Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik Software Engineering

Integrations- und Bereitstellungsautomatisierung von Cloud-Anwendungen für Composable-Enterprise-Architekturen im Kontext der SAP Business Technology Platform

Bachelorarbeit

Im Rahmen der Prüfung zum Bachelor of Science (B. Sc.)

Verfasser:	Rafael Martin
Kurs:	WI SE-B 2020
Dualer Partner:	SAP SE, Walldorf
Betreuer der Ausbildungsfirma:	Klaus Räwer
Wissenschaftlicher Betreuer:	Herr Ulrich Wolf
Abgabedatum:	08.05.2023

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit mit dem Thema:

Integrations- und Bereitstellungsautomatisierung von Cloud-Anwendungen für Composable-Enterprise-Architekturen im Kontext der SAP Business Technology Platform

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, 08.05.2023, _____

Rafael Martin

Inhaltsverzeichnis

Abkürzungsverzeichnis

XP	Extreme Programming
DevOps	Development & Operations
CI/CD	Cuntinuous Integration and Continuous Delivery
CI	Cuntinuous Integration
CD	Cuntinuous Delivery
DoD	Definition of Done
E2E-Tests	End-to-End-Tests

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

1.2 Zielsetzung und Abgrenzung

1.3 Aufbau der Arbeit

2 Grundlagen und Begriffserklärungen

2.1 Die Composable-Enterprise-Architektur

2.1.1 Begriffserklärung und Abgrenzung

2.1.2 Technologische Konzepte des Composable-Enterprises

2.2 Integration und Bereitstellung von Softwareentwicklungen

2.2.1 Agile und DevOps als moderne Anwendungsentwicklungskonzepte

Das Hauptaugenmerk eines Composable-Enterprises besteht darin, Resilienz und Flexibilität aufrechtzuerhalten. Damit soll sichergestellt werden, dass IT-Leistungen in einem sich stetig ändernden Umfeld schnell und risikoarm bereitgestellt werden. Das traditionelle Wasserfallmodell, welches eine sequenzielle Abfolge der Projektelemente *Anforderung*, *Design*, *Implementierung*, *Test* und *Betrieb* vorgibt, besitzt dabei signifikante Limitationen. Die in dieser Methodik detailliert durchgeführte Vorabplanung, kann in der Realität aufgrund unvorhersehbarer Externalitäten selten eingehalten werden. Auch die starre Abfolge der Projektphasen mindert insbesondere in fortgeschrittenen Zeitpunkten des Vorhabens den Spielraum für Anpassungsmöglichkeiten [8, S. 5]. Dies resultiert nicht nur einem Anstieg der Kosten, sondern führt ebenfalls dazu, dass IT-Projekte länger als geplant ausfallen [7, S. 41]. Als Reaktion haben sich innerhalb der Projektmanagementlandschaft zunehmend **agile Vorgehensmodelle** etabliert. Im Gegensatz zum Wasserfallmodell, welches eine umfassende Vorabplanung vorsieht, wird das Vorhaben in einer agilen Entwicklung in viele Teilprojekte, sog. *Sprints*, segmentiert (s. Abb. 1) [1, S. 87].

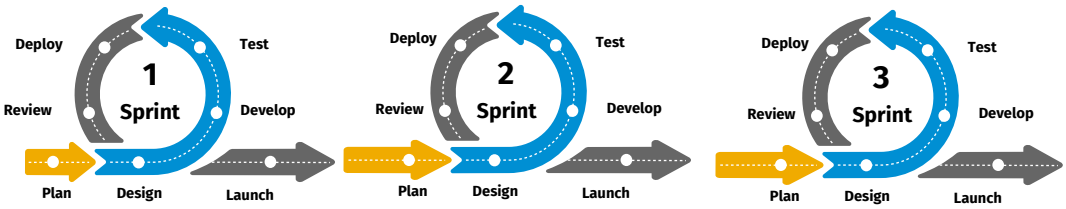

Abbildung 1: Exemplarische Abfolge eines agilen Entwicklungszykluses. In Anlehnung an K&C [10].

Sprints sind Durchläufe, welche i.d.R. einen Zeitraum von ein bis vier Wochen umfassen. Während dieses Abschnitts ist die Fertigstellung einer vor dem Teilprojekt definierten Aufgabenkontingente (*Sprint-Backlog*) vorgesehen. Nach Abschluss eines Sprints soll dabei ein potenziell an den Kunden auslieferbares Produkt zur Verfügung gestellt werden. Dies erlaubt eine schnelle Bereitstellung funktionsfähiger Software,

