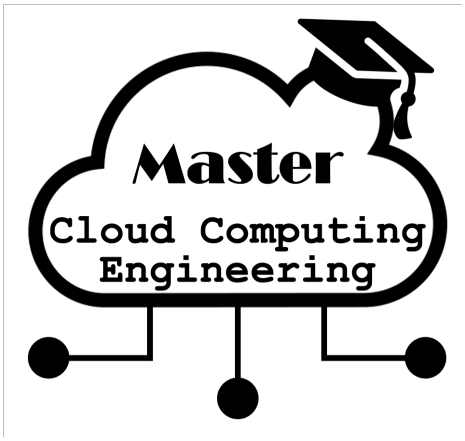


DevOps ILV - Aufgabenstellung 3

Hochschule Burgenland
Studiengang MCCE
Sommersemester 2025

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

14. Mai 2025

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

Inhaltsverzeichnis

1 Aufgabenstellung 3

1.1 Ausgangssituation

Sie sind Teil eines Qualitätssicherungsteams in einem fiktiven Unternehmen, das eine E-Commerce-Plattform betreibt. Die Plattform bietet Kameraprodukte an, nutzt KI-gestützte Produktsuche, integriert verschiedene Backend-Systeme (z. B. SAP, AWS, NetSuite, HubSpot) und führt den Nutzer durch einen komplexen, serviceorientierten Kaufprozess.Ein zentraler Bestandteil dieses Systems ist der in der beigefügten Prozessgrafik dargestellte End-to-End-Ablauf, vom Website-Besuch bis zur finalen Mail-Bestätigung. Die Architektur umfasst APIs, externe Services, Cloud-Provisionierung und ERP-Integration.

1.2 Ziel

Erarbeiten Sie als Gruppe eine ganzheitliche Teststrategie für dieses System. Die Strategie soll praktikabel sein, typische Herausforderungen im DevOps-Umfeld adressieren und die Integration verschiedener Testebenen, Tools und Umgebungen berücksichtigen.

1.3 Bearbeitungsschwerpunkte

Bitte erarbeiten Sie eine Ausarbeitung (max. 6 Seiten + Anhang), in der Sie folgende Punkte behandeln:

1.3.1 1. Testumgebungsstabilisierung

- *Wie stellen Sie in der Entwicklungs- und Integrationsphase eine stabile Testumgebung sicher, insbesondere angesichts externer Systeme (SAP, AWS, GPT, HubSpot etc.)?*
- *Wie kann Service-Virtualisierung oder Mocking zum Einsatz kommen?*
- *Welche Datenanforderungen bestehen (z. B. synthetische Testdaten, Daten-Maskierung)?*

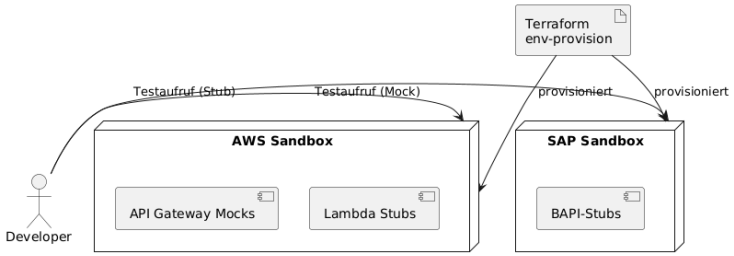
Für unsere E-Commerce-Plattform mit ihren vielfältigen Integrationen (SAP, AWS, GPT, HubSpot) setzen wir auf:

- **Infrastruktur als Code (IaC):** Wir nutzen Terraform/CloudFormation, um isolierte Testumgebungen automatisiert zu erstellen und zu verwalten.
- **Umgebungsmodellierung:** Jede Testumgebung wird mit ihren Komponenten, Konfigurationen und Testdaten dokumentiert, für bessere Transparenz und Kontrolle.
- **Containerisierung:** Kubernetes-Namespaces für kurzlebige, isolierte Testumgebungen.

- **Sandbox-Accounts:** Dedizierte Test-Accounts für Cloud-Dienste verhindern Konflikte zwischen Teams.

