Evaluation der Microservices- gegenüber der monolithischen Architektur

Mohamed Sohil
Mohamed.Sohil@stud.h-da.de
Wissenschaftliches Arbeiten II
Hochschule Darmstadt

28.01.2021

abstract ein monolithiches System funktioniert problemlos ,wenn es kleine Applikation ,untrennbare Komponenten ist,und ein kleines Team daran arbeiten ,aber wenn es skaliert(Teamgröße ,und Implementierung) oder erweitert wird,dann wächst die Komplexität ,und nimmt das bedarf dazu , das große monolithes System zu zerlegen, und in kopplungslose Komponenten ,bzw. Microservices zu migrieren. die Microservices sind klein und erledigen Aufgaben ,dies bringt viele Vorteile mit wie:dass die Entwickler unabhängig arbeiten, wenn die Microservices unabhängig voneinander sind ,und die Skalierbarkeit wird hier unterstützt, denn die Projekte , die auf Microservices-Architektur basiert sind , erlauben die Skalierbarkeit , es kann auch neue Komponenten ,bzw. Services hinzugefügt ,sowie entfernt werden .der Ausfall eines Services beeinflusst nicht die anderen Services . Neben der Vorteile gibt es auch Herausforderungen , die Microservices als Komplexe Architektur darstellen können .dieses Paper befasst sich mit dem Entwurf ,Deployment ,sowie welche Vorteile und Herausforderungen,die Microservices- gegenüber .der monolithischen Architektur mit sich bringen.

Inhaltsverzeichnis

1 Motivation

eine Frage wird heutezutage aufgeworfen , warum von Monolithischer Architektur in Microservices imigriert wird. Der Hauptgrund, warum die Entwickler sich für die Verwendung einer Microservices-Architektur entscheiden, besteht darin, die Geschwindigkeit bei der Produktion von Software zu verbessern, mit der Sie Änderungen an Ihrer Anwendung vornehmen können, wenn die monolithische Natur Hindernis für die Änderungen der Anwendungen geworden ist. Alle anderen Vorteile von Microservices ergeben sich aus dieser Grundvoraussetzung. Der Grund, warum wir unsere Änderungen an unseren Anwendungen schneller durchführen möchten, besteht natürlich darin, die Kunden schnell neue Features und Funktionen zur Verfügung zu stellen, um zu testen.Aufgrund dieser Änderungen werden die erwarteten positiven Ergebnisse erzielt können.ein Bedarf auch an Microservices-Architektur besteht darin ,Wenn die Änderungen an Software vorgenommen werden, um unseren Geschäftswert zu verbessern, und wenn dies nicht gelingt, müssen wir dies schnell wissen und fortfahren, um etwas Neues auszuprobieren. Microservices ist eine Optimierung unserer Anwendungsarchitektur, damit wir schneller vorankommen und diese Änderungen schneller vornehmen können. bei meisten Unternehmen wird festgestellt , dass ein gewisser Prozentsatz ihrer benutzerdefinierten Anwendungen von einem iterativen Fortschritt zu einer Microservices-Architektur profitiert. Diese Anwendungen werden von dieser Änderung profitieren.

2 Methodik

Eine systematische Literaturrecherche wurde durchgeführt, um die Lösung des Problems zu erforschen und folgende Fragen zu beantworten, was bedeutet monolithische Architektur , welche Vorteile ,nachteile mit sich bringen und welche Lösungen wurden gefunden , um Nachteile zu überwinden ,verfügbare , und skalierbare Systeme zu erhalten Dazu wurden Suchvorgänge in Suchmaschinen und Webseiten durchgeführt, die wissenschaftliche Forschungen und Paper zur Verfügung stellen, wie Google Scholar, Springer, IEEE Xplore .Recherchiere Nach Büchern in der Bibliothek wurde auch durchgeführt. Folgende Fragen wurden gestellt: wie wird monolithische Architektur beschrieben? Hat die Benutzung dieses Ansatzs von Software-Architektur Vorteile ? welche Herausforderung/Nachteile bringen monolithische Architektur mit sich ? Wie wird diese Architektur evaluiert? die zweite Phase der Arbeit wurd nach möglichen Lösungen erforscht, die auch in Fragen zusammengefasst wurden Gibt es Alternative zu Monolithischer Architektur? Wie wird Microservices-Architektur definiert? Welche Vorteile/Herausforderungen bringt sie mit sich ? Welche Maßnahmen werden ergriffen ,um die Herausforderung zu lösen? Wie wird diese Architektur evaluiert

3 Monolithisch :Evaluation der Architektur

3.1 Defination der Architektur

Monolithische Architektur war der traditionelle Ansatz für die Softwareentwicklung, der in der Vergangenheit von großen Unternehmen wie Netflix ,Amazon und Ebay verwendet wurde. In Monolithischer Architektur sind die Funktionen in einer einzigen Anwendung zusammengefasst ,und werden als Einheit zu implementieren und bereitzustellen. Monolithische Anwendungen sind zwar nicht kompliziert,und haben trotzdem ihre eigenen Stärke[3].

