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

USD US-Dollar

CRM-System Customer-Relationship-Managment-System

UNIX Uniplexed Information and Computing Service

SOA Serviceorientierte Architektur

API Application Programming Interface

CI Continuous Integration

CD Continuous Delivery

REST Representational State Transfer

HTTP Hypertext Transfer Protocol

URI Unified Resource Identifier

1 Einleitung

"Veränderungen begünstigen nur den, der darauf vorbereitet ist."
– Pasteur, 1933, S. 348

Die IT-Branche befindet sich in einem erheblichen Wandel. Neue Methoden und Werkzeuge revolutionieren die Software-Welt. Angefangen hat es mit der Verbreitung von Cloud Computing. Bei den damit einhergehenden großen verteilten Systemen wurde es immer schwieriger den Betrieb des Systems von der Architektur des Systems zu trennen. Daraus resultierte DevOps. Ein Ansatz, welcher das stärkere Zusammenarbeiten von Softwareentwicklung und IT-Betrieb fördert und fordert. Mit dem Architekturmuster der Microservices lassen sich die Ziele von DevOps bereits im Entwurf von Anwendungen einbringen. Containervirtualisierung erleichtert die Bereitstellung der einzelnen Microservices und neue Werkzeuge wie Kubernetes helfen dabei die große Anzahl an Containern zu managen. Zusammen bilden all diese Veränderungen den Grundstein für moderne Anwendungen bestehend aus containerisierten Microservices, welche mit Kubernetes verwaltet werden.

Die beiden Softwareentwickler Kubernetes Brendan Burns und David Oppenheimer, welche Kubernetes mitentwickelten, halten diese Veränderungen sogar ähnlich revolutionär wie die Popularisierung der objektorientierten Programmierung [Burns und Oppenheimer, 2016, S. 1]. Der Cloud-Experte John Arundel denkt, dass aufgrund dieser Revolutionen die Zukunft in containersierten, verteilten Systemen liegt, die auf der Kubernetes-Plattform laufen [Arundel und Domingus, 2019, S. 1].

In dieser Arbeit werden die Revolutionen miteinander verbunden, um die Merkmale und den Nutzen von containersisierten Microservices vom Entwurf bis zur Bereitstellung kennenzulernen. Zu Beginn der Arbeit wird in diesem Kapitel die Motivation, die Zielsetzung sowie der Aufbau der Arbeit beschrieben.

1.1 Motivation

DevOps wird von immer mehr Unternehmen adaptiert, um die Geschwindigkeit und Qualität zu erhöhen. Ein umfangreiche Umfrage geben 83% aller gefragten IT-Entscheidungsträger an, dass ihre Organisation bereits DevOps-Praktiken einsetzt [Puppet, 2021, S. 10].

Der Übergang zu Microservice-Architekturen ist in vollem Gange. Vor allem im unternehmerischen Umfeld werden immer mehr monolithische Anwendungen in Microservices aufgespalten. Die Verbreitung wird auch noch in den nächsten Jahren zunehmen und es ist nicht mit einer Trendwende zu rechnen. Das Marktvolumen für Microservices in der Cloud wurde 2020 auf 831 Millionen US-Dollar (USD) geschätzt. Bis zum Jahre 2026 soll der Markt mit einer durchschnittlichen jährlichen Wachstumsrate von 21.7% auf 2701 Millionen USD anwachsen [Mordor Intelligence, 2020, S. 7].

Die Verwendungsweise wird maßgeblich beeinflusst durch neue Technologien [Newman et al., 2015, S. 16]. Containervirtualisierung erleichtert die Bereitstellung der einzelnen Microservices. Container werden auch über Microservices hinweg verwendet und sind aus der heutigen IT-Landschaft nicht mehr wegzudenken. Google startet über zwei Milliarden Container pro Woche [Liebel, 2021, S. 43].

Kubernetes ist der Branchenstandard und die Grundlage für moderne Webanwendungen [Arundel und Domingus, 2019, Vorwort]. Bei einer Umfrage zeigt sich, dass 91% der Befragten Kubernetes zur Containerorchestrierung einsetzen [Cloud Native Computing Foundation, 2020, S. 8]. Alle großen Cloud-Anbieter wie Google Cloud, Amazon Web Services und Microsoft Azure setzen Kubernetes ein.

Es kann also zweifelsfrei behauptet werden, dass Microservices und Kubernetes im Trend sind und in Zukunft auch weiter ansteigen werden. Die Kombination dieser Methoden und Werkzeugen ergänzt sich perfekt und ist die Zukunft für große Systeme. Jedoch sind die Technologien diffizil und bringen neben zahlreichen Vorteilen auch viele Herausforderungen mit sich. Deshalb ist es von großer Bedeutung die Technologien in ihrer Gesamtheit zu verstehen und anwenden zu können. In dieser Arbeit wird sich deshalb der Entwurf und die Bereitstellung von Microservices mit Kubernetes widmen.

