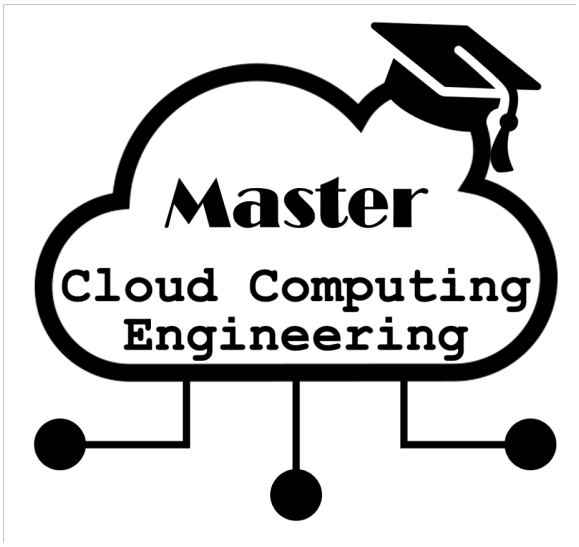

DevOps ILV - Aufgabenstellung 3

Hochschule Burgenland
Studiengang MCCE
Sommersemester 2025

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

19. Mai 2025

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

Inhaltsverzeichnis

1 Aufgabenstellung 3

1.1 Prozessorientierte Teststrategie

1.1.1 Prozessübersicht

Der zentrale Kundenprozess besteht aus folgenden Schritten:

1. User goes to website
2. Browse for products
3. Add camera to cart
4. Create account
5. Selects shipping
6. Adds photo storage service
7. Completes Checkout
8. Mail confirmation

1.1.2 Testebenen und -arten

Für jeden Prozessschritt werden folgende Testebenen angewendet:

Testebene	**Beschreibung**
Unit-/Component-Tests	Tests einzelner Komponenten und Funktionen, isoliert im Entwicklungsprozess
API-Tests	Prüfung der Schnittstellen zwischen Services und externen Systemen
Integrationstests	Test der Zusammenarbeit von Komponenten und externen Diensten
UI/UX-Tests	Testen der Benutzeroberfläche und Nutzungserfahrung
End-to-End-Tests	Simulation kompletter Benutzerflüsse über alle Prozessschritte hinweg
Performance-Tests	Prüfung des Systemverhaltens unter Last und Stress
Security-Tests	Prüfung auf Sicherheitsprobleme wie XSS, CSRF, etc.

1.1.3 Detaillierte Test-Definition je Prozessschritt

A. User goes to website

Service-Virtualisierung: GPT-Mock für schnelles Testen, simulierte Userprofile

Testarten	**Testfokus**
Unit/Component	Performance der Startseiten-Komponenten, Responsive Design, Werbeelemente
API	GPT-Integration für personalisierte Empfehlungen, Content-API-Abrufe
Nicht-funktional	XSS-Vulnerabilities, Content Security Policy, Ladezeit unter 2 Sekunden, WCAG 2.1 AA-Konformität

B. Browse for products

Service-Virtualisierung: GPT-Mock für Produktempfehlungen, simulierte Produktkatalog-API

Testarten	**Testfokus**
Unit/Component	Filterkomponenten-Verhalten, Suchfunktionalität, Produktkatalog-Darstellung
API	Produkt-API-Antwortzeiten und -struktur, Suchoptimierung und -relevanz
Nicht-funktional	Reaktionszeit der Suche ¡ 0.5 Sekunden, Verhalten bei gleichzeitiger Suche von 1000+ Nutzern

C. Add camera to cart

Service-Virtualisierung: SAP-Mock für Verfügbarkeitsprüfungen, simulierte Verzögerungen

Testarten	**Testfokus**
Unit	Warenkorb-Komponente, Preisberechnung
Integration	SAP-Anbindung zur Verfügbarkeitsprüfung
API	Warenkorb-API, SAP-API für Lagerbestandsprüfung
Nicht-funktional	Reaktionszeit der Warenkorb-Operationen, Konsistenz bei gleichzeitigen Zugriffen

D. Create account

Service-Virtualisierung: NetSuite-Mock für Account-Erstellung, HubSpot-Mock

Testarten	**Testfokus**
Unit	Registrierungsformular, Validierungslogik
Integration	NetSuite-Account-Erstellung, HubSpot-Anbindung
API	Account-Erstellungs-API, Authentifizierungs-Workflows
Nicht-funktional	Passwort-Policies, Account-Übernahme-Schutz, DSGVO-Konformität

