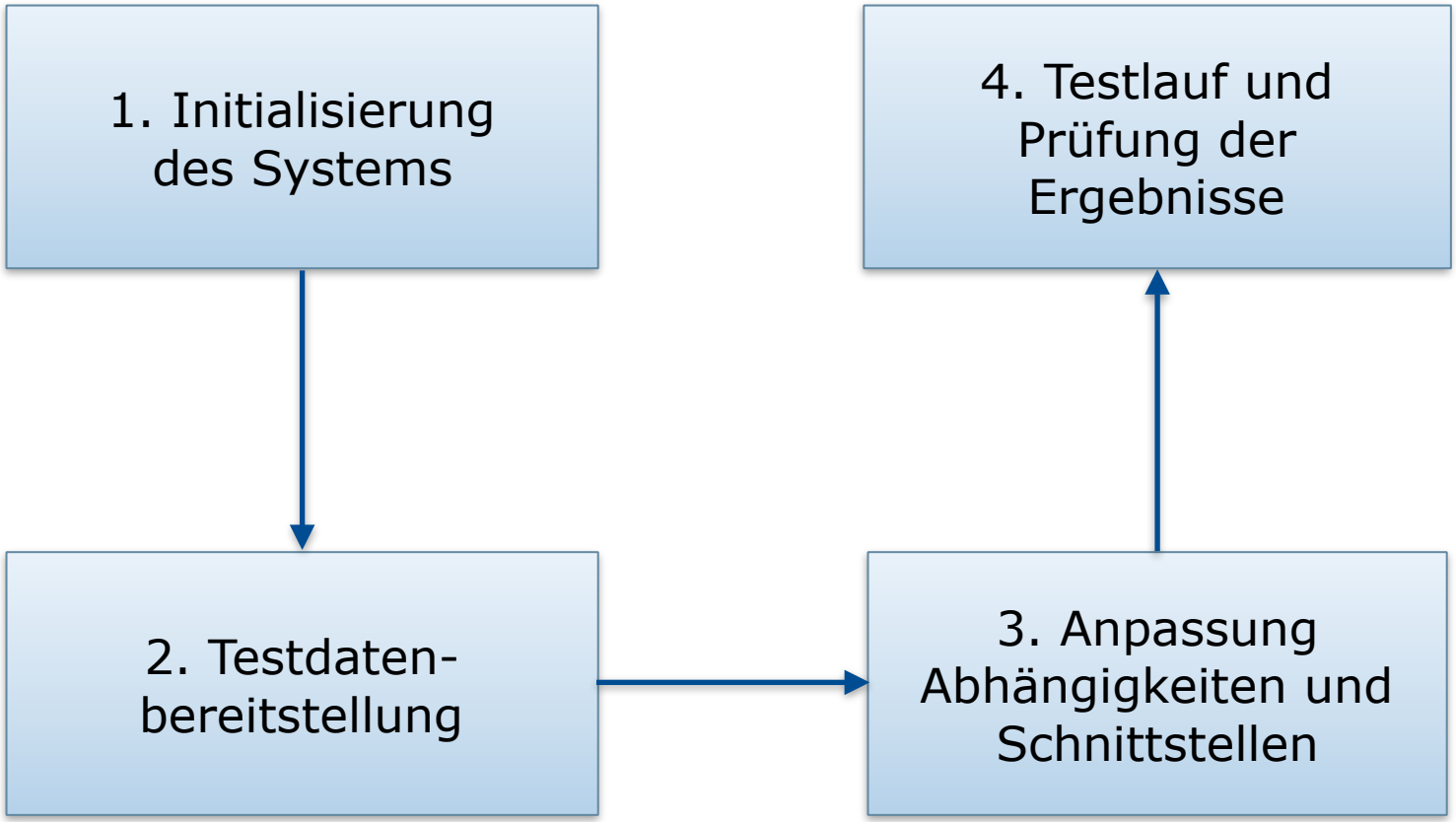


Demo

Agenda

- Effiziente Verteilung der Tests: die Testpyramide
- Vorstellung der Beispielapplikation
- **Vier Teilschritte eines Integrationstests**
- 1. Initialisierung von Testsystemen
- 2. Testdatenbereitstellung
- 3. Abhängigkeiten und Kommunikationsschnittstellen
- 4. Testlauf und Validieren der Ergebnisse

Teilschritte eines Integrationstests



Agenda

- Effiziente Verteilung der Tests: die Testpyramide
- Vorstellung der Beispielapplikation
- Vier Teilschritte eines Integrationstests
- **1. Initialisierung von Testsystemen**
- 2. Testdatenbereitstellung
- 3. Abhängigkeiten und Kommunikationsschnittstellen
- 4. Testlauf und Validieren der Ergebnisse

1. Teilschritt: Initialisierung

Ziel: Das „System under Test“ in einen definierten Zustand bringen, um Tests ausführen zu können

- Einfach zu realisieren auf Unit-Test-Ebene

- Bei Integrationstests werden **zentrale** Test-Systeme genutzt

  - Testläufe und manuelle Benutzung beeinflussen sich gegenseitig
  - Parallelisierung daher oft nicht möglich
  - Nutzung des Systems während der Testläufe eingeschränkt
  - Nach Testläufen ist ggf. eine Bereinigung erforderlich

Komponententests mit Spring / CDI

- Spring beinhaltet Unterstützung für Integrationstests
- Im CDI-Umfeld: Test-Frameworks wie CDI-Unit oder Arquillian

Vorteil:

- Geeignet für das Testen einzelner Komponenten

Nachteile:

- Begrenzte Möglichkeit für Integrationstests
  (Tests der Persistenzschicht typischerweise gegen In-Memory-Datenbanken)

- Weniger geeignet für End-To-End-Tests

1. Teilschritt: Initialisierung

End-To-End Tests

- Als Grundlage wird die Anwendung in ein Artefakt verpackt und in einer Testumgebung installiert
- Im Nachgang werden die Tests gegen diese Umgebung ausgeführt
- Herausforderung: Realisierung einer unabhängigen, zuverlässigen und reproduzierbaren Umgebung
- Unterscheidung in drei Kategorien möglich:
  - „Klassische" Server-Systeme
  - Frameworks wie Spring Boot oder Play
  - Containerized Applikationen

End-To-End: „Klassische“ Serversysteme

- Maven-Cargo-Plugin oder Arquillian
- Application-Server werden aus dem Testlauf heraus angesteuert
- Build-Prozess kann automatisiert Server heranziehen und starten
  - Vorkonfigurierte Server können in einem Artefakt-Repository liegen
  - Maven-Dependency-Plugin kann verwendet werden
- Build-Prozess kann lokal (auf Entwicklungsrechnern) und im Continuous-Integration-System (z.B. Jenkins) verwendet werden
- Kein Einfluss auf Abhängigkeiten und Schnittstellen

1. Teilschritt: Initialisierung

End-To-End:

- Anwendungen laufen nicht in Application Server, sondern enthalten einen eingebetteten Server
- Einfaches Aufsetzen und Starten einer Testumgebung
- Anwendung kann in jeder Umgebung wie eine beliebige Java-Anwendung gestartet werden
  - `java –jar …`
- Auch hier: Kein Einfluss auf die externe Umgebung (abhängige Systeme) möglich

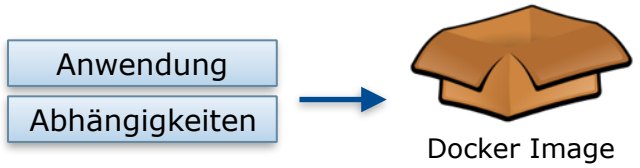
1. Teilschritt: Initialisierung

End-To-End: „Containerized"-Applikationen

- Applikationen oder auch Applikationsserver werden in sogenannte Images verpackt
- Start und Testen unabhängig vom eingesetzten Betriebssystem unter gleichen Bedingungen möglich
- Auch bei Legacy-Systemen anwendbar
- Die Testumgebung kann im Build-Prozesses mittels Docker-Maven-Plugin gestartet werden
- Es können auch gesamte „Systemlandschaften" bestehend aus mehreren Containern gestartet werden