1.2 Zielsetzung

Das Ziel dieser Bachelorarbeit ist es eine mit dem aktuellen Stand der Technik entsprechende moderne Webanwendung nach der Microservice-Architektur zu entwerfen und mithilfe von Kubernetes bereitzustellen. Dazu soll zuerst ein aktueller Stand der Technik beschrieben werden um anschließend eine Fallstudie durchzuführen. Die Fallstudie wird am Beispiel eines Customer-Relationship-Managment-Systems (CRM-Systems) durchgeführt. In der Fallstudie soll ein Verfahren vom Entwurf bis zur Bereitstellung in einem DevOps-Umfeld implementiert werden. Um Aussagen zum Anwendungsgebiet und der Realisierung von containerisierten Microservices mit Kubernetes zu treffen, sollen geklärt werden,

- welche Anwendungsmöglichkeiten sowie Vorteile Microservices bieten,
- wobei Containervirtualisierung sowie Kubernetes den Einsatz von Microservices unterstützt,
- wie eine Microservice-Architektur entworfen werden kann,
- wie Microservices mit Kubernetes bereitgestellt werden können und
- welche Nachteile und Herausforderungen sich daraus ergeben.

1.3 Aufbau der Arbeit

Als Erstes wird in Kapitel 2 der aktuelle Stand der Technik beschrieben. Es wird DevOps, das Architekturmuster der Microservices, Containervirtualisierung sowie Kubernetes genauer

erklärt. Auf Basis dieser theoretischen Grundlagen wird die Fallstudie durchgeführt. In Kapitel 3 wird zuerst die Problemstellung beschrieben. Danach wird in Kapitel 4 der Entwurf und in Kapitel 5 die Implementierung der Microservices erläutert. Anschließend wird in Kapitel 6 die Bereitstellung mit Kubernetes erklärt. Zum Schluss wird in Kapitel 7 ein Fazit gezogen und die Ergebnisse diskutiert.

2 Theoretische Grundlagen

Um die Fallstudie durchzuführen, wird zunächst der theoretische Rahmen der Arbeit erläutert. In diesem Kapitel wird zunächst der DevOps-Ansatz und Microservices beschrieben. Anschließend wird Containervirtualisierung sowie Kubernetes erklärt.

2.1 DevOps

Bevor auf die Architektur und die verwendete Technologien eingegangen wird, ist es wichtig den weiteren Kontext zu betrachten. Wie das Kofferwort ”DevOps”bereits andeutet, beschreibt er einen Ansatz für eine effektivere und stärkere Zusammenarbeit zwischen Softwareentwicklung (Development) und IT-Betrieb (Operations). Dabei ist DevOps nicht klar definiert und ist ein Überbegriff für Denkweisen, Kultur, Methoden, Technologien und Werkzeuge. DevOps stellt den Kundennutzen in den Mittelpunkt. Das Ziel ist es ein Unternehmen anpassungsfähiger zu machen und trotzdem geordnete Prozesse zu wahren. Die Anpassungsfähigkeit wird hier häufig mit der Time to Market gemessen. Diese Kennzahl sagt aus wie lange es dauert eine Änderung auf die Produktionsumgebung zu bringen.

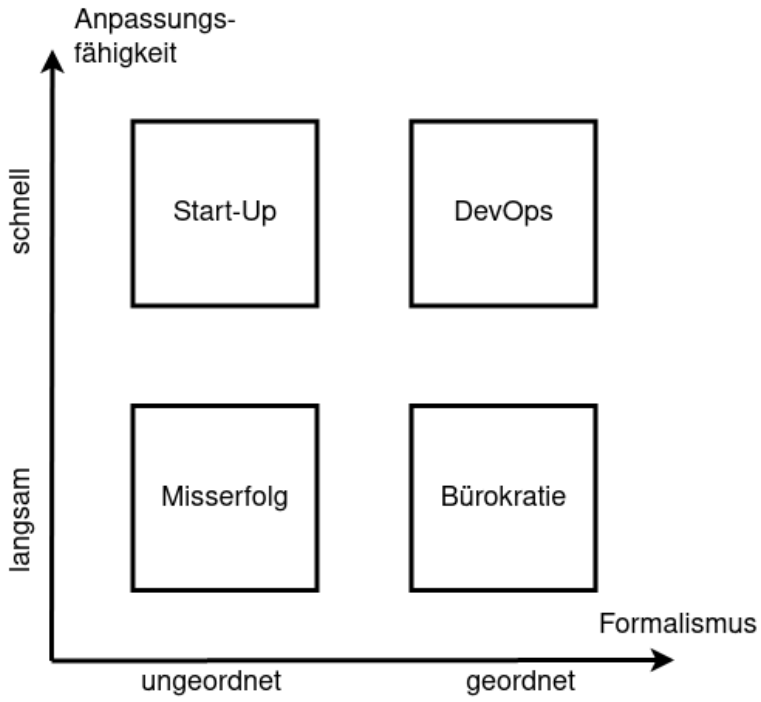

Abbildung 1: Kategorisierung von Unternehmen

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 Modularisierung 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.

