Hochschule für angewandte Wissenschaften München
Fakultät für Informatik und Mathematik

Bachelorarbeit zum Thema:

Entwurf und Bereitstellung von Microservices mit Kubernetes am Beispiel eines CRM-Systems

Zur Erlangung des akademischen Grades Bachelor of Science

Vorgelegt von: Simon Hirner

Matrikelnummer: 02607918

Studiengang: Wirtschaftsinformatik

Betreuer: Prof. Dr. Torsten Zimmer

Abgabedatum: 04.02.2022

Abstract

Bei immer mehr moderne Webanwendungen findet das Architekturmuster der Microservices anwendung. Dieser Trend verändert nicht nur den Entwurf und die Implementierung von Anwendungen, sondern hat auch erhebliche Auswirkungen auf die Bereitstellung und den Betrieb.

In dieser Bachelorarbeit wird der Entwurf und die Bereitstellung von Microservices mithilfe von Kubernetes analysiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

UNIX Uniplexed Information and Computing Service

CRM Application Programming Interface

CRM Customer Relationship Managment

LoC Lines of Code

SOA Service Oriented Architecture

CI Continuous Integration

CD Continuous Delivery

REST Representational State Transfer

HTTP Hypertext Transfer Protocol

URI Unified Resource Identifier

1 Einleitung

"Es gibt zwei Wege einen Softwareentwurf zu erstellen, entweder so einfach, dass er offensichtlich keine Schwächen hat, oder so kompliziert, dass er keine offensichtlichen Schwächen hat. Die erste Methode ist weitaus schwieriger."

– Tony Hoare

Zunächst wird in diesem Kapitel näher auf die Motivation sowie der Zielsetzung der vorliegenden Bachelorarbeit eingegangen.

Kurzbeschreibung
Gliederung & Struktur der Arbeit

1.1 Motivation

Die Grundidee Anwendungen in einzelne Komponenten einzuteilen ist nichts Neues. Microservices verfolgen diesen Ansatz noch extremer. Durch neue Werkzeuge und Technologien
Durch Container, Microservices und Kubernetes hat sich in den letzten Jahren ein erheblicher Wandel in der IT-Branche vollzogen. Containervirtualisierung erleichtert das Bauen großer verteilter Systeme durch das Verbinden von vielen kleinen Services. Im Gegensatz zu monolithischen Anwendungen erleichtern Microservices den Einsatz von verschiedenen Technologien, die Skalierung und den modularen Aufbau (Newman et al., 2015, S. 24 ff.). Um die große Anzahl an Services zu steuern, kommen Orchestrierungssysteme zum Einsatz. In diesem Bereich hat sich in den letzten Jahren Kubernetes zum Standard entwickelt. Das alles sind Trends die DevOps unterstützen und für ein zunehmendes Verschmelzen von Softwareentwicklern und Systemadministratoren sorgen. Heutzutage sind DevOps-Prinzipien tief in modernen Anwendungen verwurzelt und haben erhebliche Auswirkungen auf alle Phasen des Entwicklungszyklus. Vor allem der Entwurf und die Bereitstellung wurden enorm beeinflusst und haben sich grundsätzlich verändert. Den containerisierten, verteilten Systemen gehört die Zukunft (Arundel und Domingus, 2019, S. 1). In dieser Bachelorarbeit wird deshalb der Fokus auf dem Entwurf und der Bereitstellung von modernen Microservice-basierten Anwendungen mit dem Branchenstandard Kubernetes liegen.

1.2 Zielsetzung

Ziel dieser Bachelorarbeit ist es eine Übersicht über
Die
Die Bachelorarbeit widmet sich dem Entwurf und der Bereitstellung von Microservices mit Kubernetes. Die Arbeit wird zuerst die nötigen Methoden und Technologien beschreiben, um diese im Anschluss anhand einer konkreten Fallstudie einzusetzen. Das Ziel der Fallstudie ist es, ein auf Microservices basierendes Customer-Relationship-Management-System (CRM-System) zu entwerfen und dieses mithilfe von Kubernetes bereitzustellen. Dabei soll ein

Verfahren, das dem aktuellen Stand der Technik entspricht, vom Entwurf bis zur Bereitstellung von modernen Microservice-basierten Anwendungen implementiert werden.

