



Abschlussprüfung Winter 2024

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Entwicklung von Dev Kickstarter

**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	III
Tabellenverzeichnis	IV
Glossar	V
Listings	VII
Abkürzungsverzeichnis	VIII
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung	2
1.4 Projektschnittstellen	2
1.5 Projektabgrenzung	3
2 Projektplanung	3
2.1 Projektphasen	3
2.2 Ressourcenplanung	4
2.3 Entwicklungsprozess	4
3 Analysephase	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	5
3.2.1 „Make or Buy“-Entscheidung	5
3.2.2 Projektkosten	5
3.2.3 Amortisationsdauer	6
3.3 Nutzwertanalyse	6
4 Entwurfsphase	7
4.1 Zielplattform	7
4.2 Architekturdesign	9
4.3 Entwurf der E-Mail	9
4.4 Maßnahmen zur Qualitätssicherung	9
5 Implementierungsphase	10
5.1 Implementierung der GitLab Systemhook	10
5.2 Implementierung des Dev-Kickstarter	11
5.3 Implementierung der E-Mail	12
6 Einführungsphase	12

6.1	EKS-Deployment	12
6.2	AWS-Deployment	12
7	Abnahmephase	13
8	Dokumentation	13
9	Fazit	14
9.1	Soll-/Ist-Vergleich	14
9.2	Lessons Learned	14
9.3	Ausblick	14
	Literaturverzeichnis	15
	Eidesstattliche Erklärung	16
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Ressourcen Planung	ii
A.3	Iterationsplan	iii
A.4	Oberflächenentwürfe	iii
A.5	Entwicklerdokumentation	vi
A.6	Initialisierung der Hook-Struktur	viii
A.7	Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS	ix
A.8	Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS	xi
A.9	Initialisierung der Kickstarter-Struktur	xiii
A.10	Verarbeitung des Gitlab Events und senden der E-Mail/Hinzufügen zur Teams Gruppe	xv
A.11	Deployment yaml für die Systemhook	xviii
A.12	Terraform Infrastructure as Code	xix
A.13	Benutzerdokumentation	xxi

Abbildungsverzeichnis

1	Finaler Entwurf des Infrastruktur-Designs und Soll-Zustand	8
2	Implementierung der GitLab Systemhook	10
3	Implementierung des Dev-Kickstarter	11
4	Erster Entwurf der E-Mail	iv
5	Umsetzung in HTML mit CSS	iv
6	Finales Design der E-Mail	v

Tabellenverzeichnis

1	Grobe Zeitplanung	3
2	Kostenaufstellung	6
3	Soll-/Ist-Vergleich	14

Glossar

- Microsoft Teams: Kommunikations-Tool, das für Teamarbeit und Kommunikation verwendet wird.
- CI: Continuous Integration - Ein Entwicklungsansatz, bei dem Änderungen an einer Software regelmäßig in ein gemeinsames Repository integriert werden.
- CD: Continuous Deployment - Eine Softwarebereitstellungsmethode, die das automatisierte Deployment von Codeänderungen ermöglicht.
- CSS: Cascading Style Sheets - Eine Stylesheet-Sprache, die verwendet wird, um das Layout von Webdokumenten zu gestalten.
- ERM: Entity-Relationship-Modell - Ein konzeptionelles Datenmodell, das die Beziehungen zwischen Entitäten beschreibt.
- Go: Programmiersprache Go - Eine von Google entwickelte Programmiersprache.
- HTML: Hypertext Markup Language - Die Standardauszeichnungssprache für Dokumente, die im Web angezeigt werden.
- GitLab: Eine Plattform für Versionskontrolle und DevOps, die es Entwicklern ermöglicht, Code zu speichern, zu verwalten und zu teilen.
- GitLab-Systemhooks: Mechanismen in GitLab, die es ermöglichen, auf bestimmte Ereignisse (wie Pushes, Merge-Requests usw.) zu reagieren und automatisierte Aktionen auszulösen, z.B. das Auslösen von CI/CD-Pipelines oder Benachrichtigungen in externen Systemen.
- Onboarding: Der Prozess, durch den neue Mitarbeiter in ein Unternehmen integriert werden und die notwendige Einarbeitung erhalten.
- User Experience (UX): Die Gesamterfahrung eines Benutzers bei der Interaktion mit einem Produkt, insbesondere in Bezug auf Benutzerfreundlichkeit und Zufriedenheit.
- Schnittstelle: Ein definierter Punkt, an dem zwei Systeme oder Komponenten miteinander kommunizieren und Daten austauschen.
- Automatisierung: Der Einsatz von Technologien, um Prozesse ohne menschliches Eingreifen auszuführen, um Effizienz und Genauigkeit zu erhöhen.
- Jira: Eine Projektmanagement-Software, die für die Planung und Nachverfolgung von Aufgaben in Softwareentwicklungsprojekten genutzt wird.
- agil: Ein Entwicklungsansatz, der auf Flexibilität und Anpassungsfähigkeit abzielt und kurze Iterationen sowie regelmäßiges Feedback fördert.

- iterativ: Ein Ansatz, bei dem ein Prozess in wiederholten Zyklen durchgeführt wird, um kontinuierliche Verbesserungen basierend auf Feedback zu ermöglichen.

Listings

1	Initialisierung der Hook-Struktur	viii
2	Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS	ix
3	Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS	xi
4	Initialisierung der Kickstarter-Struktur	xiii
5	Initialisierung der Kickstarter-Struktur	xv
6	Deployment yaml für die Systemhook	xviii
7	Terraform Infrastructure as Code	xix

Abkürzungsverzeichnis

API	Application Programming Interface
TUI	Touristik Union International

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, 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.