E. Selects shipping

Service-Virtualisierung: SAP-Mock für Versandverifizierung, Simulation internationaler Versandoptionen

Testarten	**Testfokus**
Unit	Versandoptionen-Komponenten
Integration	SAP-Integration für Versandoptionen
API	Versandkosten-Berechnung-API, Lieferzeit-Prognose-API
Nicht-funktional	Reaktionszeiten, korrekte Versandoptionen bei verschiedenen Ländercodes

F. Adds photo storage service

Service-Virtualisierung: AWS-Service-Mock für Provisioning-Simulation

Testarten	**Testfokus**
Unit	Storage-Plan-Auswahl-Komponenten
Integration	AWS-Service-Provisioning
API	Storage-Service-API-Aufrufe, Pricing-API für additive Services
Nicht-funktional	Service-Aktivierungszeiten, Fehlerbehandlung bei AWS-Provisioning-Problemen

G. Completes Checkout

Service-Virtualisierung: Payment-Service-Mock, NetSuite-Mock für Order-Updates

Testarten	**Testfokus**
Unit	Bezahlprozess-Komponenten, Zusammenfassung
Integration	Integration mit NetSuite für Bestellaktualisierung
API	Checkout-API, Zahlungsverarbeitungs-API
Nicht-funktional	PCI-DSS-Konformität, Zahlungssicherheit, Performance, Transaktions-Rollback

H. Mail confirmation

Service-Virtualisierung: E-Mail-Service-Mock für Zustellungssimulation

Testarten	Testfokus
Unit	E-Mail-Template-Rendering
Integration	E-Mail-Versandsystem-Integration
API	E-Mail-API-Aufrufe, Bestätigungs-Tracking-API
Nicht-funktional	E-Mail-Zustellung-Rate > 99,5%, E-Mail-Versandzeit < 30 Sekunden

1.2 End-to-End-Testszenarien

1.2.1 Hauptszenarien

Standardablauf: Kompletter Kaufprozess

- **Beschreibung:** Vollständiger Durchlauf vom Website-Besuch bis zur E-Mail-Bestätigung
- **Testdaten:** Neuer Benutzer, verfügbares Kameraprodukt, Standard-Versand, Basic-Speicherpaket
- **Erwartetes Ergebnis:** Erfolgreicher Kaufabschluss, korrekte Bestellbestätigung per E-Mail
- **Automatisierungsgrad:** 90% automatisiert (Playwright/Cypress)
- **Testumgebung:** Staging mit teilweise virtualisierten Services

Szenario: Rückkehrender Kunde mit Login

- **Beschreibung:** Login eines Bestandskunden und Durchführung eines Kaufs mit gespeicherter Adresse
- **Testdaten:** Existierender Account, Produktvariante mit Zubehör, vorhandener Fotospeicher
- **Erwartetes Ergebnis:** Korrekte Nutzung gespeicherter Daten, erfolgreicher Checkout
- **Automatisierungsgrad:** 85% automatisiert
- **Testumgebung:** Staging mit Echtanbindung an HubSpot

Szenario: Foto-Kursbuchung

- **Beschreibung:** Buchung eines Fotografie-Kurses statt eines physischen Produkts
- **Testdaten:** Neuer Benutzer, Online-Kurs "DSLR-Fotografie für Einsteiger"
- **Erwartetes Ergebnis:** Erfolgreiche Kursbuchung, Aktivierungslink in der E-Mail
- **Automatisierungsgrad:** 80% automatisiert
- **Testumgebung:** Staging mit AWS-Integration

1.2.2 Fehlerszenario-Tests

Produkt während Checkout nicht mehr verfügbar

- **Beschreibung:** Simulation eines Lagerbestandsproblems während des Checkouts
- **Testdaten:** Produkt mit kritischem Lagerbestand, der während des Prozesses auf 0 sinkt
- **Erwartetes Ergebnis:** Nutzerfreundliche Fehlerbehandlung, alternative Produktvorschläge
- **Automatisierungsgrad:** 70% automatisiert, 30% manuelle Validierung
- **Testumgebung:** Staging mit manipuliertem SAP-Mock