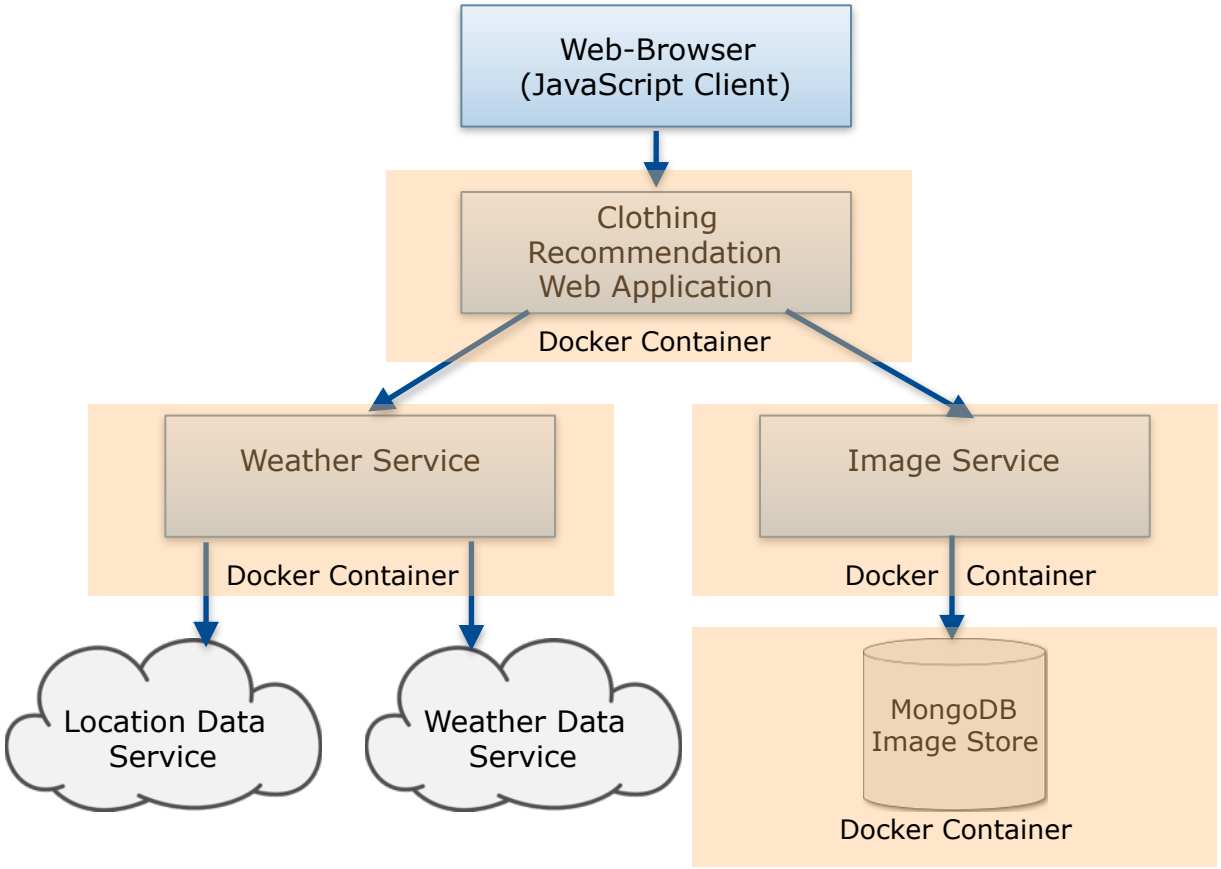
Docker

- Build, Ship, Run:
  - Verschiedenste Programmiersprachen
  - Anwendung und Abhängigkeiten in einem Container
  - Überall lauffähig (Portabilität!)
- Zahlreiche fertige Images können als Grundlage verwendet werden (Docker Hub Repository)
- Docker vs. virtuelle Maschinen: Docker-Container enthalten nicht das gesamte Betriebssystem!



Beispielapplikation

Architektur



Demo

Agenda

- Effiziente Verteilung der Tests: die Testpyramide
- Vorstellung der Beispielapplikation
- Vier Teilschritte eines Integrationstests
- 1. Initialisierung von Testsystemen
- **2. Testdatenbereitstellung**
- 3. Abhängigkeiten und Kommunikationsschnittstellen
- 4. Testlauf und Validieren der Ergebnisse

2. Teilschritt: Testdatenbereitstellung

Testdatenbereitstellung

Ziel:

- **Testdaten** und zugehörige **Datenstrukturen** in das System einspeisen
- Reproduzierbare Ausgangssituation für Tests erzeugen
- Automatisierte Verifikation der Testergebnisse basierend auf explizit definierten Annahmen

2. Teilschritt: Testdatenbereitstellung

Testdatenbereitstellung

Werkzeuge:

Relationale Datenbanken: **Struktur und Daten**

- Maven-SQL-Plug-In, Flyway, Liquibase

Ähnliche Werkzeuge existieren auch bei nicht-relationalen Datenbanken (bspw. NoSql): **Daten**

- mongeez, mongobee

2. Teilschritt: Testdatenbereitstellung

Wahl der Datenspeicherinstanz

Ziel: Seiteneffekte durch andere Tests/Klienten vermeiden

Unterscheidung **dreier Ansätze**:

- Produktionsnahe (zentrale) Datenbanken
- Leichtgewichtige In-Memory-Datenbanken
- Containerized Datenbanken

2. Teilschritt: Testdatenbereitstellung

Produktionsnahe (zentrale) Datenbanken

Vorteil:

- Produktionsnahes Verhalten
- ggf. geringer Setup-Aufwand
- ggf. bereits mit sinnvollen Daten bestückt

Nachteile:

- Müssen zur Laufzeit gestartet werden bzw. verfügbar sein
- Entwickler können nicht unabhängig voneinander testen
- Daten werden durch Tests verändert
- Testsetup nicht zwingend stabil

2. Teilschritt: Testdatenbereitstellung

In-Memory-Datenbanken

Vorteil:

- Keine Installation erforderlich
- Schnelle und einfache Initialisierung von In-Memory-Datenbanken
- Einfache Testdaten-Initialisierung, keine Bereinigung erforderlich
- Wiederholbares Testsetup
- Parallele Testläufe

Nachteile:

- In-Memory-Datenbanken verhalten sich ggf. anders als produktiv eingesetzte Datenbanken

2. Teilschritt: Testdatenbereitstellung

Containerized-Datenbanken

Was tun, falls keine In-Memory-Variante verfügbar?

- Container als Alternative
- Vorteile der beiden erstgenannten Ansätze können vereint werden indem Datenbanken ebenfalls in Container verpackt werden

Demo

Agenda

- Effiziente Verteilung der Tests: die Testpyramide
- Vorstellung der Beispielapplikation
- Vier Teilschritte eines Integrationstests
- 1. Initialisierung von Testsystemen
- 2. Testdatenbereitstellung
- **3. Abhängigkeiten und Kommunikationsschnittstellen**
- 4. Testlauf und Validieren der Ergebnisse

3. Teilschritt
Abhängigkeiten und Schnittstellen

Kommuniziert das zu testende System mit externen Systemen oder Anwendungen, so ist es nicht immer möglich das Gesamtsystem in einen testbaren Zustand zu bringen.