1.2 Projektziel

Das Ziel dieses Projekts ist es, die Effizienz und **User Experience** beim **Onboarding** neuer Entwickler durch automatisierte Prozesse zu verbessern. Der aktuelle Onboarding-Prozess ist überwiegend manuell und erfordert zeitaufwändige Schritte wie das Hinzufügen neuer Entwickler zu **Microsoft Teams** Gruppen, das Versenden von E-Mails mit wichtigen Informationen zu internen Prozessen und das Bereitstellen von Zugriffen auf **GitLab**-Repositories sowie weitere Entwicklungsressourcen. Diese manuellen Tätigkeiten sind nicht nur zeitintensiv, sondern bergen auch das Risiko von Fehlern.

Im Rahmen dieses Projekts wird eine Anwendung entwickelt, die auf **GitLab**-Events reagiert und automatisch neue Entwickler in die relevanten **Microsoft Teams** Gruppen integriert. Zusätzlich werden E-Mails mit zentralen Informationen zu internen Abläufen und Dokumentationsstandorten

verschickt, um den Einstieg für neue Entwickler zu erleichtern. Dadurch wird die Produktivität neuer Mitarbeiter gesteigert und gleichzeitig der manuelle Aufwand für das IT-Team deutlich reduziert.

Am Ende des Projekts soll eine Lösung stehen, die sowohl den aktuellen Anforderungen gerecht wird als auch einfach verwaltbar und skalierbar ist, um auf zukünftige Bedürfnisse flexibel reagieren zu können.

1.3 Projektbegründung

Der derzeit manuelle [Onboarding](#)-Prozess für neue Entwickler ist sowohl zeitaufwendig als auch fehleranfällig. 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 bei [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.

Die Motivation hinter dem Projekt ist, die Einarbeitungszeit neuer Entwickler zu verkürzen und gleichzeitig eine höhere Konsistenz und Qualität im [Onboarding](#)-Prozess zu gewährleisten. So wird der Einstieg für neue Entwickler erleichtert, und sie können schneller produktiv arbeiten.

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](#)-Ereignisse 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](#) 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](#).

2.2 Ressourcenplanung

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

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 Kollegen.

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 Feedback direkt in die nächste Phase einfließen zu lassen. Dies förderte nicht nur eine hohe Anpassungsfähigkeit, sondern auch eine kontinuierliche Verbesserung der Qualität des Projekts.

Die [agile](#) Vorgehensweise half dabei, schnelle Anpassungen an neuen Anforderungen vorzunehmen, die in den regelmäßigen Besprechungen und durch Feedback der Kollegen eingebracht wurden. Durch den Einsatz von Tools wie [Microsoft Teams](#) und [Jira](#) wurde die Zusammenarbeit und das Projektmanagement unterstützt, sodass der Fortschritt transparent blieb und die Arbeitsschritte effektiv geplant und dokumentiert werden konnten.

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.

Um dieses Problem zu lösen, haben wir beschlossen, eine GitLab Systemhook zu implementieren, die auf `user_create`-Ereignisse 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-Ereignisse 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-Ereignisse reagieren. Diese Lösung bietet einen zentralen Punkt für GitLab-Ereignisse 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 InfoTec GmbH 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 existieren zwar bereits Softwarelösungen, die Teile des Onboardings automatisieren könnten, jedoch gibt es kein Produkt, das die spezifischen Anforderungen der TUI InfoTec GmbH vollständig abdeckt. Besonders die Integration mit den bereits genutzten [GitLab-SystemHooks](#) und die spezifischen Anpassungen, die für die internen Abläufe nötig sind, erfordern eine maßgeschneiderte Lösung. Aus diesem Grund wurde beschlossen, das Projekt intern umzusetzen, um eine optimale Integration in die bestehende Infrastruktur und eine hohe Flexibilität für zukünftige Anpassungen sicherzustellen.

3.2.2 Projektkosten

Die Kosten für die Durchführung des Projekts setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Laut Tarifvertrag verdient ein Auszubildender im dritten Lehrjahr pro Monat 1450 € Brutto.

$$8 \text{ h/Tag} \cdot 220 \text{ Tage/Jahr} = 1760 \text{ h/Jahr} \quad (1)$$

$$1450 \text{ €/Monat} \cdot 13.3 \text{ Monate/Jahr} = 19285 \text{ €/Jahr} \quad (2)$$

$$\frac{19285 \text{ €/Jahr}}{1760 \text{ h/Jahr}} \approx 10.96 \text{ €/h} \quad (3)$$

Die Kosten, die während der Entwicklung des Projekts anfallen, werden im Folgenden kalkuliert. Die für die Realisierung des Projekts benötigten Personal- und Ressourcenkosten sind von der Personalabteilung festgelegte Pauschalsätze, die nicht weiter angepasst werden dürfen. Der Stundensatz eines Auszubildenden beträgt demzufolge 10.96€, der eines Mitarbeiters 25€. Die Ressourcennutzung umfasst einen Büroarbeitsplatz, die Hardware- und Softwarenutzung sowie Stromkosten. Hierfür wird von der Personalabteilung ein pauschaler Stundensatz von 15€ vorgegeben. Die folgende Tabelle zeigt die detaillierten Projektkosten auf, die für die einzelnen Vorgänge anfallen.

Vorgang	Mitarbeiter	Zeit (h)	Personal (€) ⁶	Ressourcen (€) ⁷	Gesamt (€)
Entwicklungskosten	1 x Auszubildender	80	800,00	1.050,00	1.750,00
Fachgespräch	2 x Mitarbeiter, 1 x Auszubildender	3	180,00	135,00	315,00
Code-Review	1 x Mitarbeiter	4	100,00	60,00	160,00
Abnahme	2 x Mitarbeiter	1	50,00	30,00	80,00
Projektkosten gesamt					2.305,00

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

Das Projekt bringt erhebliche Zeiteinsparungen mit sich, insbesondere für das IT-Team und die neuen Entwickler. Bei einer durchschnittlichen Zeiteinsparung von 10 Minuten pro Tag und 220 Arbeitstagen im Jahr für jeden der 25 Anwender ergibt sich eine jährliche Zeiteinsparung von etwa 917 Stunden. Bei einem kombinierten Stundensatz von 40 € (inklusive Mitarbeiter- und Ressourcenkosten) ergibt sich eine jährliche Einsparung von 36.680 €. Die Amortisationsdauer des Projekts beträgt somit etwa 4 Wochen, was das Projekt sehr kosteneffizient macht.

