Fakultät Informatik

Evaluation von Contract Testing in Continuous Integration Pipelines zur Sicherstellung der Interoperabilität in einer Microservice-Architektur

Bachelorarbeit im Studiengang Informatik

vorgelegt von

Jonas Lang

Matrikelnummer 363 0314

Erstgutachter:	Prof. Dr.-Ing. Matthias Meitner
Zweitgutachter:	Prof. Dr. Ronald Petrlic
Betreuer:	Sebastian Foersch
Unternehmen:	msg systems ag

Kurzdarstellung

Kurze Zusammenfassung der Arbeit, höchstens halbe Seite. Deutsche Fassung auch nötig, wenn die Arbeit auf Englisch angefertigt wird.

Abstract

Only if thesis is written in English.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Problemstellung und Motivation

1.2 Zielsetzung der Arbeit

1.3 Aufbau der Arbeit

Kapitel 2

Grundlagen

2.1 Microservice Architektur

Die Microservice-Architektur ist ein Architekturstil für die Softwareentwicklung, bei dem eine Anwendung als eine Sammlung unabhängiger, kleiner Dienste aufgebaut wird. Jeder dieser Dienste ist eigenständig und erfüllt eine spezifische Geschäftsanforderung. Diese Architektur hat sich als Alternative zur monolithischen Architektur etabliert, bei der alle Funktionalitäten einer Anwendung in einer einzigen, großen Codebasis zusammengefasst sind.

Ein wesentliches Merkmal der Microservice-Architektur ist die Unabhängigkeit der Dienste. Jeder Microservice kann eigenständig entwickelt, getestet, bereitgestellt und skaliert werden. Die lose Kopplung zwischen den Diensten wird durch wohl definierte Application Programming Interfaces (APIs) sichergestellt, die häufig auf REST oder Messaging-Protokollen basieren [1]. Zudem wird die funktionale Trennung betont, da jeder Service einen klar definierten Anwendungsteil abdeckt und von einem eigenen Entwicklungsteam verwaltet werden kann. Diese Struktur ermöglicht es, unterschiedliche Technologien oder Programmiersprachen für verschiedene Services einzusetzen, wodurch eine technologische Vielfalt innerhalb eines Systems entsteht. So entstehen mehrere kleine Anwendungen, die einfacher zu verstehen und zu warten sind als ein großer Technologie-Stack wie in einer monolithischen Architektur.

Die Microservice-Architektur bietet zahlreiche Vorteile. Sie ermöglicht eine hohe Flexibilität, da Änderungen an einem Service unabhängig von anderen vorgenommen werden können. Die Fehlertoleranz wird verbessert, da ein Fehler in einem Microservice nicht zwangsläufig die gesamte Anwendung beeinträchtigt. Durch die gezielte Bereitstellung von Ressourcen für bestimmte Dienste kann die Skalierbarkeit optimiert werden. Zudem erleichtert die überschaubare Größe einzelner Microservices deren Wartung, da der Code besser verständlich und weniger komplex ist.

Trotz ihrer Vorteile bringt die Microservice-Architektur auch Herausforderungen mit sich. Die Kommunikation zwischen den einzelnen Diensten erfordert robuste Schnittstellen und ein zuverlässiges API-Management, um eine reibungslose Interaktion zu gewährleisten. Die

Verwaltung von Daten kann komplex sein, da die Konsistenz zwischen verschiedenen Microservices sichergestellt werden muss. Darüber hinaus erfordert die Vielzahl kleiner Dienste leistungsfähige Deployment- und Monitoring-Tools, um deren Verwaltung effizient zu gestalten.

Die Microservice-Architektur bildet die Grundlage für moderne verteilte Systeme und spielt eine zentrale Rolle für Cloud-Umgebungen. Da bei dieser Architektur die Kommunikation zwischen den einzelnen Diensten im Vordergrund steht, ist eine zuverlässige Validierung der Schnittstellen essenziell.

2.2 Arten des Softwaretests

Um die Qualität und Funktionsfähigkeit von Software sicherzustellen, kommen verschiedene Arten von Softwaretests zum Einsatz. Diese lassen sich nach ihrem Fokus und ihrer Testtiefe kategorisieren.