Zahlungsabbruch und Wiederaufnahme

- **Beschreibung:** Test der Wiederaufnahme eines unterbrochenen Zahlungsprozesses
- **Testdaten:** Bestandskunde mit vollständigem Warenkorb, Zahlungsabbruch
- **Erwartetes Ergebnis:** Persistenz des Warenkorbs, erfolgreiche Wiederaufnahme, Benachrichtigung
- **Automatisierungsgrad:** 65% automatisiert
- **Testumgebung:** Staging mit Payment-Service-Mock

AWS-Service-Aktivierungsfehler

- **Beschreibung:** Test des Fehlerverhaltens bei AWS-Provisioning-Problemen
- **Testdaten:** Erfolgreich abgeschlossene Bestellung mit Fotospeicherdienst
- **Erwartetes Ergebnis:** Fehlerbenachrichtigung, automatisierter Wiederholungsversuch, Support-Ticket
- **Automatisierungsgrad:** 75% automatisiert
- **Testumgebung:** Staging mit fehlersimulierendem AWS-Mock

1.3 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.

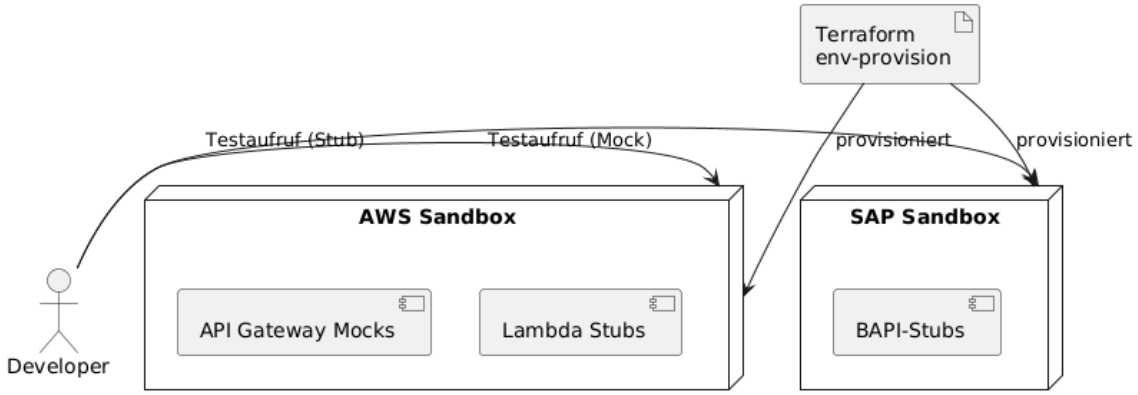

Abbildung 1: Deployment-Diagramm: Virtuelle Testumgebungen via IaC

Tabelle 2: Testumgebungen

Umgebung	**Hauptzweck**	**Virtualisierung**	**Testdaten**	**Ausführung**
Entwicklung	Entwicklertests, Komponententests	Alle Services virtualisiert	Synthetische Testdaten	Kontinuierlich
CI/CD-Pipeline	Unit/API-Tests, statische Codeanalyse	Alle Services virtualisiert	Synthetische Testdaten	Bei jedem Commit
Integration	Service-Integration, API-Tests	Kritische externe Services virtualisiert	Gemischte Testdaten	Täglich
Staging	E2E-Tests, UAT	Minimale Virtualisierung	Produktionsähnliche Daten	Bei Release-Kandidaten
Produktionsähnlich	Performance, Last, Sicherheit	Keine Virtualisierung	Vollständige Testdaten	Wöchentlich

1.3.1 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.

Die Virtualisierung abhängiger Services erfolgt phasengesteuert: Während der Build-Phase kommen WireMocks zum Einsatz, um Unit- und API-Tests in der Pipeline oder Entwicklungsumgebung zu unterstützen. In der IaC-Testumgebung emuliert AWS LocalStack Cloud-APIs, um Integrationstests effizient durchzuführen. Zusätzlich werden SAP-Schnittstellen bereits vor Testbeginn durch Parasoft Virtualize gestubbt - dies reduziert Latenzen bei der Testvorbereitung und erfüllt