3.2 Vorteile

die Implementierung , das Testen und die Bereitstellung der Anwendung einfach ist , außerdem die Kommunikation zwischen Komponenten in der Monolithiscer Architektur einfach ist , da sie nur in der gleichen Anwendung erfolgt. Auch Teamgröße spielt eine Rolle , wenn das Team klein ist und am gleichen Ort (Platz) arbeitet ,außerdem ist eine monolithische-basierte Anwendung leicht bereitzustellen , und zu testen[17]. Das Problem besteht jedoch darin, dass viele Unternehmen heutezutage noch ihre Software als Monolithisch haben, trotz der Problemen im Zusammenhang mit Monolithisch. Die meisten dieser Software sind schwer zu verwalten und weiterzuentwickeln, was Unternehmen dazu zwingt, neue Software zu kaufen, anstatt neue Funktionen in den eigenen zu entwickeln. Die Migration von Monolithische Architektur zu Microservices Architektur kann eine Lösung sein, und ihre Software wird wieder verfügbar und bietet alle Vorteile der Microservices-Architektur. Einfache Bereitstellung - Ein Entwickler musste lediglich die WAR-Datei auf einen Server kopieren, auf dem Webserver wie Tomcat oder Glasfish installiert wurde[17].

3.3 Nachteile

Die Nachteile können aufgetretten werden : Wenn die Anwendung jedoch komplizierter wird, wächst die Monolithstruktur an Größe und wird zu einer großen, schwer zu verwaltenden und skalierbaren Software. Monolithische Architektur wie erwähnt war immer der traditionelle Weg für die Softwareentwicklung, der in der Vergangenheit von großen Unternehmen verwendet wurde. In monolithischer Architektur sind die Funktionen in einer einzigen Anwendung gekapselt. Wenn der Monolith klein ist und nur über wenige Funktionen verfügt, kann er seine eigenen Vorteile haben, z. B. einfache Entwicklung, Tests, Bereitstellung und Skalierung. Wenn zum Beispiel mehr Rechenleistung benötigt wird , wird einfach den gesamten Monolithen dupliziert und mit der geringen Größe des Systems nur einen geringen Overhead haben. Schwächen dieses Systems treten auf, wenn die Monolithische Anwendung aktualisiert und weiterentwickelt wird, wobei Größe und Anzahl der Funktionen zunehmen. Sobald dies geschieht, überwiegen die Nachteile von MA die Vorteile , Wenn beispielsweise mehr Rechenleistung benötigt wird, führt die Skalierung eines großen Monolithen zu einem größeren Overhead in Bezug auf einen kleinen Monolithen. Ein weiterer Nachteil, der durch die Größe des Monolithen verursacht wird, besteht darin, dass die Entwicklung verlangsamt werden kann und aufgrund der längeren Startzeit ein Hindernis für den kontinuierlichen Einsatz darstellt, ein großes Nachteil besteht auch darin, wenn ein Software-Fehler (Bug) auftaucht ,kann das ganze System beeinträchtigt. Für eine Änderung in einer kleinen Teilmenge des Systems muss das gesammte neu erstellt, erneut getestet und erneut bereitgestellt werden. Die kontinuierliche Bereitstellung ist schwierig und verhindert häufige und kleine Updates[3].

3.4 Evaluation der Architektur

In einer monolithisch-basierten Anwendung mit vielen Codezeilen wird immer die Entwicklung neuer Funktionen langsamer,da die implementierten Funktionen groß sind und die Modularität normalerweise abnimmt.außerdem ist die Überwachung der Fehler langsam kompilziert, es ist nicht einfach ,einen Fehler zu entdecken ,und zu beheben. ein Fehler beeinträchtigt die gesammte Anwendung ,auch wenn andere komponenten fehlerfrei funktionieren. die Komplexität ist auch deutlich bei Skalierung ,oder Aktualisierung der Version.

3.5 Lösung

Das Problem besteht jedoch darin, dass viele Unternehmen heutezutage noch ihre Software als Monolithisch haben, trotz der Problemen im Zusammenhang mit Monolithisch. Die meisten dieser Software sind schwer zu verwalten und weiterzuentwickeln, was Unternehmen dazu zwingt, neue Software zu kaufen, anstatt neue Funktionen in den eigenen zu entwickeln. Die Migration von Monolithisch zu Microservice-Architektur kann eine Lösung sein, und ihre Software wird wieder verfügbar und bietet alle Vorteile der Microservices-Architektur[12].

