Abschlussprüfung Winter 2024

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Automatisierte Onboarding Prozesse

Automatisierte Onboarding- und Ressourcenmanagement-Prozesse bei TUI Group

Abgabetermin: Hannover, den 18.11.2024

Prüfungsbewerber:
Paul Glesmann
Schopenhauerstraße 15
30625 Hannover

Ausbildungsbetrieb:
TUI InfoTec GmbH
Karl-Wichert-Allee 23
30627 Hannover

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Microsoft Teams Tool für Teamkommunikation.

CI Continuous Integration – regelmäßige Codeintegration.

CD Continuous Deployment – automatisierte Bereitstellung.

CSS Sprache zur Gestaltung von Weblayouts.

Inline CSS CSS-Stile direkt im HTML-Element.

Stylesheets Dokumente mit CSS-Regeln.

Kubernetes System zur Container-Orchestrierung.

YAML Lesbares Datenformat für Konfigurationen.

Container Image Paket für ausführbare Anwendungen.

Ports Virtuelle Kommunikationspunkte in Netzwerken.

Umgebungsvariablen Werte für Prozessumgebungen.

Kubernetes Ingress Regelt HTTP/HTTPS-Zugriff im Cluster.

Deployment Sicherstellung der Pod-Ausführung in Kubernetes.

Infrastructure as Code Infrastrukturverwaltung per Code.

Terraform Tool für Infrastrukturverwaltung als Code.

Go Programmiersprache von Google.

HTML Standardsprache für Webdokumente.

GitLab Plattform für Versionskontrolle und DevOps.

OneSource Unternehmens-GitLab-Instanz.

Runway Interne Entwicklerplattform und Dokumentationsquelle.

GitLab-Systemhooks Reaktionen auf GitLab-Events.

GitLab Event Events innerhalb von GitLab.

Git Commit Speicherung von Codeänderungen.

Onboarding Integration neuer Mitarbeiter.

User Experience (UX) Benutzererfahrung bei der Produktnutzung.

Schnittstelle Kommunikationspunkt zwischen Systemen.

Automatisierung Automatisierte Prozessausführung.

Jira Projektmanagement-Software.

agil Flexibler Entwicklungsansatz.

iterativ Prozess mit wiederholten Zyklen.

Microservices Architektur aus unabhängigen Diensten.

Code Review Überprüfung von Quellcode.

Unit Tests Tests einzelner Softwarekomponenten.

Integration Tests Tests des Zusammenspiels von Komponenten.

Best Practices Erprobte Verfahren in der Softwareentwicklung.

SQS AWS-Dienst für Nachrichtenwarteschlangen.

SNS AWS-Dienst für Benachrichtigungen.

AWS Cloud-Plattform von Amazon.

SES AWS-E-Mail-Dienst.

EKS Verwalteter Kubernetes-Dienst von AWS.

HTTP Protokoll für Datenübertragung im Web.

HTTP POST Methode zum Senden von Daten an Server.

Filter Policies Filter für Nachrichtenattribute.

Graph Client Client für Microsoft Graph API.

Payload Daten in einer Nachricht.

Message Attributes Zusätzliche Nachrichteninformationen.

SQS Consumer Komponente zur Verarbeitung von SQS-Nachrichten.

Flag Kennzeichnung für Zustände oder Optionen.

Endpunkte URLs für API-Kommunikation.

Listings

1 Einleitung

Diese Projektdokumentation beschreibt den Ablauf des Projektes „Implementierung automatisierter Onboarding- und Ressourcenmanagement-Prozesse zur Optimierung der User Experience für neue Entwickler bei Touristik Union International (TUI) Group“.

Die vorliegende Dokumentation wurde projektbegleitend erstellt und dient der Abschlussprüfung im Ausbildungsberuf Fachinformatiker Anwendungsentwicklung. Sie erläutert die Ziele, den Ablauf und die Ergebnisse des Projektes zur Automatisierung des Onboarding-Prozesses für neue Entwickler. Des Weiteren werden die eingesetzten Technologien und die Interaktionen mit verschiedenen Systemen beschrieben.

Die blau markierten Begriffe werden nicht direkt im Text erklärt, sondern im angehängten Glossar.

1.1 Projektumfeld

Die TUI InfoTec GmbH ist eine Tochtergesellschaft der TUI AG, einem weltweit führenden Unternehmen im Bereich Tourismus und Reisen. Als interner IT-Dienstleister übernimmt die TUI InfoTec GmbH die IT-Betreuung und -Optimierung für die gesamte TUI AG. Sie ist verantwortlich für die Bereitstellung und Wartung der IT-Infrastruktur sowie für die Entwicklung und Implementierung von Softwarelösungen, die die Geschäftsprozesse innerhalb des Konzerns unterstützen.

Die TUI InfoTec GmbH beschäftigt zurzeit etwas mehr als 600 Mitarbeiter[1], die in unterschiedlichen Bereichen der IT tätig sind. Die **Shared Services** Abteilung, die als Auftraggeber für dieses Projekt fungiert, bietet zentrale IT-Services für verschiedene Abteilungen des Unternehmens an. Dabei konzentriert sich diese Abteilung besonders darauf, den Entwicklern eine optimale Arbeitsumgebung bereitzustellen, um sie von zeitaufwendigen, wiederkehrenden Aufgaben zu entlasten und ihre Produktivität zu steigern.

Die gesamte TUI Group beschäftigt hingegen über 65.413 Mitarbeiter[2], was die Größe und die Bedeutung des Unternehmens im globalen Tourismussektor unterstreicht.

1.2 Projektziel

Ziel dieses Projekts ist es, die Effizienz und User Experience beim Onboarding neuer Entwickler durch automatisierte Prozesse zu verbessern. Der bisherige Onboarding-Prozess erfolgt größtenteils manuell und umfasst häufig den zeitaufwändigen Schritt, neuen Entwicklern die Informationen zu internen Abläufen, Dokumentationsstandorten und weiteren wichtigen Ressourcen manuell im Laufe des Einstiegs zu vermitteln. Zusätzlich müssen neue Entwickler manuell in die relevanten Microsoft Teams Gruppen aufgenommen werden.

[1] Stand 2021.
[2] Stand 2023.

Im Rahmen des Projekts wird eine Anwendung entwickelt, die auf GitLab Events reagiert und eine automatisierte Willkommens-E-Mail an neue Entwickler sendet. Diese E-Mail enthält wichtige Informationen zu internen Prozessen, Dokumentationsstandorten und weiteren zentralen Ressourcen, sodass neue Entwickler direkt beim Einstieg eine strukturierte Übersicht erhalten und effizient auf die notwendigen Informationen zugreifen können. Zusätzlich fügt die Anwendung die neuen Entwickler automatisiert den passenden Microsoft Teams-Gruppen hinzu. Diese Automatisierung steigert die Produktivität neuer Mitarbeiter und reduziert zugleich den manuellen Aufwand für das IT-Team erheblich.

Das Projektergebnis soll eine Lösung sein, die sowohl den aktuellen Anforderungen gerecht wird als auch einfach zu verwalten und skalierbar ist, um flexibel auf zukünftige Anforderungen reagieren zu können.

1.3 Projektbegründung

Der aktuelle Onboarding-Prozess für neue Entwickler ist größtenteils manuell und erfordert erheblichen Zeitaufwand, was das IT-Team zusätzlich belastet und den Einstieg der neuen Mitarbeiter erschwert. Neue Entwickler werden oft nicht sofort in die richtigen Teams und Kommunikationskanäle integriert und erhalten möglicherweise nicht alle relevanten Informationen zum Einstieg in die Entwicklungsprozesse der TUI. Dies kann zu Verzögerungen führen, die den Produktivitätseintritt der Entwickler behindern. Ein automatisierter Prozess würde sicherstellen, dass jeder neue Entwickler sofort alle nötigen Ressourcen und Teammitgliedschaften erhält und gleichzeitig unnötige manuelle Arbeit für das IT-Team entfällt.

Die Automatisierung des Onboarding-Prozesses bietet somit klare Vorteile: - **Zeitersparnis**: Die Automatisierung reduziert die Zeit, die das IT-Team für manuelle Aufgaben aufwenden muss. Neue Entwickler können ohne Verzögerung in die relevanten Gruppen aufgenommen und erhalten automatisch alle notwendigen Informationen. - **Kosteneffizienz**: Durch die Reduktion des manuellen Aufwands werden nicht nur Fehler vermieden, sondern auch Ressourcen effizienter eingesetzt. Das IT-Team kann seine Kapazitäten für wichtigere Aufgaben nutzen.

1.4 Projektschnittstellen

Die entwickelte Anwendung interagiert mit verschiedenen Systemen, um den Onboarding-Prozess für neue Entwickler zu automatisieren. Ein zentraler Bestandteil ist die Integration mit GitLab, das bei TUI sowohl für die Versionskontrolle als auch für Continuous Integration und Continuous Deployment (CI/CD) genutzt wird. Um die Automatisierung zu ermöglichen, werden Schnittstellen in der Programmiersprache Go entwickelt, die auf GitLab Events reagieren und die entsprechenden Prozesse auslösen.

Das Projekt wurde durch den Head of Technology der **Shared Services** Abteilung genehmigt. Innerhalb der Abteilung standen zudem Mitarbeiter zur Verfügung, um bei Rückfragen zu unterstützen und regelmäßiges Feedback während der Entwicklung zu geben.

Die **Benutzer** der Anwendung sind neue Entwickler, die automatisch in den Onboarding-Prozess integriert werden, um eine reibungslose Einarbeitung und den Zugang zu den erforderlichen Ressourcen zu gewährleisten.

1.5 Projektabgrenzung

Die aktuelle Refaktorisierung des bestehenden Systems ist nicht Teil dieses Projekts. Zukünftige Anpassungen des aktuellen Systems sind erforderlich, um die GitLab-Systemhooks, die wir derzeit verwenden, entsprechend zu überarbeiten und anzupassen. Diese Änderungen werden separat behandelt und sind nicht im Rahmen der gegenwärtigen Automatisierungsprojekte enthalten.