was neben der Erhöhung der Kundenzufriedenheit ebenfalls in einer Optimierung der Planungsprozesse resultiert. So kann das nach Ablauf eines Sprints an die Stakeholder ausgelieferte Artefakt als Feedback-Grundlage verwendet und im unmittelbaren Folge-Sprint eingearbeitet werden [10, S. 39]. Dieser Sprint-Review-Zyklus wird ausgeführt, bis alle Kundenbedürfnisse erfüllt sind und das Vorhaben als abgeschlossen gilt. Innerhalb der letzten Dekade haben sich diverse auf agilen Prinzipien basierenden Vorgehensmodelle, wie Scrum, Kanban oder Extreme Programming (XP) in der Softwareentwicklung etabliert. Obwohl einige dieser Methoden zur erfolgreichen Zusammenarbeit innerhalb der Entwicklungsteams beigetragen haben, bleibt das sog. *Problem der letzten Meile* bestehen [11]. Traditionell erfolgt eine funktionale Trennung der Entwickler- und IT-Betriebler-Teams. Das Problem der letzten Meile beschreibt dabei, dass aufgrund ausbleibender Kooperation der Entwicklungs- und Betriebsteams der Programmcode nicht auf die Produktivumgebung abgestimmt ist. Erkenntnisse aus der Praxis zeigen, dass solche organisatorischen Silos häufig in einer schlechten Softwarequalität und somit in einem geminderten Ertragspotenzial bzw. in einer Erhöhung der Betriebskosten resultieren [2, S. 1]. So geht aus der von McKinsey veröffentlichen Studie *The Business Value of Design 2019* hervor, dass durchschnittlich 80 Prozent des Unternehmens-IT-Budgets zur Erhaltung des Status quo, also zum Betrieb bestehender Anwendungen verwendet wird. Stattdessen fordert das Beratungshaus eine Rationalisierung der Bereitstellung von Software, um finanzielle Mittel für wertschöpfende Investitionen zu maximieren [13]. Abhilfe schaffen kann das in der Literatur als **Development & Operations (DevOps)** bekannte Aufbrechen organisatorischer Silos zwischen Entwicklung und dem IT-Betrieb [2, S. 1]. Dabei stellt DevOps keine neue Erfindung dar. Stattdessen werden einzelne bereits bewährte Werkzeuge, Praktiken und Methoden, wie z.B. die agile Softwareentwicklung, zu einem umfassenden Rahmenwerk konsolidiert. DevOps zielt dabei auf eine Optimierung des gesamten Applikationslebenszykluses, von Planung bis Bereitstellung der Software, ab. Neben der Bereitstellung von IT-Services umfasst DevOps ebenfalls Aktivitäten zu IT-Sicherheit, Compliance und Risikomanagement. Prägnant zusammenfassen lässt sich das DevOps-Konzept durch das Akronym

CAMS: *Culture (Kultur)*, *Automation (Automatisierung)*, *Measurement (Messung)* und *Sharing (Teilen)* [2, S. 5]. Dabei gilt *Kultur* als das wohl wesentlichste DevOps-Erfolgselement. Diese Norm bezweckt eine Kollaborationsmentalität, welche sich über alle Ebenen eines Unternehmens erstreckt. Operative Entscheidungen sollen dabei auf die Fachebenen herunter delegiert werden, welche aufgrund ihrer spezifischen Expertise am geeigneten sind, Dispositionen zu verabschieden [2, S. 5]. Eine *Automatisierung* der Softwarebereitstellungsprozesse ermöglicht, sich wiederholende manuelle Arbeit zu eliminieren. Dies kann ebenfalls zur Rationalisierung und damit zur Senkung der IT-Betriebskosten beitragen. Der dabei erzielte Einfluss wird anhand verschiedener DevOps-Kennzahlen bemessen (*Messung*). Neben der Systemverfügbarkeit und der Instandsetzungszeit ist insbesondere der *Time-to-Market* eine signifikante Metrik [2, S. 7].

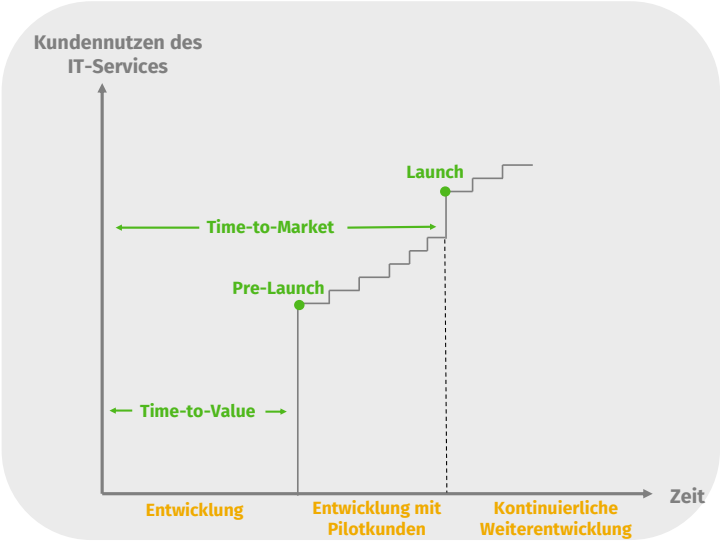

Abbildung 2: Zeitliche Darstellung des Kundennutzens von IT-Services. In Anlehnung an Halstenberg [2, S. 9].