1.3 Abgrenzung

Die folgenden Aspekte werden nicht betrachtet:

- Sicherheit (Authentifizierung)
- Cloud Computing

Das Ziel dieser Bachelorarbeit ist es mit neuen Methoden und Werkzeugen eine moderne Webanwendung zu gestalten. Dabei soll DevOps-Bezogen der gesamte Softwareprozess von der Entwicklung bis zum Betrieb analysiert und umgesetzt werden.

2 Theoretischer Rahmen

DevOps, Cloud und Container Microservices und Kubernetes
Kubernetes ist der Branchenstandard und die Grundlage für moderne Webanwendungen (Arundel und Domingus, 2019, Vorwort).
Die Verwendungsweise wird maßgeblich beeinflusst durch neue Technologien. (Newman et al., 2015, S. 16)
DevOps, Kubernetes und Microservices sind zwar komplett unterschiedliche Dinge haben jedoch zweifelsfrei Verbindungen und können in Zusammenarbeit ihre Stärken bestmöglich ausnutzen.

2.1 DevOps

Bevor auf die Architektur und verwendete Technologien eingegangen wird, ist es wichtig den Kontext dieser neuen Entwicklungen zu verstehen.
Das Kofferwort DevOps ist eine Sammlung von Lösungsansätzen wie IT-Leistungen schnell und risikoarm bereitgestellt werden können. Wie das Kofferwort ”DevOps”bereits andeutet, bewirkt es eine stärkere Verschmelzung zwischen Softwareentwicklung (Development) und IT-Betrieb (Operations). Dabei ist DevOps nich klar definiert. DevOps stellt den Kundennutzen in den Mittelpunkt und erinnert dabei an agile Softwareentwicklungsmethoden. DevOps und aglie Entwicklung schließen sich auch nicht gegenseitig aus unterscheiden sich jedoch durch die einbezogenen Gruppen und Abteilungen. Den während DevOps bezieht sich wie bereits erklärt nicht nur auf den Entwicklungsprozess und die Softwareentwickler sondern auf den gesamten Softwareprozess.
Um DevOps in Unternehmen umzusetzen reicht es nicht die entsprechenden Werkzeuge einzuführen, sondern es muss zu einem Kulturwandel kommen.
DevOps ist eine Sammlung an Methoden und Denkweisen, welche sich immer weiter verbreiten. Das Vorgehen in der Fallstudie dieser Arbeit kommen auch DevOps-Werkzeuge zum Einsatz. Da die gewählten Technologien und Architekturmuster sich nicht nur auf den Entwurf sondern auch
Eine der wichtigsten Werkzeuge von DevOps ist Continuous Delivery.

2.2 Microservices

Im Mittelpunkt dieser Arbeit stehen Microservices. Bei Microservices handelt es sich um ein Architekturmuster zur Modualisierung von Software (Newman et al., 2015, S. 15). Obwohl der Begriff Microservices noch realtiv jung ist, sind die dahinterstehenden Konzepte bereits älter (Newman et al., 2015, S. 15). Zur Verständlichkeit und leichteren Weiterentwicklung werden große Systeme werden schon lange in kleine Module unterteilt. Die Besonderheit von

Microservices liegt darin, dass die Module einzelne Programme sind. Ein solches einzelnes Programme wird als Microservice bezeichnet.

