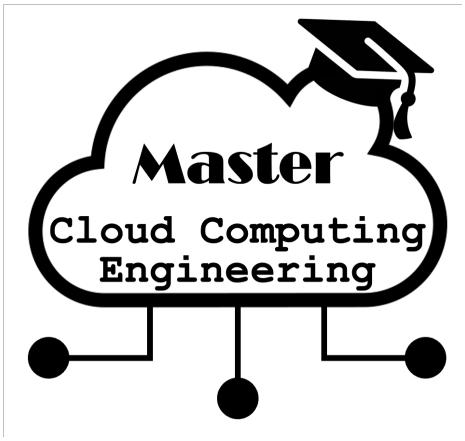

DevOps ILV - Aufgabenstellung 3

Hochschule Burgenland
Studiengang MCCE
Sommersemester 2025

Harald Beier* Susanne Peer† Patrick Prugger‡
Philipp Palatin§

17. Mai 2025

*2410781028@hochschule-burgenland.at
†2410781002@hochschule-burgenland.at
‡2410781029@hochschule-burgenland.at
§2310781027@hochschule-burgenland.at

Inhaltsverzeichnis

1 Aufgabenstellung 3

1.1 Testumgebungsstabilisierung

Für unsere E-Commerce-Plattform mit ihren vielfältigen Integrationen (SAP, AWS, GPT, HubSpot) setzen wir auf:

- **Infrastruktur als Code (IaC):** Wir nutzen Terraform/CloudFormation, um isolierte Testumgebungen automatisiert zu erstellen und zu verwalten.
- **Umgebungsmodellierung:** Jede Testumgebung wird für bessere Transparenz und Kontrolle mit ihren Komponenten, Konfigurationen und Testdaten dokumentiert.
- **Containerisierung:** Kubernetes-Namespaces für kurzlebige, isolierte Testumgebungen.
- **Sandbox-Accounts:** Dedizierte Test-Accounts für Cloud-Dienste verhindern Konflikte zwischen den Teams.

Zusammenfassend isolieren und verwalten wir die Infrastruktur mittels IaC (z. B. Terraform/CloudFormation) und nutzen dedizierte Sandbox-Accounts, um Konflikte zu vermeiden. Service-Virtualisierung ist eine zentrale Strategie die im nächsten Abschnitt beschrieben wird. Für eine beispielhafte Darstellung der Umgebung siehe Abbildung 1.

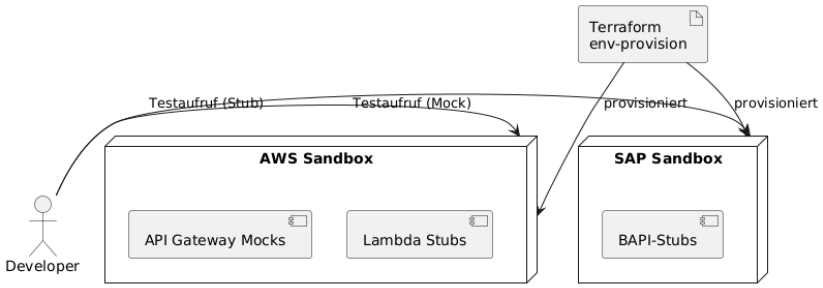

Abbildung 1: Deployment-Diagramm: Virtuelle Testumgebungen via IaC

1.1.1 Service-Virtualisierung und Mock-Ansätze

Service-Virtualisierung ist entscheidend, wenn externe Systeme nicht verfügbar, instabil oder die Kosten für die Nutzung für Tests zu hoch sind:

- **REST/HTTP-Interfaces:** WireMock oder Mountebank für die Simulation von REST-APIs WireMock (2025).
- **Cloud-APIs:** AWS LocalStack für lokale Emulation von AWS-Services Services (2021).

- **ERP-Integration:** Virtualisierung von SAP BAPIs und speziellen Schnittstellen.
- **API-Gateway:** Nutzung von AWS API Gateway zur Erstellung von Mock-Endpunkten.

Für abhängige Systeme (ERP oder externe APIs) können Virtualisierungstools realistische Interaktionen simulieren. Open-Source-Lösungen wie WireMock oder Mountebank (siehe dazu Byars (2018)) stubben REST-/HTTP-Schnittstellen, während Cloud-Tools (z. B. AWS LocalStack) Cloud-APIs lokal emulieren. Der Ablauf ist in Abbildung 2 dargestellt.