4 Microservices :Evaluation der Architektur

4.1 Defination der Architektur

Microservices sind kleine Services (Dienste) , Jeder Dienst konzentriert sich nur auf eine Funktionalität. Diese Art von Ansatz macht deutlich, wo die Grenzen zwischen verschiedenen Diensten liegen und wohin Codeänderungen gehen sollten. Folglich sind Microservices von Natur aus losegekoppelt . Durch die lose-Kopplung können Entwickler unabhängige Änderungen an Services vornehmen, ohne den Rest der Codebasis zu beeinflussen. Da Microservices nicht miteinander verbunden sind, können sie unabhängig voneinander skaliert und bereitgestellt werden. Eine solche Architektur ist ein Ansatz für die Erstellung vom Software, das die traditionellen, normalerweise monolithischen Architektur ersetzt und verlegt[9]. Eine andere Defination der Microservices-Architektur lautet: Microservices werden als kleine autonome Services betrachtet, die auf folgene Prizipien basiert sind : - das Ausblenden interner Implementierungsdetails - das Isolieren von Fehlern und - Unabhängiges Bereitstellen und Beobachten von Services. eine andere Defination von der Microservices-Architektur ist: Microservices werden als „einen Ansatz zur Entwicklung einer einzelnen Anwendung definiert , die jeweils in einem eigenen Prozess ausgeführt werden und mit einfachen Mechanismen kommunizieren entweder asynchron oder synchron.diese Services sind unabhänigng voneineinader zu implementieren , und bereitzustellen,darüberhinaus werden die Services in verschiedenen Programmiersprachen geschrieben und mit unterschiedliche Datenspeichertechnologien (wie SQL oder Nosql) verwenden können[11].

4.2 Vorteile

4.2.1 Microservices sind klein

Microservices sind als kleineren Diensten betrachtet ,die skaliert oder entfernt dürfen können , wenn das Bedarf daran ist, sodass andere Teile des Systems auf kleinerer, weniger leistungsfähiger Hardware ausführen können[6].

4.2.2 Microservices sind autonom

es wird festgestellt, dass Microservices autonom sind - dies bedeutet , dass jeder Dienst funktioniert und sich unabhängig von anderen Diensten ändern kann . Um diese Autonomie sicherzustellen, müssen die Dienste so entworfen werden dass sie: ¡losegekoppelt sind - Durch Interaktion über klar definierte Schnittstellen oder durch veröffentlichten Ereignissen bleibt jeder Microservice unabhängig von der internen Implementierung der Mitarbeiter im Unternehmen. Zum Beispiel den von uns eingeführten Bestellservice früher sollte nicht über die Implementierung der Kontotransaktionen informiert sein Bedienung.unabhängig bereitstellbar - Dienste werden parallel entwickelt, häufig von mehreren Teams. Sie müssen sie im Gleichschritt oder in einer orchestrierten Formation einsetzen würde zu riskanten und

ängstlichen Einsätzen führen. Idealerweise möchten Sie Ihre verwenden kleinere Dienste für schnelle, häufige und kleine Releases[8].

4.2.3 Skalierbarkeit

Anwendungen müssen für die Skalierung bereit sein. Bei einer Monolith-Architektur ist dies teuer, da das gesamte System skaliert werden muss. Mit Microservices wird das System in Services aufgespaltet werden ,und jeder Dienst kann unabhängig skaliert werden. Der Server kann mehrere Instanzen von Microservices bereitstellen, die den größten Teil der Last haben. Beispielsweise erzeugt Amazon Web Services automatisch mehr Instanzen eines Mikrodienstes, wenn der Mikrodienst stark ausgelastet ist. Mit dieser Art der Skalierung können Sie die gesamte Anwendung auf weniger leistungsfähiger Hardware ausführen[4].

4.2.4 Austauschbarkeit

Da die einzelnen Services klein sind, sind die Kosten für den Ersatz durch eine bessere Implementierung oder sogar für das Löschen insgesamt erheblich einfacher zu verwalten. Sehr oft haben die Entwickler an einem Tag mehr als hundert Codezeilen gelöscht und sich nicht zu viele Sorgen gemacht, Da Microservices häufig von ähnlicher Größe sind, sind die Hindernisse für das Umschreiben oder Entfernen von Diensten sehr gering. Teams, die Microservice-Ansätze befolgen, können Services bei Bedarf komplett neu schreiben und einen Service nur dann beenden, wenn er nicht mehr benötigt wird. Wenn eine Codebasis nur ein paar hundert Zeilen lang ist, ist es für Menschen schwierig, sich emotional an sie zu binden, und die Kosten für das Ersetzen sind ziemlich gering[3].

4.2.5 Time-to-Market

Microservices ermöglichen eine kurze Markteinführungszeit. Wie bereits erwähnt wurde, Microservices können einzeln in Produktion gebracht werden. Wenn Entwicklungsteam an einem Großem System arbeitet , die von Monolithisch zu Microservices migriert, ist jedes Mitglied für einen oder mehrere Microservices verantwortlich und wenn ein Bedarf an Änderungen oder Verbesserung nur an einen Service, kann das Team ihn unabhängig entwickeln und einbringen. Das ermöglicht Viele Teams parallel an zahlreichen Funktionen zu arbeitet ,und bringen mehr ausführbare Funktionen in kürzerer Zeit in Markt als dies mit einem Einsatzmonolithen möglich gewesen wäre[3].