3.3 Nutzwertanalyse

Neben den monetären Vorteilen bietet das Projekt auch erhebliche nicht-monetäre Vorteile. Die Automatisierung des Onboardings sorgt dafür, dass neue Entwickler sofort Zugriff auf alle relevanten Ressourcen haben, ohne dass Mitarbeiter ständig an das Einrichten von Zugängen erinnert werden müssen. Dies reduziert Ablenkungen und Unterbrechungen im täglichen Arbeitsablauf, was zu einer höheren Produktivität und einer angenehmeren Arbeitsatmosphäre führt. Dadurch wird nicht nur der Einstieg für neue Entwickler erleichtert, sondern auch die Belastung der bestehenden Teams deutlich verringert.

⁶Personalkosten pro Vorgang = Anzahl Mitarbeiter · Zeit · Stundensatz.

⁷Ressourcenbeitrag pro Vorgang = Anzahl Mitarbeiter · Zeit · 15 € (Ressourcenbeitrag pro Stunde).

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-Ereignisse an ein **Amazon Simple Notification Service (SNS)**-Thema sendet. Zweitens eine Anwendung, die spezifische Ereignisse aus einer **Amazon Simple Queue Service (SQS)**-Warteschlange konsumiert und darauf reagiert, wie z.B. die Verarbeitung von `user_create`-Ereignissen. Beide Schnittstellen haben wir in **Go** entwickelt, um unsere bestehende Expertise zu nutzen und den Entwicklungsprozess zu beschleunigen.

Die Wahl der Programmiersprache **Go** 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-Systemereignisse nutzen wir **AWS SNS** in Kombination mit **AWS SQS**. Der SNS-Dienst sammelt die relevanten GitLab-Ereignisse und verteilt sie über SQS an verschiedene Anwendungen. Diese Architektur garantiert eine skalierbare und effiziente Verarbeitung der Ereignisse.

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.

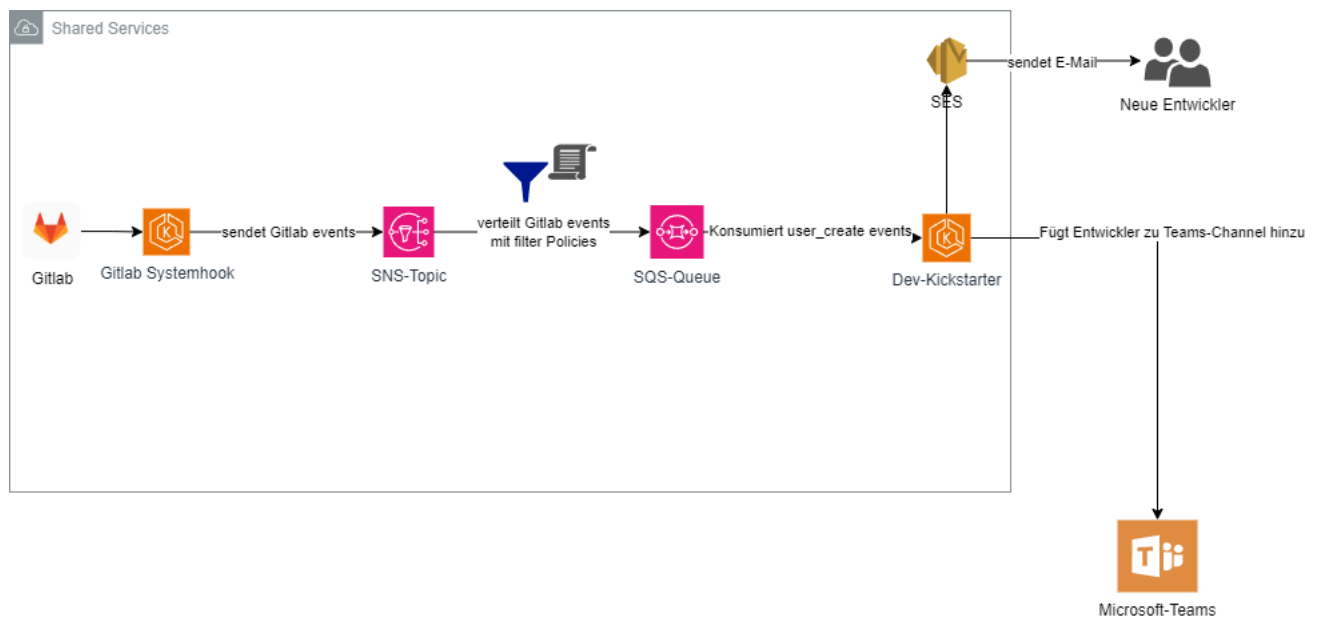


Abbildung 1: Finaler Entwurf des Infrastruktur-Designs und Soll-Zustand

4.2 Architekturdesign

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

Ein in **Go** geschriebener Service konsumiert die relevanten Ereignisse von der SQS-Warteschlange. Bei bestimmten Ereignissen, wie z.B. `user_create`, versendet dieser Service Willkommens-E-Mails über **AWS SES** und fügt neue Entwickler zu Microsoft Teams hinzu. Filter Policies auf den SQS-Warteschlangen sorgen dafür, dass nur relevante Ereignisse 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. 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 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.4: Oberflächenentwürfe auf Seite iii.

4.4 Maßnahmen zur Qualitätssicherung

Um die Qualität des Projektergebnisses zu sichern, wurden verschiedene Maßnahmen ergriffen. 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, Ereignisse wie die Erstellung neuer Benutzer in GitLab zu erfassen und an ein SNS (Simple Notification Service) Topic weiterzuleiten. Dabei stand im Vordergrund, die Systemhook effizient, erweiterbar und in der Lage zu gestalten, Echtzeit-Benachrichtigungen für verschiedene GitLab-Ereignisse zu verarbeiten.