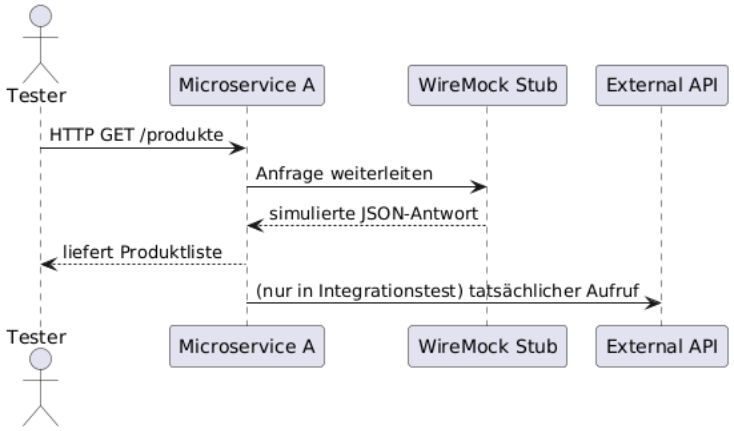

Abbildung 2: Sequenzdiagramm: Service-Virtualisierung mit WireMock

1.1.2 Testdatenmanagement

Um konsistente und compliance-konforme Tests zu gewährleisten, können folgende Ansätze verwendet werden:

- **Datenmaskierung:** Ersetzen von Personally Identifiable Information (PII) mit realistischen fiktiven Werten unter Beibehaltung der referentiellen Integrität. Tricentis (2024) empfiehlt Format-preserving Encryption für PII
- **Synthetische Daten:** Künstliche Datensätze für Spezialfälle und Randszenarien. BrowserStack (2025) kombiniert synthetische und maskierte Daten.
- **Hybridansatz:** Maskierte Produktionsdaten für Basistests, ergänzt durch synthetische Daten für Edge-Cases.
- **API-Integration:** CI/CD-Pipeline kann Testdaten über APIs auffrischen, zurücksetzen oder klonen.

1.2 Testarten und Abdeckung

1.2.1 Funktionale Tests

Zur Abdeckung funktionaler Anforderungen setzen wir folgende Testtypen ein:

- **Unit-Tests:** Für einzelne Module/Services, laufen bei jedem Commit.
- **API-Tests:** Validierung jeder Microservice-Schnittstelle gegen ihre Spezifikation (mit JUnit, pytest oder Postman/Newman).
- **Integrationstests:** Testen zusammenhängender Dienste (z.B. Inventarsynchronisation von SAP zu NetSuite).
- **End-to-End-Tests:** Simulation realer Benutzerszenarien (Produktsuche, Checkout etc.) mit Selenium, Cypress oder Playwright.

1.2.2 Nicht-funktionale Tests

Zur Prüfung von Performance, Security, Verfügbarkeit und Datenintegrität verwenden wir:

- **Performance/Last-Tests:** Apache JMeter oder Gatling zur Simulation von Verkehrsspitzen und Messung der Skalierbarkeit.
- **Security-Tests:** Kombination aus statischer (SAST) und dynamischer (DAST) Analyse mit Tools wie SonarQube, Snyk oder OWASP ZAP.
- **Verfügbarkeitstests:** Monitoring der Systemverfügbarkeit unter verschiedenen Lastbedingungen.
- **Datenintegritätstests:** Validierung der Datenkonsistenz zwischen SAP, NetSuite und AWS.

1.2.3 CI/CD-Pipeline Integration

Tests sind in der Pipeline wie folgt integriert:

- **Build-Phase:** Unit-Tests und API-Tests laufen bei jedem Build.
- **Deployment-Phase:** Integrations- und Smoke-Tests nach Deployment in Testumgebung.
- **Post-Deployment:** Aufwändige Tests (Performance, Security) in parallelen oder separaten Pipelines.
- **Nightly-Jobs:** Umfassende Tests, die mehr Zeit beanspruchen.

1.3 Testeffizienz und Wartbarkeit

1.3.1 Strukturierung der Tests für Systemveränderungen

Um flexibel auf Änderungen (wie SAP-Upgrades oder Microservice-Updates) reagieren zu können:

- **Modulare Testarchitektur:** Tests sind nach Komponenten oder Services organisiert.
- **Shared Libraries:** Gemeinsame Funktionen und Daten-Fixtures vermeiden Duplikationen.
- **Page Object Model:** Kapselung von UI-Interaktionen für bessere Wartbarkeit.
- **API-Client-Bibliotheken:** Wiederverwendbare Clients für API-Tests.