4.2.6 freie Wahl der Technologie

Wenn Microservices als Ansatz in der Entwicklung verwendet werden, gibt es keine Einschränkungen hinsichtlich des Einsatzes von Technologien. Dies bietet die Möglichkeit, eine neue Technologie innerhalb eines einzelnen Microservices zu implementieren,und zu testen, ohne andere Dienste zu beeinträchtigen.ein Vorteil ist auch ,Das Risiko wird bei der Einführung neuer Technologien und neuer Versionen bereits verwendeter Technologien verringert, da diese neuen Technologien in einer begrenzten Umgebung eingeführt und getestet werden, um die Kosten niedrig zu halten. Darüber hinaus ist es möglich, bestimmte Technologien für bestimmte Funktionen einzusetzen, beispielsweise eine bestimmte Datenbank. Das Risiko ist gering, da der Microservice bei Problemen ausgetauscht oder entfernt werden kann. Die neue Technologie ist auf einen oder wenige leistungen beschränkt. Dies reduziert das Risiko und ermöglicht unabhängige Technologieentscheidungen für verschiedene Microservices. Noch ein wichtiger Punkt ist, dass die Entscheidung, neuer hochinnovativer Technologien auszuprobieren und zu bewerten ist , wird erleichtert. Dies erhöht die Produktivität der Entwickler und verhindert, dass die Technologieplattform veraltet ist. Darüber hinaus wird der Einsatz moderner Technologien gut qualifizierte Entwickler anziehen[1].

4.2.7 Größe des Teams

Der unabhängige Einsatz von Microservices und die Aufteilung des Entwicklungsaufwands in Teams führen zu einer Obergrenze für die Größe eines einzelnen Microservices. Ein Team sollte in der Lage sein, Funktionen in einem Microservice zu implementieren und zu Testen .diese Funktionen sind unabhängig von anderen Teams in der Produktion bereitzustellen. Auf diese Weise ermöglicht diese Architektur die Skalierung der Entwicklung, ohne dass zu viel Koordination und Anstrengung zwischen den Teams erforderlich sind. Ein Team muss in der Lage sein, Funktionen unabhängig von den anderen Teams zu implementieren. Auf den ersten Blick scheint der Mikroservice daher groß genug zu sein, um die Implementierung verschiedener Funktionen zu ermöglichen.weil Microservices klein sind, kann Ein Team für mehrere Microservices verantwortlich sein,. Eine Untergrenze für die Größe des Microservices ergibt sich nicht aus der unabhängigen Bereitstellung und der Aufteilung in Teams. Daraus ergibt sich jedoch eine Obergrenze: Wenn ein Microservice eine bestimmte Größe erreicht hat, die seine weitere Entwicklung durch ein einzelnes Team verhindert, ist er zu groß. Bei Angelegenheit sollte ein Team so groß sein, dass seine Größe gut für agile Prozesse geeignet ist, normalerweise besteht das Team aus drei bis neun Personen. Ein Microservice sollte daher niemals so groß werden, dass ein Team von drei bis neun Personen ihn nicht selbst weiterentwickeln kann. Neben der Größe spielt die Anzahl der Features eine wichtige Rolle, die in einem einzelnen Microservice implementiert werden sollen. Immer wenn innerhalb kurzer Zeit an großer Anzahl von Änderungen Bedarf ist, kann ein Team schnell überlastet werden. Ein Alternativ dafür , dass mehrere Teams an demselben Microservice arbeiten können. Im Allgemeinen sollte ein Microservice jedoch niemals so groß werden, dass mehrere Teams erforderlich sind, um daran zu arbeiten[1].

4.2.8 Elastizität

Der Aufbau eines Software-Produkt ,das auf Microservice-Architektur basiert ist , ermöglich Komponenten bzw. Services einzeln und unabhängig voneinander zu entwicklen und bereitzustellen ,daher kann die Ausfallsicherheit des gesamten Systems durch eine bessere Isolierung von Fehlern erhöhen, in dem es eine Isolierung von Fehlern bietet. Ein System aus Microservices funktioniert möglicherweise, selbst wenn ein Dienst ausfällt ,Im Vergleich zu einer monolithischen Architektur, bei der das gesamte System abstürzt, wenn seine Komponente in eine fatale Situation gerät. Andererseits kann ein abgestürzter Microservice Kaskadenfehler im gesamten System verursachen, wenn er nicht gut isoliert ist. Ein kaskadierender Fehler stürzt das System ab oder blockiert das gesamte Microservice-Netzwerk. Um Kaskadenfehler zu vermeiden, verwenden Microservices Timeouts, Leistungsschalter, um Fehlersituationen zu isolieren. Darüber hinaus erfordert Microservices wie jedes andere verteilte System eine aktive Überwachung, um den Status aller Dienste anzuzeigen. Also Durch die Aufteilung Ihrer Anwendung in mehrere Dienste können Fehler zwar eingegrenzt werden, es werden jedoch auch Fehlerstellen entdeckt ,darüberhinaus ist es leicht zu berücksichtigen ,was passiert , wenn ein Fehler auftritt und Fehler ,ohne das ganze System zu beeinträchtigen[1].