2 Projektplanung

2.1 Projektphasen

Für die Durchführung des Projekts standen insgesamt 80 Stunden zur Verfügung. Diese Stunden wurden vor Projektbeginn auf verschiedene Phasen der Softwareentwicklung verteilt. Eine Übersicht der groben Zeitplanung und der Hauptphasen ist in Tabelle 1 zu finden. Darüber hinaus können die einzelnen Hauptphasen in kleinere Unterphasen unterteilt werden. Eine detaillierte Darstellung dieser Phasen ist im Anhang A.1: Detaillierte Zeitplanung auf Seite i zu finden.

1

Projektphase	**Geplante Zeit**
Analysephase	6 h
Entwurfsphase	6 h
Implementierungsphase	34 h
Test- und Kontrollphase	18 h
Dokumentation + Nachbearbeitung	6 h
Erstellen der Projektdokumentation und Präsentation	8 h
Pufferzeit	2 h
Gesamt	**80 h**

Tabelle 1: Grobe Zeitplanung

Eine detailliertere Zeitplanung findet sich im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

2.2 Ressourcenplanung

Die Planung der benötigten Ressourcen ist ein wesentlicher Bestandteil der Projektorganisation.[3] Dazu gehören sowohl Hard- und Software als auch die erforderlichen Räumlichkeiten. Eine detaillierte Übersicht über die verwendeten Ressourcen finden Sie im Anhang A.2: Ressourcen Planung auf Seite ii.

2.3 Entwicklungsprozess

Für die Durchführung des Projekts wurde ein agiler Entwicklungsprozess gewählt. Dieser Ansatz ermöglichte es, flexibel auf sich ändernde Anforderungen und Herausforderungen während der Entwicklung zu reagieren. Der Arbeitsprozess war stark iterativ geprägt, mit regelmäßigen Rücksprachen und enger Zusammenarbeit mit den Senior-Entwicklern der Abteilung.

Der iterative Zyklus bestand darin, Arbeitsschritte zu planen, umzusetzen, zu evaluieren und anhand des erhaltenen Feedbacks kontinuierlich zu verbessern. Diese kurzen Iterationszyklen erlaubten es, Ergebnisse frühzeitig zu präsentieren und Rückmeldungen direkt in die nächste Phase einfließen zu lassen.

Die agile Vorgehensweise unterstützte zudem schnelle Anpassungen an neue Anforderungen, die in den regelmäßigen Besprechungen sowie durch Feedback der Senior-Entwickler eingebracht wurden. Durch den Einsatz von Tools wie Microsoft Teams und Jira wurde die Zusammenarbeit und das Projektmanagement gefördert. Diese Tools sorgten dafür, dass der Fortschritt transparent blieb und die Arbeitsschritte effizient geplant und dokumentiert werden konnten[4].

3 Analysephase

3.1 Ist-Analyse

Der bisherige Onboarding-Prozess stellt neue Mitarbeiter vor die Herausforderung, dass zentrale Informationen über interne Prozesse, wie die Standorte von Dokumentationen, eingesetzte Technologien und wichtige Einstiegshilfen, nicht rechtzeitig zur Verfügung stehen. Obwohl wir eine Entwicklerplattform anbieten, auf der alle notwendigen Dokumentationen gesammelt sind, entdecken neue Entwickler diese häufig erst spät. Zudem sind neue Mitarbeiter oft nicht in die relevanten Microsoft Teams-Kanäle integriert, was die Kommunikation und den Zugriff auf wichtige Informationen zusätzlich verzögert.

Da jeder neue Entwickler früher oder später in unserem GitLab System arbeitet, besteht hier die Chance, den Onboarding-Prozess zu automatisieren. Momentan fehlen jedoch automatisierte Prozesse, die neue Entwickler nahtlos in die vorhandenen Systeme einbinden.

[3] Kleinschmidt, H., *Projektmanagement: Grundlagen, Methoden, Techniken*, 2012; Schrey, S., *Projektmanagement: Ein Lehrbuch*, 2011.
[4] Pressman, R. S., *Software Engineering: A Practitioner's Approach*, 2014.

Um dieses Problem zu lösen, haben wir beschlossen, eine GitLab-Systemhook zu implementieren, die auf `user_create` Events lauscht und entsprechend reagiert. Zum Zeitpunkt des Projektbeginns waren bereits drei separate Systemhooks im Einsatz. Daher entschieden wir uns für die Entwicklung einer zentralen Systemhook, die alle GitLab Events an ein zentrales AWS SNS Topic (Simple Notification Service) sendet. Von dort aus können verschiedene Anwendungen durch SQS Queues (Simple Queue Service) und spezifische Filterregeln gezielt auf die GitLab Events reagieren. Diese Lösung bietet einen zentralen Punkt für GitLab Events und ermöglicht es uns, den Onboarding-Prozess effizient zu erweitern und zu automatisieren.

3.2 Wirtschaftlichkeitsanalyse

Das Projekt zur Automatisierung des Onboardings ist für die TUI Group von großem wirtschaftlichen Nutzen. Die Automatisierung sorgt dafür, dass neue Entwickler sofort Zugriff auf alle relevanten Ressourcen erhalten, ohne dass manuelle Eingriffe notwendig sind. Dadurch werden sowohl Verzögerungen als auch die Gefahr, Mitarbeiter zu übersehen oder Zugänge zu vergessen, deutlich reduziert. Dies steigert nicht nur die Effizienz der neuen Entwickler, sondern entlastet auch das IT-Team, das sich auf wichtigere Aufgaben konzentrieren kann. Durch die Einsparung von Zeit und die Reduktion der Arbeitsunterbrechungen werden langfristig Kosten gesenkt und die Produktivität gesteigert.

3.2.1 „Make or Buy“-Entscheidung

Es existiert keine fertige Softwarelösung, die die spezifischen Anforderungen der TUI Group vollständig abdeckt. Besonders die Integration mit den bereits genutzten GitLab-Systemhooks und die spezifischen Anpassungen, die für die internen Abläufe notwendig sind, erfordern eine stark angepasste, maßgeschneiderte Lösung. Aus diesem Grund wurde entschieden, das Projekt intern umzusetzen, um eine optimale Integration in die bestehende Infrastruktur sowie eine hohe Flexibilität für zukünftige Anpassungen sicherzustellen.

3.2.2 Projektkosten

Die Kosten für die Durchführung des Projekts setzen sich aus Personal- und Ressourcenkosten zusammen. Bei der Berechnung des Stundensatzes sowohl für Auszubildende als auch für Senior Entwickler werden Arbeitgeberanteile und Sozialversicherungen berücksichtigt, um den tatsächlichen Stundensatz realistisch darzustellen.
Ein Auszubildender im dritten Lehrjahr verdient laut Tarifvertrag monatlich 1486 € brutto, was einem Stundenlohn von etwa 11,68 € entspricht:

$$\frac{(1486\,€ \cdot 1.21)}{20\,\text{Tage/Monat} \cdot 7.7\,\text{Stunden/Tag}} = \frac{1.798.06\,€/\text{Monat}}{154\,\text{Stunden/Monat}} \tag{1}$$

$$\approx 11,68\,€/\text{Stunde} \tag{2}$$

Für die Mitarbeit eines Senior Entwicklers, der für Fachgespräche konsultiert wurde, wird ein monatliches Gehalt von 6069 € zugrunde gelegt, was einem Stundenlohn von etwa 45,86 € ergibt:

$$\frac{(6069\,€ \cdot 1,21)}{20\,\text{Tage/Monat} \cdot 7.7\,\text{Stunden/Tag}} = \frac{7.343,49\,€/\text{Monat}}{154\,\text{Stunden/Monat}} \tag{1}$$

$$\approx 47,69\,€/\text{Stunde} \tag{2}$$

Zusätzlich zu den Personalkosten werden pauschale Ressourcenkosten von 15 € pro Stunde berücksichtigt, um die Nutzung von Büroarbeitsplätzen, Hardware, Software und Strom abzudecken. Der Stundensatz für einen Auszubildenden beläuft sich auf 10,96 €, während der eines Senior Entwicklers 45,86 € beträgt. Für die Ressourcennutzung wird ein einheitlicher Stundensatz von 15 € angesetzt. Die nachfolgende Tabelle zeigt eine detaillierte Übersicht der Projektkosten für die einzelnen Arbeitsschritte.

Vorgang	**Mitarbeiter**	**Zeit (h)**	**Personal (€)[6]**	**Ressourcen (€)[7]**	**Gesamt (€)**
Entwicklungskosten	1 x Auszubildender	80	934,4	1.200	2.134,4
Fachgespräch	2 x Senior Entwickler 1 x Auszubildender	3	321,18	135,00	456,18
Code-Review	1 x Senior Entwickler	4	190,76	60,00	250,76
Abnahme	2 x Senior Entwickler	1	95,38	30,00	125,38
				Projektkosten gesamt	**2.966,72**

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

Im folgenden Abschnitt soll ermittelt werden, ab welchem Zeitpunkt sich die Entwicklung der Anwendung amortisiert hat. Dieser Wert ermöglicht es, die wirtschaftliche Sinnhaftigkeit des Projekts zu bewerten und zu erkennen, ob sich durch die Umsetzung langfristige Kostenvorteile ergeben. Die Amortisationsdauer wird berechnet, indem die Anschaffungskosten durch die laufende Kostenersparnis geteilt werden, die das neue Produkt erzielt.

Durch die Automatisierung des Übertragungsprozesses ließe sich erhebliche Bearbeitungszeit einsparen, was zu einer Reduzierung der Personalkosten führt.

[6]Personalkosten pro Vorgang = Anzahl Mitarbeiter · Zeit · Stundensatz.
[7]Ressourcenbeitrag pro Vorgang = Anzahl Mitarbeiter · Zeit · 15 € (Ressourcenbeitrag pro Stunde).

Im Anhang A.4: Amortisation auf Seite iv ist dargestellt, dass wir monatlich etwa 150 neue Benutzer hinzugewinnen. Für jeden neuen Benutzer werden etwa 5 Minuten benötigt, um ihnen die relevanten Informationen zu internen Prozessen und Dokumentationsstandorten bereitzustellen und sie in die Teamgruppe hinzuzufügen. Durch den Automatisierungsprozess kann dieser Zeitaufwand vollständig eingespart werden.