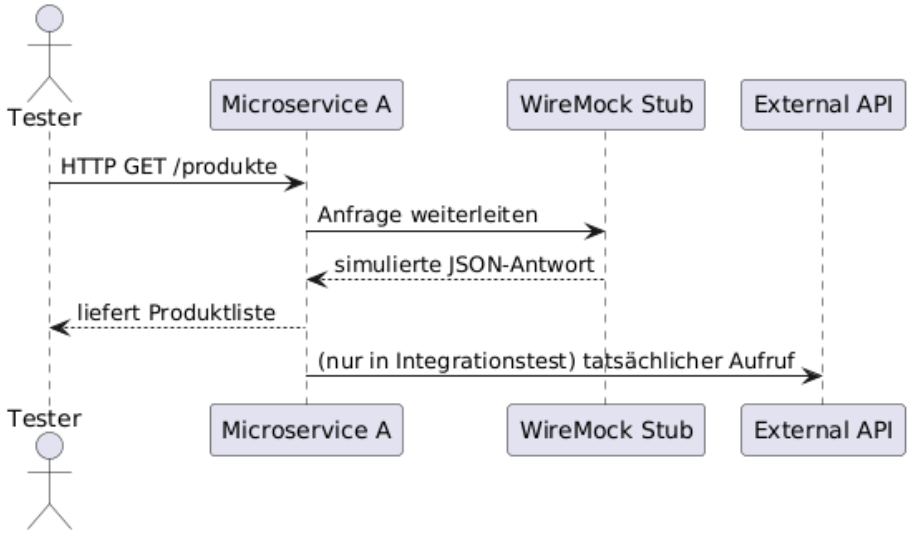

Abbildung 2: Sequenzdiagramm: Service-Virtualisierung mit WireMock

Complance-Anforderungen wie Datenmaskierung. Der gestaffelte Einsatz virtueller Komponenten sichert konsistente Testbedingungen, unabhängig von externen Systemabhängigkeiten.

Tabelle 3: Virtualisierungsansatz pro Prozessschritt

System	**Prozessschritt**	**Virtualisierungsansatz**	**Tool**	**Umgebung**
GPT	User goes to website	Vortrainierte Antworten für Produktsuche	WireMock	Entwicklung, CI/CD
SAP	Add camera to cart	Statische Verfügbarkeitsantworten	LocalStack	Entwicklung, Integration
SAP	Selects shipping	Konfigurierbare Versandoptionen	WireMock	Integration, Last-Tests
HubSpot	Create account	Service-Provisionierung	Mountebank	Entwicklung, Integration
NetSuite	Create account	Account-Erstellung	WireMock	Alle außer Prod
NetSuite	Completes Checkout	Bestellaktualisierung	WireMock	Integration, Staging
AWS	Adds photo storage	Service-Provisioning	LocalStack	Entwicklung, Integration

1.3.2 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.

Tabelle 4: Prozessspezifische Testdaten

Prozessschritt	**Testdaten-Anforderungen**	**Quelle**	**Bereitstellung**
User goes to website	User-Personas, Browser-Profile	Synthetisch	Fixture-Files
Browse for products	Produktkatalog mit Varianten	Prod-Kopie	TDM-Tool (Delphix)
Add camera to cart	Produktvarianten mit Lagerbestandsstufen	Synthetisch	API-Seeding
Create account	Benutzerprofile, E-Mail-Domains	Synthetisch	Faker.js + API
Selects shipping	Verschiedene Länder/Regionen	Prod-Kopie	CSV-Import
Adds photo storage	Storage-Pläne mit Preisen	Aktuelle Konfiguration	Configuration-as-Code
Completes Checkout	Zahlungsmethoden, Gutscheincodes	Synthetisch	API-Seeding
Mail confirmation	E-Mail-Templates, Sprachen	Aktuelle Version	Git-Repository

Testdaten-Governance Zur Sicherstellung von Datenqualität und Compliance:

- **Daten-Klassifizierung:** Kategorisierung von Testdaten nach Sensitivität und Verwendungszweck
- **Automatisierte DSGVO-Konformität:** Automatisches Maskieren von personenbezogenen Daten
- **Versionsverwaltung:** Testdaten-Snapshots mit Versionierung für Reproduzierbarkeit
- **Selbstbedienungsportal:** GUI für Tester zum Anfordern und Verwalten von Testdaten-Sets
- **Nutzungsverfolgung:** Logging und Monitoring der Testdatennutzung für Audits

1.4 Testarten und Abdeckung

1.4.1 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.

1.4.2 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.