1.3.2 Effizienzansätze

Zur Optimierung des Testaufwands setzen wir ein:

- **Impact Analysis:** Identifikation relevanter Tests nach Code-Änderungen durch Version-Control-Hooks und spezielle Tools.
- **Risikobasiertes Testen:** Priorisierung von Features mit hoher Geschäftsrelevanz oder bekannter Komplexität.
- **Sprint-basierte Testplanung:** QA und Entwicklung bewerten gemeinsam Änderungsauswirkungen und passen Testpläne an.

1.4 Reporting & Testtransparenz

1.4.1 Dokumentation und Auswertung

Für transparentes Reporting nutzen wir:

- **CI/CD-Dashboard:** Unit- und Integrationstestergebnisse (Pass/Fail, detaillierte Logs) im CI-Dashboard.
- **Coverage-Reports:** JaCoCo, Coverage.py für Codeabdeckungsanalysen.
- **Test-Framework-Reports:** HTML/XML-Reports von Frameworks wie pytest, TestNG oder Cucumber.
- **Aggregationstools:** Allure oder ReportPortal für umfassendere Analysen.
- **Monitoring-Dashboards** Grafana/Kibana zur Visualisierung von Performance-Metriken.

1.4.2 Stakeholder-spezifische Sichten

- **Entwickler:Innen:** Detaillierte Fehlerprotokolle und Stack-Traces zur schnellen Fehlerbehebung.
- **QA-Leads und Team:** Übersichts-Dashboards mit Testfallstatus, Defect-Counts und Coverage (z.B. in TestRail, Xray oder Zephyr).
- **Operations:** Monitoring-Tools (CloudWatch, Prometheus/Grafana) für Performance und Systemgesundheit.
- **Management:** Hochrangige Indikatoren wie Testbestehensraten, Coverage-Prozentsätze und Business-Risikobewertungen.
- **DevOps-Metriken:** DORA-Metriken (Deployment-Frequenz, Change-Failure-Rate) neben Testmetriken.

Durch automatisierte Berichterstellung (per E-Mail, Slack oder interne Dashboards) stellen wir Rechenschaftspflicht und zeitnahes Feedback sicher.

Diese Strategien ermöglichen eine schlanke, aber effektive Testsuite, die sich an verändernde Anforderungen anpasst und gleichzeitig den Wartungsaufwand kontrolliert.

1.5 Toolauswahl und Integration

1.5.1 Testautomatisierung

- **UI-Tests:** Selenium WebDriver oder Playwright für browserübergreifende Tests
- **Unit/Integration:** JUnit/TestNG oder pytest
- **BDD:** Cucumber oder Behave
- **API-Testing:** Postman/Newman oder REST-assured
- **Cloud-Testing:** LambdaTest für Tests auf verschiedenen OS/Browser-Kombinationen

1.5.2 Performance-Testing

- **Protokollebene:** Apache JMeter für verteilte Lasttests
- **Code-basiert:** Gatling für programmierbare Lastszenarien
- **Cloud-Services:** BlazeMeter, k6 Cloud für On-Demand-Skalierung

1.5.3 Service-Virtualisierung

- **HTTP-Stubbing:** WireMock oder Mountebank
- **Cloud-API-Emulation:** AWS API Gateway Mocks, LocalStack
- **Enterprise-Protokolle:** Parasoft Virtualize, Tricentis StubWeb für komplexe Unternehmensschnittstellen

1.5.4 Testdatenmanagement

- **Enterprise-Plattformen:** Informatica, Delphix oder open source Lösungen wie Data Masker
- **Datengenerierung:** Faker-Bibliotheken wie Mockaroo, dbForge Data Generator
- **Datenbank-Cloning:** Dockerisierte Test-DBs für isolierte Testdatenbanken

1.5.5 Reporting & Testmanagement

- **Orchestrierung:** GitLab CI oder Jenkins für CI/CD-Pipelines
- **Reporting:** Allure, ReportPortal oder Grafana/Kibana für Dashboards
- **Code-Qualität:** SonarQube für statische Code-Analyse
- **Testmanagement:** TestRail, Xray oder Zephyr für Testfallmanagement
- **Monitoring:** Prometheus/Grafana für Performance- und Verfügbarkeitsüberwachung

Alle gewählten Tools unterstützen DevOps-Praktiken: Sie integrieren sich in CI/CD-Pipelines, bieten REST-APIs oder Plugins und skalieren in der Cloud.

1.6 Zusammenfassung

Unsere Teststrategie für die E-Commerce-Plattform stellt durch ein umfassendes Konzept sicher, dass alle funktionalen und nicht-funktionalen Anforderungen effektiv getestet werden. Wir setzen auf stabile, automatisierte Testumgebungen, Service-Virtualisierung für externe Systeme und ein effizientes Testdatenmanagement. Die Integration verschiedener Testarten in die CI/CD-Pipeline, verbunden mit einer modularen, wartbaren Testarchitektur und transparentem Reporting, garantiert kontinuierliches Qualitätsfeedback und hält mit agilen Lieferanforderungen Schritt.