Microservices sind ein Architekturmuster. Architekturmuster beschreiben die Grundstruktur von Systemen in der Softwareentwicklung. Microserices werden als Architekturmuster in die Kategorie der verteilten Systeme eingeordnet. Die einzelnen Microservices laufen zumeist auf vielen unterschiedlichen Rechnern. Die Microservices sind dabei voneinander unabhängig und kommunizieren in einem Netzwerk über festgelegte Schnittstellen miteinander.

Microservices sind also das Gegenteil von klassischen monolithischen Softwarearchitekturen. Monolithische Software ist eine einzelne, zusammenhängende und untrennbare Einheit. Dies macht die Erweiterbarkeit und Warbarkeit deutlich schwieriger, da Teile des Systems nur mit erheblichem Aufwand angepasst werden können. Es kann zu nicht vorhersehbaren Abhängigkeiten kommen. Die Wiederverwendbarkeit von Teilen der Software gestaltet sich aufwendiger. Lastverteilung und Skalierung ist problematisch. Durch Modualisierung lassen sich viele dieser Nachteile abschwächen, können jedoch nicht komplett ausgemerzt werden. Um diese Nachteile zu umgehen sind Microservices entstanden. Doch neben vielen Vorteilen kommen Microservices auch mit Herausforderungen. Im nachfolgenden Abschnitt werden die genauen Vor- und Nachteile genauer erläutert.

Der Begriff Microservice ist nicht fest definiert (Wolff, 2018, S. 2), deshalb werden im nachfolgenden Kapitel die wichtigsten Eigenschaften und Merkmale betrachtet.
Dabei werden Microservices häufig im geschäftlichen Umfeld eingesetzt (Newman et al., 2015, S. 15).

Marktwachstumsprognose?

Microservices sollen nur eine Aufgabe erledigen, diese jedoch bestmöglich. Dieser Ansatz ist nicht neu und entstammt der UNIX-Philosophie: ”Mache nur eine Sache und mache sie gut”(Douglas McIlroy). Wie der Name ”Microservices”bereits andeutet, handelt es sich dabei offensichtlich um kleine Services. Eine genaue Festlegung wie groß die Services sein sollten gibt es jedoch nicht. Die Anzahl der Codezeilen (Lines of Code) können einen Hinweis geben, jedoch sind derartige Kriterien stark von der Programmiersprache und dem verwendeten Technologie-Stack abhängig. Stattdessen sollte sich die Größe an fachliche Gegebenheiten anpassen. Je kleiner die Services gestaltet werden, umso stärker kommen die in den nachfolgenden Abschnitten beschriebenen Vor- und Nachteile zur Geltung. Des Weiteren sollte ein Microservice nur so Groß sein, dass er von einem einzigen Entwicklerteam betreut werden kann. Die Größe eines Microservices ist für die Definition also nicht zwangsläufig entscheidend (Wolff, 2018, S. 2).

Der Name ”Microservices”deutet schon an, dass es sich offensichtlich um kleine Services handelt. Eine objektive Messung der Größe ist jedoch nicht möglich. Um die Größe eines Microservices zu bestimmen, könnte man beispielsweise die Anzahl der Codezeilen (LoC) verwenden. Doch die Lines of Code hängen starkt von der verwendeten Programmiersprache

und dem Technologie-Stack ab. Eine Messung der Größe nach rein objektiven Kriterien macht demnach keinen Sinn (Wolff, 2018, S. 31)(Newman et al., 2015, S. 22). Stattdessen sollte sich die Größe an die fachlichen Gegebenheiten richten. Je kleiner die Services werden desto mehr kommen die in den nächsten Abschnitten beschriebenen Vor- und Nachteile zur Geltung. Ein Microservice sollte von einem einzigen Enticklerteam gehandhabt werden (Newman et al., 2015, S. 23). Falls das nicht mehr möglich ist, könnte es darauf hindeuten, dass der Microservice zu groß ist.