4.2.9 Kontinuierliche Lieferung

Im Gegensatz zu monolithischen Anwendungen, in denen qualifizierte Teams an Funktionen wie Benutzeroberfläche, Datenbank, Business Logik und technologischen Schichten arbeiten, verwendet Microservices funktionsübergreifende Teams, um den gesamten Lebenszyklus einer Anwendung mithilfe eines kontinuierlichen Bereitstellungsmodells abzuwickeln. Wenn ein Entwickler, Betriebs- und Testteams gleichzeitig an einem einzigen Dienst arbeiten, wird das Testen und Debuggen einfach und sofort. Mit diesem Ansatz der

inkrementellen Entwicklung wird Code kontinuierlich weiterentwickelt, getestet und bereitgestellt, Bugs leicht zu entdecken und Sie können Code aus vorhandenen Bibliotheken verwenden, anstatt das Rad neu zu erfinden[1].

4.3 Herausforderungen

4.3.1 Code-Abhängigkeit

Ein großer Vorteil einer auf Mikroservices basierenden Architektur ist die Möglichkeit, die einzelnen Services unabhängig voneinander (losekopplung) zu implementieren. Dieser Vorteil kann jedoch durch Code-Abhängigkeit beeinträchtigt werden. Wenn eine Bibliothek von mehreren Microservices verwendet wird und eine neue Version dieser Bibliothek eingeführt werden soll, ist möglicherweise eine koordinierte Bereitstellung mehrerer Microservices erforderlich - eine Situation, die vermieden werden soll. Dieses Szenario kann aufgrund von binären Abhängigkeiten auftreten, bei denen verschiedene Versionen nicht mehr kompatibel sind. Die Bereitstellung muss zeitlich so abgestimmt sein, dass alle Microservices in einem bestimmten Zeitintervall und in einer definierten Reihenfolge bereitgestellt werden. Die Code-Abhängigkeit muss auch in allen Microservices geändert werden. Dieser Prozess muss von alle beteiligten Teams priorisiert und berücksichtigt werden. Eine Abhängigkeit auf Binärebene ist eine sehr enge technische Kopplung, die zu einer sehr engen Kopplung führt, die vermieden werden soll[1].

4.3.2 Monitoring

Jeder Microservice muss ebenfalls überwacht werden. Nur so können zur Laufzeit und Testen , Probleme mit dem Dienst diagnostiziert , und festgestellt werden. Mit einem Bereitstellungsmonolithen ist die Überwachung des Systems relativ einfach. Wenn Probleme auftreten, kann sich der Administrator beim System anmelden und mithilfe bestimmter Werkzeuge Fehler analysieren und korrigieren. Microservice-basierte Systeme enthalten so viele Systeme, dass dieser Ansatz nicht leicht durchführbar ist. Folglich muss es ein Überwachungssystem geben, das Überwachungsinformationen von allen Diensten zusammenbringt. Diese Informationen sollten nicht nur die typischen Informationen des Betriebssystems und der E / A zur Festplatte und zum Netzwerk enthalten, sondern auch einen Blick in die Anwendung basierend auf Anwendungsmetriken ermöglichen. Nur so können herausgefunden werden, an welcher Stelle von der Anwendung korrekt ist und wo es derzeit Probleme gibt[6].

4.3.3 Die Kommunikationen zwischen Komponenten sind kompliziert

Da Microservices aus unabhängigen Komponenten sind , müssen Sie Anfragen zwischeneinander sorgfältig bearbeiten. In einem solchen Szenario müssen Entwickler möglicherweise zusätzlichen Code schreiben, um Störungen zu vermeiden. Im Laufe der Zeit treten Komplikationen auf, wenn bei Fernanrufen eine Latenz auftritt. Die Kommunikation unter der Services kann in zwei Kategorien unterteilt werden [15]:

Kategorie 1: Synchron , und Asynchron : Synchron basiert auf Request/Response : Service A schickt ein Anfrage an Service B , und wartet auf die Antwort. Es wird zu dieser Kommunikationsform REST Protokoll verwendet. Asynchron basiert Publish/subscribe :Service A schickt ein Anfrage an Service B ,und wartet nicht auf die Antwort ,sondern arbeitet weiter.Es wird zu dieser Kommunikationsform AMQP and STOMP Protokoll verwendet.

Kategorie 2:

One To One : ein Client sendet eine Anfrage an Service ,und wird von dem Service verarbeitet. One To Many :ein Client sendet eine Anfrage an Services ,und wird von ihnen

verarbeitet.

4.3.4 Verwalten von Microservices

Mit zunehmender Anzahl von Microservices wird die Verwaltung immer schwieriger. Es ist wichtig, dass das Management vor oder während des Aufbaus von Microservices geplant wird. Während die Modularität hilft, können Dinge sehr schnell außer Kontrolle geraten, wenn sie nicht gut verwaltet werden. Es wird festgestellt, dass das Missmanagement dieser Dienste ebenso ein Problem darstellt wie Probleme, die in den ersten Phasen der Transformation von monolithischen Anwendungen auftreten. Die Entwicklung von Microservices-Management-Tools auf eigene Faust ist zwar eine gute Option, kann jedoch komplex und umständlich sein. Es wird in diesem Fall empfohlen, eine Plattform zu erwerben, oder zu entwicklen , deren Funktionen das Microservices-Management umfassen[2].