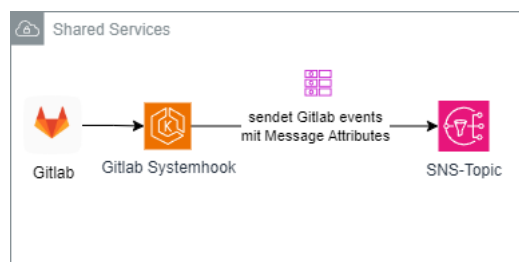


Abbildung 2: Implementierung der GitLab Systemhook

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-Struktur verarbeitet, die im Anhang A.6 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 Webhook-API erhalten. Codeauschnitte des Authentifizierungsprozesses befinden sich im Anhang A.8.

Sobald ein GitLab-Ereignis erkannt 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, wie in Anhang A.8 veranschaulicht. Damit die Nachrichten gezielt gefiltert und ausgewertet werden können, werden sogenannte `MessageAttributes` erstellt. Diese Attribute enthalten Informationen über das jeweilige Ereignis, wie z.B. den Ereignisnamen oder spezifische Merkmale des Benutzers, der das Ereignis ausgelöst hat. Im Falle einer Benutzererstellung wird beispielsweise geprüft, ob der neue Benutzer ein Bot ist, was ebenfalls als Attribut weitergeleitet wird.

Die **MessageAttributes** spielen eine entscheidende Rolle bei der Filterung und ermöglichen eine granulare Weiterleitung von GitLab-Ereignissen. 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, verschiedene Benutzerereignisse zu verarbeiten und Aktionen wie das Versenden von Willkommens-E-Mails und das Hinzufügen neuer Benutzer zu Microsoft Teams-Gruppen automatisiert auszuführen.

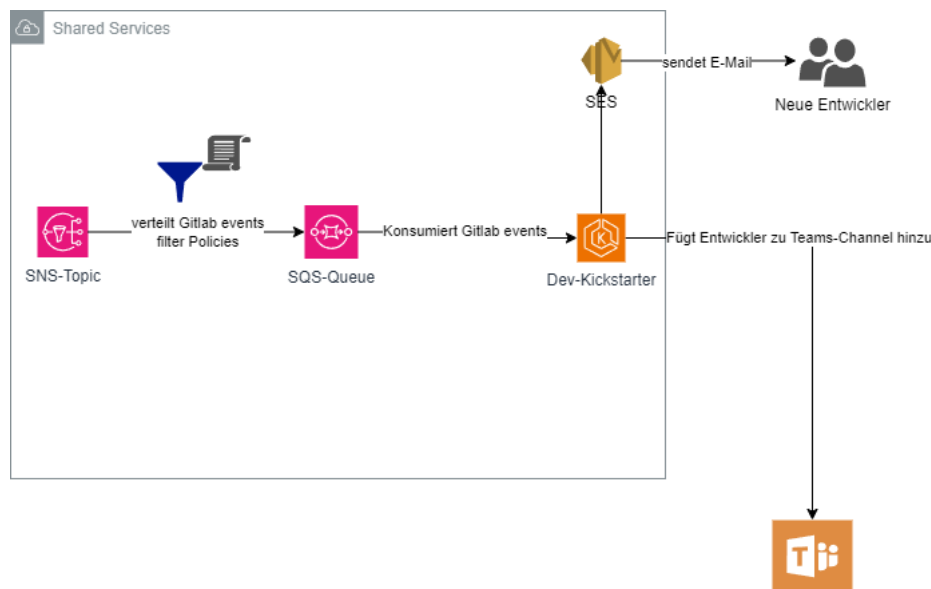


Abbildung 3: Implementierung des Dev-Kickstarter

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 AWS SQS-Queue, die mit Filter Policies erstellt wurde, um nur relevante Ereignisse zu verarbeiten. Die Definition der **Kickstarter**-Struktur ist im Anhang A.9 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 zu versenden. Der zugehörige Code befindet sich im Anhang A.10. Anschließend wird der Benutzer automatisch zu einer definierten Microsoft Teams-Gruppe hinzugefügt, indem die Microsoft Graph API verwendet wird.

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.

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 A.6 eingesehen werden.

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 committet, 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.11 eingesehen werden.

6.2 AWS-Deployment

Für das AWS-Deployment wurden die erforderlichen Infrastrukturkomponenten in der AWS-Umgebung konfiguriert. Dazu gehören Amazon SNS (Simple Notification Service) zur Kommunikation und Amazon SQS (Simple Queue Service) zur Nachrichtenverarbeitung.

Die Bereitstellung dieser Ressourcen erfolgte mithilfe von Infrastructure as Code (IaC) mit Terraform. 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 (Simple Email Service) 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.12 eingesehen werden.

7 Abnahmephase

- Welche Tests (z. B. Unit-, Integrations-, Systemtests) wurden durchgeführt und welche Ergebnisse haben sie geliefert (z. B. Logs von Unit Tests, Testprotokolle der Anwender)?
- Wurde die Anwendung offiziell abgenommen?

Beispiel Ein Auszug eines Unit Tests befindet sich im Anhang A.6: Initialisierung der Hook-Struktur auf Seite viii. Dort ist auch der Aufruf des Tests auf der Konsole des Webservers zu sehen.

8 Dokumentation

- Wie wurde die Anwendung für die Benutzer/Administratoren/Entwickler dokumentiert (z. B. Benutzerhandbuch, API-Dokumentation)?
- Hinweis: Je nach Zielgruppe gelten bestimmte Anforderungen für die Dokumentation (z. B. keine IT-Fachbegriffe in einer Anwenderdokumentation verwenden, aber auf jeden Fall in einer Dokumentation für den IT-Bereich).

