htw saar

EmRoLab
Embedded Robotics Lab

Ausarbeitung

Software-Architektur

an der Hochschule für Technik und Wirtschaft des Saarlandes

im Studiengang Kommunikationsinformatik

der Fakultät für Ingenieurwissenschaften

Continuous Integration / Continuous Delivery and Deployment

vorgelegt von

Sayed Mustafa Sajadi

Mhd Yaman Aljazairi

Anas Alajaji

betreut und begutachtet von

Prof. Dr. Markus Esch

Saarbrücken, Tag. Monat Jahr

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, Tag. Monat Jahr

Sayed Mustafa Sajadi
Mhd Yaman Aljazairi
Anas Alajaji

Zusammenfassung

Kurze Zusammenfassung des Inhaltes in deutscher Sprache, der Umfang beträgt zwischen einer halben und einer ganzen DIN A4-Seite.

Orientieren Sie sich bei der Aufteilung bzw. dem Inhalt Ihrer Zusammenfassung an Kent Becks Artikel: `http://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html`.

Inhaltsverzeichnis

1 Einleitung

1.0.1 Motivation

Die Notwendigkeit von Software-Updates ist sehr wichtig, um die korrekte Funktion der eingebetteten Systeme fehlerfrei und problemlos zu gewährleisten, denn die Software-Bugs hatten katastrophale Folgen, insbesondere in den Bereichen Luft- und Raumfahrt, Automotive und Medizin. Zwischen Juni 1985 und Januar 1987 verabreichte ein computergesteuertes Therapiegerät namens "Therach-25" sechs Personen eine große Anzahl von Überdosierungen, die zu schweren Verletzungen und zum Tod führten. Der Software-Entwicklern sollten Funktionen zur Verfügung gestellt werden, die den Software-Entwicklungsprozess beschleunigen und Software-Updates über das Internet ermöglichen. Denn ein manuelles Update, zum Beispiel über USB-Sticks, zum einen verlangsamt der Übertragungsprozess und zum anderen kann während der Übertragung zu menschlichen Fehlern führen. Darüber hinaus ist dieser Prozess anfällig, wie Fiat Chrysler 2015 damit begonnen hat, einen Hotfix für Millionen von Fahrzeugen per Post über einen USB-Stick zu verteilen, der es Hackern ermöglicht, Briefe und USB-Sticks zu fälschen.

1.0.2 Aufgabenstellung und Zielsetzung

In dieser Arbeit soll die verschiedenen Release-Management-Methoden erklärt und insbesondere die Methode Continuous Integration, Delivery and Deployment (CICD) herausgearbeitet und die verschiedenen dabei eingesetzten Tools sowie deren Unterschiede erläutert werden. Als nächstes wird auf die Frage eingegangen, welche CI/CD-Auswirkungen die Softwarearchitektur und die Entwicklungsprozesse von haben. Darüber hinaus wird die Anwendung dieser Methode anhand eines Fallbei-

spiels näher untersucht, in dem der Fall der Anwendung eines Software-Updates eines Endgeräts über das Internet tatsächlich betrachtet und unter Verwendung einer Container-Technologie namens Docker umgesetzt wird.

2 Grundlagen

3 Release Management Methoden

4 Continuous Integration und Continuous Delivery

5 Fallstudie

Es sollen zwei Ampeln nachgebaut werden, die in einer experimentellen Umgebung die Funktionsweise realer Ampeln zeigen. Eine der Ampeln soll die gelbe LED blinken und die andere soll im normalen Betrieb sein. In diesem Projekt soll der CI/CD Methode von Development bis zum Deployment angewendet werden.

5.1 Anforderungen

In diesem Abschnitt werden die funktionale und nicht-funktionale Anforderungen für die Realisierung des Projekts analysiert.

5.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die gewünschte Funktionalität und das Verhalten des Systems, die in Tabelle5.1 aufgelistet sind.

5.1.2 Nicht-Funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben die Qualität der oben genannten Funktionen, die erreicht werden müssen. Daher haben sie einen erheblichen Einfluss auf Ressourcenverbrauch, Entwicklung und Wartung. Darüber hinaus tragen diese Anforderungen dazu bei, die Akzeptanz des Systems zu verbessern. Einige dieser Anforderungen werden im Folgenden aufgelistet und erörtert.

- **Zuverlässig:** Zuverlässigkeit stellt die Grundvoraussetzung für die Akzeptanz des Systems dar. Das korrekte Verhalten und der Übergang in einen sicheren

Must-have	1. Simulierung zwei Verkehrsampeln mithilfe von Raspberry Pi's und LEDs 2. Automatische Steuerung der LEDs durch GPIO Schnittstelle 3. Entwicklung einer Software für das Blinken der gelben LED 4. Entwicklung einer Software für den Normalbetrieb der Ampel 5. Automatische Software-Test 6. Automatische Bereitstellung (release) der Software 7. Implementierung von CI/CD Software-Entwicklungsmethode
Should-have	1. Automatische Containerisierung der für das Ampelsystem entwickelten Software mithilfe von Docker 2. Automatische Veröffentlichung von containerisierten Software auf Docker Hub 3. Orchestrierung der Docker-Containers mithilfe von Kubernetes
Could-have	1. Automatische Übertragung (deploy) der Software an das Endgerät

Tabelle 5.1: Funktionale Anforderungen

Zustand im Fehlerfall muss immer gewährleistet sein. Falls die Übertragung der Software in einem inkorrekten Zustand endet, müssen Entwickler in der Lage sein, das System in den korrekten Zustand wiederherzustellen.