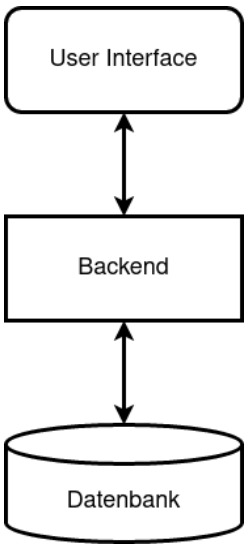

Abbildung 2: Aufbau einer monolithischen Architektur

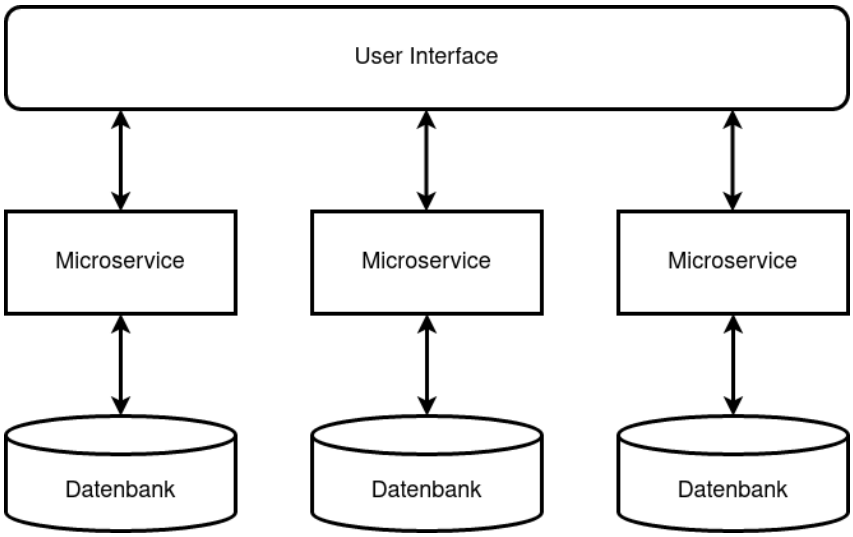

Abbildung 3: Aufbau einer Microservice-Architektur

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).

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 Containervirtualisierung

Früher wurden Anwendungen auf physischen Rechnern ausgeführt. Wenn mehrere Anwendungen auf einem Rechner liefen, konnte eine Anwendung Ressourcen wie die Rechenleistung der anderen Anwendung wegnehmen. Durch Virtualisierung von Hardware konnte das Problem gelöst werden, indem mehrere virtuelle Maschinen auf einem physischen Rechner betrieben werden. Virtuelle Maschinen besitzen dieselben Funktionen wie ein physischer Rechner und führen ein eigenes Betriebssystem aus. Dadurch ist diese Art der Virtualisierung nicht leichtgewichtig. Bei der Containervirtualisierung werden nur die Ressourcen des Betriebssystems des Betriebssystems zur Verfügung gestellt, welche auch wirklich benötigt werden. Containervirtualisierung ist deshalb leichtgewichtig.
Ein weiteres Problem ist, wie Microservices aber auch andere verteilte Systeme über große Netzwerke mit vielen verschiedenartigen Rechnern bereitgestellt werden können. Es wurde also eine zuverlässige Methode gesucht wie Software auf diversen Rechnern immer gleich läuft. Auch dieses Problem schafft Containervirtualisierung aus dem Weg, indem es Software mit allen Abhängigkeiten, Bibliotheken, Compilern und Konfigurationen in eine standardisierte und einheitliche Komponente verpackt: Der Container. Ein Image einer virtuellen Maschine besitzt auch alle diese Eigenschaften, enthält aber noch viel mehr als wie die Anwendung wirklich für die Ausführung benötigt.
Vorteile zu VMs: - Container laufen auf der realen CPU, somit kein zusätzlicher Aufwand für Virtualisierung - Container enthält nur die Dateien, die wirklich benötigt werden, daher viel kleiner - Dateisystem-Layern, könen von mehreren Container genutzt und wiederverwendet werden, effiziente Platznutzung
Eigenschaften von Images: - Wiederverwendbar - Kann überall laufen wo die entsprechende Container-Engine läuft und ausreichend Ressourcen vorhanden sind - Läuft überall gleich (Container-Format muss unterstützt werden) - Keine Abhängigkeit von unterschiedlichen Distributionen, Bibliotheken und Software-Versionen auf dem Rechner (lediglich abhängig von Betriebssystem-Kernel)
Ein weiteres Problem was letztendlich zur Containervirtualisierung führte, ist die Frage wie verteilte Systeme (microservices) über große Netzwerke mit verschiedenartigen Rechnern bereitgestellt werden können.