Einzelne Subsysteme oder Kollaborateure müssen daher durch geeignete Attrappen ersetzt werden:

- Stabiles Testsetup
- Eigene Komponente kann kontrolliert getestet werden
  —> auch Ausnahme und Fehlersituationen
- **ABER**: Implizite Annahmen über transient abhängige Systeme

3. Teilschritt

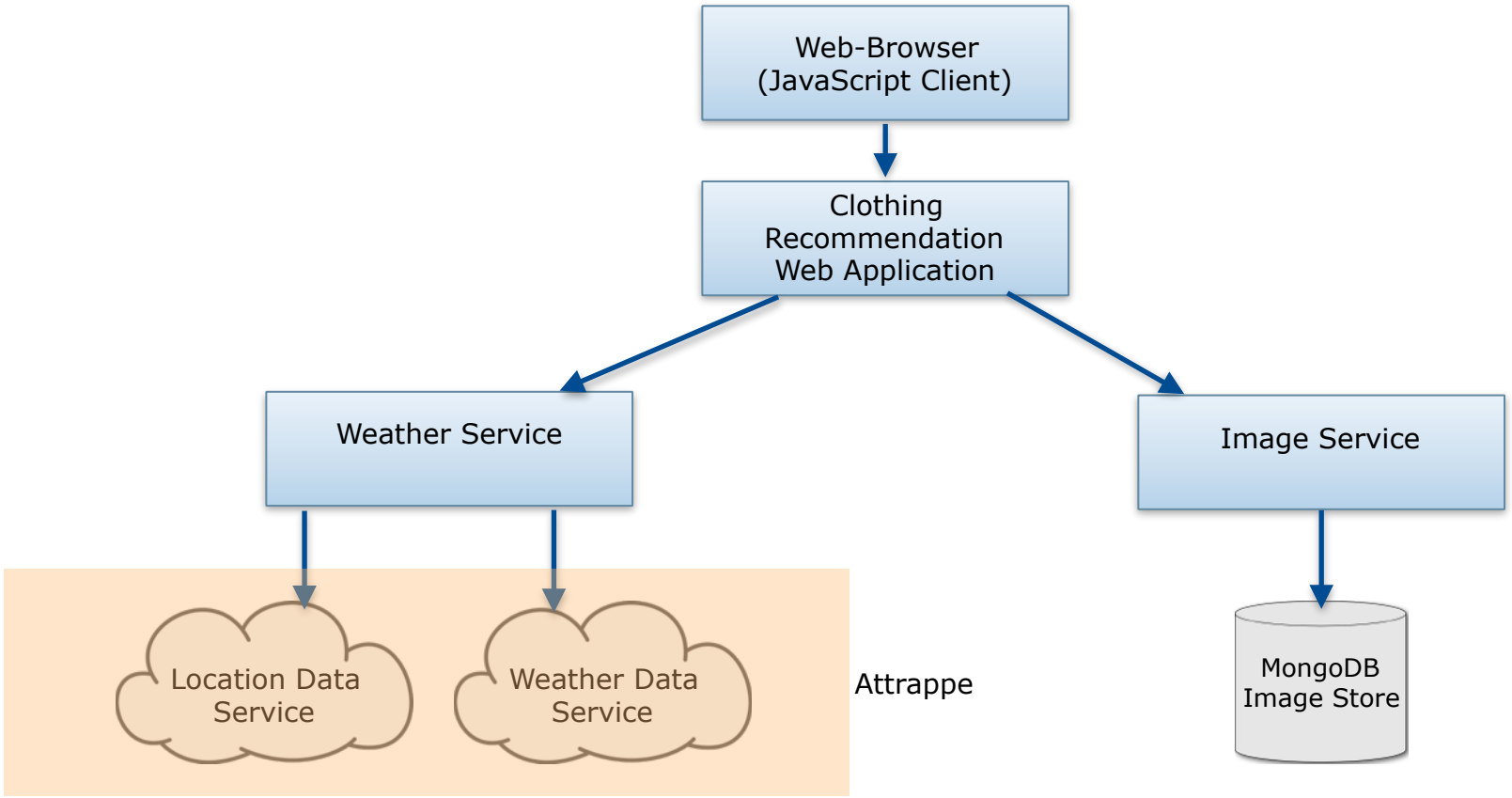
Verwendung von Attrappen

- **Programmatischer Ansatz:**
  - Abhängigkeiten werden durch explizit für Testzwecke implementierte Komponenten ersetzt
  - Geeignet für einfache Szenarien und Systeme, die in der Entstehung sind
  - Bei Legacy-Systemen nicht in jedem Fall möglich

- **Capture and Replay Ansatz:**
  - Antworten von Kollaborateuren werden im Betrieb aufgezeichnet und während des Testlaufs „abgespielt"
  - Auch teilweise geeignet bei Legacy-Systemen

Beispielapplikation

Architektur



Demo

3. Teilschritt

Testen des Kommunikationsverhaltens

- Integrative Tests, bei denen das zu testende System als Provider einer Schnittstelle fungiert, benötigen während des Testlaufs Werkzeuge mit denen Aufrufer einfach simuliert werden können

- Im Umfeld von RESTful Webservices kann beispielsweise REST-assured verwendet werden

- Bei SOAP-Webservices können aus der Schnittstellenbeschreibung generierte Test-Clients für den Testlauf genutzt werden

REST-assured

- Java-Framework zum Testen und Validieren von REST Services
- Unterstützung für JSON und XML (XPath)

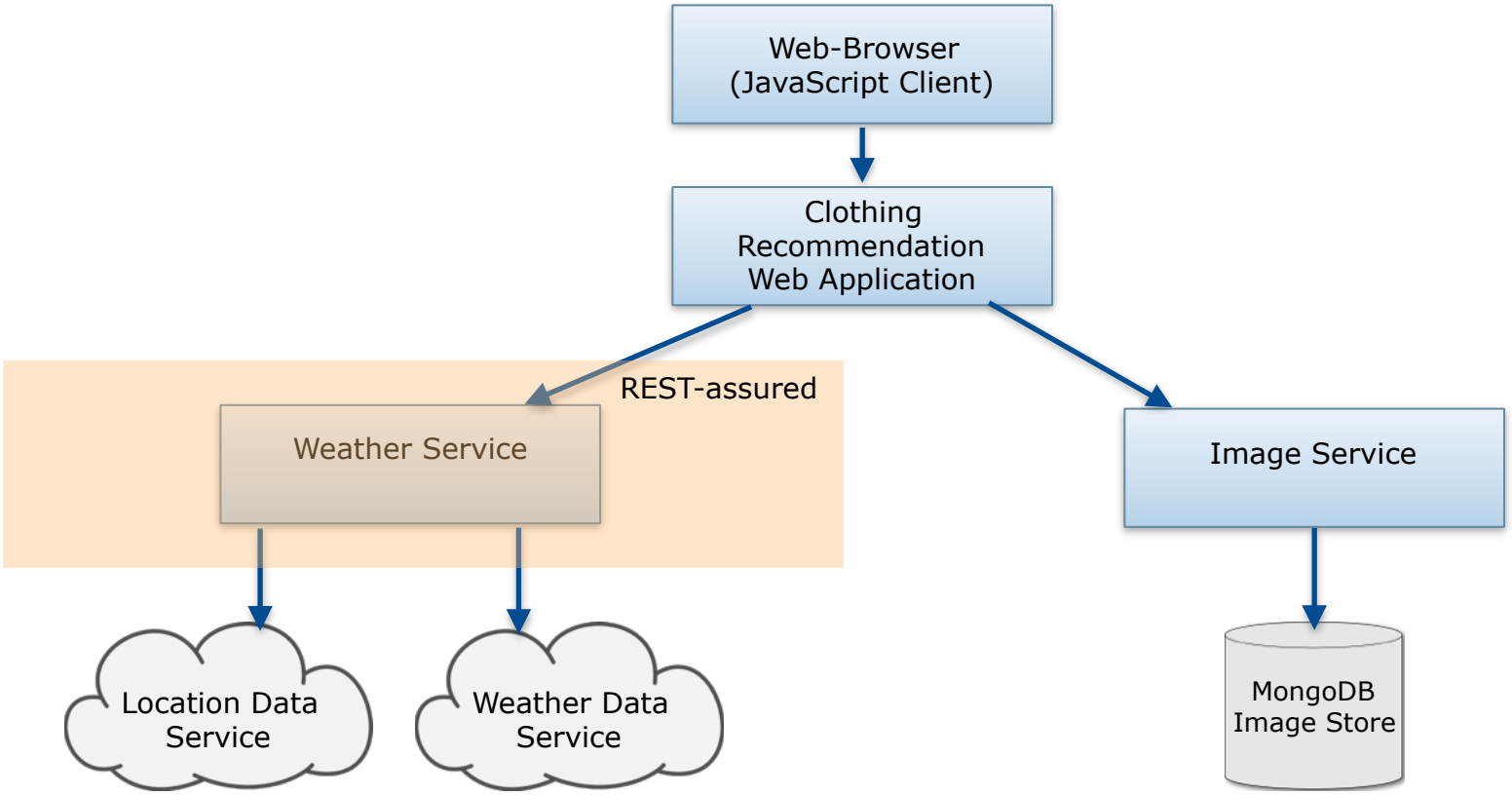
- Fluent-Interface-Notation:

```
get(„/book/1“).then().assertThat().body(„book.name", equalTo(„REST“));

given().parameters("firstName", "John", "lastName", „Doe“).
when().post("/greetMe").
then().body(hasXPath(„/greeting/firstName[text()='John']"));
```