Die Teamgröße beschrenkt die Größe eines Microservices, um die Unabhängigkeit der Entwicklerteams zu gewährleisten. Ein einzelner Entwickler sollte in der Lage sein den kompletten Microservice zu verstehen. Die Infrastruktur begrenzt auch die Größe. Ist die Bereitstellung eines Services sehr aufwendig, sollte die Anzahl der Microservices eher geringer sein. Auch die Kommunikation zwischen den Services und somit die Netzwerkauslastung nimmt mit der Anzahl zu. Das ist ebenfalls ein Grund für nicht zu kleine Services.
Microservices laufen als eigenständige Programme und müssen unabhängig voneinander deploybar sein. Jeder Microservice ist ein eigener Prozess, welcher isoliert von den anderen abläuft. Die Isolierung trägt dazu bei, dass verteilte System besser zu verstehen aber auch keine unbemerkten Abhängigkeiten zwischen den Services entstehen zu lassen. Neue Technologien wie Containervirtualisierung können dies erleichtern. Die Kommunikation der Services erfolgt über ein Netzwerk und mittels sprachunabhängiger Schnittstellen (API).

Ein Microservice bildet eine eigenständige und klar definierte Funktion einer Applikation ab. Der Zugriff auf den Service erfolgt ausschließlich über eine klar definierte Schnittstelle (Tremp, 2021, S. 64). Wo genau die Aufteilung in Microservices vorgenohmen werden kann wird in einem späteren Kapitel genauer diskutiert. Es ist jedoch durchaus eine komplexe und schwierige Aufgabe.
Eine ausreichende Eigenständigkeit ist nur gegeben, sobald die Services unabhängig voneinander verändert und bereitgestellt werden können. Ein Entwicklerteam sollte sich bei der Implementierung von Änderungen oder neuen Funktionen nicht mit anderen Teams absprechen müssen.

Das Gesetz von Conaway von Melvin Edward Conway besagt, dass Örganisationen die Systeme entwerfen sind gewzungen Entwürfe zu erstellen die die Kommunikationsstrukturen dieser Organsiationen abbilden."Vereinfacht gesagt, wenn zwei Entwicklerteams in einem Unternehmen ein neues Softwaresystem entwickeln, dann wird mit großer Wahrscheinlichkeit das neue Softwaresystem unabhängig von den fachlichen Anforderungen auch aus zwei großen Komponenten bestehen. Die Schnittstellen zwischen diesen zwei Komponenten werden eine ähnliche Qualität wie die zwischenmenschlichen Kommunikation zwischen den beiden Entwicklerteams haben. Studien der Havard Business School belegen, dass diese These in der Realität meistens zutreffend ist.

Das CAP-Theorem kann auf alle verteilte Systeme und somit auch auf Microservices angewendet werden. Es besagt, dass es niemals möglich ist, gleichtzeitig die drei Eigenschaften Konsistenz (Consistency), Availability (Verfügbarkeit) und Ausfalltoleranz (Partition

Tolerance) zu gewährleisten.

2.2.1 Serviceorientierte Architekturen

An dieser Stelle muss ein kurzer Exkurs zu serviceorientierten Architekturen (SOA) gemacht werden, da der Begriff häufig im Zusammenhang genannt wird.
Serviceorientierte Architektur ist ein Designansatz (Newman et al., 2015, S. 29). Wie der Name schon andeutet ist das Prinzip sehr ähnlich zu Microservices. Auch bei SOA gibt es Services, welche über das Netzwerk miteinander kommunizieren.
Es gibt durchaus Definitionen welche Microservice und SOA als identisch ansehen.
Services bei SOA werden jedoch nicht neu entwickelt. Die Funktionalität ist in den betrieblichen Anwendungen bereits vorhanden. Durch einen SOA-Ansatz sollen diese Funktionalitäten durch Services von außerhalb der Anwendung zugreifbar gemacht werden. Das Ziel ist es eine flexiblere Gesamtstruktur der IT in einem Unternehmen zu haben und Services wiederverwenden zu können. Es bedeutet jedoch nicht zwanglsäufig das große monolithische Anwendungen in kleine Services aufgeteilt werden. Eine große Anwendung kann auch aus mehreren Service-Komponenten besitzen, die dessen Dienste für andere Anwendungen verfügbar machen, ohne den eigentlichen Monolithen aufzuteilen. Über die Service sollen Geschäftsprozesse abgebildet werden (Wolff, 2018, S. 2).
Der größte Unterschied liegt jedoch auf der Ebene an der beide Ansätze ansetzen. Bei Microservices handelt es sich wie bereits erklärt um Architekturmuster, also einen konkreten Ansatz wie eine Anwendung entworfen werden kann. SOA setzt bei der gesamten IT eines Unternehmens an. SOA beschreibt wie viele Systeme in einem Unternehmen miteinander interargieren. Microservices beschreiben die Architektur von einem einzigen System.