1.7 Anhang

1.7.1 Tool-Landschaft Überblick

Tabelle 1: Tool-Landschaft Überblick

Kategorie	Tools	Anwendungsbereich
Testautomatisierung	Selenium, Playwright, Cypress, JUnit/TestNG, pytest, Cucumber, Postman/Newman	UI- und Funktionstest, API-Test
Performance-Testing	Apache JMeter, Gatling, k6, BlazeMeter	Last- und Performance-Tests
Service-Virtualisierung	WireMock, Mountebank, AWS LocalStack, API Gateway Mocks, Parasoft Virtualize	Mock-Services für APIs und Systeme
Testdatenmanagement	Delphix, Informatica TDM, Redgate Data Generator, Faker-Bibliotheken	Datenmaskierung, Synthetische Daten
Reporting & Testmanagement	Jenkins/GitLab CI, Allure Report, ReportPortal, Xray/Zephyr/TestRail, Grafana/ELK	Testdokumentation, Auswertung

Ein Beispiel für eine für eine wissenschaftliches Paper das Tools vergleicht ist Software Testing: 5th Comparative Evaluation: Test-Comp 2023 von Beyer (2023)

1.7.2 Testarchitektur und Komponentendiagramm

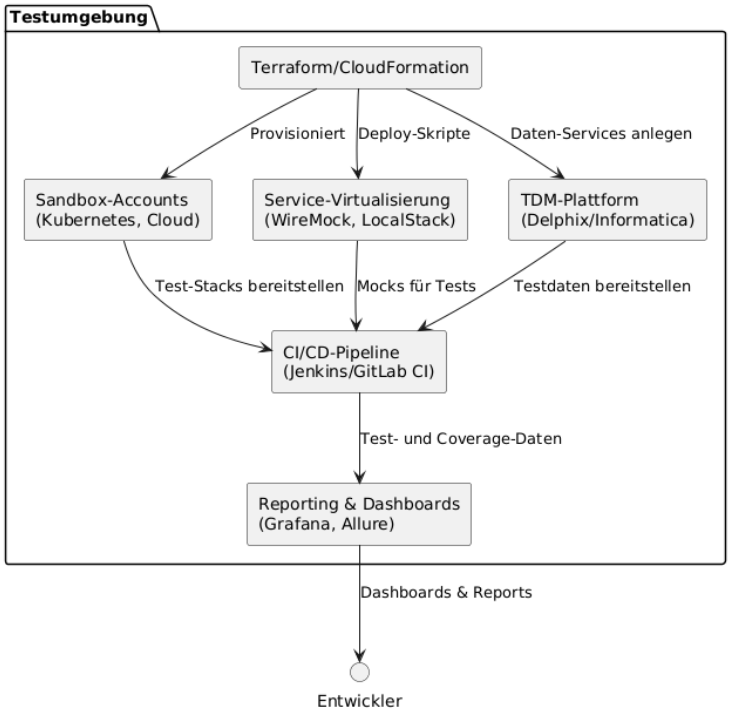

Abbildung 3: Architektur: Komponentenübersicht der Testumgebung

In Abbildung 3 ist unsere Testarchitektur dargestellt. Diese Architektur zeigt die verschiedenen Komponenten, die in der Testumgebung verwendet werden, einschließlich der Testautomatisierung, Service-Virtualisierung und Testdatenmanagement-Tools. Die Architektur ist so gestaltet, dass sie eine klare Trennung zwischen den verschiedenen Schichten der Testumgebung ermöglicht und gleichzeitig eine einfache Integration in die CI/CD-Pipeline gewährleistet.

- **IaC** (Terraform/CloudFormation) stellt Sandbox-Accounts, Virtualisierungs- und TDM-Komponenten bereit.
- **Service-Virtualisierung** (WireMock, LocalStack) simuliert externe Systeme und liefern damit isolierte Umgebungen und Mock-Services.
- **TDM** versorgt die Pipeline mit maskierten oder synthetischen Daten.
- **CI/CD-Pipeline** (Selenium, JUnit, Postman) orchestriert Tests und stellt Ergebnisse ins Reporting.
- **Reporting** (Allure, Grafana) aggregiert Testergebnisse und stellt sie in Dashboards dar.