Container revolutionieren also die Bereitstellung von Anwendungen, haben jedoch auch eine große Auswirkung auf die Anwendungsarchitektur und entfalten so beispielsweise das volle Potential von Microservices. In der steigenden Beliebtheit von Microservice-basierten Anwendungen bestehend aus Container sehen manche eine ähnliche Revolution wie die in den 1990er-Jahren objektorientierte Programmierung. (Burns und Oppenheimer, 2016, S. 1)
Vorteile
Container eignen sich aufgrund dieser Eigenschaften perfekt, Microservice-basierte Anwendungen zusammenzustellen und auszuführen.
Container sind normalerweise unveränderlich. Soll ein Container geändert werden, so wird der alte Container gegen einen neuen ausgetauscht.
Ein Container besitzt sein eigenes Dateisystem, Anteil an CPU, Speicher und Prozessraum.
Containervirtualisierung

2.4 Kubernetes

Der Name Kubernetes ist Griechisch und bedeutet Steuermann. Kubernetes wird häufig auch mit dem Kürzel ”K8säbgekürzt. Kubernetes ist jedoch kein Betriebssystem. Es benötigt ein installiertes Betriebssystem auf den Nodes.
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.
Kubernetes bietet viele Funktionen, die helfen eine entkoppelte Microservice-Architektur zu bauen.

2.4.1 Aufbau

2.4.2 Objekte

2.4.3 Wobei hilft Kubernetes im Bezug auf Microservices

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

2.4.4 Kubectl

2.4.5 Minikube

Minikube ist ein Werkzeug, um ein lokales Kubernetes Cluster zu betreiben. Minikube erstellt ein Cluster bestehend aus nur einem Node in einer virtuellen Maschine. Der Node fungiert dabei sowohl als Master sowie auch als Worker. Minikube unterstützt mittlerweile auch den Betrieb mit mehreren Nodes. Des Weiteren kann das Cluster auch in einem Docker Container anstatt in einer virtuellen Maschine betrieben werden. Da Minikube über eine virtuelle Maschine oder in einem Docker Container läuft, kann minikube auch über Linux hinaus auf Windows oder MacOS betrieben werden.

3 Erläuterung Fallstudie

Nachdem die theoretischen Grundlagen nun erläutert worden sind, wird mit der Fallstudie begonnen. In diesem Kapitel wird das Problem der Fallstudie beschrieben.

4 Entwurf der Microservices

4.1 Makro-Architektur

4.2 Micro-Architektur

4.3 Integration

5 Implementierung

5.1 Services

5.2 Frontend

5.3 Datenbank

6 Bereitstellung mit Kubernetes

6.1 Containerisierung

6.2 Bereitstellung

6.3 Skalierung

6.4 Lastverteilung

7 Schlussbetrachtung

7.1 Diskussion

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.

7.2 Ausblick

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

Pasteur, L. (1933). *Oeuvres de Pasteur*. Médiathèque scientifique de l'Institut Pasteur. Verfügbar 26. Januar 2022 unter https://gallica.bnf.fr/ark:/12148/bpt6k6211139g

Burns, B., & Oppenheimer, D. (2016). Design Patterns for Container-based Distributed Systems. https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/burns

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

Puppet. (2021). *State of DevOps*. Verfügbar 26. Januar 2022 unter https://puppet.com/resources/report/2021-state-of-devops-report/

Mordor Intelligence. (2020). *Global Cloud Microservices Market*. Hyderabad. Verfügbar 26. Januar 2022 unter https://www.mordorintelligence.com/industry-reports/cloud-microservices-market

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

Liebel, O. (2021). *Skalierbare Container-Infrastrukturen: Das Handbuch für Administratoren* OCLC: 1241672797.

Cloud Native Computing Foundation. (2020, 17. November). *Cloud Native Survey*. San Francisco. Verfügbar 26. Januar 2022 unter https://www.cncf.io/blog/2020/11/17/cloud-native-survey-2020-containers-in-production-jump-300-from-our-first-survey/

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 26. Januar 2022