- **Skalierbar:** Das Projekt soll skalierbar sein. Wenn man eine neue Ampel nachbauen möchte, sollte es einfach und nicht kompliziert sein.
- **Fehlertoleranz:** Im Falle eines Fehlers sollte die Pipeline die weitere Ausführung anderer Schritte stoppen, wodurch Fehler vermieden werden, die aufgrund einer nachfolgenden Ausführung auftreten können.
- **Robust:** Der Pull-Request soll nicht gemergt werden, bevor die Fehlerfreiheit des Programms durch bestehenden Unit-Test bestätigt wird.
- **Zeiteffizient:** Der gesamte Prozess, von der Entwicklung bis zur Auslieferung,

muss in kurzer Zeit und ohne Unterbrechung erfolgen.

- **Echtzeitüberwachung:** Während der Übertragung des Software-Updates soll dieses in Echtzeit überwacht werden.

5.2 Konzept

5.2.1 Geeignete Software-Update-Strategie der Ampelanlage

Heutzutage sind mehr Geräte mit dem Internet verbunden als je zuvor, was bedeutet, dass Software-Updates über das Internet und nicht über eine traditionelle Schnittstelle bereitgestellt werden müssen. Das spart viel Zeit und Geld bei der Wartung der Software und entlastet vor allem den Facharbeitern. Darüber hinaus wird die Fehlersuche im Fehlerfall durch den einfachen Fernzugriff auf das betroffene System erheblich vereinfacht. Diese Strategie wird als Softwar Over The Air (SOTA) benannt. Durch diese Strategie wird die Echtzeitüberwachung der Übertragung ermöglicht, was das ganze Software-Update der Ampelanlage übersichtlicher macht. Darüber hinaus wird durch SOTA Flexibilität bei der Software-Update erreicht, was bedeutet, dass jede Softwareversion mit einem einzigen Klick auf das System übertragen werden kann. Dies macht die Strategie einfach zu handhaben. Neben all diesen Vorteilen hat das System noch einen weiteren Vorteil, nämlich die Anzahl der Updates ist nicht begrenzt und somit nicht zeitabhängig.

5.2.2 Software Update-Architektur der Ampelanlage

Um die oben genannten Vorteile nutzen zu können, ist es notwendig, eine Internet-Schnittstelle zur Außenwelt zu schaffen. In diesem Zusammenhang und aufgrund der im Raspberry Pi integrierten Internetschnittstelle eignet sich der Raspberry Pi für dieses System, und weil es sich um ein Minicomputer handelt, erfüllt die in diesem Projekt aufgelistete Anforderungen. Die Abbildung 5.1 zeigt, wie ein Entwickler, über das im Raspberry Pi integrierte Internetschnittstelle ein Update auf der Ampelanlage übertragen kann, nachdem er die Software auf GitLab hochgeladen hat.

Die Verteilung der korrekten Software-Version an den entsprechenden Ampelanlage erfolgt über Raspberry Pis, die direkt mit den Ampelanlagen angeschlossen sind.

Die Software-Update-Architektur in Abbildung5.1 zeigt deutlich, dass für die Anforderungen, die im Abschnitt 5.1 aufgelistet sind, eine hierarchische Software-Deployment Topologie dafür geeignet ist.

Die Architektur von Kubernetes, die im Kapital ?? beschrieben wurde, bietet die Möglichkeit ein hierarchische Firmware-Deployment aufzubauen.