2.2.2 Vorteile

Das Aufteilen von Software bringt wichtige Vorteile mit sich.

Modularisierung Bei klassischen Software-Monolithen, welche aus Komponenten zusammengestellt wird, entstehen schnell unerwünschte Abhängigkeiten. Die viele Abhängigkeiten erschweren die Wartung oder Weiterentwicklung. Da die einzelnen Microservices eigene Programme sind, herrscht eine starke Modularisierung. Die Programme sind eigenständig und kommunizieren nur über explizite Schnittstellen. Ungewollte Abhängigkeiten entstehen hier deutlich schwerer. (Wolff, 2018, S. 3)
In der Praxis wird die Architektur von Deployment-Monolithen meistens zunehmend schlechter. (Wolff, 2018, S. 3)

Austauschbarkeit Da Microservices nur über eine explizite Schnittstelle genutzt werden, können sie einfach durch einen Service, der die selbe Schnittstelle anbietet ersetzt werden. Bei der Ersetzung ist der neue Service nicht an den Technologie-Stack des alten Service gebunden. Auch die Risiken werden geringer, da bei schwerwiegenden Fehlentscheidungen in der Entwicklung, ein Austausch mit weniger Aufwand verbunden ist. (Wolff, 2018, S. 4)
Sie können auch deutlich schneller eingesetzt werden. (Time to Market)

Das gesamte neu schreiben eines Microservices ist in der Regel nicht besonders schwer. Viele Unternehmen scheitern an der Wartung oder Erweiterungen von alten monolithischen Systemen (Newman et al., 2015, S. 29). Da Problem ist das die Ablösung dieser großen Systeme eine fast unmöglichen Mammutaufgabe ist.

Skalierbarkeit Durch die Unabhängigkeit der Microservices können sie auch unabhängig voneinander skaliert werden. So kann eine einzelne Funktionalität, welche stärker genutzt wird, skaliert werden, ohne das gesamte System zu skalieren. (Wolff, 2018, S. 5)
Dadurch können gezielt Falschenhälse in der Anwendung entsprechend hochskaliert werden. Beim Einsatz von Cloudanbietern wie AWS ist es möglich eine Skalierung nach dem genauen Bedarf vorzunehmen. So wird nur die tatsächlich Gebrauchte Leistung des Systems bezahlt. Aber auch bei on-Premise Lösungen profitiert man von einer effektiveren Lastausnutzung. Auch die Verteilung der Last ist einfacher, da die Services auf unterschiedlichen Maschinen laufen.
Es gibt wenige Architekturmuster wie dieses, welche so eng mit Kosteneinsparungen verbunden sind. (Newman et al., 2015, S. 27).