1.4.3 CI/CD-Pipeline Integration

Tests sind in der Pipeline wie folgt integriert:

- **Build-Phase:** Unit-Tests (lokal) und API-Tests (gegen Mocks) laufen bei jedem Build.
- **Deployment-Phase:** Integrations- und Smoke-Tests, in IaC-Umgebung nach Deployment in Testumgebung (mit SAP/AWS-Virtualisierung).
- **Post-Deployment:** E2E-Tests in Staging-Umgebung (mit SAP/AWS-Virtualisierung).
- **Nachtests:** Performance- und Security-Tests in isolierten Cloud-Sandboxes.

Tabelle 5: Automatisierungsstrategie pro Prozessschritt

Prozessschritt	Automatisierungsgrad	Frameworks	CI/CD-Integration
User goes to website	90%	Jest, Lighthouse	Commit-Stage
Browse for products	85%	TestCafe, Cypress	Täglich
Add camera to cart	90%	Cypress, API-Tests	Commit-Stage
Create account	80%	Playwright, API-Tests	Täglich
Selects shipping	85%	Cypress, API-Tests	Täglich
Adds photo storage	75%	Selenium, API-Tests	Täglich
Completes Checkout	70%	Cypress, API-Tests	Release-Gate
Mail confirmation	95%	API-Tests, E-Mail-Tests	Release-Gate

1.4.4 Testausführungsorte

Die Tests werden an verschiedenen Ausführungsorten ausgeführt:

1.5 Testeffizienz und Wartbarkeit

1.5.1 Strukturierung der Tests für Systemveränderungen

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

- **Modulare Testarchitektur:** Tests sind nach Komponenten (Unit), Systemen (Integration) und E2E-Prozessen organisiert, um Änderungen in SAP oder Microservices isoliert zu validieren

Tabelle 6: Testausführungsorte und virtualisierte Services

Testtyp	Ausführungsort	Virtualisierte Services
Unit-Tests	Lokale Entwicklerumgebung / Build-Pipeline	WireMock (externe APIs) Lokale DB-Clones
API-/Integrationstests	IaC-basierte Testumgebung (AWS Sandbox)	AWS LocalStack (Cloud-Services) SAP-Mocks
End-to-End-Tests	Staging-Umgebung (Kubernetes-Namespace)	HubSpot-& GPT-Virtualisierung via API Gateway
Performance/Security	Dedizierte Lastumgebung (BlazeMeter/k6 Cloud)	Vollständig isolierte Sandbox-Accounts

- **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.

1.5.2 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.

1.6 Reporting & Testtransparenz

1.6.1 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.

1.6.2 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.

1.7 Toolauswahl und Integration

1.7.1 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

1.7.2 Performance-Testing

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

1.7.3 Service-Virtualisierung

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

1.7.4 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

1.7.5 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.

1.8 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.

1.9 Anhang

1.9.1 Tool-Landschaft Überblick

Tabelle 7: Tool-Landschaft Überblick mit Testebenen

Kategorie	Tools	Anwendungsbereich	Testebene
Testautomatisierung	Selenium, Playwright, Cypress, JUnit/TestNG, pytest, Cucumber, Postman/Newman	UI- und Funktionstest, API-Test	Unit/Component, System, E2E
Performance-Testing	Apache JMeter, Gatling, k6, BlazeMeter	Last- und Performance-Tests	System, End-to-End
Service-Virtualisierung	WireMock, Mountebank, AWS LocalStack, API Gateway Mocks, Parasoft Virtualize	Mock-Services für APIs und Systeme	Build-Phase, Integrationstests
Testdatenmanagement	Delphix, Informatica TDM, Redgate Data Generator, Faker-Bibliotheken	Datenmaskierung, Synthetische Daten	Alle Testebenen, Pipeline-übergreifend
Reporting & Testmanagement	Jenkins/GitLab CI, Allure Report, ReportPortal, Xray/Zephyr/TestRail, Grafana/ELK	Testdokumentation, Auswertung	Pipeline-übergreifend, Stakeholder-Dashboards

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

1.9.2 Testarchitektur und Komponentendiagramm

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.