Der Time-to-Market beschreibt die Zeitspanne zwischen Entwicklungsentstehungsprozess und der Markteinführung von IT-Services [6, S. 141]. Auch der *Time-to-Value* erhält zunehmend Bedeutung in der Softwareentwicklung. Im Gegensatz zum Time-to-Market wird hier nicht die Zeit bis zur Komplett-Einführung, sondern bis zum ersten Kundennutzen bemessen. Obwohl der im Time-to-Value bereitgestellte IT-Service möglicherweise Verbesserungspotenzial besitzt, überwiegt der mit

der initialen Auslieferung herbeigeführte Mehrwert. Dabei ermöglicht eine solche Früheinführung einen Vorsprung gegenüber Konkurrenten. So ist es dem Softwareunternehmen bereits gelungen, erste Kunden zu akquirieren, deren Input und Feedback möglichst rasch erfasst und verarbeitet werden kann [2, S. 9]. Softwareunternehmen können IT-Services ab dem Pre-Launch somit sukzessive und ressourcenoptimiert unter Zusammenarbeit mit den Pilotkunden erweitern. Auch Adam Caplan, leitender Strategieberater bei Salesforce, empfiehlt angesichts der bei Softwareintegration entstehenden Komplexität, Software schnellst möglichst in Produktivumgebungen zu testen [6]. Aus diesen Erfahrungen sollen Best-Practises entwickelt werden, welche innerhalb von Teams und organisationsübergreifend weitergegeben werden (*Teilen*) [2, S. 7].

2.2.2 Pipelines zur Integrations- und Bereitstellungsautomatisierung

DevOps beschreibt eine Philosophie zur Förderung der Zusammenarbeit zwischen Entwicklungs- und Betriebsteams. Ein integraler Bestandteil des DevOps-Rahmenwerks ist *Cuntinuous Integration and Continuous Delivery (CI/CD)*. CI/CD ist eine Softwareentwicklungsmethode, welche zur Verbesserung der Qualität bzw. zur Senkung der Entwicklungszeit von IT-Services beitragen soll. Abhilfe schaffen soll dabei eine Pipeline, welche alle Schritte von Code-Integration bis Bereitstellung der Software automatisiert. Hauptaugenmerk liegt dabei auf einer zuverlässigen und kontinuierlichen Bereitstellung von Software [9, S. 471].

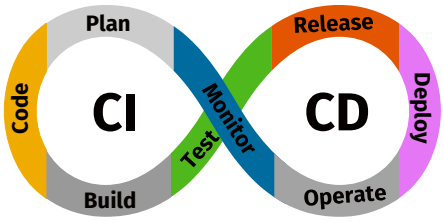

Abbildung 3: Aktivitäten im CI/CD-Prozess. In Anlehnung an Synopsys [16].

Alle in diesem Prozess anfallenden Aktivitäten sind dabei dem CI/CD-Zyklus zu entnehmen (s. Abb. 3). Der CI-Prozess (Continuous-Integration-Prozess) bezweckt, dass lokale Quellcodeänderungen in kurzen Intervallen und so schnell wie möglich in eine zentrale Codebasis geladen werden. Das frühzeitige Integrieren von Code soll dabei zu einer unmittelbaren und zuverlässigen Fehlererkennung innerhalb des Entwicklungsvorhabens beitragen [9, S. 471]. Der erste Schritt des CI-Prozesses umfasst die Planung zu entwickelnder Services (*Plan*: s. Abb. 3). Dabei soll festgestellt werden, welche Anforderungen eine Lösung besitzt bzw. welche Softwarearchitekturen sowie Sicherheitsmaßnahmen implementiert werden sollten. Um sicherzustellen, dass die in der Planung entworfene Anwendungsarchitektur auf das Design des Produktivsystems abgestimmt ist, sollte zu jedem Zeitpunkt das Know-how der Betriebsteams einbezogen werden [2, S. 16]. Nach erfolgreichem Entwurf zu implementierender Anwendungsfeatures beginnt die Entwicklung der IT-Services (*Code*: s. Abb. 3). Arbeiten hierbei mehrere Entwickler parallel an demselben IT-Service, wird der entsprechende Quellcode in sog. *Repositories* wie Github oder Bitbucket ausgelagert. Ein Repository stellt dabei einen zentralen Speicherort dar, welcher das Verfolgen sowie Überprüfen von Änderungen und ein paralleles bzw. konkurrierendes Arbeiten an einer gemeinsamen Codebasis ermöglicht [5, S. 31]. Der in dem Repository archivierte Hauptzweig (*Master-Branch*) stellt dabei eine aktuelle und funktionsfähige Version des Codes dar. Dieser mit verschiedenen Validierungsprozessen überprüfte Code, kann dabei zu jeder Zeit in der Produktionsumgebung bereitgestellt werden. (s. Abb. 4). Im Sinne der agilen Entwicklung werden dabei große Softwareanforderungen, sog. *Epics*, in kleine Features segmentiert, welche in separate Feature-Branches ausgelagert werden. Diese sind unabhängige Kopien des Hauptzweiges, in welcher ein Entwickler Änderungen vornehmen kann, ohne Konflikte in der gemeinsamen Codebasis zu verursachen. Nach Fertigstellung der Funktionalitäten sollte der um die Features erweiterte Quellcode so schnell wie möglich in den Hauptzweig integriert werden. So wird sichergestellt, dass der Code stets stabil, also funktionsfähig ist und keine Konflikte mit dem aktuellen Code des Hauptzweiges aufweist [5, S. 169]. Dabei wird eine Validierung des Codes gemäß der *Definition of Done (DoD)* forciert. Die DoD ist eine in der Planungsphase festgelegte Anforde-