Zusammenfassend isolieren wir und verwalten die Infrastruktur mittels Infrastructure as Code (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

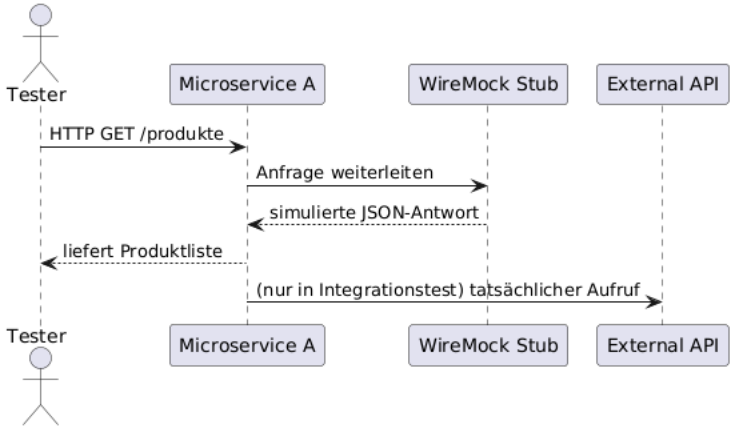


Service-Virtualisierung und Mock-Ansätze Service-Virtualisierung ist entscheidend, wenn externe Systeme nicht verfügbar oder instabil sind:

- **REST/HTTP-Interfaces:** WireMock oder Mountebank für die Simulation von REST-APIs.
- **Cloud-APIs:** AWS LocalStack für lokale Emulation von AWS-Services.
- **ERP-Integration:** Virtualisierung von SAP BAPIs und speziellen Schnittstellen.
- **API-Gateway:** Nutzung von AWS API Gateway zur Erstellung von Mock-Endpunkten.

Wenn abhängige Systeme (ERP oder externe APIs) instabil oder nicht verfügbar sind, simulieren Virtualisierungstools realistische Interaktionen. Open-Source-Lösungen wie WireMock oder Mountebank 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



Testdatenmanagement Um konsistente und compliance-konforme Tests zu gewährleisten:

- **Datenmaskierung:** Ersetzen von PII mit realistischen fiktiven Werten unter Beibehaltung der referentiellen Integrität.
- **Synthetische Daten:** Künstliche Datensätze für Spezialfälle und Randszenarien.
- **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.

1.3.2 2. Testarten und Abdeckung

- *Welche Testarten (z. B. Unit-, API-, Integrations-, E2E-, Load-, Performance-, Security-tests) sind erforderlich, um:*
  - *die funktionalen Anforderungen abzudecken?*
  - *nicht-funktionale Anforderungen wie Performance, Security, Verfügbarkeit und Datenintegrität zu prüfen?*
- *Wo und wie werden diese Tests innerhalb der CI/CD-Pipeline ausgeführt?*

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.

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.

1.3.3 3. Testeffizienz und Wartbarkeit

- Wie strukturieren Sie Tests, um gezielt auf Systemveränderungen (z. B. SAP-Upgrade) reagieren zu können?
- Wie nutzen Sie z. B. Impact Analysis, modulare Architekturen oder risikobasiertes Testen, um Wiederverwendbarkeit und Selektivität zu ermöglichen?

1.3.4 4. Reporting & Testtransparenz

- Wo und wie sollen Testergebnisse dokumentiert und ausgewertet werden (z. B. Dashboards, Logs, automatisierte Reports)?
- Wer sind die Stakeholder für das Reporting (Dev, QA, Ops, Management)?

1.3.5 5. Toolauswahl und Integration

- Welche Testtools (open source und/oder kommerziell) schlagen Sie für die Umsetzung vor für z. B.:
  - Testautomatisierung
  - Performance-Testing
  - Service-Virtualisierung
  - Testdatenmanagement
  - Reporting & Testmanagement

1.4 Teamarbeit & Rollenverteilung

Versucht in der Ausarbeitung die folgenden Rollen einzunehmen:

- QA-Architekt: übergreifendes Testkonzept & Architektur
- Testanalyst: Spezifikation von Testfällen und Daten
- Tool-Integrator: Toolauswahl & CI/CD-Verknüpfung

Die Aufteilung ist eine Empfehlung, aber keine Pflicht.

1.5 Abzugebende Materialien

- Schriftliches Konzept (PDF, max. 6 Seiten ohne Anhang)
- Architektur- oder Ablaufdiagramm(e) zur Teststrategie
- Tabelle mit empfohlener Tool-Landschaft

1.6 Vorgesehene Bearbeitungsdauer

Gesamt in etwa 5 Stunden pro Person.

1.7 Lernziele

- Eine effektive Teststrategie im Kontext von DevOps
- Den Einsatz moderner Testtools konzeptionell bewerten
- Risiken in komplexen Integrationslandschaften erkennen und adressieren
- Testgetriebene CI/CD-Prozesse planen und visualisieren

Literaturverzeichnis