Beispiel Ein Ausschnitt aus der erstellten Benutzerdokumentation befindet sich im Anhang A.13: Benutzerdokumentation auf Seite xxi. Die Entwicklerdokumentation wurde mittels PHPDoc¹ automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang A.5: Entwicklerdokumentation auf Seite vi.

¹Vgl. PHPDOC.ORG [2010]

9 Fazit

9.1 Soll-/Ist-Vergleich

- Wurde das Projektziel erreicht und wenn nein, warum nicht?
- Ist der Auftraggeber mit dem Projektergebnis zufrieden und wenn nein, warum nicht?
- Wurde die Projektplanung (Zeit, Kosten, Personal, Sachmittel) eingehalten oder haben sich Abweichungen ergeben und wenn ja, warum?
- Hinweis: Die Projektplanung muss nicht strikt eingehalten werden. Vielmehr sind Abweichungen sogar als normal anzusehen. Sie müssen nur vernünftig begründet werden (z. B. durch Änderungen an den Anforderungen, unter-/überschätzter Aufwand).

Beispiel (verkürzt) Wie in Tabelle 3 zu erkennen ist, konnte die Zeitplanung bis auf wenige Ausnahmen eingehalten werden.

Phase	Geplant	Tatsächlich	Differenz
Entwurfsphase	19 h	19 h	
Analysephase	9 h	10 h	+1 h
Implementierungsphase	29 h	28 h	-1 h
Abnahmetest der Fachabteilung	1 h	1 h	
Einführungsphase	1 h	1 h	
Erstellen der Dokumentation	9 h	11 h	+2 h
Pufferzeit	2 h	0 h	-2 h
Gesamt	70 h	70 h	

Tabelle 3: Soll-/Ist-Vergleich

9.2 Lessons Learned

- Was hat der Prüfling bei der Durchführung des Projekts gelernt (z. B. Zeitplanung, Vorteile der eingesetzten Frameworks, Änderungen der Anforderungen)?

9.3 Ausblick

- Wie wird sich das Projekt in Zukunft weiterentwickeln (z. B. geplante Erweiterungen)?

Literaturverzeichnis

phpdoc.org 2010

PHPDOC.ORG: *phpDocumentor-Website*. Version: 2010. <http://www.phpdoc.org/>, Abruf:
20.04.2010

Eidesstattliche Erklärung

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

Entwicklung von Dev Kickstarter – 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 Oberflächenentwürfe