rungsspezifikation, deren Erfüllung als notwendige Voraussetzung für den Abschluss eines Features gilt. Im Rahmen dieser Norm sind Entwickler dazu angehalten, für jedes implementierte Feature einen der DoD entsprechenden Test zu entwerfen.

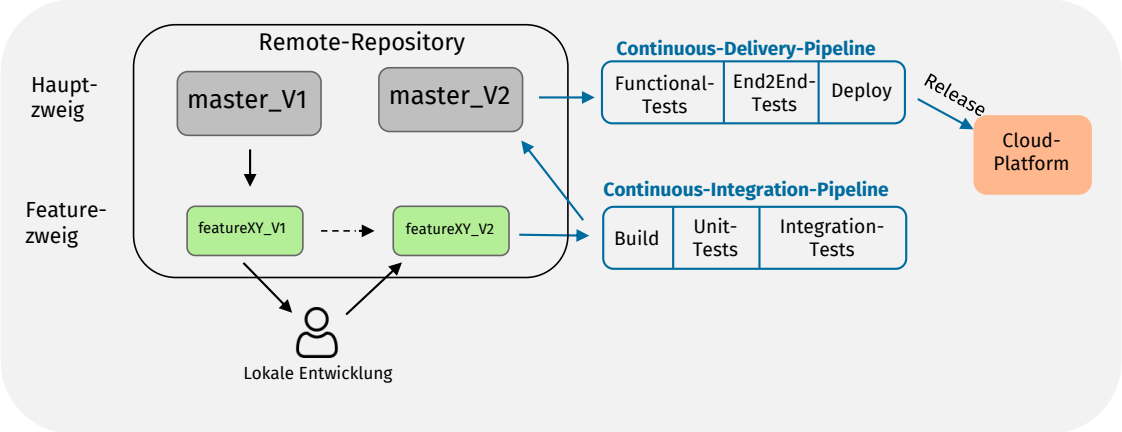

Abbildung 4: Versionskontrollsysteme zur Verwaltung von Quellcode. Eigene Darstellung.

Die Einbindung des Feature-Branchs in den Hauptzweig resultiert i.d.R. in einem unmittelbaren Start des *CI/CD-Pipeline-Prozesses*. Bei der CI/CD-Pipeline handelt es sich dabei um eine vom Repository unabhängige Recheninstanz, welche auf einer virtuellen Maschine oder in einer containerisierten Computing-Umgebung betrieben wird. Verwaltet wird diese Pipeline entweder von dem Unternehmen selbst (*On-Premise*) oder auf einer externen Cloud-Infrastrukur wie AWS, Azure oder Google Cloud [4, Kap. 1.2]. Im ersten Schritt des Pipeline-Prozesses wird die Applikationen zu einem ausführbaren Programm kompiliert (*Artefakt*) anhand welchem verschiedene Tests ausgeführt werden (*Build*: s. Abb. 3). Dafür können je nach Programmiersprache verschiedene Build-Tools, wie Maven für Java oder NPM für Javascript verwendet werden [4, Kap. 7.1]. Die dabei durchgeführte Validierung, auch *Smoke-Test*, soll sicherstellen, dass zu jeder Zeit ein rudimentär getesteter Code bereitsteht und grundlegende Funktionalitäten sowie Schnittstellen erwartungsgemäß ausgeführt werden [2, S. 19]. Der in dem Entwicklungszweig bereitgestellte Code wird dabei überwiegend anhand schnell durchführbarer Tests überprüft. Die Abwicklung solcher ressourcenschonender Validierungen hat dabei insbesondere zwei