Multipliziert man die monatlichen neuen Benutzer (150) mit den 5 Minuten pro Benutzer, ergibt das jährlich:

$$150\,\text{Benutzer} \times 5\,\text{Minuten} \times 12\,\text{Monate} = 9000\,\text{Minuten/Jahr} \quad \text{oder} \quad 150\,\text{Stunden/Jahr}$$

Da der Onboarding-Prozess meist von Senior-Entwicklern durchgeführt wird, wird für die Berechnung der Einsparungen der Stundensatz eines Senior Entwicklers von 47,69 € verwendet. Das führt zu einer jährlichen Einsparung von:

$$\text{Einsparung} = 150\,\text{Stunden} \times 47{,}69\,€ \approx 7.153{,}5\,€$$

Die Amortisationsdauer des Projekts kann berechnet werden als:

$$\text{Amortisationsdauer (in Jahren)} = \frac{\text{Anschaffungskosten}}{\text{jährliche Einsparung}} = \frac{2.966,72\,€}{7.153,5\,€} \approx 0,415\,\text{Jahre}$$

Um die Amortisationsdauer in Tagen zu erhalten, multiplizieren wir mit 365:

$$\text{Amortisationsdauer (in Tagen)} \approx 0,415\,\text{Jahre} \times 365\,\text{Tage/Jahr} \approx 151\,\text{Tage}$$

Die Amortisationsdauer des Projekts beträgt somit etwa 151 Tage, was die wirtschaftliche Effizienz des Projekts verdeutlicht. Eine graphische Darstellung ist im Anhang A.4 zu finden.

3.3 Nicht-monetäre Vorteile

Die Wirtschaftlichkeitsanalyse hat bereits eine ausreichende Grundlage für die Realisierung des Projekts geliefert, sodass auf eine detaillierte Analyse nicht-monetärer Vorteile verzichtet wird. Allerdings sind die qualitativen Verbesserungen, die das Projekt mit sich bringt, von großer Bedeutung.

Diese unmittelbare Verfügbarkeit von Informationen führt dazu, dass neue Teammitglieder schneller produktiv werden können und sich auf ihre Kernaufgaben konzentrieren, anstatt Zeit mit administrativen Tätigkeiten zu verlieren. Zudem wird die Belastung der bestehenden Entwicklerteams deutlich verringert, da weniger Zeit für die Einrichtung und Verwaltung von Benutzerzugängen aufgewendet werden muss. Diese Effekte tragen nicht nur zu einer besseren Zusammenarbeit innerhalb der Teams bei, sondern

fördern auch eine angenehme Arbeitsatmosphäre, in der sich alle Teammitglieder auf ihre wichtigen Aufgaben konzentrieren können.

4 Entwurfsphase

4.1 Zielplattform

Aus der Ist-Analyse ging hervor, dass wir zwei wesentliche Schnittstellen benötigen, um die Anforderungen des Projekts zu erfüllen. Erstens eine GitLab System Hook, die GitLab Events an ein Amazon Simple Notification Service (SNS) Topic sendet. Zweitens eine Anwendung, die spezifische Events aus einer Amazon Simple Queue Service (SQS) Queue konsumiert und darauf reagiert, wie z.B. die Verarbeitung von `user_create` Events. Beide Schnittstellen haben wir in Go entwickelt, um unsere bestehende Expertise zu nutzen und den Entwicklungsprozess zu beschleunigen.

Die Wahl der Programmiersprache Go[5] basiert auf einer Reihe von technischen und praktischen Überlegungen. Da Go die primär in unserer Abteilung verwendete Programmiersprache ist und das Team über umfangreiche Erfahrung damit verfügt, konnten wir die Einarbeitungszeit minimieren und effizient arbeiten.

Für die Bereitstellung der Anwendung haben wir uns für Amazon Web Services (AWS) entschieden. AWS bietet eine breite Palette an Diensten, die unseren Projektanforderungen ideal entsprechen, und ist bereits tief in den Prozessen unseres Unternehmens verankert. Diese Vertrautheit mit AWS ermöglicht eine reibungslose Implementierung und die Nutzung bestehender Best Practices und interner Ressourcen.

Für die zentrale Erfassung und Verarbeitung der GitLab Events nutzen wir AWS SNS in Kombination mit AWS SQS. Der SNS-Dienst sammelt die relevanten GitLab Events und verteilt sie über SQS an verschiedene Anwendungen. Diese Architektur garantiert eine skalierbare und effiziente Verarbeitung der Events.

Für den Versand von E-Mails setzen wir AWS Simple Email Service (SES) ein. Die E-Mails werden mithilfe von HTML und CSS gestaltet, um ansprechende und benutzerfreundliche Inhalte zu erzeugen.

Die Schnittstellen selbst werden in einem bestehenden Amazon Elastic Kubernetes Service (EKS) Cluster bereitgestellt, der bereits für andere Anwendungen in unserer Abteilung genutzt wird. Durch die Wiederverwendung dieser Infrastruktur können wir zusätzliche Ressourcen sparen und den Verwaltungsaufwand minimieren. Der Soll-Zustand wird als Aktivitätsdiagramm in Anhang A.5 dargestellt.

[5] The IT Solutions - How Golang is Perfect for Cloud Native Applications, The IT Solutions [2024].

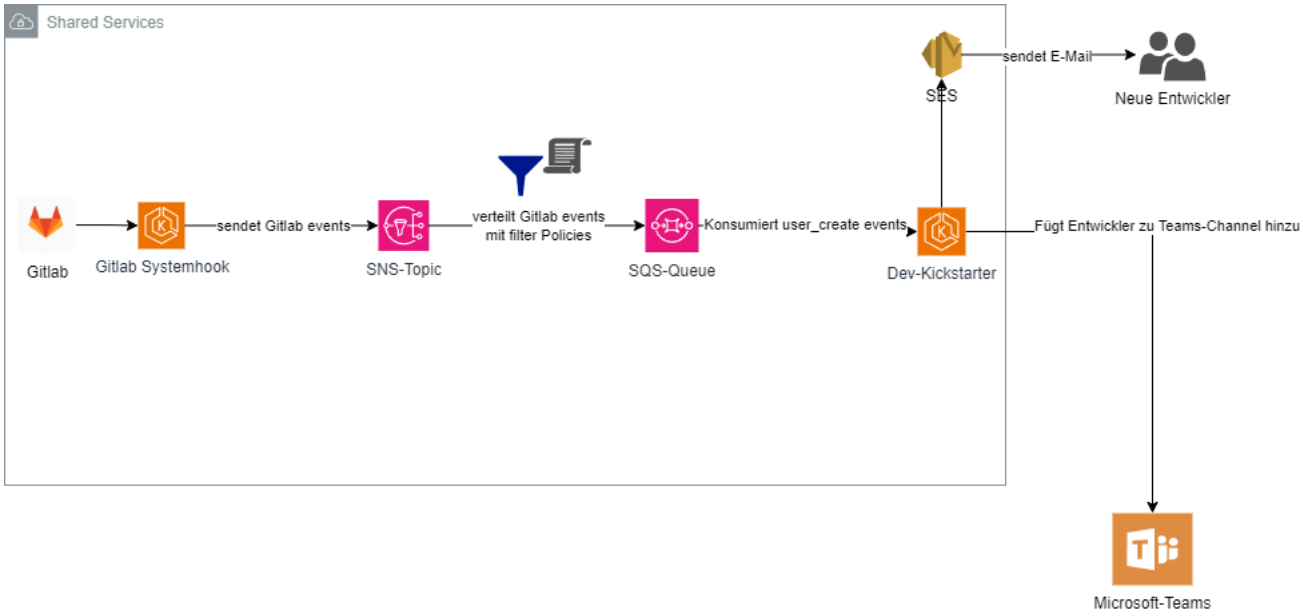

Abbildung 1: Finaler Entwurf des Infrastruktur-Designs und Soll-Zustand

4.2 Architekturdesign

Für das Projekt wurde eine **ereignisgesteuerte Architektur (Event Driven Architecture)** gewählt.[6] Diese Architektur nutzt lose gekoppelte Microservices, die Informationen durch das Erzeugen und Konsumieren von Events austauschen. GitLab sendet Events über einen System Hook an ein AWS SNS-Topic, von dem aus die Events an verschiedene SQS Queues weitergeleitet werden. Diese asynchrone Kommunikation ermöglicht eine flexible und skalierbare Verarbeitung.

Ein in Go geschriebener Service konsumiert die relevanten Events von der SQS Queues. Bei bestimmten Events, wie `user_create`, versendet dieser Service Willkommens-E-Mails über AWS SES und fügt neue Entwickler zu Microsoft Teams hinzu. Filter Policies auf den SQS Queues sorgen dafür, dass nur relevante Events verarbeitet werden. Diese Architektur gewährleistet Skalierbarkeit, Entkopplung und effiziente Ereignisverarbeitung.

4.3 Entwurf der E-Mail

Das Design der E-Mail wurde zu Beginn der Entwurfsphase skizzenhaft entworfen, um sicherzustellen, dass sie benutzerfreundlich und einladend wirkt.[7]Besonders wichtig war es, die Informationen so aufzubereiten, dass sie nicht überladen sind und die Empfänger, insbesondere neue Entwickler, schnell und einfach erfassen können.

Die Gestaltung fokussierte sich auf eine klare und ansprechende Struktur, die dazu einlädt, die E-Mail aufmerksam zu lesen. Durch regelmäßige Rücksprache mit dem Auftraggeber und iteratives Feedback

[6] Amazon Web Services - Event Driven Architecture, Amazon Web Services [2024d].
[7] Hub Spot - The Ultimate Guide to Email Design, Hub Spot [2024].

konnte das Design kontinuierlich optimiert werden. Letztlich wurde die E-Mail in HTML und CSS umgesetzt, um eine moderne und einladende Benutzererfahrung zu gewährleisten.

Beispielentwurf findet sich im Anhang A.6: Oberflächenentwürfe auf Seite vi.

4.4 Maßnahmen zur Qualitätssicherung

Um die Qualität des Projektergebnisses zu sichern, wurden verschiedene Maßnahmen ergriffen.[8]. Dazu gehören die Implementierung von Integrationstests und Unit-Tests, die automatisch bei jedem Git-Commit ausgeführt werden. Diese Tests gewährleisten, dass Änderungen am Code keine bestehenden Funktionen beeinträchtigen.