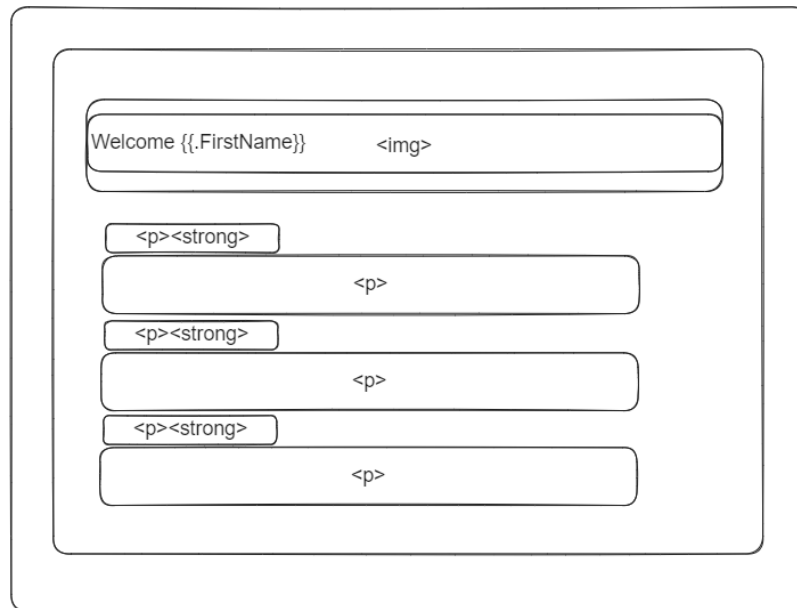


Abbildung 4: Erster Entwurf der E-Mail

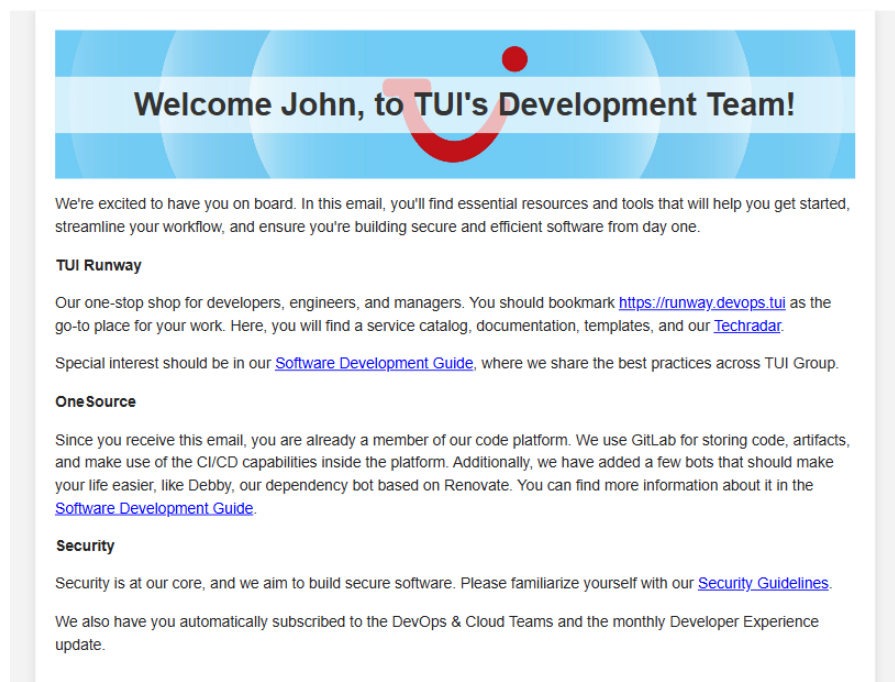


Abbildung 5: Umsetzung in HTML mit CSS

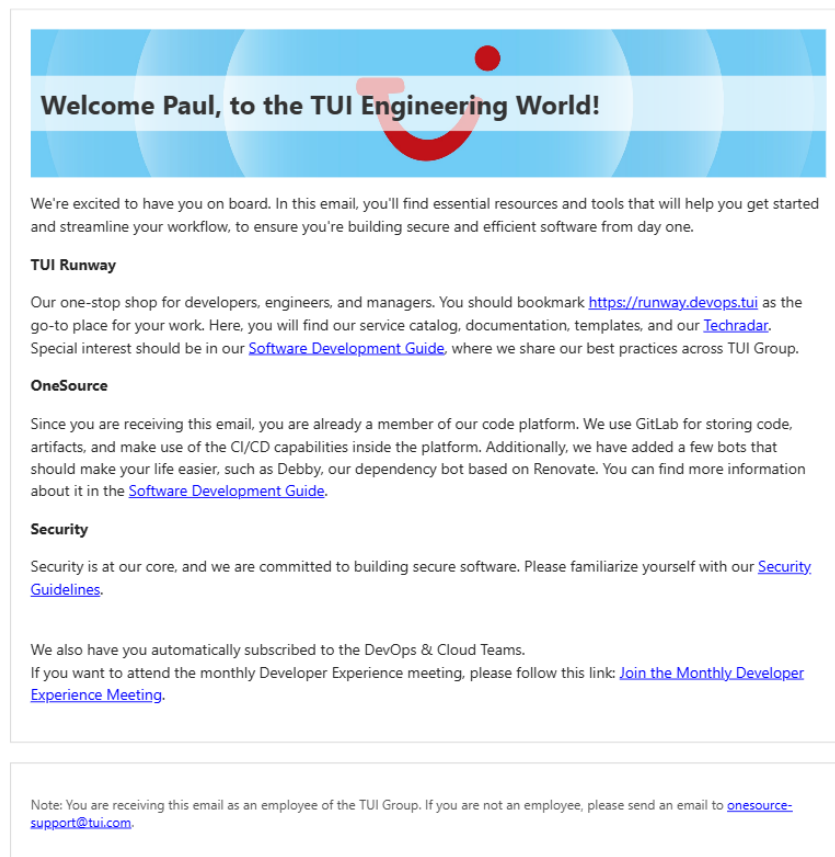


Abbildung 6: Finales Design der E-Mail

A.5 Entwicklerdokumentation

lib-model

[class tree: lib-model] [index: lib-model] [all elements]

Packages:
lib-model

Files:
Naturalmodulename.php

Classes:
Naturalmodulename

Class: Naturalmodulename

Source Location: /Naturalmodulename.php

Class Overview

```

BaseNaturalmodulename
|
--Naturalmodulename

```

Subclass for representing a row from the 'NaturalModulename' table.

Methods

- [__construct](#)
- [getNaturalTags](#)
- [getNaturalWikis](#)
- [loadNaturalModuleInformation](#)
- [__toString](#)

Class Details

[line 10]
Subclass for representing a row from the 'NaturalModulename' table.

Adds some business logic to the base.

[\[Top \]](#)

Class Methods

constructor [__construct](#) [line 56]

```
Naturalmodulename __construct( )
```

Initializes internal state of Naturalmodulename object.

Tags:

see: parent::__construct()
access: public

[\[Top \]](#)

method [getNaturalTags](#) [line 68]

```
array getNaturalTags( )
```

Returns an Array of NaturalTags connected with this Modulename.

Tags:

return: Array of NaturalTags
access: public

[\[Top \]](#)

method getNaturalWikis [line 83]

```
array getNaturalWikis( )
```

Returns an Array of NaturalWikis connected with this Modulename.

Tags:

return: Array of NaturalWikis
access: public

[\[Top \]](#)

method loadNaturalModuleInformation [line 17]

```
ComparedNaturalModuleInformation  
loadNaturalModuleInformation( )
```

Gets the ComparedNaturalModuleInformation for this NaturalModulename.

Tags:

access: public

[\[Top \]](#)

method __toString [line 47]

```
string __toString( )
```

Returns the name of this NaturalModulename.

Tags:

access: public

[\[Top \]](#)

Documentation generated on Thu, 22 Apr 2010 08:14:01 +0200 by [phpDocumentor 1.4.2](#)

A.6 Initialisierung der Hook-Struktur

```
1 type Hook struct {
2     *box.Box
3     gitlab *gitlab.Client
4     snsClient *sns.Client
5 }
6
7 func New() (*Hook, error) {
8     hook := &Hook{
9         Box: box.New(
10             box.WithWebServer(),
11         ),
12     }
13
14     token, ok := os.LookupEnv("GITLAB_TOKEN")
15     if !ok {
16         return nil, errors.New("GITLAB_TOKEN must be set")
17     }
18
19     httpClient := &http.Client{
20         Timeout: 30 * time.Second,
21     }
22     var err error
23     hook.gitlab, err = gitlab.NewClient(token, gitlab.WithBaseURL(os.Getenv("GITLAB_URL")), gitlab.
24         WithHTTPClient(httpClient))
25     if err != nil {
26         return nil, fmt.Errorf("failed to create gitlab client %w", err)
27     }
28
29     awsCfg, err := awsConfig.LoadDefaultConfig(hook.Context)
30     if err != nil {
31         return nil, fmt.Errorf("failed to load AWS config: %w", err)
32     }
33
34     hook.snsClient = sns.NewFromConfig(awsCfg)
35
36     hook.WebServer.POST("/api/v1/hook", hook.handleHook)
37
38     return hook, nil
39 }
```