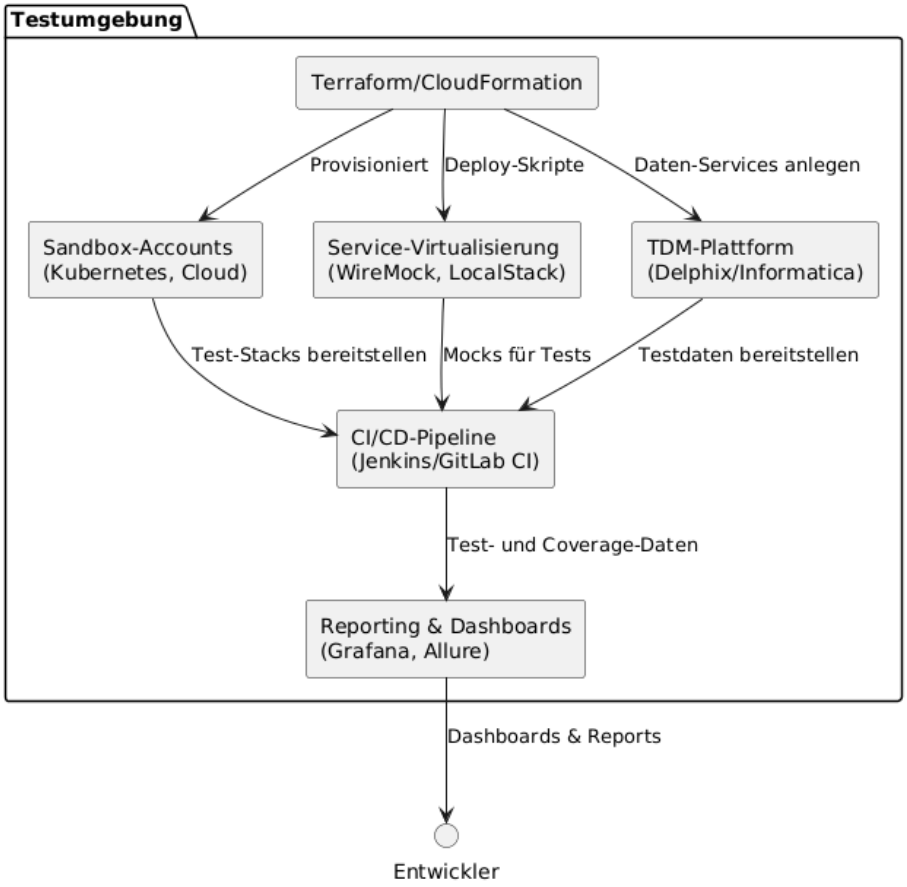

Abbildung 3: Architektur: Komponentenübersicht der Testumgebung

1.9.3 Ablaufflow der Continuous-Testing-Pipeline

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

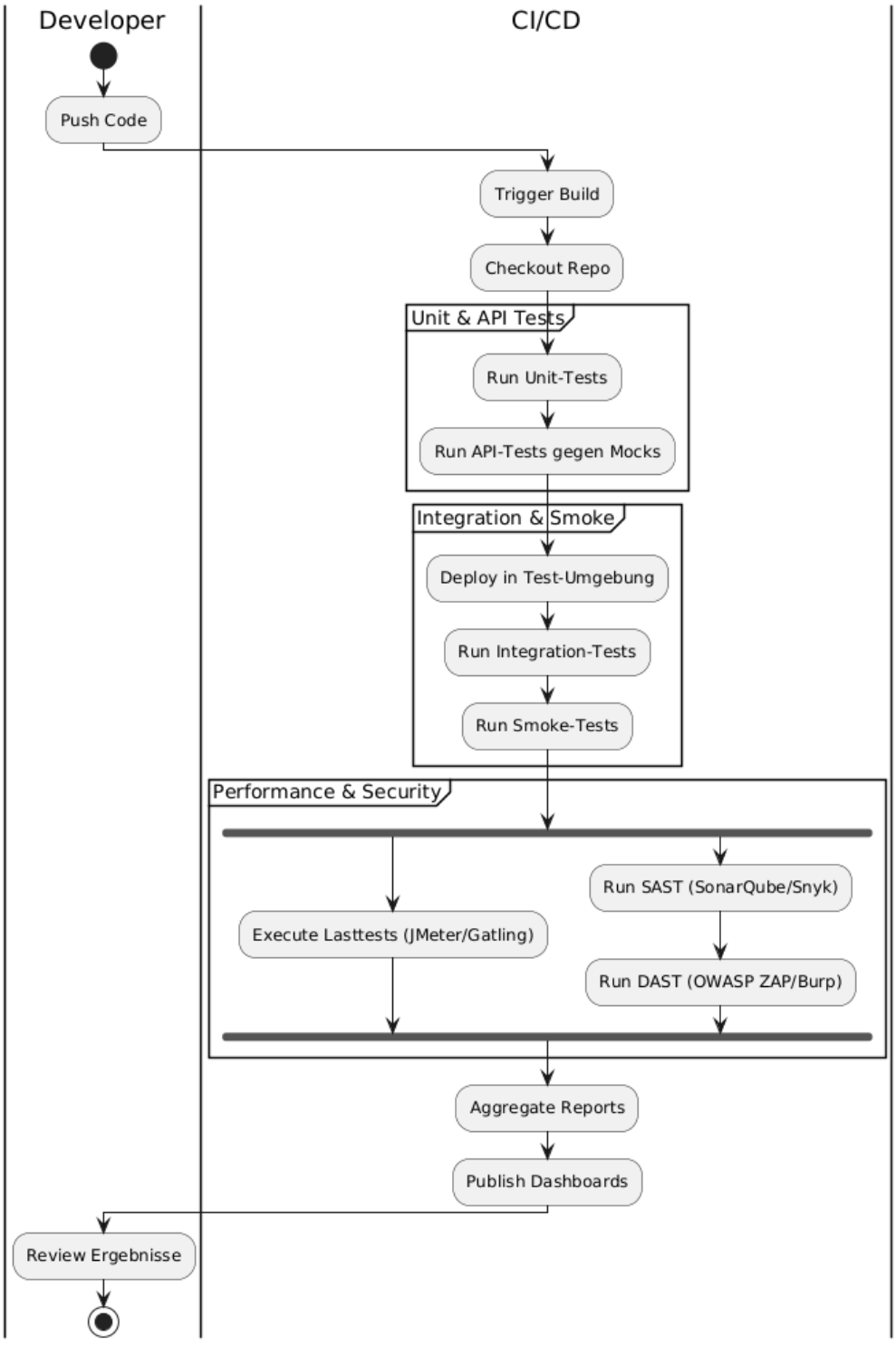