Darüber hinaus fand eine iterative Überprüfung der Codequalität durch erfahrene Mitarbeiter statt. Durch regelmäßige Code-Reviews konnte sichergestellt werden, dass der Code den Qualitätsstandards entspricht und Best Practices eingehalten werden. Diese Kombination aus automatisierten Tests und manuellem Feedback trägt maßgeblich zur hohen Qualität des Projekts bei.

5 Implementierungsphase

Bevor mit der Implementierung begonnen wurde, wurde ein Iterationsplan erstellt. In diesem wurden die einzelnen Schritte und deren Reihenfolge festgelegt. In jeder Iteration wurde eine spezifische Funktionalität umgesetzt und am Ende der jeweiligen Iteration dem Team präsentiert. Dieses Vorgehen folgt den in Abschnitt 2.3 beschriebenen Prinzipien der agilen Softwareentwicklung. Der vollständige Iterationsplan befindet sich im Anhang A.3: Iterationsplan auf Seite iii.

5.1 Implementierung der GitLab Systemhook

Im Rahmen der Implementierung der GitLab Systemhook wurde ein Prozess entwickelt, der es ermöglicht, GitLab Events wie die Erstellung neuer Benutzer in GitLab zu erfassen und an ein SNS-Topic weiterzuleiten. Dabei lag der Schwerpunkt darauf, die Systemhook effizient und erweiterbar zu gestalten, um verschiedene GitLab Events verarbeiten zu können.

Abbildung 2: Implementierung der GitLab Systemhook

[8] TU Wien - Methoden der Qualitätssicherung, TU Wien [2007]

Das `Hook`-Struct sowie die Initialisierung der erforderlichen Komponenten, wie dem GitLab-Client und dem AWS SNS-Client, wurden gemäß den Anforderungen definiert. Der Endpunkt, an den die HTTP-POST-Anfragen gesendet werden, ist `/api/v1/hook`. Dieser wird durch die Methode `handleHook` in der `Hook`-Struct verarbeitet, die im Anhang A.8: Initialisierung der Hook-Struktur auf Seite ix dargestellt wird. Der Ablauf beginnt mit der Authentifizierung der HTTP-POST-Anfragen durch die Überprüfung des Tokens, um sicherzustellen, dass nur autorisierte Systeme Zugriff auf die API erhalten. Codeauschnitte des Authentifizierung- prozesses befinden sich im Anhang A.9: Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS auf Seite x.

Sobald ein GitLab Event erfasst wurde, erfolgt die Verarbeitung der Payload asynchron. Durch die asynchrone Verarbeitung werden eingehende Anfragen schnell entgegengenommen und weiterverarbeitet, ohne das System zu blockieren. Im nächsten Schritt wird die empfangene Payload an den SNS-Dienst gesendet[9], wie in Anhang A.9: Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS auf Seite x veranschaulicht. Damit die Nachrichten gezielt gefiltert und ausgewertet werden können, werden sogenannte Message Attributes erstellt. Diese Attribute enthalten Informationen über das jeweilige Event, wie z.B. den Eventnamen oder spezifische Merkmale des Benutzers, der das Event ausgelöst hat. Im Falle einer Benutzererstellung wird beispielsweise geprüft, ob der neue Benutzer ein Bot ist, was ebenfalls als Attribut weitergeleitet wird.

Die Message Attributes spielen eine entscheidende Rolle bei der Filterung und ermöglichen eine granulare Weiterleitung von GitLab Events. Auf Basis dieser Attribute können spezifische Filter Policies angewendet werden, sodass nur relevante Events an nachgelagerte Systeme oder Prozesse weitergeleitet werden.

5.2 Implementierung des Dev-Kickstarter

Im Rahmen der Implementierung des Dev-Kickstarter wurde eine robuste Architektur entworfen, die es ermöglicht, **user_create** Events zu verarbeiten und Aktionen wie das Versenden von Willkommens-E-Mails und das Hinzufügen neuer Benutzer zu Microsoft Teams-Gruppen automatisiert auszuführen.

Das `Kickstarter`-Struct sowie die Initialisierung der erforderlichen Komponenten, wie dem AWS SES-Client, SQS-Consumer und Microsoft Graph-Client, wurden gemäß den Anforderungen definiert. Der SQS-Consumer empfängt Nachrichten aus einer SQS Queues, die mit Filter Policies erstellt wurde, um nur relevante Ereignisse zu verarbeiten.[10] Die Definition der `Kickstarter`-Struct ist im Anhang A.10: Initialisierung der Kickstarter-Struktur auf Seite xii zu finden.

Nach dem Empfang einer Nachricht wird die Payload asynchron verarbeitet. Dabei wird die E-Mail-Adresse des Benutzers extrahiert, um eine Willkommens-E-Mail über den AWS SES-Dienst[11] zu versenden. Anschließend wird der Benutzer automatisch zu einer definierten Microsoft Teams-Gruppe hinzugefügt, indem die Microsoft Graph API verwendet wird. Der zugehörige Code befindet sich im

[9] AWS - Amazon Simple Notification Service Documentation, Amazon Web Services [2024b].
[10] AWS - Amazon Simple Queue Service Documentation, Amazon Web Services [2024c].
[11] AWS - Amazon Simple Email Service Documentation, Amazon Web Services [2024a].

Anhang A.11: Verarbeitung des Gitlab Events und senden der E-Mail/Hinzufügen zur Teams Gruppe auf Seite xiv.

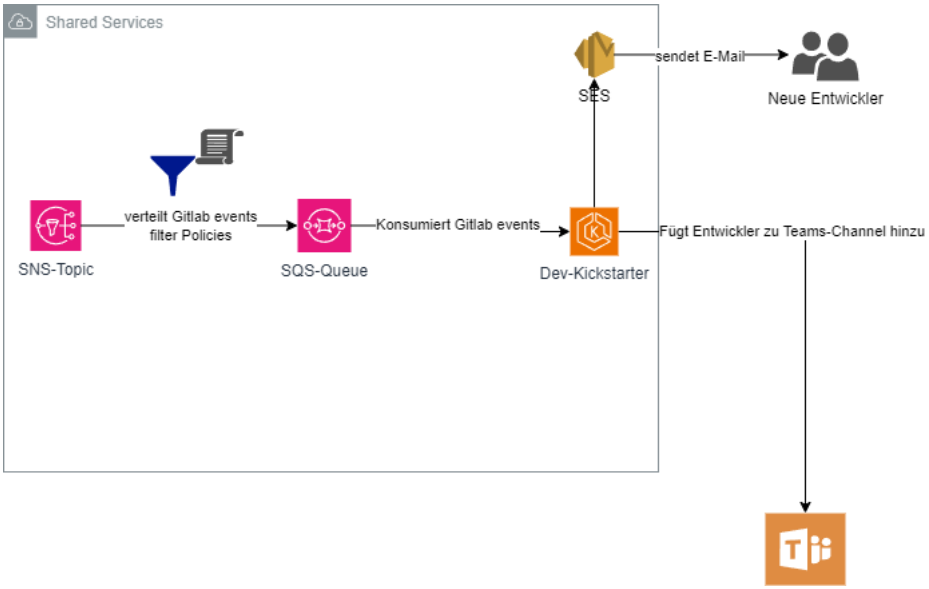

Abbildung 3: Implementierung des Dev-Kickstarter

5.3 Implementierung der E-Mail

Die E-Mail wurde als HTML-Dokument erstellt, um sicherzustellen, dass sie auf verschiedenen Endgeräten und E-Mail-Clients korrekt angezeigt wird. Dabei wurde inline CSS verwendet, da viele E-Mail-Clients externe Stylesheets nicht vollständig unterstützen. Dies gewährleistet eine konsistente Darstellung der E-Mail auf allen Plattformen.[12]

Ein wichtiges Merkmal der Implementierung ist die dynamische Personalisierung der Inhalte. Dazu wurde der Platzhalter `{{.FirstName}}` eingefügt, um den Vornamen des Empfängers einzufügen. Dieser Wert wird zur Laufzeit gefüllt, indem der Code des Empfängers ausgewertet und der entsprechende Vorname dynamisch in die Vorlage eingefügt wird.

Zusätzlich wurden nützliche Ressourcen und Links in die E-Mail integriert, wie z.B. der Zugriff auf interne TUI-Plattformen wie Runway und OneSource, um den neuen Teammitgliedern den Einstieg zu erleichtern. Sicherheitsaspekte wurden ebenfalls berücksichtigt, indem Links zu den TUI-Sicherheitsrichtlinien eingebunden wurden. Dies unterstreicht die Wichtigkeit des sicheren Arbeitens innerhalb der Organisation.

Wie die finale E-Mail aussieht, kann im Anhang 9: Finales Design der E-Mail auf Seite viii eingesehen werden.

[12] Hub Spot - The Ultimate Guide to Email Design, Hub Spot [2024].

6 Einführungsphase

6.1 EKS-Deployment

Die GitLab-Systemhook und der Dev-Kickstarter wurden im Rahmen einer automatisierten CI/CD-Pipeline containerisiert und anschließend in die GitLab-Registry hochgeladen. Um die Anwendungen im Kubernetes-Cluster (EKS) bereitzustellen, wurde eine yaml-Konfigurationsdatei erstellt. Diese Datei definiert alle notwendigen Ressourcen und Parameter, wie z.B. die Container-Images, Ports, Umgebungsvariablen und Ingress-Konfigurationen.

Die fertige yaml-Datei wurde dann in ein internes Repository commitet, welches für alle unsere EKS-Deployments verwendet wird. Sobald der Commit erfolgt ist, wird das Deployment im Cluster automatisch ausgelöst und die Anwendungen werden dort bereitgestellt.

Ein Beispiel für die verwendete yaml-Konfiguration kann im Anhang A.12: Deployment yaml für die Systemhook auf Seite xvii eingesehen werden.

6.2 AWS-Deployment

Für das AWS-Deployment wurden die erforderlichen Infrastrukturkomponenten in der AWS-Umgebung konfiguriert. Dazu gehören Amazon SNS zur Kommunikation und Amazon SQS zur Nachrichtenverarbeitung.