Listing 1: Initialisierung der Hook-Struktur

A.7 Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS

```

1 func (h *Hook) handleHook(ctx echo.Context) error {
2     err := validateToken(ctx.Request().Header)
3     if err != nil {
4         h.Logger.Warn("unauthorized access", slog.String("remoteAddr", ctx.Request().RemoteAddr))
5         return ctx.NoContent(http.StatusUnauthorized)
6     }
7
8     payload, err := io.ReadAll(ctx.Request().Body)
9     if err != nil {
10        return fmt.Errorf("failed to read all bytes from body: %w", err)
11    }
12
13    go h.publishMessage(payload)
14
15    return nil
16 }
17
18
19 func (h *Hook) publishMessage(payload []byte) {
20     var event event
21
22     err := json.Unmarshal(payload, &event)
23     if err != nil {
24         h.Logger.Error("failed to unmarshal body", slog.String("error", err.Error()))
25         return
26     }
27
28     h.Logger.Debug("event", slog.Any("event", event))
29     messageAttributes, err := h.createMessageAttributes(event)
30     if err != nil {
31         h.Logger.Error("failed to create message attributes", slog.String("eventName", event.EventName), slog.String("error", err.Error()))
32     }
33
34     input := &sns.PublishInput{
35         Message:      aws.String(string(payload)),
36         TopicArn:      aws.String(os.Getenv("SNS_TOPIC_ARN")),
37         MessageAttributes: messageAttributes,
38     }
39
40     message, err := h.snsClient.Publish(context.Background(), input)
41     if err != nil {
42         h.Logger.Error("error publishing to SNS", slog.String("error", err.Error()))
43         return
44     }
45 }

```

```

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.8 Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS

```

1 func (h *Hook) handleHook(ctx echo.Context) error {
2     err := validateToken(ctx.Request().Header)
3     if err != nil {
4         h.Logger.Warn("unauthorized access", slog.String("remoteAddr", ctx.Request().RemoteAddr))
5         return ctx.NoContent(http.StatusUnauthorized)
6     }
7
8     payload, err := io.ReadAll(ctx.Request().Body)
9     if err != nil {
10        return fmt.Errorf("failed to read all bytes from body: %w", err)
11    }
12
13    go h.publishMessage(payload)
14
15    return nil
16 }
17
18
19 func (h *Hook) publishMessage(payload []byte) {
20     var event event
21
22     err := json.Unmarshal(payload, &event)
23     if err != nil {
24         h.Logger.Error("failed to unmarshal body", slog.String("error", err.Error()))
25         return
26     }
27
28     h.Logger.Debug("event", slog.Any("event", event))
29     messageAttributes, err := h.createMessageAttributes(event)
30     if err != nil {
31         h.Logger.Error("failed to create message attributes", slog.String("eventName", event.EventName), slog.String("error", err.Error()))
32     }
33
34     input := &sns.PublishInput{
35         Message:      aws.String(string(payload)),
36         TopicArn:      aws.String(os.Getenv("SNS_TOPIC_ARN")),
37         MessageAttributes: messageAttributes,
38     }
39
40     message, err := h.snsClient.Publish(context.Background(), input)
41     if err != nil {
42         h.Logger.Error("error publishing to SNS", slog.String("error", err.Error()))
43         return
44     }
45 }

```

```

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 3: Verarbeitung eingehender Webhooks und Veröffentlichung von Ereignissen über AWS SNS

A.9 Initialisierung der Kickstarter-Struktur

```

1 type Config struct {
2     DryRun      bool
3     HasAzureCreds bool
4     EmailSource string
5     SqsUrl      string
6     MsTeamsGroupId string
7 }
8
9 type Kickstarter struct {
10    *box.Box
11    sqsConsumer *consumer.Consumer
12    sesClient   *ses.Client
13    emailTemplate *template.Template
14    graphClient  *msgraphsdk.GraphServiceClient
15    dryRun       bool
16    hasAzureCreds bool
17    emailSource  string
18    sqsUrl       string
19    msTeamsGroupId string
20 }
21
22 func New(config *Config) (*Kickstarter, error) {
23     kickstarter := &Kickstarter{
24         Box: box.New(
25             box.WithWebServer(),
26         ),
27         dryRun:      config.DryRun,
28         hasAzureCreds: config.HasAzureCreds,
29         emailSource:  config.EmailSource,
30         sqsUrl:       config.SqsUrl,
31         msTeamsGroupId: config.MsTeamsGroupId,
32     }
33
34     awsCfg, err := awsConfig.LoadDefaultConfig(context.Background())
35     if err != nil {
36         return nil, fmt.Errorf("failed to load AWS config: %w", err)
37     }
38
39     if err := kickstarter.initSQSConsumer(awsCfg); err != nil {
40         return nil, fmt.Errorf("failed to initialize SQS consumer: %w", err)
41     }
42
43     kickstarter.sesClient = ses.NewFromConfig(awsCfg)
44
45     if kickstarter.hasAzureCreds {
46         cred, err := azidentity.NewDefaultAzureCredential(nil)
47         if err != nil {
48             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 4: Initialisierung der Kickstarter-Struktur

A.10 Verarbeitung des Gitlab Events und senden der E-Mail/Hinzufügen zur Teams Gruppe