Beispielapplikation

Architektur



Demo

3. Teilschritt

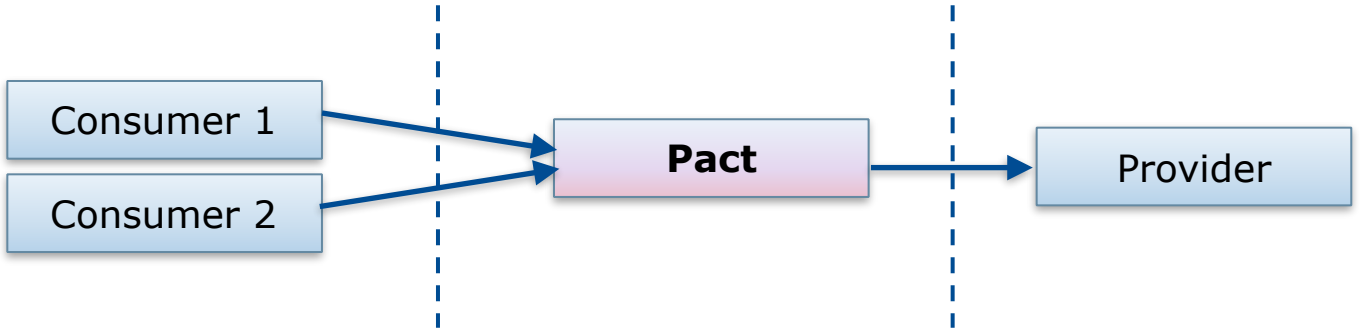
Testen des Kommunikationsverhaltens

- Verifikation mit Hilfe von Kontrakten, ermöglicht einfaches - **unabhängiges** - Testen des Clients sowie des Servers
- Schnittstelle wird anhand einer Spezifikationsdatei beschrieben
- Während des Build-Prozesses können Client und Server anhand der Spezifikationsdatei mit Daten versorgt und geprüft werden
- Im Microservice-Umfeld wird dazu häufig das Werkzeug Pact verwendet

Pact

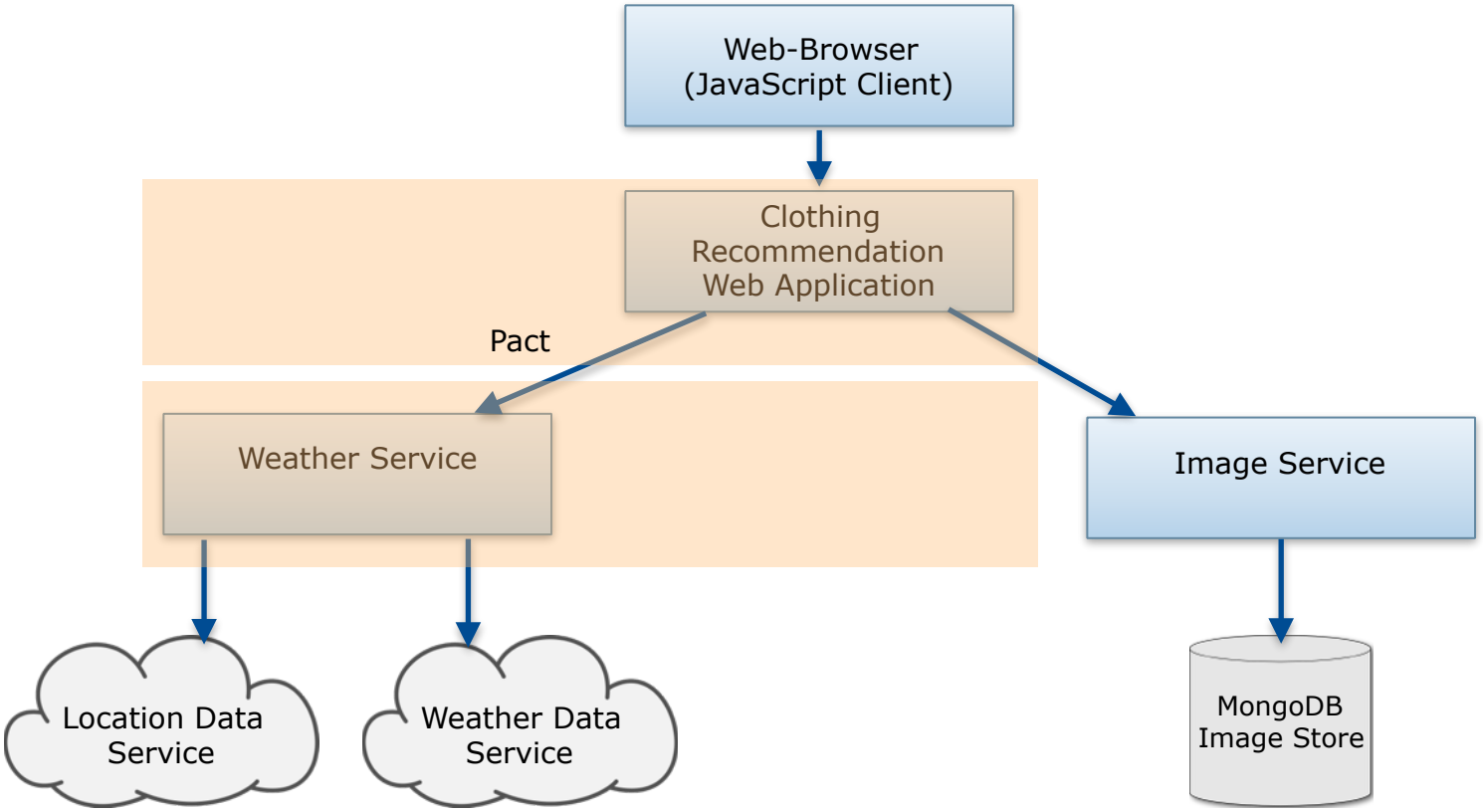
- Spezifikation für **„Consumer driven contract"-**Testing

- JSON-basierte Pact-Dateien als Definition eines Contracts zwischen Client und Server
- Pact-Implementierungen erlauben Client- und Servertests
- Implementierungen in diversen Sprachen verfügbar



Beispielapplikation

Architektur



Demo

Agenda

- Effiziente Verteilung der Tests: die Testpyramide
- Vorstellung der Beispielapplikation
- Vier Teilschritte eines Integrationstests
- 1. Initialisierung von Testsystemen
- 2. Testdatenbereitstellung
- 3. Abhängigkeiten und Kommunikationsschnittstellen
- **4. Testlauf und Validieren der Ergebnisse**

4. Teilschritt

Funktionale Tests

- Oftmals wird für Integrationstests, Oberflächentests und Systemtests ein und dasselbe Werkzeug verwendet
- Aus **Entwicklersicht** können somit mit wenig Aufwand unterschiedlichste Tests entstehen
- Aus **Fachbereichssicht** leidet hierbei häufig die Transparenz
  - Testinhalte sind für außenstehende nicht mehr ohne Entwickler-Unterstützung einsehbar
  - Eine Anpassung und Erweiterung der Tests ist ohne Hilfe von Entwicklern nicht möglich