Technologiefreiheit Des Weiteren führt die Unabhängigkeit auch zu einer großen Technologiefreiheit. Die verwendeten Technologien müssen schließlich nur in der Lage sein die explizite Schnittstelle anzubieten. (Wolff, 2018, S. 5)
Statt auf einen Technologie-Stack als Kompormiss zu verwenden, kann für jeden Service die am besten geeigneten Technologien ausgewählt werden.
Außerdem können neue Technologien auch leichter angewendet werden. Bei einer großen monolithischen Anwendung ist eine Umstellung auf beispielsweise ein neue Programmiersprache ein schwieriger und langwieriger Prozess. Bei Microservices kann die neue Sprache zuerst an einem einzelnen Service getestet werden und dann schrittweise das ganze System umgestellt werden. Oder eben nur die Services für welche die neue Sprache auch wirklich Vorteile besitzt. (Newman et al., 2015, S. 25).
Ein Nebeneffekt von dieser Technologiefreiheit ist auch, dass die Auswirkungen von falschen Entscheidungen über die Einführung von neuen Technologien deutlich reduziert.

Continuous Delivery Ein wesentlicher Grund für die Einführung ist Continuous Delivery. Die kleinen Microservices können leichter deployt werden und das Deployment bietet weniger Gefahren und ist einfacher abzusichern, als bei einem Monolithen. (Wolff, 2018, S. 5)

Das schnelle und größtenteils automatisierte Ausliefern von Software ist wichtig. In der der schnelllebigen Zeit ist man nur im Vorteil wenn neue Features und Fehlerbehebungen so schnell wie ihren Weg zum Benutzer finden. Microservices können schneller und leichter deployt werden als große Monolithen. Vor allem bei geringen Änderungen von einigen Codezeilen ist der gesamte Deployment Prozess von Monolithen sehr nervig.
Durch Ihre Unabhängigkeit können nur einzelne Services nach Änderungen neu bereitgestellt werden. Auch hier ist sind die Auswirkungen von Fehlern geringer. Ist eine Auslieferung fehlerhaft ist nicht das ganze System davon betroffen, sondern lediglich ein Service. Auch

kann bei Microservices leicht noch eine alte Version desselben Services betrieben werden, bis die fehlerfreie Funktion der neuen Version gewährleistet ist. Bei einem Monolithen wäre der Ressourcenverbrauch in so einem Fall doppelt so hoch, wie die eigentliche Anwendung benötigt.

2.2.3 Herausforderungen

Doch natürlich haben Microservices wie jedes Architekturmuster auch einige Nachteile.

Versteckte Beziehungen

Refactoring

Fachliche Architektur

Komplexität

Verteilte Systeme

2.2.4 Architektur

Die Architektur bei Micrservices ist das Finden von Kompromissen. Die einzelnen Entwicklerteams sollten viel Freiheit haben, um die nach ihrer Meinung am Besten geeigneten Technologien für ihren Service zu verwenden. Es macht jedoch Sinn gewisse Vorgaben und Rahmenbedingungen vorzugeben. Die Beschränkung auf eine Auswahl von beispielsweise fünf Programmiersprachen macht Sinn, um Wissen nicht zu weit zu verstreuen. Für die Schnittstellen ist es sinnvoll einige wenige Schnittstellenarten festzulegen, die von allen Services verwendet wird.

Die Mikroarchitektur, also die Architektur mit der ein einzelner Service implementiert wurde, sollte von außen nicht sichtbar sein und besitzt somit für das Gesamtsystem keine Relevanz.

Die fachliche Archtiektur, also die Aufteilung in verschiedene Bereiche, die jeweils von einem Microservice umgesetzt werden. Wo bei dieser Aufteilung die Grenzen gezogen werden ist eine der zentralen Herausforderungen (Wolff, 2018, S. 102). Das entscheidende Kriterium für die Aufteilung ist, dass Änderungen möglichst nur einen Service betreffen und somit von nur einem Team durchgeführt werden können. Dadurch ist wenig Abstimmung zwischen den Entwicklerteams notwendig und die Vorteile der Microservices kommen erst richtig zur geltung. Jeder Microservice solle also eine fachlichen Kontext darstellen, der eine abgeschlossene Funktionalität darstellt.