Abbildung 4: Ablaufflow: Continuous-Testing-Pipeline

1.9.4 Prozessorientierte Testarchitektur für Foto-Webshop

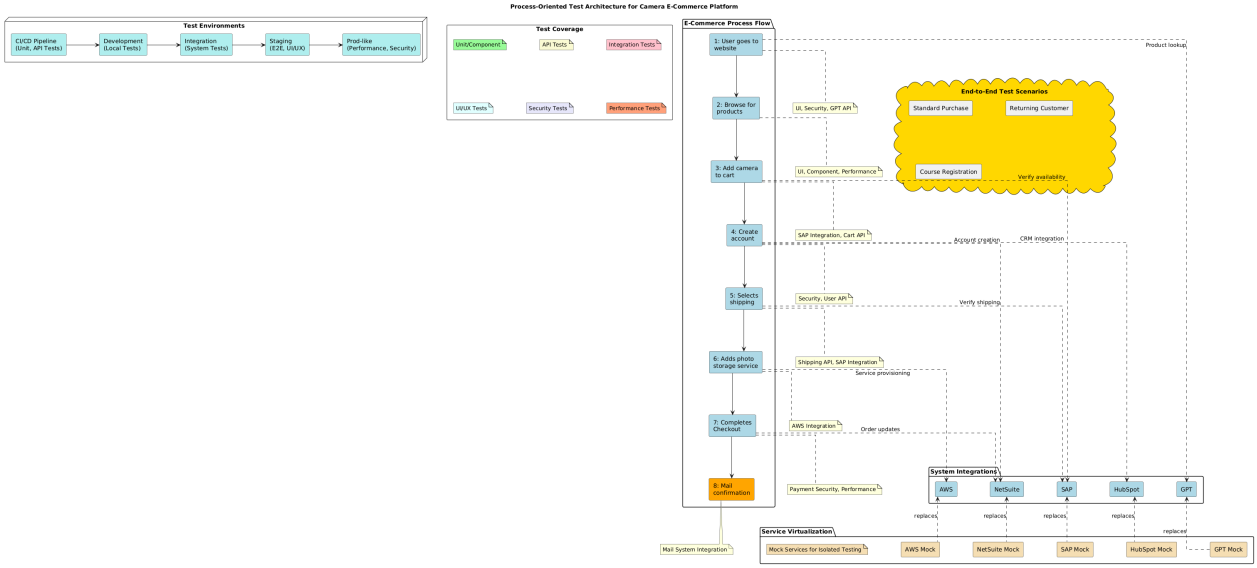

Abbildung 5: Prozessorientierte Testarchitektur

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.