Die Bereitstellung dieser Ressourcen erfolgte mithilfe von Infrastructure as Code mit Terraform.[13] Dabei wurden SNS-Topics und SQS-Queues eingerichtet, um die Kommunikation zwischen den Diensten zu ermöglichen. Eine Filter-Policy wurde verwendet, um sicherzustellen, dass nur spezifische Ereignisse, wie das Erstellen eines Benutzers (eventName: `user_create`), an die SQS-Queue weitergeleitet werden.

Zusätzlich wurde eine IAM-Policy erstellt, die dem Dev-Kickstarter die erforderlichen Berechtigungen gewährt, um E-Mails über Amazon SES zu senden und Nachrichten aus der SQS-Queue zu empfangen.

Die Terraform-Ressourcen sind in einem internen Repository gespeichert. Ein Beispiel für die verwendeten Terraform-Ressourcen kann im Anhang A.13: Terraform Infrastructure as Code auf Seite xviii eingesehen werden.

7 Abnahmephase

Vor der Endabnahme wurden umfangreiche Tests durchgeführt, um die Qualität der Anwendung sicherzustellen. Unit-Tests prüften die Funktionalität einzelner Codebestandteile isoliert, deren Logs im Anhang A.8: Initialisierung der Hook-Struktur auf Seite ix zu finden sind. Integrationstests überprüften,

[13] Spacelift - Infrastructure as Code, Spacelift [2024].

ob die einzelnen Module korrekt zusammenarbeiten und die definierten Schnittstellen erfolgreich miteinander kommunizieren.

Nachdem die Anwendung fertiggestellt war, konnte sie zur Endabnahme dem Fachbereich vorgelegt werden. Durch die agile Entwicklungsmethode hatte der Fachbereich nach jeder Iteration Zugriff auf aktuelle Versionen der Anwendung und konnten so frühzeitig Feedback geben. Diese regelmäßigen Rücksprachen führten nicht nur zu einer vertrauten Nutzung, sondern auch zu einem tiefen Verständnis der Funktionsweise des Gesamtsystems. Anregungen und Kritik konnten direkt während der Entwicklung berücksichtigt werden, wodurch bereits viele Probleme frühzeitig behoben werden konnten.

Da die Fachbereiche die Anwendung bereits gut kannten, verlief die Endabnahme reibungslos. Zur zusätzlichen Qualitätssicherung wurde vor dem Deployment ein Code Review durch einen weiteren Entwickler durchgeführt, und die Einführung der Anwendung konnte erfolgreich abgeschlossen werden.

8 Dokumentation

Die **Projektdokumentation** beschreibt den gesamten Verlauf der Projektumsetzung, einschließlich der Planung, Durchführung und Tests. Sie dient als Referenz für alle Projektbeteiligten und dokumentiert die wichtigsten Phasen und Entscheidungen im Projekt. IT-spezifische Fachbegriffe werden dabei, soweit möglich, vermieden, um die Verständlichkeit für alle Beteiligten zu gewährleisten.

Die **Entwicklerdokumentation** enthält die technischen Anforderungen, Konfigurationsparameter und Endpunkte der Anwendung. Sie bietet detaillierte Informationen zu Umgebungsvariablen, Flags und Endpunkte für die Einrichtung und das Deployment der Anwendung. Ein Auszug dieser Dokumentation ist im Anhang A.14: Entwicklerdokumentation (Auszug) auf Seite xx zu finden.

Die Dokumentationen sind so formuliert, dass sie den Anforderungen der jeweiligen Verwendung gerecht werden. Die Projektdokumentation ist allgemein verständlich formuliert, während die Entwicklerdokumentation technische Details und Anweisungen für die Konfiguration und den Betrieb der Anwendung enthält.

9 Fazit

9.1 Soll-/Ist-Vergleich

Bei einer rückblickenden Betrachtung des Projekts lässt sich festhalten, dass die festgelegten Anforderungen weitgehend erreicht wurden. Alle wesentlichen Funktionen und Anpassungen wurden gemäß den definierten Zielen und Anforderungen umgesetzt.

Die Projektplanung konnte insgesamt eingehalten werden, obwohl kleinere Abweichungen in einzelnen Phasen auftraten. So war in der Dokumentationsphase ein Mehraufwand erforderlich, während die Test- und Kontrollphase weniger Zeit in Anspruch nahm. Diese Abweichungen konnten innerhalb des geplanten Zeitrahmens ausgeglichen werden, sodass das Projekt planmäßig abgeschlossen werden konnte. Die folgende Tabelle zeigt den Soll-/Ist-Vergleich der Projektphasen:

Tabelle 3: Soll-/Ist-Vergleich

Projektphase	**Geplante Zeit**	**Tatsächliche Zeit**	**Differenz**
Analysephase	6 h	6 h	0 h
Entwurfsphase	6 h	6 h	0 h
Implementierungsphase	34 h	34 h	0 h
Test- und Kontrollphase	18 h	16 h	-2 h
Dokumentation + Nachbearbeitung	6 h	8 h	+2 h
Erstellen der Projektdokumentation und Präsentation	8 h	10 h	+2 h
Pufferzeit	2 h	0 h	-2 h
Gesamt	**80 h**	**80 h**	**0 h**

Der Auftraggeber zeigte sich mit dem Projektergebnis zufrieden, da alle zentralen Anforderungen erfüllt und interne Abläufe optimiert wurden.

9.2 Lessons Learned

Im Verlauf des Projekts konnten wertvolle Erfahrungen gesammelt werden, insbesondere hinsichtlich der effizienten Planung und Einhaltung von Projektphasen in einem agilen Umfeld. Die kontinuierliche Rücksprache mit dem Fachbereich erwies sich als entscheidend für die erfolgreiche Umsetzung der Projektanforderungen. Auch die Anwendung der agilen Entwicklungsweise und die regelmäßige Einbeziehung der Endnutzer sorgten für eine zielgerichtete und bedarfsorientierte Entwicklung.

9.3 Ausblick

Obwohl alle definierten Anforderungen im Projekt umgesetzt wurden, gibt es Potenzial für zukünftige Erweiterungen. Eine mögliche Weiterentwicklung wäre die Ergänzung zusätzlicher Schnittstellen, um eine automatisierte Anbindung weiterer interner Systeme zu ermöglichen. Auch die Weiterentwicklung der bestehenden Funktionen für eine noch tiefere Integration in die vorhandene Infrastruktur wäre sinnvoll.

Aufgrund des modularen Aufbaus der Anwendung, wie im Abschnitt 4.2 beschrieben, können diese Anpassungen und Erweiterungen problemlos vorgenommen werden. Der flexible Aufbau der Anwendung gewährleistet somit eine gute Wartbarkeit und Erweiterbarkeit für zukünftige Anforderungen.

Literaturverzeichnis

Amazon Web Services 2024a
Amazon Web Services: *Amazon Simple Email Service Documentation*, 2024. https://docs.aws.amazon.com/ses/index.html. – Last Time Accessed: 2024-10-30

Amazon Web Services 2024b
Amazon Web Services: *Amazon Simple Notification Service Documentation*, 2024. https://docs.aws.amazon.com/sns/index.html. – Last Time Accessed: 2024-10-30

Amazon Web Services 2024c
Amazon Web Services: *Amazon Simple Queue Service Documentation*, 2024. https://docs.aws.amazon.com/sqs/index.html. – Last Time Accessed: 2024-10-30

Amazon Web Services 2024d
Amazon Web Services: *Event-Driven Architecture*. https://aws.amazon.com/de/event-driven-architecture/. Version: 2024. – Last Time Accessed: 2024-10-30

Hub Spot 2024
Hub Spot: *The Ultimate Guide to Email Design*. https://blog.hubspot.com/marketing/email-design. Version: 2024. – Last Time Accessed: 2024-10-30

Spacelift 2024
Spacelift: *Infrastructure as Code*. https://spacelift.io/blog/terraform-infrastructure-as-code. Version: 2024. – Last Time Accessed: 2024-10-30

The IT Solutions 2024
The IT Solutions: *How Golang is Perfect for Cloud Native Applications*. https://www.theitsolutions.io/blog/go-for-cloud-native-applications. Version: 2024. – Last Time Accessed: 2024-10-30

TU Wien 2007
TU Wien: *Methoden der Qualitätssicherung*. https://qse.ifs.tuwien.ac.at/wp-content/uploads/2007_UBIT_4a.pdf. Version: 2007. – Last Time Accessed: 2024-11-02

Eidesstattliche Erklärung

Ich, Paul Glesmann, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

> *Automatisierte Onboarding Prozesse – Automatisierte Onboarding- und Ressourcenmanagement-Prozesse bei TUI Group*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Hannover, den 18.11.2024

Paul Glesmann

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase		**6 h**
1. Besprechung der bestehenden GitLab-System-Hooks und Identifikation von Verbesserungsmöglichkeiten.		
2. Feedback und Anforderungen der Entwickler und Fachbereiche einholen.		
Entwurfsphase		**6 h**
1. Design der Systemarchitektur		
2. Entwurf der Infrastruktur auf AWS		
3. Detaillierte Analyse der GitLab-Events		
Implementierungsphase		**34 h**
1. Entwicklung des Webservers (Go)	14 h	
2. Implementierung der GitLab-System-Hook	10 h	
3. Infrastruktur-Bereitstellung mit AWS & EKS	10 h	
Test- und Kontrollphase		**18 h**
1. Erstellung von Unit-Tests	8 h	
2. Integrationstests	10 h	
Dokumentation + Nachbearbeitung		**6 h**
1. Dokumentation der Codebasis, API-Integrationen und Infrastruktur	3 h	
2. Vorstellung der Software im Team und Übergabe	3 h	
Erstellen der Projektdokumentation und Präsentation		**8 h**
Puffer		**2 h**
1. Puffer für unvorhergesehene Ereignisse: Zeitreserve für unerwartete Herausforderungen oder Verzögerungen im Projektverlauf.		
Gesamt		**80 h**