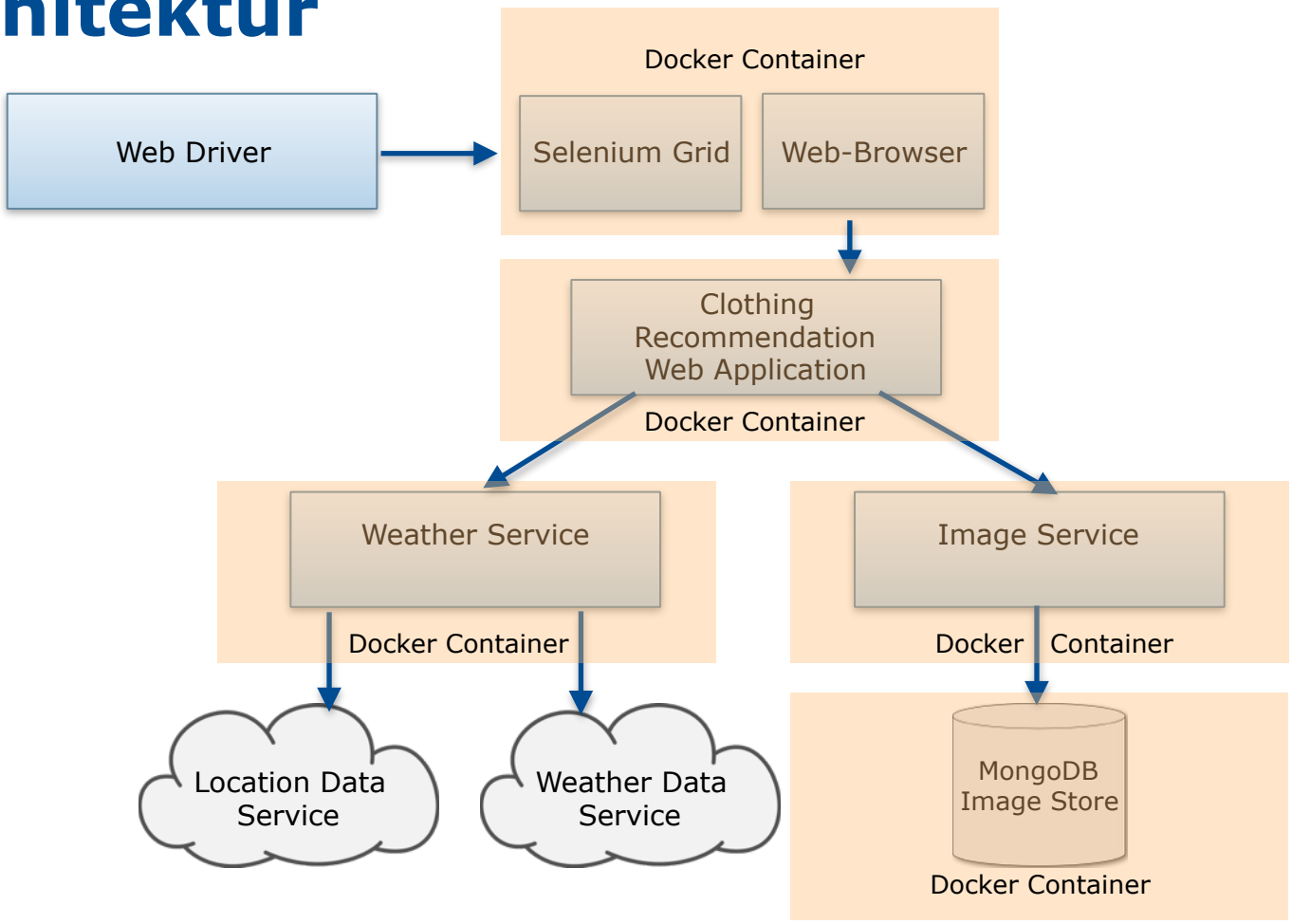
4. Teilschritt

Funktionale Tests

- Werkzeuge wie FitNesse und Cucumber ermöglichen es Fachbereichen Tests in domänen-spezifischer Sprache zu formulieren
- „ausführbare“ Anforderungen entstehen
- Randfälle und besondere Konstellationen können auch im Nachgang der Implementierung ergänzt und getestet werden
- Einfaches Testen und Auffinden von Spezifikationslücken möglich