```

1 // handler is passed to the sqs consumer
2 // if error returned, message will not be deleted from SQS
3 func (k *Kickstarter) handler(ctx context.Context, message types.Message) error {
4     if message.Body == nil {
5         k.Logger.Error("received message with empty body")
6     }
7     userEmail, err := k.extractEmail(*message.Body)
8     if err != nil {
9         k.Logger.Error("failed to extract email", slog.String("error", err.Error()))
10        return nil
11    }
12
13    if userEmail == "" {
14        k.Logger.Error("email is invalid or empty", slog.String("email", userEmail))
15        return nil
16    }
17
18    k.Logger.Info("sending mail to", slog.String("Email:", userEmail))
19    err = k.sendMail(ctx, userEmail)
20    if err != nil {
21        k.Logger.Error("failed to send email", slog.String("error", err.Error()))
22    }
23
24    err = k.addUserToMSTeamsGroup(ctx, userEmail, k.msTeamsGroupId)
25    if err != nil {
26        k.Logger.Error("failed to add user to microsoft group", slog.String("error", err.Error()))
27    }
28
29    return nil
30 }
31
32 // SendEmail sends an email using AWS SES
33 func (k *Kickstarter) sendMail(ctx context.Context, recipient string) error {
34     EmailSource := k.emailSource
35
36     subject := "Let's Get Started in the TUI Engineering World!"
37
38     firstName, lastName := k.getName(recipient)
39
40     var buf bytes.Buffer
41
42     err := k.emailTemplate.Execute(&buf, templatedata{FirstName: firstName, LastName: lastName})
43     if err != nil {
44         return fmt.Errorf("failed to execute template: %w", err)
45     }
46
47     input := &ses.SendEmailInput{

```



```

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

```

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

Listing 5: Initialisierung der Kickstarter-Struktur

A.11 Deployment yaml für die Systemhook

```

1 apiVersion: atlas.devops.tui/v1beta1
2 kind: Application
3 metadata:
4   annotations:
5     appify.devops.tui/version: 1.11.2
6   labels:
7     app: gitlab-to-sns-webhook
8     name: gitlab-to-sns-webhook
9     namespace: onesource
10 spec:
11   iamRole: arn:aws:iam::500123565959:role/ehda-test-gitlab-to-sns-webhook
12   image: registry.source.tui/dev-exp/onesource/gitlab-to-sns-webhook:v1.5.1
13   scaling:
14     vertical:
15       enabled: true
16       disruptive: true
17   port: 8443
18   command:
19     - /opt/app/app
20   args:
21     - --log-level=debug
22     - --listen-address=:8443
23     - --tls-cert-file=/opt/app/volumes/tls/tls.crt
24     - --tls-key-file=/opt/app/volumes/tls/tls.key
25   env:
26     - name: SNS_TOPIC_ARN
27       value: arn:aws:sns:eu-central-1:500123565959:ehda-test-gitlab-to-sns-webhook
28     - name: GITLAB_URL
29       value: https://test.source.tui
30     - name: GITLAB_TOKEN
31       valueFrom:
32         secretKeyRef:
33           name: sns-webhook
34           key: GITLAB_TOKEN
35   secrets:
36     - name: k8s-gitlab-to-sns-webhook
37       source: SecretsManager
38   ingress:
39     host: gitlab-to-sns-webhook.test.devops.tui
40     path: /api
41   metrics:
42     provider: Prometheus

```

Listing 6: Deployment yaml für die Systemhook

A.12 Terraform Infrastructure as Code

```

1 resource "aws_sns_topic" "gitlab_to_sns_webhook" {
2   name           = "${var.service_name}-gitlab-to-sns-webhook"
3   kms_master_key_id = aws_kms_key.dev_kickstarter.key_id
4   tags = var.tags
5 }
6
7 resource "aws_sqs_queue" "dev_kickstarter" {
8   name           = "${var.service_name}-dev-kickstarter"
9   kms_master_key_id = aws_kms_key.dev_kickstarter.key_id
10  kms_data_key_reuse_period_seconds = 300
11  tags = var.tags
12 }
13
14 resource "aws_sns_topic_subscription" "dev_kickstarter_sns_subscription" {
15   topic_arn = aws_sns_topic.gitlab_to_sns_webhook.arn
16   protocol  = "sqs"
17   endpoint  = aws_sqs_queue.dev_kickstarter.arn
18   filter_policy = jsonencode({
19     "eventName" : ["user_create"],
20     "isBot" : ["false"]
21   })
22 }
23
24 data "aws_iam_policy_document" "dev_kickstarter_policy_document" {
25   statement {
26     effect = "Allow"
27
28     actions = [
29       "kms:Decrypt",
30     ]
31
32     resources = [
33       aws_kms_key.dev_kickstarter.arn,
34     ]
35   }
36
37   statement {
38     effect = "Allow"
39
40     actions = [
41       "sqs:ReceiveMessage",
42       "sqs:DeleteMessage",
43       "sqs:GetQueueAttributes",
44     ]
45
46     resources = [
47       aws_sqs_queue.dev_kickstarter.arn,
48     ]






```

```
49  }
50
51  statement {
52    effect = "Allow"
53
54    actions = [
55      "ses:SendEmail",
56      "ses:SendRawEmail",
57    ]
58
59    resources = [
60      "*"
61    ]
62  }
63 }
```

Listing 7: Terraform Infrastructure as Code

A.13 Benutzerdokumentation

Ausschnitt aus der Benutzerdokumentation:

Symbol	Bedeutung global	Bedeutung einzeln
	Alle Module weisen den gleichen Stand auf.	Das Modul ist auf dem gleichen Stand wie das Modul auf der vorherigen Umgebung.
	Es existieren keine Module (fachlich nicht möglich).	Weder auf der aktuellen noch auf der vorherigen Umgebung sind Module angelegt. Es kann also auch nichts übertragen werden.
	Ein Modul muss durch das Übertragen von der vorherigen Umgebung erstellt werden.	Das Modul der vorherigen Umgebung kann übertragen werden, auf dieser Umgebung ist noch kein Modul vorhanden.
	Auf einer vorherigen Umgebung gibt es ein Modul, welches übertragen werden kann, um das nächste zu aktualisieren.	Das Modul der vorherigen Umgebung kann übertragen werden um dieses zu aktualisieren.
	Ein Modul auf einer Umgebung wurde entgegen des Entwicklungsprozesses gespeichert.	Das aktuelle Modul ist neuer als das Modul auf der vorherigen Umgebung oder die vorherige Umgebung wurde übersprungen.