A.2 Ressourcen Planung

Hardware

- Büroarbeitsplatz mit Thin-Client

Software

- Windows 7 Enterprise mit Service Pack 1 - Betriebssystem
- Visual Studio Code - Hauptentwicklungsumgebung
- Docker - Containerisierung von Anwendungen
- Go - Programmiersprache zur Entwicklung des Webservers
- AWS - Cloud-Infrastruktur
- EKS (Elastic Kubernetes Service) - Orchestrierung der Container
- Terraform - Infrastruktur als Code
- GitLab - Versionskontrolle und CI/CD
- MS Teams - Zusammenarbeit im Team
- Jira - Projektmanagement

Personal

- Entwickler - Umsetzung des Projektes
- Anwendungsentwickler - Review des Codes

A.3 Iterationsplan

Der folgende Iterationsplan beschreibt die Schritte der Implementierungsphase und deren Reihenfolge. Jede Funktionalität wurde zunächst auf der Testumgebung implementiert und nach erfolgreicher Validierung in die Produktionsumgebung auf dem bestehenden **Amazon EKS Cluster** ausgerollt:

1. **Implementierung der GitLab System Hook:** Entwicklung der GitLab System Hook, die relevante GitLab-Ereignisse an ein SNS Topic sendet.
2. **Erstellung des SNS Topics mit Terraform:** Definition und Bereitstellung des SNS Topics mittels Terraform, um die GitLab-Ereignisse zu empfangen.
3. **Erstellung der SQS Queue mit Filter Policy:** Definition und Bereitstellung einer SQS Queue mit Terraform, inklusive einer Filter Policy, um nur `user_create`-Ereignisse vom SNS Topic zuzulassen.
4. **Implementierung des Dev-Kickstarter Services:** Entwicklung des in Go geschriebenen Dev-Kickstarter Services, der die `user_create`-Ereignisse von der SQS Queue konsumiert.
5. **Extraktion und Verarbeitung der relevanten Daten:** Extraktion der relevanten Informationen, wie die E-Mail-Adresse des erstellten Benutzers, aus den empfangenen GitLab-Ereignissen.
6. **Versand von E-Mails und Hinzufügen zu MS Teams:** Implementierung der Logik, um mithilfe von AWS SES eine Willkommens-E-Mail an den neuen Benutzer zu senden und ihn zu einem MS Teams-Channel hinzuzufügen.

Jeder dieser Schritte wurde iterativ auf der Testumgebung getestet und anschließend auf dem Amazon EKS Cluster in die Produktionsumgebung ausgerollt. Der vollständige Iterationsplan befindet sich im Anhang A.15: Iterationsplan auf S. xiii.

A.4 Amortisation

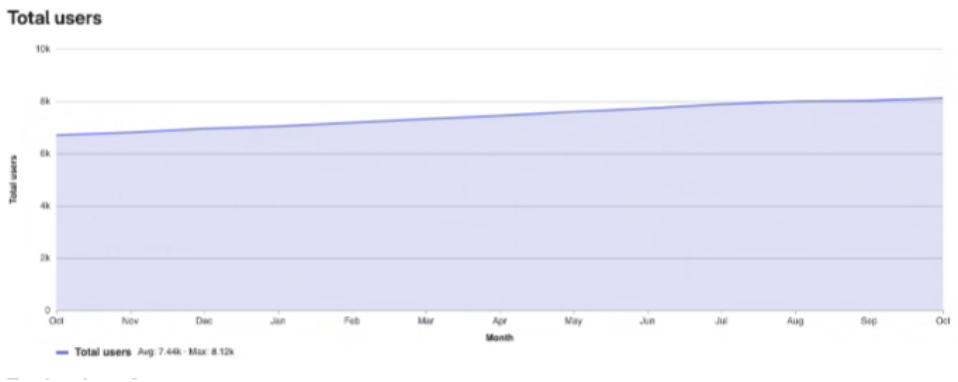

Abbildung 4: Anstieg der registrierten Benutzer in der GitLab-Instanz

Ein durchschnittlicher monatlicher Zuwachs von 150 Nutzern zeigt die wachsende Akzeptanz und Nutzung der Plattform. Dieser Trend verdeutlicht den steigenden Bedarf an GitLab als zentrale Lösung für die Projektverwaltung und -zusammenarbeit.