Ursachen. Das Hauptaugenmerk von CI liegt insbesondere in der Realisierung einer hohen Code-Integrationsfrequenz. Aufwendige und langsame Tests würden Entwickler somit hemmen, den Code häufig in dem zentralen Repository bereitzustellen. Des Weiteren soll sichergestellt werden, dass der gesamte lokale Code nicht erst unmittelbar vor Release zusammengeführt wird (*Merge Day*) [15]. Stattdessen sollte der Entwickler ein zeitnahes Feedback auf seine Erweiterungen erhalten, um Fehler und Konflikte so schnell wie möglich entdecken und beheben zu können. Die in der CI-Pipeline abgewickelten Validierungen umfassen i.d.R. *Unit-* sowie *Integration-Tests* [4, Kap. 1.2].

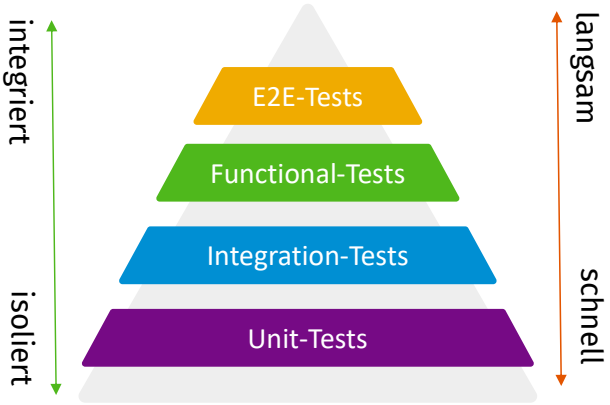

Abbildung 5: Hierarschische Darstellung von Softwaretests.
In Anlehnung an Paspelava [14].

Unit-Tests befinden sich dabei auf unterster Hierarchieebene der Test-Pyramide (s. Abb. 5). Somit besitzen diese eine kurze Ausführungsdauer, werden jedoch ausschließlich in einer isolierten Testumgebung abgewickelt. Mit Unit-Tests wird die funktionale Korrektheit kleinster Einheiten, wie z.B. Methoden einer Klasse, überprüft. Der Zweck der Unit-Tests besteht dabei in einer von externen Einflüssen und Daten unabhängigen Überprüfung der einzelnen Komponenten [3, Kap. 2]. Um bei der Bereitstellung neuer Funktionalitäten ebenfalls das Zusammenspiel verschiedener Komponenten zu überprüfen, werden *Integration-Tests* durchgeführt. Bei diesen Tests können Aspekte, wie der Austausch eines Nachrichtenmodells zweier Web-

Services oder das Response-Objekt einer Datenbankabfrage untersucht werden [3, Kap. 2]. Nachdem einzelne Funktionalitäten entwickelt und alle Tests erfolgreich absolviert wurden, werden die validierten Änderungen im Hauptzweig zusammengeführt. Mit diesem Prozessschritt beginnt der *Continuous-Delivery-Workflow (CD-Workflow)*. Während CI den Prozess der kontinuierlichen Integration des Quellcodes in das zentrale Repository verwaltet, steuert der CD-Workflow die Automatisierung der Anwendungsbereitstellung. Applikationen sollen somit ohne große Verzögerungen in die Produktivumgebung und somit zum Kunden ausgeliefert werden. Im Sinne des DevOps-Rahmenwerkes wird der CD-Prozess automatisch und unmittelbar nach Ablauf aller CI-Aktivitäten angestoßen. In der Praxis wird hierbei jedoch häufig ein manueller Schritt zwischengeschalten [2, S. 20]. Damit soll sichergestellt werden, dass das Ausrollen der Anwendung erst nach Überprüfung und Genehmigung aller Entwicklungs-Stakeholder beginnt. Im ersten Schritt des CD-Prozesses wird das in die Produktivumgebung bereitzustellende Artefakt über die Deployment-Pipeline in eine *Staging-Area* geladen. Bei der Staging-Area handelt es sich dabei um ein System, welches zwischen Entwicklungs- und Produktivumgebung liegt. Die Staging-System-Konfigurationen werden dabei so angelegt, dass diese der Produktionsumgebung möglichst ähnlich sind [4, Kap. 1.3]. Neben den Datenbanken werden hierbei ebenfalls Serverkonfigurationen, wie Firewall- oder Netzwerkeinstellungen von dem Produktivsystem übernommen. Somit soll sichergestellt werden, dass eine neue Anwendungsversion unter produktions-ähnlichen Bedingungen getestet wird. Analog zum CI-Prozess werden innerhalb des CD-Workflows ebenfalls Unit- und Integration-Tests abgewickelt. Diese sind i.d.R. deutlich rechenintensiver und besitzen längere Ausführungszeiten. Somit werden im CD-Prozess essenzielle, jedoch während des Entwicklungsworkflows zu aufwendige Validierungen durchgeführt [2, S. 20]. In der Staging Area werden unterdessen auch in der Test-Pyramide (s. Abb. 5) höher positionierte, also rechenintensivere Tests ausgeführt [3, Kap. 2]. Dazu gehören *Functional-Tests*. Mit diesen werden die in der Planungsphase festgelegten Anforderungen bzw. Funktionen der Anwendung überprüft. So kann z.B. evaluiert werden, ob bei Eingabe einer Benutzer-Passwort-Kennung ein korrekter Autorisierungsto-