4.3.5 Service-Erkennung

Auch wenn die Anzahl der Microservices für System wächst, wächst auch die komplexität des Systems , und kann es schwierig sein , einen geeigneten Service für bestimmte Aufgabe zu finden es soll eine standardmäßige und konsistente Komponente oder ein einheitliches Modul geben , bei denen alle Microservices zusammen mit ihren Details und ihrem Status des Dienstes registrieren sollen . Dies wird den Microservices und API-Gateways bei der Erkennung und Verfügbarkeit von Diensten helfen[14].

4.3.6 Sicherheit

die Sicherheit wird als eine große Herausforderung in jedem Softwaresystem betrachtet . Bei mehreren großen Microservices ist es jedoch eine ernsthafte Herausforderung, den unbefugten Zugriff auf Geschäftsvorgänge auf verschiedenen Ebenen des Systems (Anwendung, Bereitstellungsplattformen usw.) zu verhindern. Jeder Microservice muss die Authentizität des Clients überprüfen und validieren, der auf seine Geschäftsvorgänge zugreift. Um die Sicherheit zu gewährleisten, ist daher ein Modul erforderlich, das die autorisierten Benutzer identifiziert kann und die sicheren Sitzungen zwischen autorisierten Benutzern und Microservices bietet[4].

5 Lösungsvorschläge

5.1 Frameworks

Warum wird Framework von Unternehmen heutezutage verwendet , um Anwendung zu entwicklen?
1- Die Komplexität bei der Entwicklung hat stark zugenommen ,Customer erwarten, dass alle beteiligten bei der Organization weiß , wer sie sind.
2-Kunden möchten eine schnellere Lieferung ihres Produktes - Kunden möchten nicht mehr auf die update ,oder eine neue Version eines Softwarepakets,das jährliche Veröffentlicht werden , warten.
3-Kunden erwarten, dass ihre Anwendungen immer verfügbar sind.
4-Performanz und Skalierbarkeit.
5-Fehler oder Probleme in einem Teil der Anwendung dürfen nicht das gesamte System beeinträchtigen
aus diesen Gründen wird heutigen software Produkte anders entworfen und entwickelt , es wird ein anderer Ansatz befolgt , um Anwendungen zu implementieren.
Ein Framework wie Spring kommt sehr häufig in Produktion ,Spring Framework bietet viele Features ,um Entwicklung der Anwenung zu erleichtern ,es wurde erwähnt ,welche

Herausforderungen Microservices-Architektur durch Benutzung einbringt , Spring Framework bietet auch Möglichkeiten ,um dies zu überwinden , beispielsweise [13]:

5.1.1 Monitoring

Spring Framework ermöglicht es unterschiedliche Komponenten zu verwalten .

5.1.2 Service Discovery

durch Eureka können die Services einander finden , und es kann einen Griff auf jedes Service haben[15].

5.1.3 circuit breakers

es ist erforderlich ein System zu entwerfen , das Fehler oder Ausfall eines Services behandeln kann , dieses Michanismus ist sehr notwendig ,wenn zwei oder mehrere Services miteinander kommunizieren ,wenn ein Service ausfällt ,wird den Fehler entdeckt ,und das andere Service wird nicht beeinträchtigt[20].

5.1.4 Sicherheit

Spring ermöglicht auch die Sicherheit der Komponenten durch authentifizierungs- und Autorisierung , es bietet OAuth2 es ist ein a token-based security framework wer ist erlaubt , auf die Services zugreifen darf , und was wird mit diesem Service gemacht werden

5.2 Container : Hosting der Microservices

folgendes ist die Defination von Container ,und welche Lösungen es bietet [10]: Docker ist in der Lage, die Herausforderungen zu lösen, die mit Microservices wie Komplexität, Fehlertoleranz usw. verbunden sind. Die Nutzung von Docker für die Bereitstellung , oder Hosting von Microservices eignet sich gut für die Bereitstellung von Anwendungen in der Cloud. Um Microservices mit Docker bereitzustellen, müssen wir zuerst alle Microservices für eine Anwendung erstellen und dann Docker-Images für diese unabhängigen Services erstellen. Wir können auch einen Dienst und seine abhängigen Module als Dienst in einem Stapel zusammenfassen und ein einzelnes Image erstellen. Mit diesen Docker-Images werden Container erstellt, die angeben, welcher Prozess gestartet werden soll und welche anderen Konfigurations-Metadaten benötigt werden, wenn der Container bereitgestellt oder gestartet wird. Mit Docker wird jedes Microservice-Image auf einem Container bereitgestellt und kann als Microservice-Instanz aufgerufen werden. Die einzelnen Container werden miteinander verknüpft, um die zusammenhängende Anwendung zu bilden. Die Anwendung kann einfach skaliert werden, indem ein oder mehrere Container mit den entsprechenden Service-Images anstelle der gesamten Anwendung bereitgestellt werden. Es gibt mehrere Vorteile, die wir durch die Bereitstellung von Microservices mit Docker erzielen können.