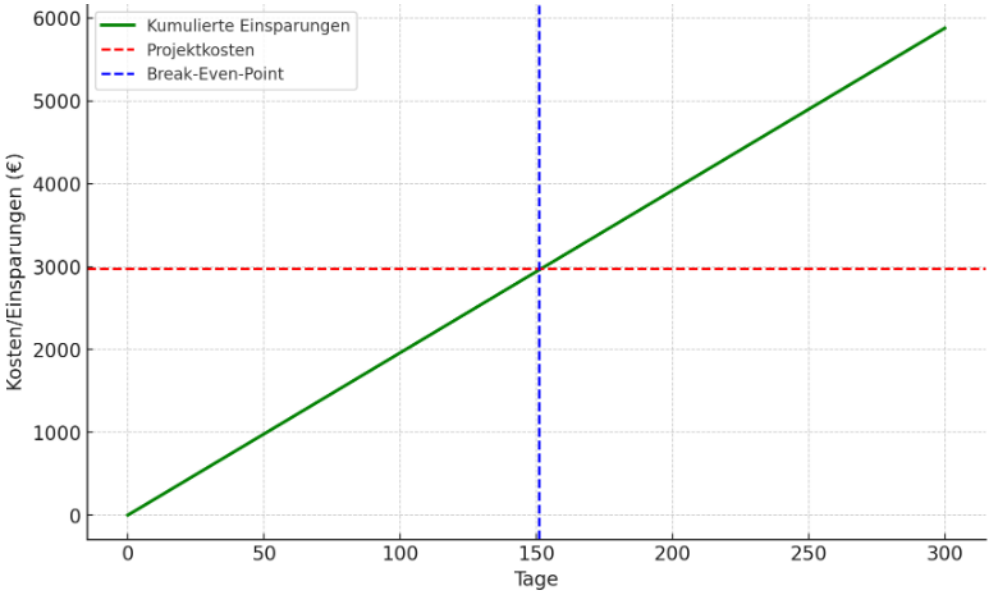

Abbildung 5: Graphische Darstellung der Amortisation

A.5 Aktivitätsdiagramm für den Soll-Zustand

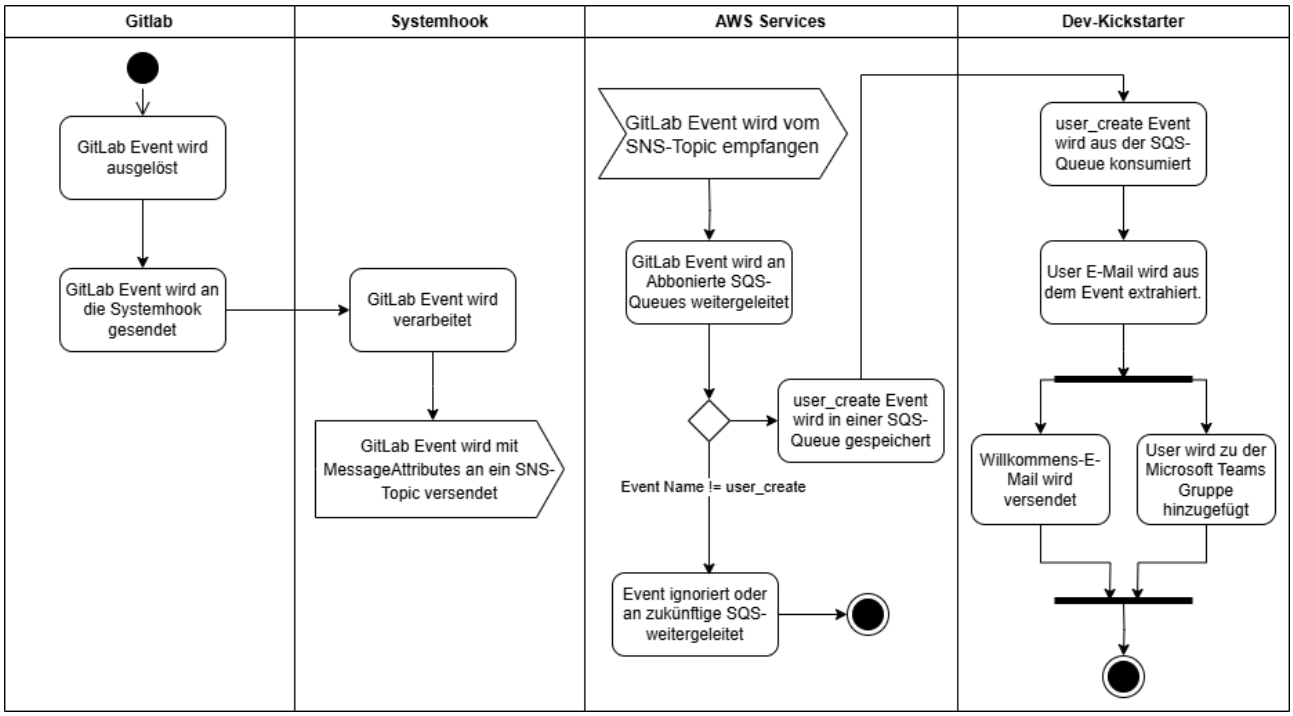

Abbildung 6: Aktivitätsdiagramm für den Soll-Zustand

A.6 Oberflächenentwürfe

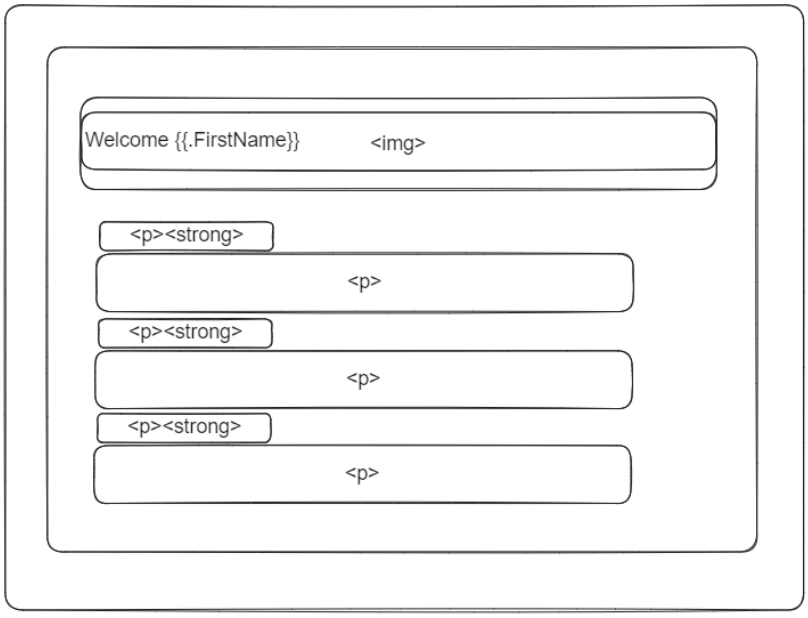

Abbildung 7: Erster Entwurf der E-Mail

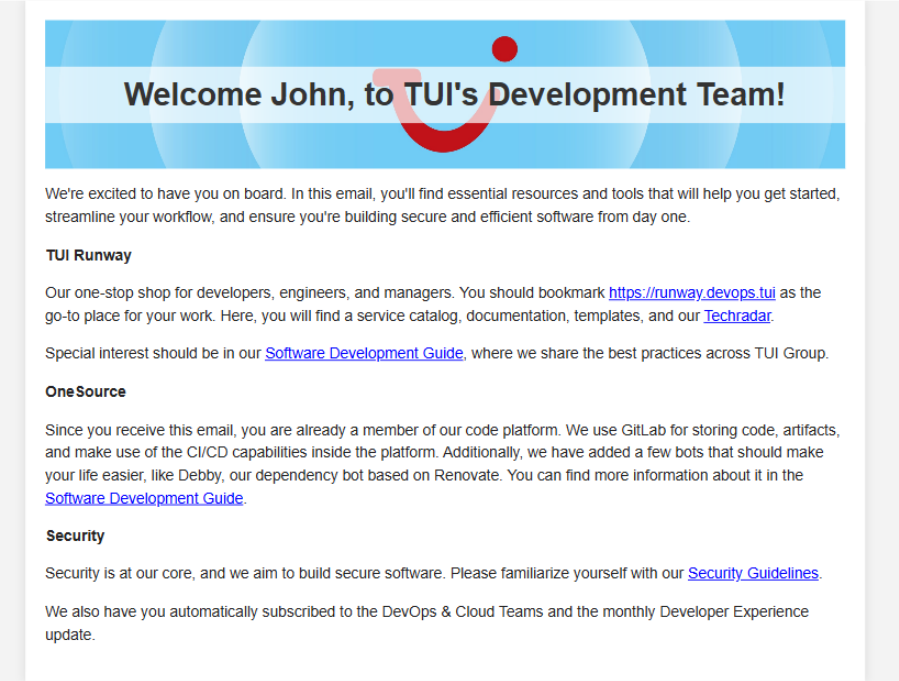

Abbildung 8: Umsetzung in HTML mit CSS

A.7 Screenshot der versendeten E-Mail

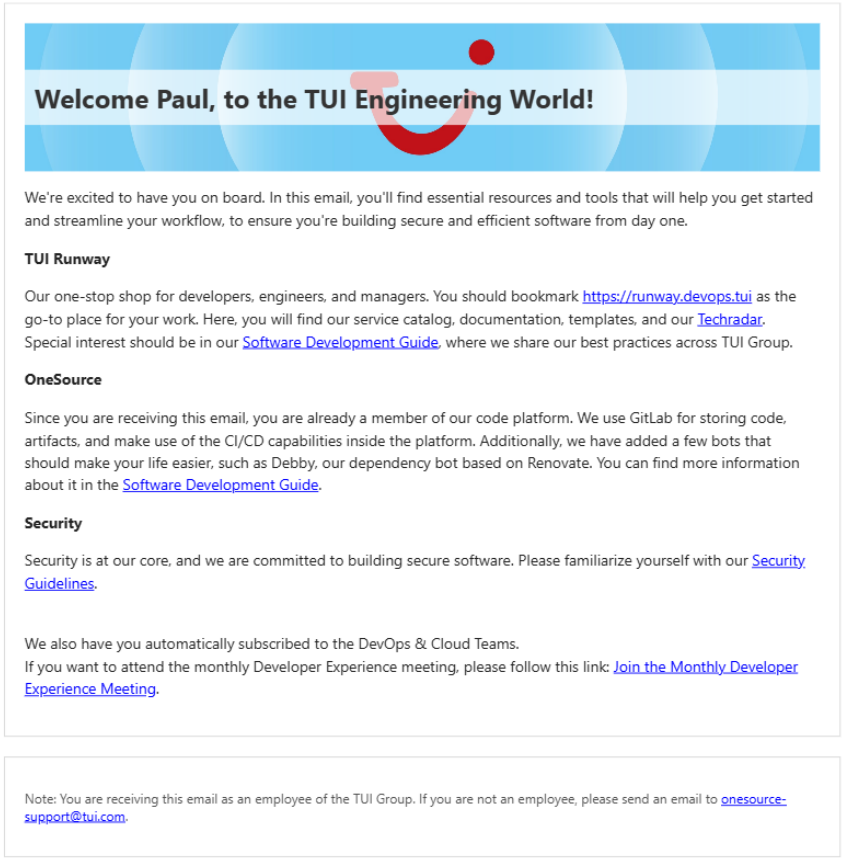

Abbildung 9: Finales Design der E-Mail

A.8 Initialisierung der Hook-Struktur

```
type Hook struct {
  *box.Box
  gitlab     *gitlab.Client
  snsClient *sns.Client
}

func New() (*Hook, error) {
  hook := &Hook{
    Box: box.New(
      box.WithWebServer(),
    ),
  }

  token, ok := os.LookupEnv("GITLAB_TOKEN")
  if !ok {
    return nil, errors.New("GITLAB_TOKEN must be set")
  }

  httpClient := &http.Client{
    Timeout: 30 * time.Second,
  }
  var err error
  hook.gitlab, err = gitlab.NewClient(token, gitlab.WithBaseURL(os.Getenv("GITLAB_URL")), gitlab.
      WithHTTPClient(httpClient))
  if err != nil {
    return nil, fmt.Errorf("failed to create gitlab client %w", err)
  }

  awsCfg, err := awsConfig.LoadDefaultConfig(hook.Context)
  if err != nil {
    return nil, fmt.Errorf("failed to load AWS config: %w", err)
  }

  hook.snsClient = sns.NewFromConfig(awsCfg)

  hook.WebServer.POST("/api/v1/hook", hook.handleHook)

  return hook, nil
}
```

Listing 1: Initialisierung der Hook-Struktur

A.9 Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS

```
func (h *Hook) handleHook(ctx echo.Context) error {
  err := validateToken(ctx.Request().Header)
  if err != nil {
    h.Logger.Warn("unauthorized access", slog.String("remoteAddr", ctx.Request().RemoteAddr))
    return ctx.NoContent(http.StatusUnauthorized)
  }

  payload, err := io.ReadAll(ctx.Request().Body)
  if err != nil {
    return fmt.Errorf("failed to read all bytes from body: %w", err)
  }

  go h.publishMessage(payload)

  return nil
}


func (h *Hook) publishMessage(payload []byte) {
  var event event

  err := json.Unmarshal(payload, &event)
  if err != nil {
    h.Logger.Error("failed to unmarshal body", slog.String("error", err.Error()))
    return
  }

  h.Logger.Debug("event", slog.Any("event", event))
  messageAttributes, err := h.createMessageAttributes(event)
  if err != nil {
    h.Logger.Error("failed to create message attributes", slog.String("eventName", event.EventName), slog.String("
        error", err.Error()))
  }

  input := &sns.PublishInput{
    Message:           aws.String(string(payload)),
    TopicArn:          aws.String(os.Getenv("SNS_TOPIC_ARN")),
    MessageAttributes: messageAttributes,
  }

  message, err := h.snsClient.Publish(context.Background(), input)
  if err != nil {
    h.Logger.Error("error publishing to SNS", slog.String("error", err.Error()))
    return
  }

```

```
46    h.Logger.Info("successfully published message to sns", slog.String("eventName", event.EventName), slog.String("
          messageId", *message.MessageId))
47    h.Logger.Debug("Published message details",
48      slog.String("messageId", *message.MessageId),
49      slog.String("eventName", event.EventName),
50      slog.String("message", string(payload)),
51      slog.String("Message Attributes", formatMessageAttributes(messageAttributes)),
52    )
53  }
54
55  func (h *Hook) createMessageAttributes(event event) (map[string]types.MessageAttributeValue, error) {
56    messageAttributes := make(map[string]types.MessageAttributeValue)
57
58    messageAttributes["eventName"] = types.MessageAttributeValue{
59      DataType:    aws.String("String"),
60      StringValue: aws.String(event.EventName),
61    }
62    switch strings.ToLower(event.EventName) {
63    case "user_create":
64      user, _, err := h.gitlab.Users.GetUser(event.UserID, gitlab.GetUsersOptions{}, nil)
65      if err != nil {
66        return nil, fmt.Errorf("failed to get gitlab user: %w", err)
67      }
68      slog.Info("successfully retrieved gitlab user", slog.String("UserName", user.Name))
69      messageAttributes["isBot"] = types.MessageAttributeValue{
70        DataType:    aws.String("String"),
71        StringValue: aws.String(strconv.FormatBool(user.Bot)),
72      }
73      return messageAttributes, nil
74    default:
75      return messageAttributes, nil
76    }
77  }
```