Beispielapplikation

Architektur

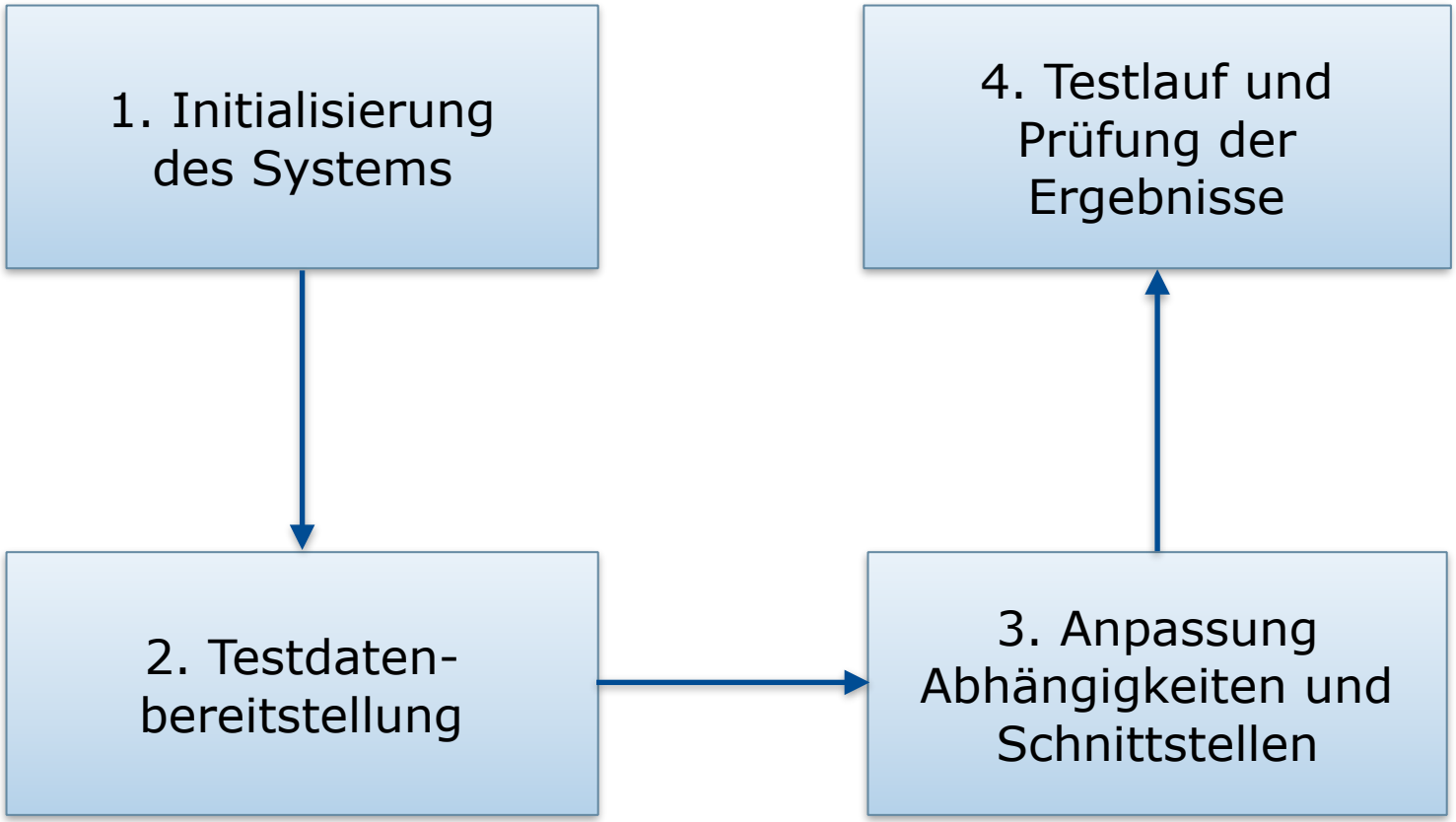


Demo

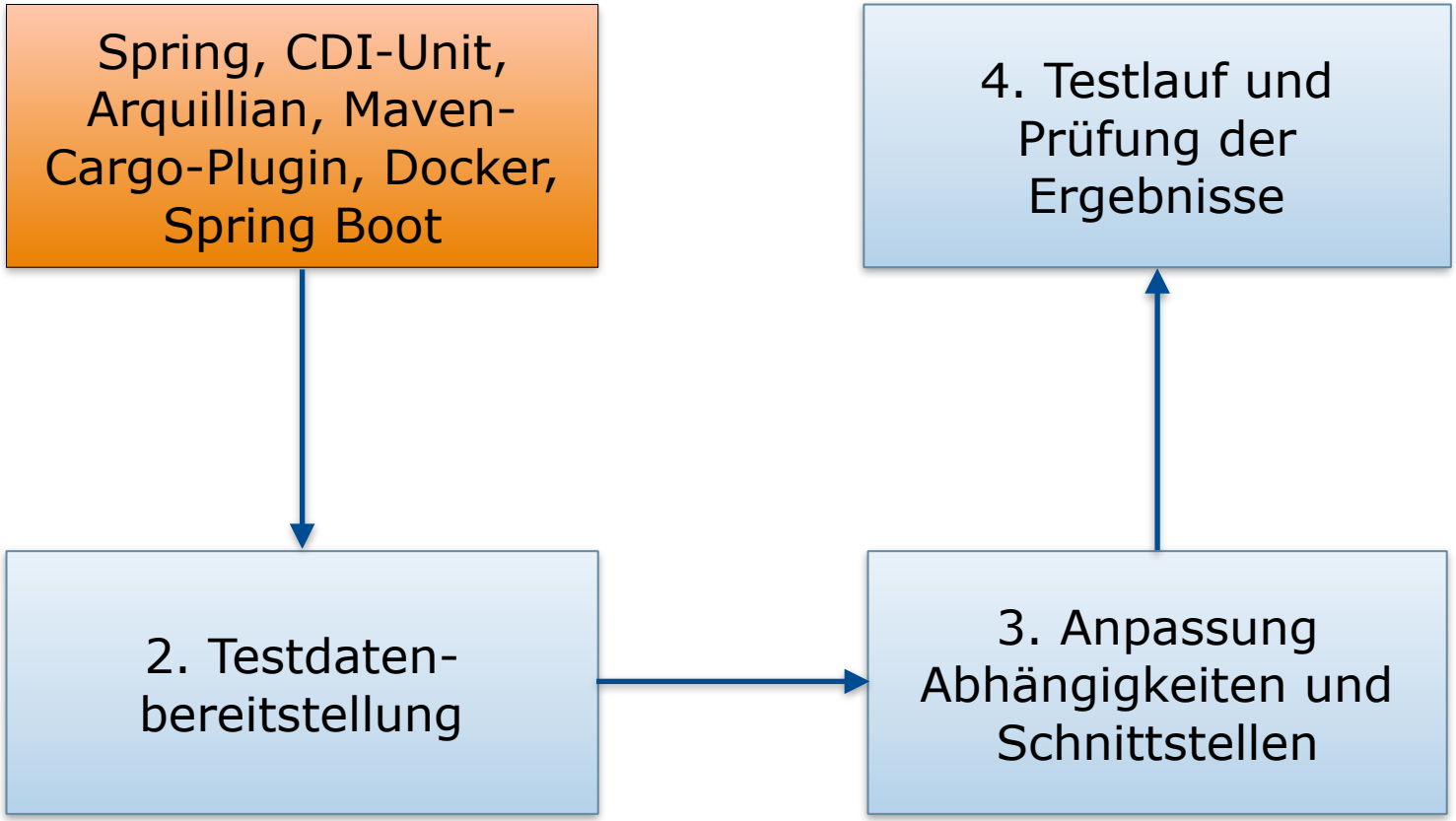
Rückblende: Motivation

- Wie kann man beim Testen vorgehen?
- Welche Arten von Tests sind zu berücksichtigen?
- Welche Bestandteile enthält ein Test?
- Wie kann man Tests leichtgewichtig implementieren?
- Welche Werkzeuge eignen sich für die unterschiedlichen Testtypen?