Natürlich sind die Microservices nie vollständig voneinander unabhängig. Es wird Services

geben die, die Funktionalität anderer Services aufrufen. Zu viele solcher Verbindungen führen jedoch zu einer hohen Abhängigkeit und widersprechen dem Microservice-Ansatz. Eine lose Kopplung, also nur wenige Abhängigkeiten sind erstrebenswert, da so sich Änderungen so nur auf einen Service auswirken. Benötigt ein Microservice viele Funktionalitäten von einem anderen Microservice kann das ein Hinweis auf eine schlechte Aufteilung sein und die Services sollten an einer anderen Stelle aufgeteilt werden oder womöglich direkt zusammen gelegt werden. Innerhalb eines Micrservice sollten die Komponenten und Module des Programms jedoch eine starke Kopplung besitzen. Dadurch wird gewährleistet, dass die Bestandteile wirklich zusammengehören.

Zyklische Abhängigkeiten sind auch zu vermeiden, da dort ohne weitere Abstimmung Änderungen in einem der beiden Services nicht möglich sind.

Eine bewährter Ansatz für den Anfang der Architektur ist es, mit großen Service zu beginnen und diese aufzuteilen. Einen großen Service aufzuteilen ist einfach. Die Gesamtarchitektur von vielen kleinen Services zu überarbeiten jedoch sehr komplex.

Herausforderung: Microservice-Systeme sind schwer änderbar (auf Makro Ebene)
Architektur eines Microservice: Erklärung Schichtenarchitektur, wenige Vorgaben, wird vom Team gehandhabt

2.2.5 Integration und Kommunikation

Microservices müssen miteinander kommunizieren. Die Integration der Services ist auf drei verschiedenen Ebenen denkbar.

UI

Datenbank

Microservices Zuletzt müssen natürlich auch Microservices direkt miteinander kommunizieren, um so Funktionalität von anderen Services aufzurufen. Die meist verbreitetste Technologie ist HTTP REST. Es gibt auch weitere Ansätze wie SOAP oder Message-Systeme.
Definition REST

2.2.6 Bereitstellung und Betrieb

2.2.7 Fazit

Viele Vorteile Wichtige Dinge um die sich gekümmert werden müssen: Skalierung, Deployment, Service Discovery werden durch Kubernetes übernommen und erleichtert.

2.3 Kubernetes

Kubernetes ist eine Open-Source-Plattform zur Orchestrierung und Verwaltung von Container-Awendungen. Wie Kubernetes bei der Bereitstellung und dem Betrieb von Microservices hilft wird in diesem Abschnitt erklärt. Zuerst muss jedoch Containervirtualisierung verstanden werden.

2.3.1 Containervirtualisierung

2.3.2 Architektur

2.3.3 Objekte

2.3.4 Wobei hilft Kubernetes im Bezug auf Microservices

- Service Discovery - Load Balancing - Skalierbarkeit - Monitoring (evtl. auch Logging) - Deployment

2.3.5 Kubectl

2.3.6 Minikube

3 Fallstudie

3.1 Problembeschreibung

3.2 Entwurf

3.3 Implementierung

3.4 Bereitstellung

4 Schlussbetrachtung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Literatur

Arundel, J., & Domingus, J. (2019). *Cloud Native DevOps mit Kubernetes: Bauen, Deployen und Skalieren moderner Anwendungen in der Cloud* (1. Auflage). dpunkt.verlag.

Newman, S., Lorenzen, K., & Newman, S. (2015). *Microservices: Konzeption und Design* (1. Aufl). mitp-Verl.

Wolff, E. (2018). *Microservices: Grundlagen flexibler Softwarearchitekturen* (2., aktualisierte Auflage). dpunkt.verlag.

Tremp, H. (2021). *Architekturen verteilter Softwaresysteme: SOA & Microservices - Mehrschichtenarchitekturen - Anwendungsintegration*. Springer Vieweg.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.

München, den 20. Januar 2022