ken übergeben wurde. Genau wie bei Integration-Tests wird während Functional-Tests das Zusammenspiel verschiedener Komponenten überprüft. Bei Integration-Tests wird dabei jedoch lediglich die generelle Durchführbarkeit einer Kommunikation verschiedener Komponenten auf Quellcode bzw. Datenbankebene überprüft. Mit Functional-Tests wird darüber hinaus die nach Übermittlung und Prozessierung der verschiedenen Komponenten generierte Ausgabe auf Ebene des Gesamtsystems validiert. Ebenfalls während des CD-Prozesses ausgeführte Validierungen sind *End-to-End-Tests (E2E-Tests)*. Mit diesen soll sichergestellt werden, dass die Anforderungen aller Stakeholder erfüllt werden. Hierbei wird ein vollständiges Anwenderszenario von Anfang bis Ende getestet. Dieses kann im Kontext eines E-Commerce-Webshops etwa das Anmelden mit Benutzername, das Suchen eines Produktes und das Abschließen einer Bestellung umfassen [12]. Nachdem alle Unit-, Integration-, sowie Functional-Tests erfolgreich absolviert wurden, werden i.d.R. verschiedene Codeanalysen angestoßen. Hierbei werden Metriken, wie die prozentuale Testabdeckung oder Schwachstellen verwendeter Code-Patterns überprüft. Nach Durchführung der Codeanalysen wird das überprüfte Artefakt auf die Cloud-Plattform geladen (*Deploy*: s. Abb. 3). Je nach Bereitstellungsstrategie (s. 2.2.3), wird die Anwendung dann unmittelbar oder erst nach weiteren Überprüfungen für den Kunden zugänglich gemacht. Der letzte Schritt des CD-Workflows umfasst die Laufzeitüberwachung der inbetriebgenommenen Anwendung (*Monitoring*: s. Abb. 3). So soll eine ordnungsgemäße Ausführung der Anwendung in der Produktionsumgebung sichergestellt werden. Wichtige Überwachungselemente sind dabei Infrastruktur- sowie Anwendungs-Monitoring. Beim Infrastruktur-Monitoring werden Metriken wie CPU-, Speicher- und Netzwerklast der Server bzw. Datenbanken untersucht. Das Anwendungs-Monitoring umfasst dabei die Überwachung der Funktionalitäten und der Applikation selbst. Hierbei werden Informationen wie Anfragen pro Sekunden, die Anzahl der Benutzer oder die in Log-Dateien gesammelte Fehlercodes analysiert [2, S. 21].

2.2.3 Strategien zur Bereitstellung von Neuentwicklungen

Nachdem das Artefakt auf der virtuellen Maschine einer Cloud-Instanz installiert und gestartet wurde, entscheiden verschiedene Strategien über die Inbetriebnahme der neuen Softwareversion. Anhand dieser wird festgelegt, mit welcher Methode und zu welchem Zeitpunkt Nutzeranfragen von der aktuellen auf die neue Anwendungsinstanz umgeleitet werden.

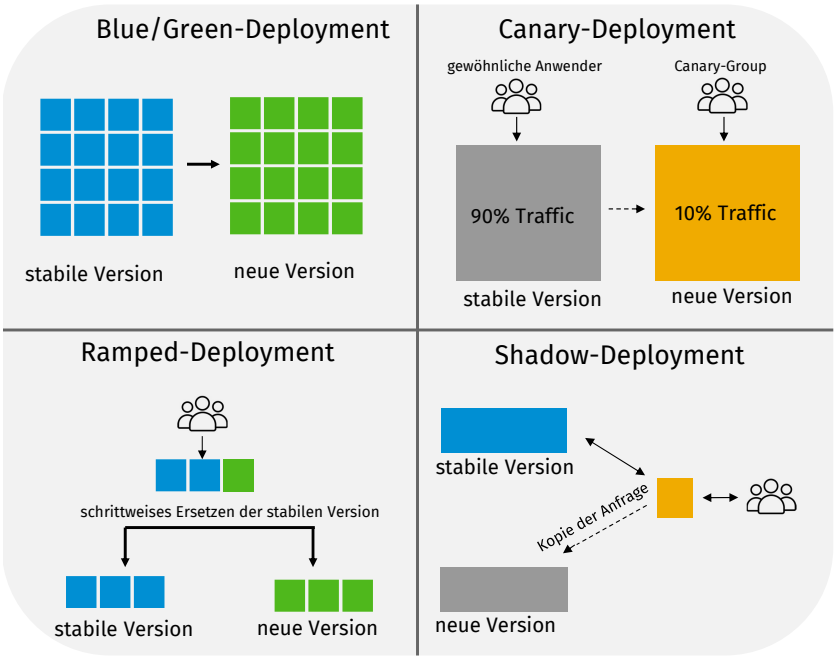

Abbildung 6: Strategien zur Bereitstellung von Software.
In Anlehnung an Ugochi [17].