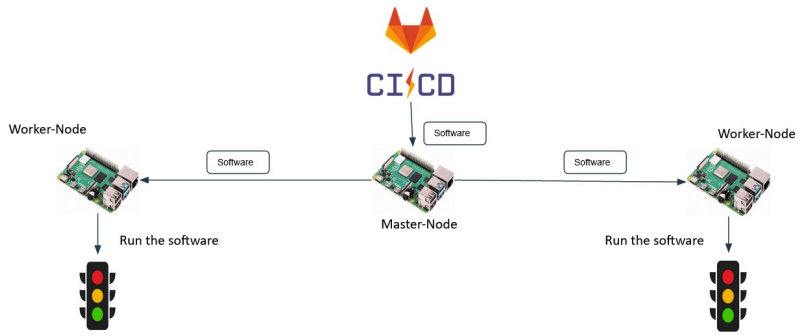

Abbildung 5.1: Übersicht der Software Update

5.2.3 Auswahl der geeigneten Release-Managementmethode

Die Unterschiede der Release-Managementmethoden zeigen, dass die CD-Methode es ermöglicht, qualitativ hochwertige Software-Release in kurze Zeit auf einem Endgerät zu übertragen. CD Methode bietet automatisierte Build- , Test- und Deploymentschritte, um vom Menschen verursachte Fehler zu vermeiden, die bei der manuellen Durchführung der Phasen auftreten können. Außerdem können in jeder Phase Probleme auftreten. Abhängigkeiten zwischen den Schritten können eine schnelle Fehlerkorrektur garantieren. Tritt in einer Phase ein Fehler auf, wird die Pipeline unterbrochen und somit weitere Fehler verhindert. Die Software kann nahtlos, sicher, zuverlässig und wiederholt auf dem Endgerät übertragen werden. Darüber hinaus kann der Entwickler den Fortschritt der Phasen beobachten, was hilft, wenn während

der Phase ein Fehler auftritt, kann der Entwickler nur ab dieser Phase und nicht von allen Phasen aus iterieren. Bei anderen Release-Managementmethoden benötigen Produkthersteller Releasemanager, um Entscheidungen darüber zu treffen, wann Releases erstellt werden, wie einzelne Schritte ausgeführt werden und wann Software ausgeliefert werden soll. Die Methode CD ermöglicht es dem Release Manager, sich mehr auf die operative Seite zu konzentrieren, wie beispielsweise die Erstellung eines automatisierten Prozesses und Workflows zur sicheren Migration von Code in das Endprodukt, anstatt sich auf die Planungs-, Entwicklungs- und Testphasen zu konzentrieren.

5.2.4 Auswahl der geeigneten Release-Strategie

Die Wahl einer geeigneten Release-Strategie hängt vom Hersteller des Produkts und dem Produkt selbst ab. Bei modernen medizinischen Geräten, Fahrzeuge und IoT Geräte sind Fehlerbehebung und neue Funktionen für einen besseren Service für die Kunden unerlässlich. Daher wird in dieser Fallstudie eine flexible Release-Strategie bevorzugt. Die Schritte im Release-Management-Prozess bleiben generisch, sodass der Wechsel zu anderen Release-Strategien einfach sein kann. Eine flexible Release-Strategie sieht laut Dr.-Ing. Kühn in der Regel die Planung von Änderungen an einem bestimmten Produkt vor[1].

5.3 Entwicklungsvorgang

Als Team an der Entwicklung von Software zu arbeiten, ist eine ziemliche Herausforderung, und braucht ein gutes Management. Diesbezüglich soll ein Konzept entwickelt werden, die verteilte Entwicklung von Software der simulierten Ampelanlage zu organisieren.

Für die Umsetzung dieses Konzeptes in der Praxis ist die richtige Wahl eines VCS entscheidend. Aufgrund der Vorteile von GitLab wird er als das richtige VCS für dieses Konzept bevorzugt. Denn GitLab vereinfacht nicht nur die Entwicklung und

Bereitstellung von Software, sondern ermöglicht auch automatisierte Release-, Build- und Deployment-Schritte.

5.4 Implementierung

Die ausgewählten und entwickelten Konzepte werden im Folgenden umgesetzt und die Details diskutiert.

5.4.1 Aufbau der experimentellen Umgebung

Zur Ausführung der vorher ausgewählten SOTA-Methode werden ein Server und ein Client benötigt. Der Server kommuniziert mit dem Client über die in das Produkt eingebaute Internetschnittstelle. Dadurch wird das Update übertragen und während der Übertragung Nachrichten ausgetauscht. Für diese Aktualisierungsmethode ist eine Testumgebung konfiguriert. Die Umgebung umfasst ein Server-zu-Client-Kommunikationssystem. Als Server und Clients werden die Raspberry Pis zunutze genommen. Ampelanlagen sind mit den Raspberry Pis, die als Client vorgesehen sind, verbunden. In diesem Projekt werden die reale Ampelanlage durch LEDs und Breadboard simuliert, um die serielle Kommunikation zu verwirklichen.

5.4.2 Implementierung des Release-Management-Prozesses

Der Release-Management-Prozess enthält alle notwendigen Schritte, bis die Software auf das Endprodukt verteilt wird. Dies sind die Entwicklung-,Bild-,Test- und Deployment-Phase. Sobald der Entwicklungsprozess abgeschlossen ist, erfolgt die automatischen weiteren Phasen wie Bild- Release- Deployment-Phase. Das Update kann sowohl für alle angeschlossenen Ampelanlagen als auch für verschiedene Ampelanlagen durchgeführt werden.

6 Evaluation

7 Zusammenfassung

Literatur

[1] Dipl.-Wirt.-Ing. Arno Theodor Kühn. „Systematik zur Release-Planung intelligenter technischer Systeme“. Doktor-Thesis. Universitätsbibliothek Paderborn, 27. Okt. 2016. URL: https://digital.ub.uni-paderborn.de/hs/content/titleinfo/2404085.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

Anhang