Unit-Tests (Modultests) überprüfen einzelne Funktionseinheiten, wie Funktionen oder Methoden, isoliert von anderen Komponenten. Sie dienen dazu, die korrekte Implementierung kleinster funktionaler Einheiten sicherzustellen. Hierbei sollten möglichst viele Einheiten getestet werden, um eine hohe Testabdeckung zu erreichen. Wegen der verhältnismäßig gering Ausführungszeit kann „[deren] Durchführung [...] sehr schnell, beliebig oft und aufwandsarm erfolgen“ [2]. Diese Tests sind in der Regel maschinell und werden häufig während der Entwicklung durchgeführt, um Fehler frühzeitig zu erkennen.

Integrationstests hingegen testen das Zusammenspiel mehrerer Module oder Systeme. Sie prüfen, ob die verschiedenen Komponenten korrekt miteinander interagieren. Integrationstests sind wichtig, um sicherzustellen, dass die einzelnen Module nicht nur isoliert, sondern auch in Kombination miteinander funktionieren. Die Testumgebung ist hierbei oft komplexer, da mehrere Module gleichzeitig zur Verfügung stehen müssen und somit ist auch die Laufzeit höher.

Systemtests betrachten die Anwendung als Ganzes. Dabei wird überprüft, ob das gesamte System konform zu den Anforderungen funktioniert. Diese Tests sind in der Regel sehr umfangreich in der Erstellung und Ausführung, da sie alle Komponenten und deren Interaktionen berücksichtigen. Darum sollten nur wenige Testfälle erstellt werden, die jedoch alle wichtigen Aspekte der Anwendung abdecken.

Akzeptanztests bilden die höchste Testebene und stellen sicher, dass die Anwendung aus Sicht der Endbenutzer den definierten Anforderungen entspricht. Diese Tests werden häufig manuell und in enger Abstimmung mit den Stakeholdern durchgeführt [3]. Sie sind entscheidend für die Validierung der Software und deren Akzeptanz durch die Benutzer. Aufgrund

des hohen Aufwands sind diese Tests nicht Teil von Entwiklungszyklen, sondern werden in der Regel erst nach der Fertigstellung des Systems durchgeführt.

Die Hierarchie der verschiedenen Teststufen wird in Abbildung 2.1 dargestellt und verdeutlicht.

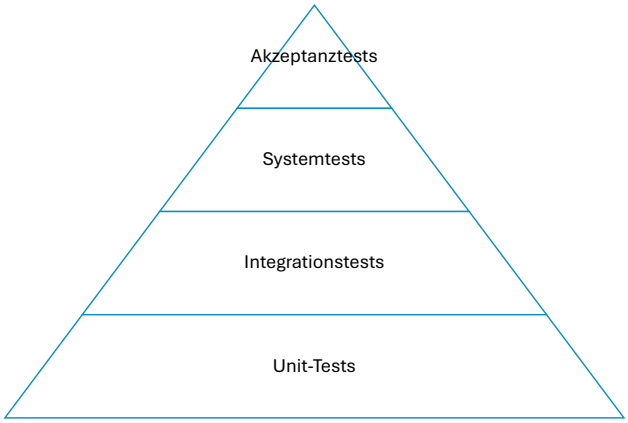

Abbildung 2.1: Testpyramide nach [2]

Neben diesen klassischen Testarten gibt es auch spezialisierte Testmethoden wie Performance-Tests, Sicherheitstests und Usability-Tests. Diese Tests konzentrieren sich auf spezifische Aspekte der Softwarequalität und sind darauf ausgelegt, bestimmte Anforderungen zu validieren. Deshalb sind diese nicht in die Testhierarchie der Testpyramide eingeordnet, sondern ergänzen diese.

Im Kontext von Microservices gewinnen automatisierte Tests und insbesondere Contract Tests an Bedeutung, da sie die Kommunikation zwischen Diensten absichern und frühzeitig Fehler in der Interaktion erkennen lassen.

2.3 Testautomatisierung

Mit dem zunehmenden Bedarf an schneller Softwareauslieferung bei gleichbleibend hoher Qualität hat sich die Testautomatisierung zu einem zentralen Bestandteil moderner Entwicklungsprozesse entwickelt. Sie beschreibt die automatisierte und kontinuierliche Durchführung von Tests, meist mithilfe spezialisierter Tools und Skripte, um manuelle Testaufwände zu reduzieren, Wiederholbarkeit zu gewährleisten und Fehler frühzeitig im Entwicklungsprozess zu erkennen.