1.7.3 Ablaufflow der Continuous-Testing-Pipeline

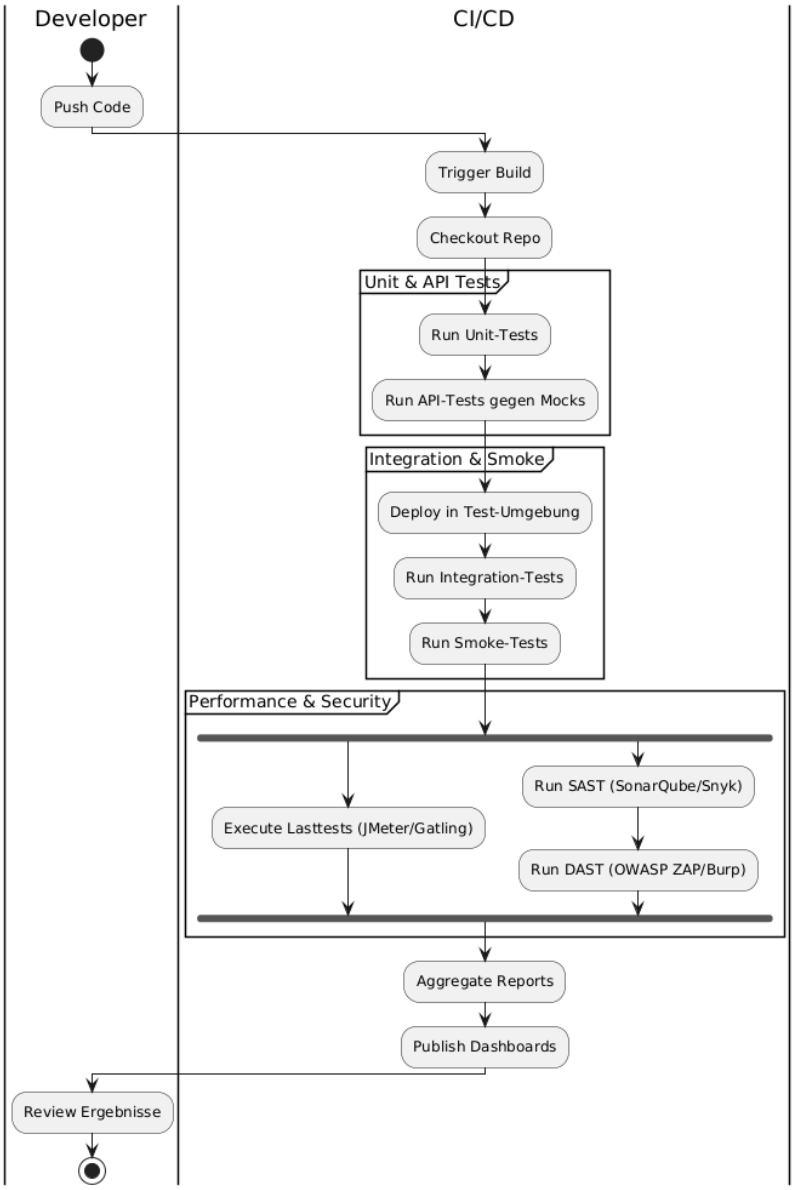

Abbildung 4: Ablaufflow: Continuous-Testing-Pipeline

In Abbildung 4 ist der Ablaufflow der Continuous-Testing-Pipeline dargestellt. Diese Pipeline zeigt die verschiedenen Schritte, die in der Testumge-

bung durchgeführt werden, nachfolgende eine kurze Beschreibung der einzelnen Schritte:

- **Unit & API Tests** laufen sofort gegen Mocks und Stubs.
- **Integration & Smoke** werden in einer auf IaC bereitgestellten Testumgebung durchgeführt.
- **Performance & Security** finden parallel in eigenen Phasen statt.
- Abschließend werden alle Reports zusammengeführt und im Dashboard veröffentlicht.

Literaturverzeichnis

Beyer, D. (2023). Software testing: 5th comparative evaluation: Test-Comp 2023. In *Proceedings of the 26th International Conference on Fundamental Approaches to Software Engineering*, pages 309–323. Springer.

BrowserStack (2025). Test data management: Strategies for effective testing. Abgerufen am 16.05.2025.

Byars, B. (2018). *Testing Microservices with Mountebank*. Manning Publications.

Services, A. W. (2021). Test aws infrastructure using localstack and terraform. Abgerufen am 16.05.2025.

Tricentis (2024). Test data management: Developing a strategy. Abgerufen am 16.05.2025.

WireMock (2025). Wiremock documentation. Abgerufen am 16.05.2025.