5.2.1 Unabhängigkeit

Jeder Docker-Container ist für einen bestimmten Dienst oder von Diensten in einem Container vorgesehen. jeder Dienst wird unabhängig voneinander in einer beliebigen Sprache mithilfe verschiedener Tools und Prozesse implementiert, die die verschiedenen von Docker bereitgestellten Plattformen nutzen.

5.2.2 Skalierbarkeit

die Anwendung kann einfach skaliert werden, indem die erforderlichen Dienstinstanzen / Docker-Container nach Bedarf gestartet werden.

5.2.3 die Automatisierung

Da das Erstellen und Bereitstellen von Docker-Containern skriptfähig ist, kann jeder Schritt der Anwendungsentwicklung mit der Bereitstellung von Diensten einfach automatisiert werden.

5.2.4 einfache Entwicklung und Bereitstellung von Microservices

Docker macht Microservices einfach tragbar und isoliert. Es gibt keine Abhängigkeitskonflikte sowie die Notwendigkeit, jede Umgebung zu konfigurieren oder Bibliotheken zu installieren.Docker ermöglicht dem Entwickler die Produktionsumgebung in der lokalen Entwicklungsumgebung problemlos zu imitieren. Wenn die Entwickler Docker in einer Produktionsumgebung verwenden möchten, stehen mehrere Tools für die Skalierung, Bereitstellung und Verwaltung dieser Container zur Verfügung. Diese Tools wie Kubernetes erleichtern die Lösung dieser Herausforderungen . Kubernetes bietet mehrere Funktionen wie horizontale Skalierung, Serviceerkennung und Lastausgleich.

5.3 Cloud Technologies

Cloud-Technologie ermöglicht die folgenden [18]:

5.3.1 Zuverlässigkeit

Die meisten Services , die von Cloud geboten sind,ermöglichen hoch verfügbarkeit und Fehlertoleranz. Wenn Sie diese Dienste benutzt werden,werden sie kostenlos erhalten.
Cloud Technologie unterstützt zuverlässigen Erstellen von Systemen. Es bietet alles, was Sie benötigen, um Ihre eigenen hochverfügbaren oder fehlertoleranten Systeme zu erstellen.

5.3.2 Service-Discovery

Cloud-Technologie bietet auch die Erkennung von Diensten,die Services können einander finden.

5.3.3 Skalierbarkeit

Durch die Verwendung der Cloud-Technologie ist es einfach Systeme zu skalieren[14].

5.4 Evaluation der Architektur

Durch Benutzung der Microservices-Architektur wird es gegenüber monolithischen Anwendungen eine Reihe von Vorteilen geben. Microservices-Architektur behebt die Probleme ,die Durch Arbeiten mit einer großen Codebasis verursacht wird, diezum beispiel ihre Entwicklungsumgebung überlasten und die Produktivität einschränken kann, Durch Benutzung der Architektur wird das Ändern des Technologie-Stacks ohne Beeinträchtigung der funktionalität der Anwendung erfolgen[7]. Mit der Microservices-Architektur kann einen neuen Technologie-Stack für einen einzelnen Service mit weniger Bedenken hinsichtlich Abhängigkeiten ausprobiert und Änderungen bei Bedarf viel einfacher rückgängig gemacht wird. Wenn ein Service ausfällt, hat dies keine Auswirkungen auf den Rest der Anwendung. Es wird aber die Sicherheitsaspekte berücksichtigt,denn in der Zukunft

werden bei Microservices-Plattformen Mechanismen zur Überwachung und Durchsetzung der Verbindungen zwischen Microservices benötigt, um das Vertrauen einzelne Microservices zu ermöglichen und den potenziellen Schaden zu begrenzen, wenn ein Microservice beeinträchtigt wird. es wird auch in der Zukunft die komplexität des Netzwerks mehr berücksichtigt,denn bei der Erstellung einer Anwendung ,die auf Microservices-Architektur basiert wird,sollen verschieden Komponenten oder Services miteinander interagieren, dies löst Schwierigkeiten bei der Sicherheit,Überwachung ,und Analyse der Fehler[19].

6 Stand der Technologie

6.1 Serverless Computing

bei Serverless Computing können die Services , oder Anwendungen im Cloud bereitgestellt,skaliert,und verwaltet. Serverless ermöglicht den Anwendungsentwickler ,dass sie nur auf den Code (Implementierung) zu konzentrieren ,ohne den Server zu berücksichtigen. Serverless Computing bietet neues Plattform , das als FaaS (Function as a Service), indem die Anwendung in Funtionen aufgeteilt werden. diese Technik bietet die Möglichkeit ,eine verfügbare ,und fehlertolerante Anwendung zu entwicklen[16].