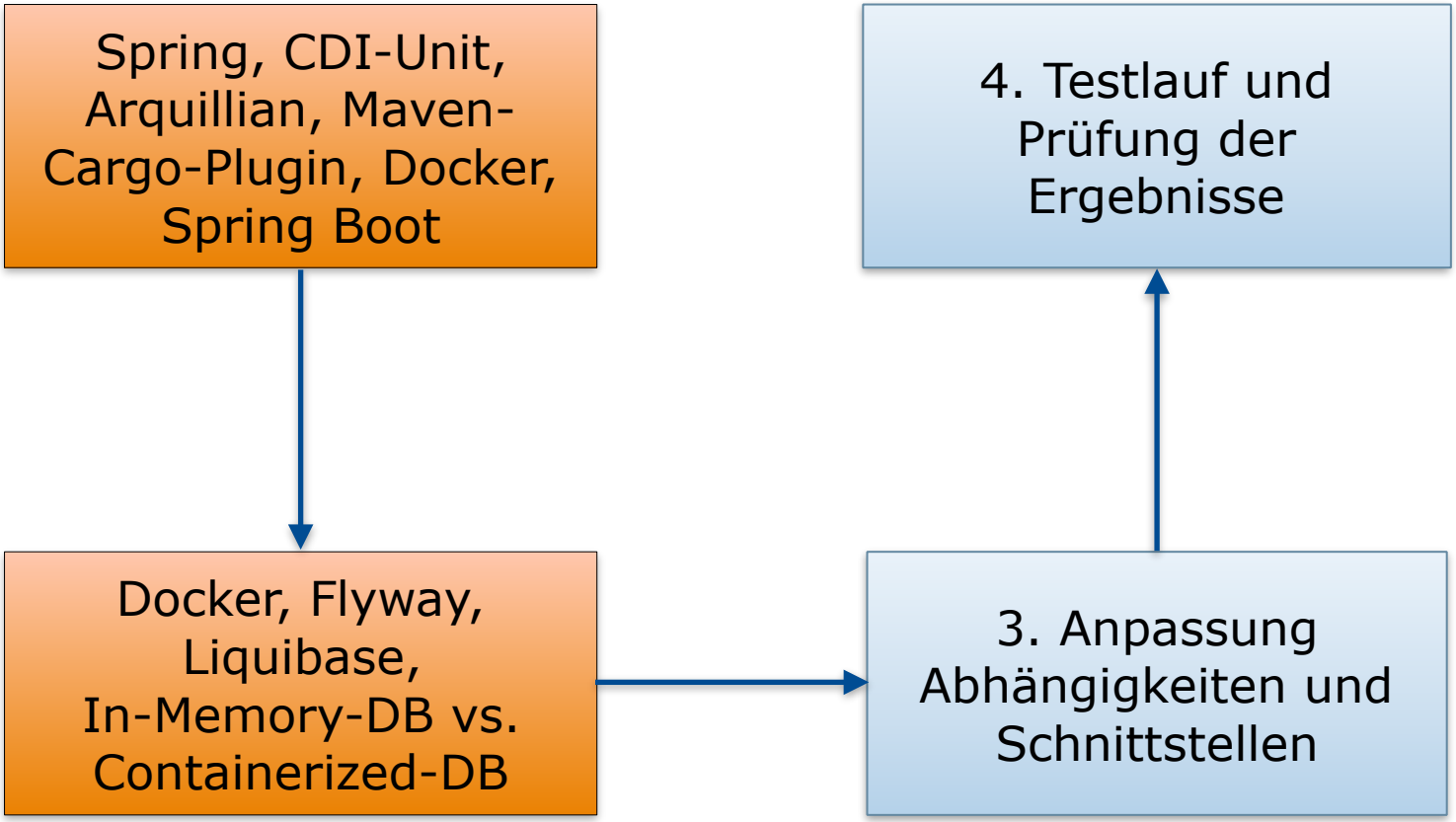
Teilschritte eines Integrationstests



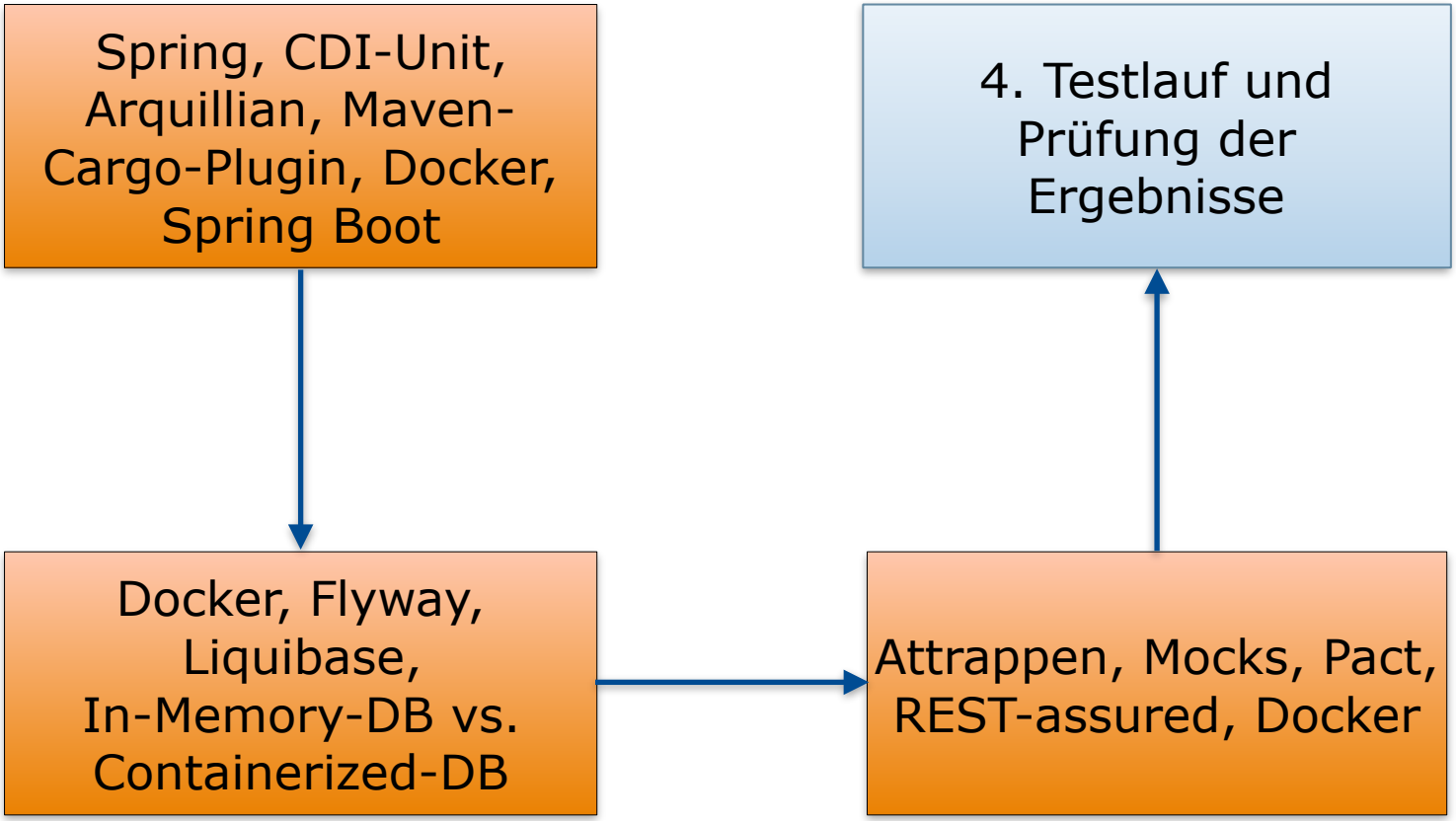
1. Initialisierung des Testsystems



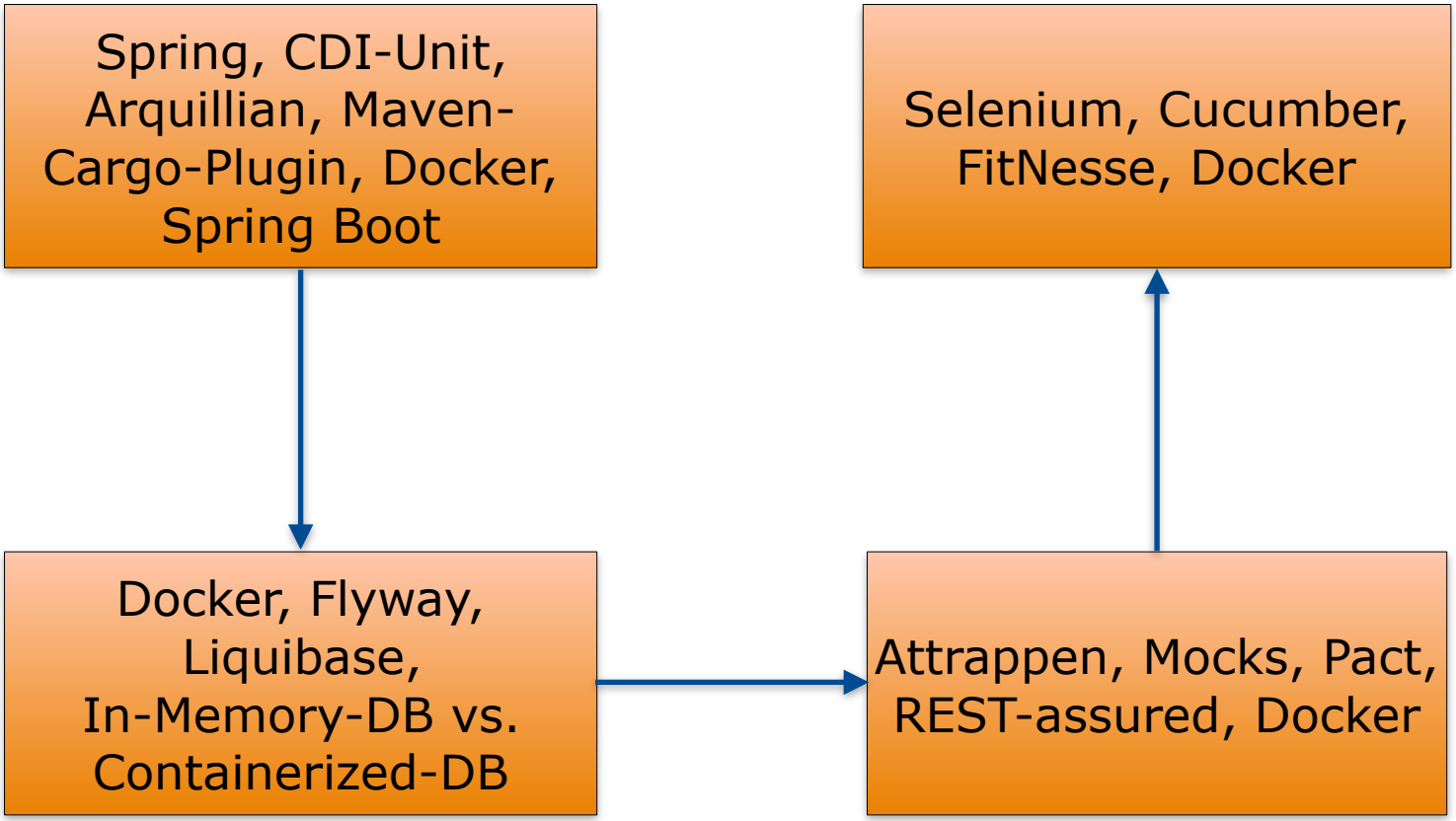
2. Testdatenbereitstellung

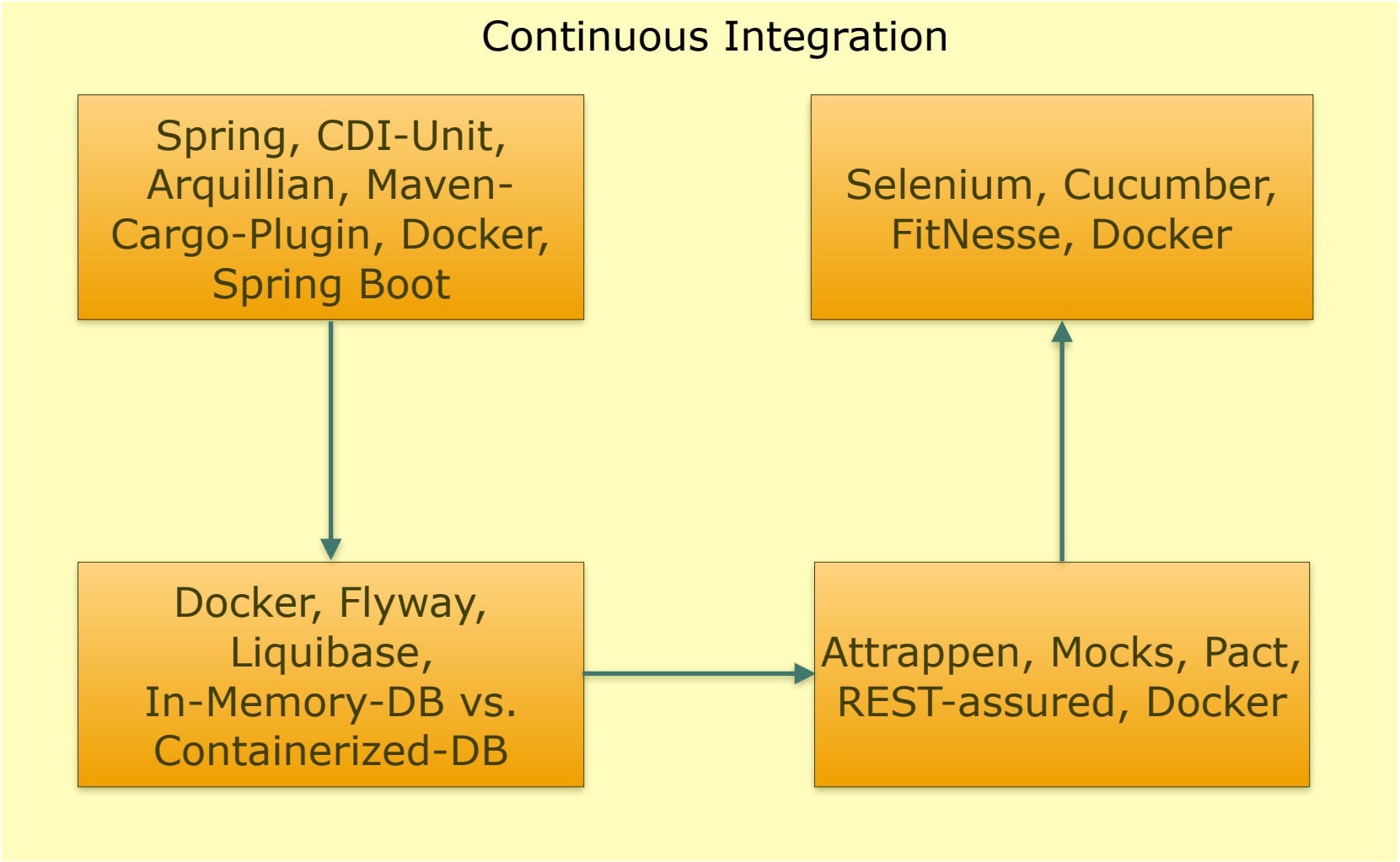


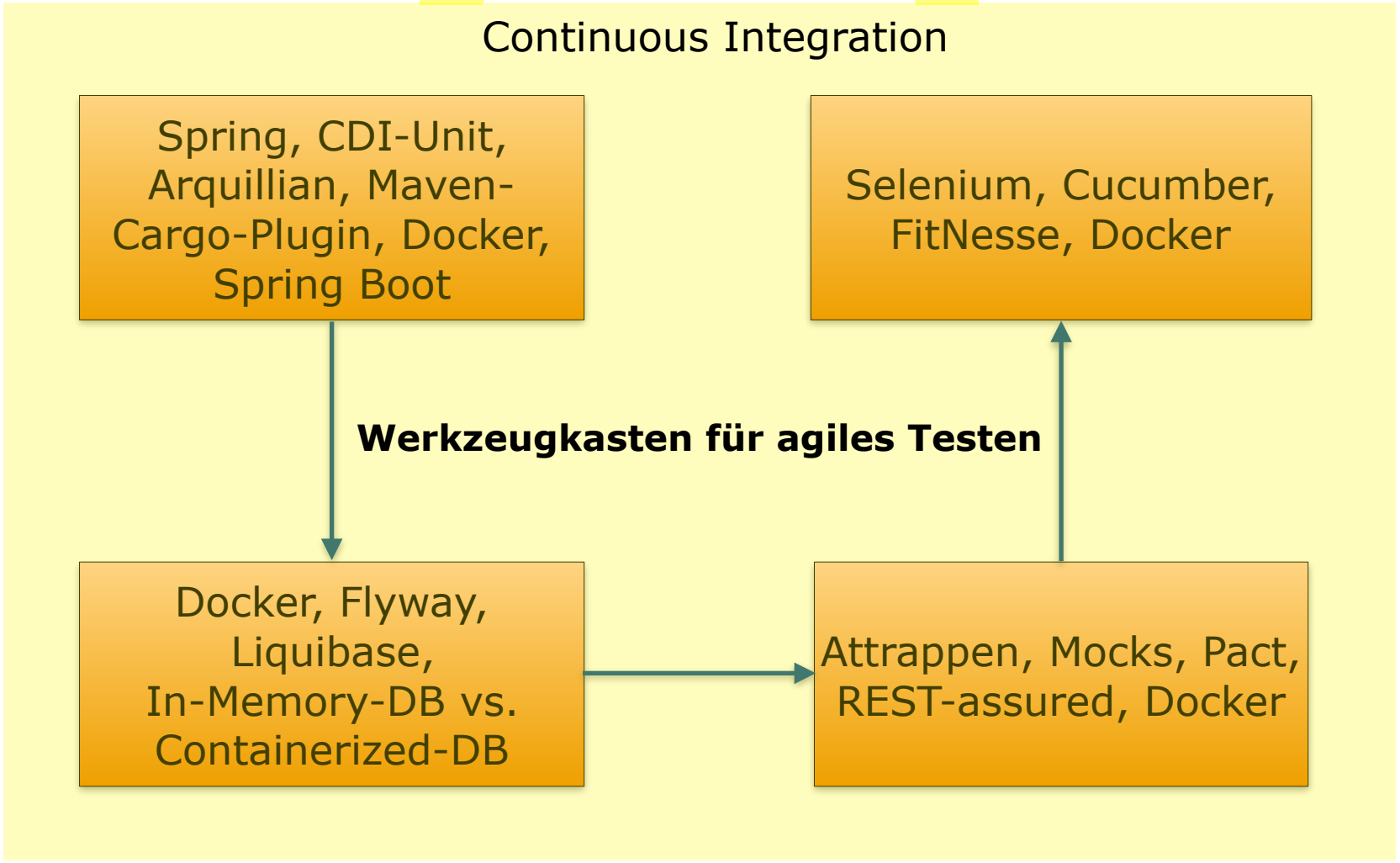
3. Abhängigkeiten und Schnittstellen



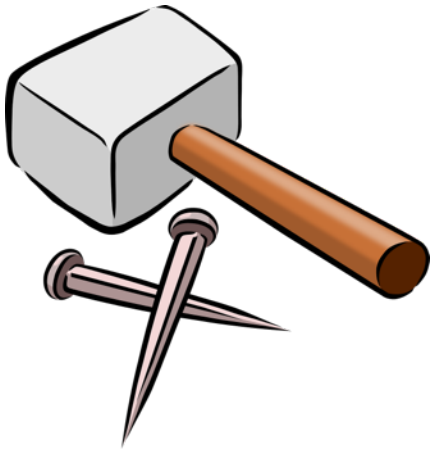
4. Funktionale Tests







Fazit



- Testtypen bewusst machen!
- Werkzeugauswahl sollte sich nach Testtyp richten
- Testparallelisierung muss gewährleistet werden

- Alle Schritte sollten **automatisiert** werden (CI !)

Links

https://github.com/andrena/testing-tools-demo.git