Solche Praktiken und Tools werden häufig under dem Begriff Zusammenarbeit zwischen Entwicklung und Betrieb (DevOps) zusammengefasst. DevOps beschreibt eine kulturelle und

technische Bewegung, die darauf abzielt, die Zusammenarbeit zwischen Entwicklungs- und Betriebsteams zu verbessern. Ziel ist es, den gesamten Software-Lebenszyklus zu optimieren, von der Entwicklung über das Testing bis hin zum und Betrieb. Dazu zählen Dinge wie Kollaboration zwischen Teams, Automatisierung von regelmäßigen Aufgaben, Überwachung von Anwendungsinfrastruktur und Auswertung von Nutzerfeedback.

Einige dieser Automatisierungen werden unter dem Begriff Continuous Integration (CI) zusammengefasst. Das bedeutet, dass Änderungen am Code kontinuierlich in das gemeinsame Repository integriert werden und dabei automatisierte und Tests ausgeführt werden, um sicherzustellen, dass der neue Code keine bestehenden Funktionalitäten bricht. Durch diese sofortige Rückmeldung können Fehler frühzeitig erkannt und behoben werden.

Continuous Delivery (CD) geht noch einen Schritt weiter: Nach der erfolgreichen Integration und dem Bestehen aller Tests wird die Anwendung automatisiert für die Auslieferung vorbereitet und zusammengebaut. Dies schließt Prozesse wie das Verpacken, Bereitstellen und Testen der Anwendung in produktionsnahen Umgebungen ein. In der weitergehenden Form, dem sogenannten Continuous Deployment, erfolgt sogar die automatische Auslieferung in die Produktionsumgebung.

Testautomatisierung ist dabei das Rückgrat dieser Abläufe. Sie ermöglicht nicht nur die schnelle und zuverlässige Durchführung von Unit-, Integrations- oder End-to-End-Tests, sondern auch die Validierung nichtfunktionaler Anforderungen wie Performance oder Sicherheit. Diese Automatisierungen werden mithilfe von CI/CD-Pipelines orchestriert, die den gesamten Prozess von der Codeänderung bis zur Bereitstellung steuern.

Gerade in Microservice-Architekturen ist die Testautomatisierung unverzichtbar, da hier viele unabhängige Komponenten miteinander interagieren. Ohne automatisierte Tests würde das Risiko fehlerhafter Integrationen oder regressiver Effekte drastisch steigen. Contract Testing, als spezieller Ansatz innerhalb der Testautomatisierung, fokussiert sich gezielt auf die Kommunikation zwischen Microservices und stellt sicher, dass Änderungen in einem Dienst keine ungewollten Auswirkungen auf andere haben.

Insgesamt trägt Testautomatisierung in Verbindung mit DevOps und CI/CD entscheidend dazu bei, die Qualität, Sicherheit und Stabilität moderner Softwareprodukte zu gewährleisten.

Kapitel 3

Contract Testing

3.1 Konzept/Prinzip

3.2 Varianten

3.3 Vor- und Nachteile

Kapitel 4

Frameworks

4.1 Übersicht verfügbarer Tools

4.2 Auswahlkriterien

4.3 Vergleich der Frameworks

Kapitel 5

Implementierung

5.1 Anwendungsumgebung

5.2 Technische Umsetzung

5.3 Probleme und Herausforderungen

Kapitel 6

Fazit und Ausblick

6.1 Bewertung der Lösung

6.2 Zusammenfassung der Ergebnisse

6.3 Ausblick auf zukünftige Entwicklungen

Anhang A

Weiterführende Informationen

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungsliste

Akronyme

API Application Programming Interface. 2

DevOps Zusammenarbeit zwischen Entwicklung und Betrieb. 4

Literatur

[1] C. Richardson, *Microservices Patterns*. Shelter Island, NY: Manning Publications Co., Nov. 2018, ISBN: 978-1-61729-454-9. Adresse: https://learning.oreilly.com/library/view/microservices-patterns/9781617294549/ (besucht am 19. 03. 2025).

[2] T. Linz und A. Spillner, *Basiswissen Softwaretest*, 7. Aufl. Heidelberg: dpunkt.verlag GmbH, Apr. 2024, 408 S., ISBN: 978-3-98889-005-4. Adresse: https://learning.oreilly.com/library/view/basiswissen-softwaretest-7th/9781098169114/ (besucht am 22. 03. 2025).

[3] Y. Singh, *Software Testing*. Cambridge: Cambridge University Press, 2011, ISBN: 978-1-107-01296-7. DOI: 10.1017/CBO9781139196185. Adresse: https://www-cambridge-org.thn.idm.oclc.org/core/books/software-testing/ADF93C6EDB4761D9ACFE2DE987A0DBD2 (besucht am 20. 03. 2025).