7 Zusammenfassung

die monolithische-und Microservices-Architektur haben ihre Vorteile und Herausforderungen.der Entwickler muss am Anfang der Entwicklungsphase einen Plan oder Konzept haben , wie das Software-Produkt implementiert wird. für monolitisch kann sich entschieden ,wenn eine einfache Anwendung zu entwicklen, und keine viele Skalierbarkeit zu erwarten.mit der Skalierbarkeit wächst die Komplexität an. für Microservices-Architektur kann sich entschieden, wenn das Software-Produkt skaliert ,aktualisiert wird. Durch Verwendung der Microservices-Architektur geniesst die Entwickler auch andere Vorteile ,wenn ein Service ausfällt, funktionieren die anderen Services.außerdem werden werden die Fehler einbeschränkt ,und einfach zu beheben. Microservice-Architektur hat auch Herausforderungen ,die auch qualifizierte Entwickler benötigt wird ,um sie zu lösen.wenn es gelungen wird ,diese Herausforderungen zu überwinden, wird eine höhe Software-Qualität,und ein in-Time Software-Produkt erwartet.

8 Literaturquellen

Literatur

[1] Eberhard Wolff,Microservices: Flexible Software Architecture,12.10.2016,Verlag: Addison-Wesley Professional

[2] Chris Richardson,Microservices Patterns: With examples in Java,19.11.2018,Verlag: Manning Publications; 1st edition

[3] Lorenzo De Lauretis:From Monolithic Architecture to Microservices Architecture,[online] https://ieeexplore.ieee.org/document/8990350,Konferenz-datum: 27-30.10.2019 ,Konferenz in: Berlin, Germany, abgerufen am (10.01.2021)

[4] Raja Mubashir Munaf, Jawwad Ahmed; Faraz Khakwani; Tauseef Rana:Microservices Architecture: Challenges and Proposed Conceptual Design,[on-line] https://ieeexplore.ieee.org/document/8737831,Konferenz-Datum : 20-21.03.2019, Konferenz in :Rawalpindi, Pakistan, abgerufen am (05.01.2021)

[5] Konrad Gos; Wojciech Zabierowski:The Comparison of Microservice and Monolithic Architecture,[on-line] https://ieeexplore.ieee.org/document/9109514,Konferenz-Datum : 22-26.04.2020 , Konferenz in :Lviv, Ukraine, abgerufen am (05.01.2021)

[6] Sam Newman:Building Microservices: Designing Fine-Grained Systems,Verlag :O'Reilly Media,veröffentlich am:02.02.2015

[7] Pooyan Jamshidi; Claus Pahl; Nabor C. Mendonça; James Lewis; Stefan Tilkov:Microservices: The Journey So Far and Challenges Ahead,[on-line] https://ieeexplore.ieee.org/abstract/document/8354433, veröffentlicht am: 04.05.2018, veröffentlicht in:IEEE Software

[8] Morgan Bruce ,Paulo A. Pereira:Microservices in Action,Verlag : Manning Publications, 1st edition, 05.11.2018

[9] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty Mike Amundsen :Microservice Architecture: Aligning Principles, Practices, and Culture ,Verlag : O'Reilly Media; 1st edition,18.07.2016

[10] Sarita, Sunil Sebastian :Transform Monolith into Microservices using Docker,[online] https://ieeexplore.ieee.org/document/8463820 ,Konferenz-Datum : 17-18.08.2017 , Konferenz in : Pune, India, abgerufen am (05.01.2021)

[11] James Lewis,Martin Fowler :Microservices a definition of this new architectural term,[on-line] https://martinfowler.com/articles/microservices.html ,veröffentlich am: 25.03.2014,abgerufen am (05.01.2021)

[12] Zhamak Dehghani:How to break a Monolith into Microservices,[online]https://martinfowler.com/articles/break-monolith-into-microservices.html?ref=wellarchitected ,veröffentlich am :24.04.2018,abgerufen am (06.01.2021)

[13] John Carnell :Spring Microservices in Action,verlag :Manning Publications; 1st edition ,veröffentlicht am :06.07.2017

[14] Vinod Pachghare,Microservices Architecture for Cloud Computing ,hochgeladen am :28.08.2018 ,veröffentlich in: Department of Computer Engineering and Information Technology, College of Engineering, Pune, India

[15] Chris Richardson:Building Microservices: Inter-Process Communication in a Microservices Architecture, [on-line]https://www.nginx.com/blog/building-microservices-inter-process-communication/, veröffentlich am :24.07.2015,abgerufen am :(13.01.2021)

[16] Maddie Stigler:Beginning Serverless Computing,Verlag: Apress,24.11.2017

[17] Erko Risthein:Migrating the Monolith to a Microservices Architecture: the Case of TransferWise Bachelor Thesis,TALLINN UNIVERISTY OF TECHNOLOGY Faculty of Information Technology Department of Informatics Chair of Information Systems,2015

[18] Andreas Wittig,Michael Wittig :Amazon Web Services in Action,Verlag :Manning Publications,veröffentlicht am:20.10.2015

[19] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara Fabrizio Montesi, Ruslan Mustafin, Larisa Safina; Microservices: yesterday, today, and tomorrow,Cornell Univercity in new York,13.06.2016

[20] Martin Fowler: Circuit Breaker [Online] https://martinfowler.com/bliki/CircuitBreaker.html, veröffentlich am :6.03.2014 ,abgerufen am 12.01.2021