Eine häufig verwendete Deployment-Strategie ist dabei das *Blue/Green-Deployment*. Hierbei wird neben der stabilen aktuellen Anwendung (*Blaue Version*) ebenfalls eine Instanz der neuen Anwendung (*Grüne Version*) betrieben. Nutzeranfragen werden dabei von dem Lastenverteilungsservice (*Load-Balancer*) erst nach Validierung aller Tests umgeschalten. Dazu gehören neben den in der CI/CD-Pipeline definierten Tests ebenfalls Überprüfungen der Qualitätssicherung. Diese umfassen manuelle Tests, in welchen Funktionen, Benutzeroberfläche sowie die Anwenderfreundlichkeit überprüft werden [17]. Im Gegensatz zum Blue/Green-Deployment, bei welchem eine neue Version simultan für die gesamte Nutzerbasis zur Verfügung gestellt

wird, gewährleistet das *Canary-Deployment* eine restriktivere Nutzlastumleitung. Hierfür wird die neue Anwendungsversion vorerst einer überschaubaren Nutzeranzahl (*Canary-Gruppe*) bereitgestellt. Dabei sollte die zusammengestellte Canary-Gruppe die Gesamtnutzerbasis möglichst gut repräsentieren. Anhand des Canary-Traffics soll der fehlerfreie Betrieb neuer Anwendungen überprüft und ggf. Anpassungen vorgenommen werden, bevor diese der gesamten Nutzerbasis zur Verfügung gestellt werden [17]. Für Anwendungen auf einer kritischen IT-Infrastruktur wird i.d.R. die *Ramped-Deployment-Strategie* verwendet. Diese ermöglicht eine präzise Kontrolle horizontal skalierter Services. Die in der horizontalen Skalierung abgewickelte Replizierung von Diensten sorgt für eine höhere Ausfallsicherheit der Anwendungen. Die neue Softwareversion wird während des Ramped-Deployment-Prozesses schrittweise auf die horizontalen Instanzen ausgerollt. Dabei werden die ersten aktualisierten Instanzen lediglich für bestimmte Anwender, eine sog. *Ramped-Gruppe*, bereitgestellt. Dabei soll das von dieser Anwendergruppe zur Verfügung gestellte Feedback während zukünftiger Planungsprozesse berücksichtigt werden [17]. Eine aufwendigere, jedoch risikoärmere Bereitstellungsstrategie stellt das *Shadow-Deployment* dar. Dabei wird neben der Instanz der aktuellen Version ebenfalls ein sog. *Shadow-Model* auf der Infrastruktur betrieben. Das Shadow-Model verwaltet die neue Version der Anwendung, kann jedoch nicht unmittelbar von den Nutzern aufgerufen werden. Diese Instanz stellt ein hinter der stabilen Version gelagertes Schattenmodell dar. Benutzeranfragen werden von dem Load-Balancer stets auf die aktuelle Version der Instanz weitergeleitet, verarbeitet und beantwortet. Gleichzeitig wird eine Kopie dieser Anfrage an das Shadow-Model weitergeleitet und von diesem prozessiert. Die Shadow-Modell-Verarbeitung des in der Produktionsumgebung abgewickelten Netzwerkverkehrs ermöglicht den Entwicklern somit eine anwendungsbezogene Überprüfung entwickelter Features [17].

3 Methodische Vorgehensweise

3.1 Prototypische Implementierung der Integrations- und Bereitstellungs-Pipelines

3.2 Evaluation der Integrations- und Bereitstellungs-Pipelines unter Anwendung des Analytischen Hierarchieprozesses

3.3 Semistrukturierte Leitfadeninterviews

4 Anwendung der Methodik auf die theoretischen Grundlagen

4.1 Prototypische Implementierung der Integrations- und Bereitstellungs-Pipelines

4.2 Evaluation der Integrations- und Bereitstellungs-Pipelines unter Anwendung des Analytischen Hierarchieprozesses

4.3 Entwicklung einer ganzheitlichen Bereitstellungsstrategie

5 Schlussbetrachtung

5.1 Fazit und kritische Reflexion

5.2 Ausblick

Literatur

Print-Quellen

[1] Joachim Goll und Daniel Hommel. *Mit Scrum zum gewünschten System*. ger. Wiesbaden: Springer Vieweg, 2015. 185 S. ISBN: 9783658107208. DOI: 10.1007/978-3-658-10721-5.