Listing 2: Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS

A.10 Initialisierung der Kickstarter-Struktur

```
type Config struct {
  DryRun          bool
  HasAzureCreds bool
  EmailSource     string
  SqsUrl          string
  MsTeamsGroupId string
}

type Kickstarter struct {
  *box.Box
  sqsConsumer    *consumer.Consumer
  sesClient      *ses.Client
  emailTemplate *template.Template
  graphClient    *msgraphsdk.GraphServiceClient
  dryRun          bool
  hasAzureCreds bool
  emailSource     string
  sqsUrl          string
  msTeamsGroupId string
}

func New(config *Config) (*Kickstarter, error) {
  kickstarter := &Kickstarter{
    Box: box.New(
      box.WithWebServer(),
    ),
    dryRun:          config.DryRun,
    hasAzureCreds: config.HasAzureCreds,
    emailSource:    config.EmailSource,
    sqsUrl:          config.SqsUrl,
    msTeamsGroupId: config.MsTeamsGroupId,
  }

  awsCfg, err := awsConfig.LoadDefaultConfig(context.Background())
  if err != nil {
    return nil, fmt.Errorf("failed to load AWS config: %w", err)
  }

  if err := kickstarter.initSQSConsumer(awsCfg); err != nil {
    return nil, fmt.Errorf("failed to initialize SQS consumer: %w", err)
  }

  kickstarter.sesClient = ses.NewFromConfig(awsCfg)

  if kickstarter.hasAzureCreds {
    cred, err := azidentity.NewDefaultAzureCredential(nil)
    if err != nil {
      return nil, fmt.Errorf("failed to load azure credentials %w", err)
```

```
49      }
50
51      kickstarter.graphClient, err = msgraphsdk.NewGraphServiceClientWithCredentials(cred, []string{"https://graph.
            microsoft.com/.default"})
52      if err != nil {
53          return nil, fmt.Errorf("failed to create microsoft graph client %w", err)
54      }
55  }
56
57  kickstarter.emailTemplate, err = template.ParseFiles("assets/email.html")
58  if err != nil {
59      return nil, fmt.Errorf("failed to parse file %w", err)
60  }
61
62  return kickstarter, nil
63 }
```

Listing 3: Initialisierung der Kickstarter-Struktur

A.11 Verarbeitung des Gitlab Events und senden der E-Mail/Hinzufügen zur Teams Gruppe

```
// handler is passed to the sqs consumer
// if error returned, message will not be deleted from SQS
func (k *Kickstarter) handler(ctx context.Context, message types.Message) error {
  if message.Body == nil {
    k.Logger.Error("received message with empty body")
  }
  userEmail, err := k.extractEmail(*message.Body)
  if err != nil {
    k.Logger.Error("failed to extract email", slog.String("error", err.Error()))
    return nil
  }

  if userEmail == "" {
    k.Logger.Error("email is invalid or empty", slog.String("email", userEmail))
    return nil
  }

  go func() {
    err := k.sendMail(ctx, userEmail)
    if err != nil {
      k.Logger.Error("failed to send email", slog.String("error", err.Error()))
    }
  }()

  go func() {
    err := k.addUserToMSTeamsGroup(ctx, userEmail, k.msTeamsGroupId)
    if err != nil {
      k.Logger.Error("failed to add user to microsoft group", slog.String("error", err.Error()))
    }
  }()

  return nil
}

// SendEmail sends an email using AWS SES
func (k *Kickstarter) sendMail(ctx context.Context, recipient string) error {
  EmailSource := k.emailSource

  subject := "Let's Get Started in the TUI Engineering World!"

  firstName := k.getName(recipient)

  var buf bytes.Buffer

  err := k.emailTemplate.Execute(&buf, templatedata{FirstName: firstName})
  if err != nil {
    return fmt.Errorf("failed to execute template: %w", err)
```

```
48    }
49
50    input := &ses.SendEmailInput{
51      Destination: &sesTypes.Destination{
52        ToAddresses: []string{recipient},
53      },
54      Message: &sesTypes.Message{
55        Body: &sesTypes.Body{
56          Html: &sesTypes.Content{
57            Charset: aws.String("UTF-8"),
58            Data:    aws.String(buf.String()),
59          },
60        },
61        Subject: &sesTypes.Content{
62          Charset: aws.String("UTF-8"),
63          Data:    aws.String(subject),
64        },
65      },
66      Source: aws.String(EmailSource),
67    }
68
69    if k.dryRun {
70      k.Logger.Info("Dry run mode, skipping sending email")
71      k.Logger.Info("Recipient:", slog.String("recipient", recipient))
72      k.Logger.Info("Subject", slog.String("subject", subject))
73      k.Logger.Info("Source", slog.String("source", EmailSource))
74      k.Logger.Debug("Email HTML Body:", slog.String("htmlBody", buf.String()))
75      return nil
76    }
77
78    _, err = k.sesClient.SendEmail(ctx, input)
79    if err != nil {
80      return fmt.Errorf("failed to send email: %w", err)
81    }
82
83    k.Logger.Info("Email sent successfully", slog.String("recipient", recipient))
84    k.Logger.Info("Source", slog.String("source", EmailSource))
85
86    return nil
87  }
88
89  func (k *Kickstarter) addUserToMSTeamsGroup(ctx context.Context, email string, groupId string) error {
90
91    userId, err := k.getUserIdByEmail(ctx, email)
92    if err != nil {
93      return fmt.Errorf("failed to get userId: %w", err)
94    }
95
96    if k.dryRun {
97      k.Logger.Info("Dry run mode, skipping adding user to Microsoft Teams group")
```

```
98      k.Logger.Info("Resolved User ID", slog.String("userId", *userId))
99      k.Logger.Info("User Email", slog.String("email", email))
100     k.Logger.Info("Microsoft Teams Group ID", slog.String("groupId", groupId))
101     return nil
102   }
103
104   requestBody := graphmodels.NewReferenceCreate()
105   odataId := fmt.Sprintf("https://graph.microsoft.com/v1.0/directoryObjects/{%s}", *userId)
106   requestBody.SetOdataId(&odataId)
107
108   err = k.graphClient.Groups().ByGroupId(groupId).Members().Ref().Post(ctx, requestBody, nil)
109   if err != nil {
110     return fmt.Errorf("failed to add user to microsoft group: %w", err)
111   }
112
113   k.Logger.Info("successfully added user to the microsoft teams group", slog.String("user", email))
114   return nil
115 }
```

Listing 4: Initialisierung der Kickstarter-Struktur

A.12 Deployment yaml für die Systemhook

```yaml
apiVersion: atlas.devops.tui/v1beta1
kind: Application
metadata:
  annotations:
    appify.devops.tui/version: 1.11.2
  labels:
    app: gitlab-to-sns-webhook
  name: gitlab-to-sns-webhook
  namespace: onesource
spec:
  iamRole: arn:aws:iam::500123565959:role/ehda-test-gitlab-to-sns-webhook
  image: registry.source.tui/dev-exp/onesource/gitlab-to-sns-webhook:v1.5.1
  scaling:
    vertical:
      enabled: true
      disruptive: true
  port: 8443
  command:
    - /opt/app/app
  args:
    - --log-level=debug
    - --listen-address=:8443
    - --tls-cert-file=/opt/app/volumes/tls/tls.crt
    - --tls-key-file=/opt/app/volumes/tls/tls.key
  env:
  - name: SNS_TOPIC_ARN
    value: arn:aws:sns:eu-central-1:500123565959:ehda-test-gitlab-to-sns-webhook
  - name: GITLAB_URL
    value: https://test.source.tui
  - name: GITLAB_TOKEN
    valueFrom:
      secretKeyRef:
        name: sns-webhook
        key: GITLAB_TOKEN
  secrets:
  - name: k8s-gitlab-to-sns-webhook
    source: SecretsManager
  ingress:
    host: gitlab-to-sns-webhook.test.devops.tui
    path: /api
  metrics:
    provider: Prometheus
```

Listing 5: Deployment yaml für die Systemhook

A.13 Terraform Infrastructure as Code

```
resource "aws_sns_topic" "gitlab_to_sns_webhook" {
  name              = "${var.service_name}-gitlab-to-sns-webhook"
  kms_master_key_id = aws_kms_key.dev_kickstarter.key_id
  tags = var.tags
}

resource "aws_sqs_queue" "dev_kickstarter" {
  name              = "${var.service_name}-dev-kickstarter"
  kms_master_key_id = aws_kms_key.dev_kickstarter.key_id
  kms_data_key_reuse_period_seconds = 300
  tags = var.tags
}

resource "aws_sns_topic_subscription" "dev_kickstarter_sns_subscription" {
  topic_arn = aws_sns_topic.gitlab_to_sns_webhook.arn
  protocol  = "sqs"
  endpoint  = aws_sqs_queue.dev_kickstarter.arn
   filter_policy  = jsonencode({
    "eventName" : ["user_create"],
    "isBot" : ["false"]
  })
}

data "aws_iam_policy_document" "dev_kickstarter_policy_document" {
  statement {
    effect  = "Allow"

     actions = [
      "kms:Decrypt",
    ]

    resources = [
      aws_kms_key.dev_kickstarter.arn,
    ]
  }

  statement {
    effect  = "Allow"

     actions = [
      "sqs:ReceiveMessage",
      "sqs:DeleteMessage",
      "sqs:GetQueueAttributes",
    ]

    resources = [
      aws_sqs_queue.dev_kickstarter.arn,
    ]
```

```
  }

  statement {
    effect = "Allow"

    actions = [
      "ses:SendEmail",
      "ses:SendRawEmail",
    ]

    resources = [
      "*"
    ]
  }
}
```

Listing 6: Terraform Infrastructure as Code

A.14 Entwicklerdokumentation (Auszug)

GitLab to SNS Webhook

This application receives webhook events from GitLab and publishes them to an AWS SNS topic. It is designed to run as a containerized service with configuration provided via environment variables and command flags.

Table of Contents

- Requirements
- Environment Variables
- Flags
- Endpoints

Requirements

- **AWS IAM Role** with permissions to publish to the SNS topic.
- **GitLab Webhook** pointing to this service's `/api/v1/hook` endpoint.

Environment Variables

Variable	Description	Required	Default
`SNS_TOPIC_ARN`	ARN of the target SNS topic	Yes	N/A
`GITLAB_URL`	GitLab instance URL	Yes	N/A
`GITLAB_TOKEN`	GitLab API token	Yes	N/A
`GITLAB_WEBHOOK_TOKEN`	Token for webhook validation	Yes	N/A
`AWS_REGION`	AWS Region for SNS	No	`eu-central-1`

Flags

Flag	Description	Default
`-log-level`	Log level (debug, info, warn)	`info`

Abbildung 10: Entwicklerdokumentation