[2] Jürgen Halstenberg. *DevOps. Ein Überblick*. ger. Unter Mitarb. von Bernd Pfitzinger und Thomas Jestädt. Essentials Ser. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2020. 159 S. ISBN: 9783658314057. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6380828.

[3] Brian Hambling und Brian, Hrsg. *Software testing. An ISTQB-BCS certified tester foundation guide*. eng. Unter Mitarb. von Brian Hambling. 3rd ed. Hambling, Brian (MitwirkendeR) Brian, (author.) Hambling, Brian, (editor.) London, England: BCS Learning & Development Limited, 2015. 11 S. ISBN: 9781780173016. URL: https://learning.oreilly.com/library/view/-/9781780172996/?ar.

[4] Mohamed Labouardy. *Pipeline as code. Continuous delivery with Jenkins, Kubernetes, and Terraform*. eng. Shelter Island, New York: Manning Publications Co, 2021. 1333 S. ISBN: 9781638350378. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6785307.

[5] Jon Loeliger und Matthew McCullough. *Version control with git. Powerful tools and techniques for collaborative software development*. en. 2nd ed. Sebastopol, CA.: O'Reilly, 2012. 434 S. ISBN: 9781449345051.

[6] Joseph T. Vesey. „Time-to-market: Put speed in product development". In: *Industrial Marketing Management* 21.2 (1992). PII: 001985019290010Q, S. 151–158. ISSN: 0019-8501. DOI: 10.1016/0019-8501(92)90010-Q. URL: https://www.sciencedirect.com/science/article/pii/001985019290010q.

[7] Wolfgang Vieweg. „Agiles (Projekt-)Management“. de. In: *Management in Komplexität und Unsicherheit*. Springer, Wiesbaden, 2015, S. 41–42. DOI: 10.1007/978-3-658-08250-5_11. URL: https://link.springer.com/chapter/10.1007/978-3-658-08250-5_11.

[8] Alberto Vivenzio, Hrsg. *Testmanagement Bei SAP-Projekten. Erfolgreich Planen * Steuern * Reporten Bei der Einführung Von SAP-Banking*. ger. Unter Mitarb. von Domenico Vivenzio. 1st ed. Vivenzio, Alberto (VerfasserIn) Vivenzio, Domenico (MitwirkendeR). Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2013. 1175 S. ISBN: 978-3-8348-1623-8. DOI: 10.1007/978-3-8348-2142-3.

[9] Fiorella Zampetti u. a. „CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study“. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) (Luxembourg). IEEE, 9/27/2021 - 10/1/2021, S. 471–482. ISBN: 978-1-6654-2882-8. DOI: 10.1109/ICSME52107.2021.00048.

Online-Quellen

[10] John Adam. *Was ist agile Softwareentwicklung?* Hrsg. von K&C. 2021. URL: https://kruschecompany.com/de/agile-softwareentwicklung/ (besucht am 05. 03. 2023).

[11] Shweta Bhanda und Abby Taylor. *The Evolution from Agile to DevOps to Continuous Delivery — Qentelli*. Hrsg. von Qentelli. 2023-03-05. URL: https://www.qentelli.com/thought-leadership/insights/evolution-agile-devops-continuous-delivery (besucht am 05. 03. 2023).

[12] Shreya Bose. *What is End To End Testing?* BrowserStack. 2023-02-20. URL: https://www.browserstack.com/guide/end-to-end-testing (besucht am 08. 03. 2023).

[13] McKinsey, Hrsg. *The business value of design*. 2019. URL: https://www.mckinsey.com/capabilities/mckinsey-design/our-insights/the-business-value-of-design (besucht am 05. 03. 2023).

[14] Darya Paspelava. *What is Unit Testing in Software. Why Unit Testing is Important*. Hrsg. von Exposit. 2021. URL: https://www.exposit.com/blog/what-unit-testing-software-testing-and-why-it-important/ (besucht am 08. 03. 2023).

[15] Red Hat, Hrsg. *Was ist CI/CD? Konzepte und CI/CD Tools im Überblick*. 2023-03-08. URL: https://www.redhat.com/de/topics/devops/what-is-ci-cd (besucht am 08. 03. 2023).

[16] Synopsys, Hrsg. *What Is CI/CD and How Does It Work?* 2023-02-01. URL: https://www.synopsys.com/glossary/what-is-cicd.html (besucht am 08. 03. 2023).

[17] Ukpai Ugochi. *Deployment Strategies: 6 Explained in Depth*. Hrsg. von Plutora. 2022. URL: https://www.plutora.com/blog/deployment-strategies-6-explained-in-depth (besucht am 08. 03. 2023).

Anhang