#backend

10. Oktober 2019

Software-Architektur-Lösungen für die Probleme der Versicherungswirtschaft

Seit jeher ist es ein Spaß zu beobachten, wie bestimmte Buzzwords in den einschlägigen Fachkreisen und Chef-Etagen die Runde drehen und danach Projekte und Entscheidungen beeinflussen. “Artificial Intelligence”, “Blockchain” und “Cloud”, um nur einige davon zu nennen. Alles sehr wichtige Themen - aber nicht in jedem Kontext die Rettung bei allen Problemen rund um IT. Zwei brandaktuelle Begriffe, die in den Kreis der alltäglichen Gesprächführung aufgenommen werden wollen: “Monolithische Architektur” und "Micro Services". Aber ist die IT-Welt wirklich so schwarz und weiß? Ist das eine gut und das andere per se schlecht? Oder liegt die Wahrheit irgendwo dazwischen und ist das überhaupt die richtige Fragestellung?

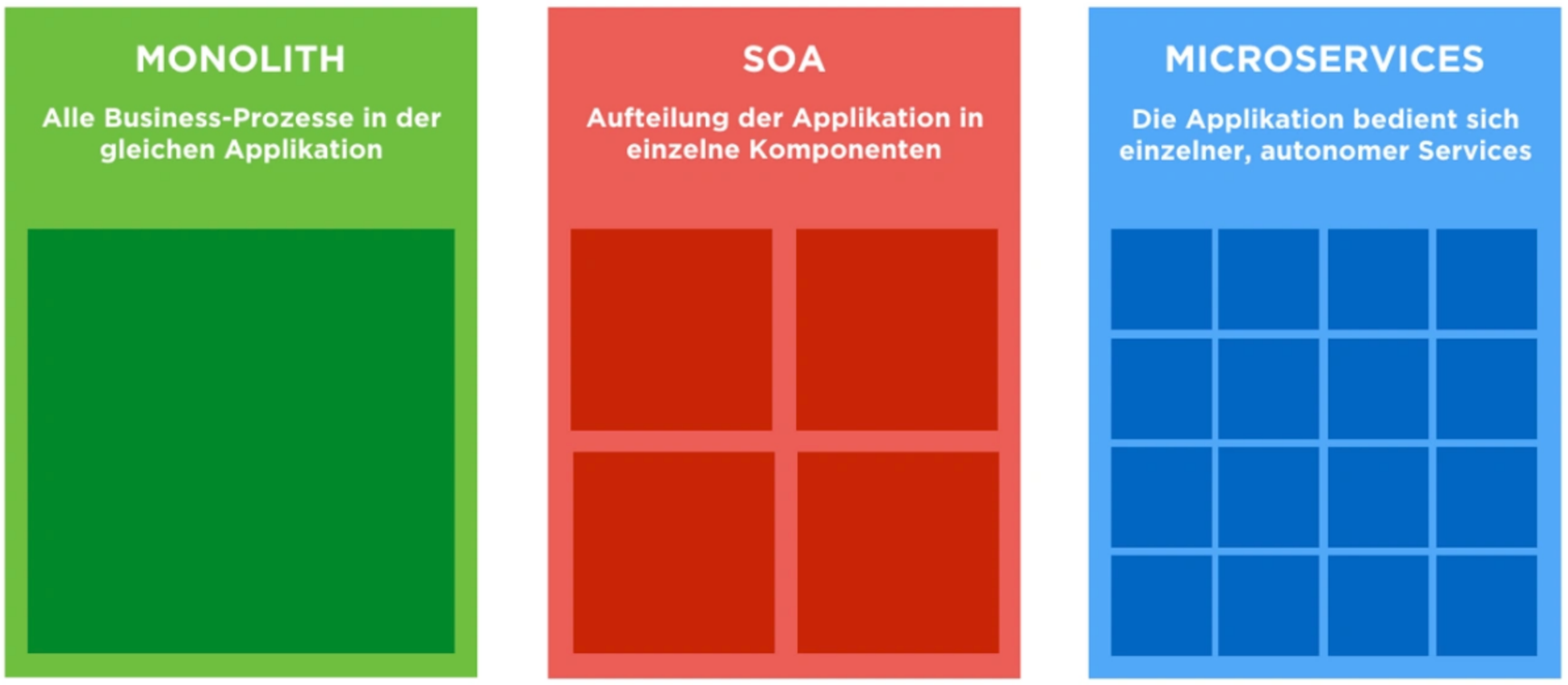
Was ist ein Monolith?

Im Kontext von Software-Architektur spricht man von einem Monolithen, wenn "alle" Geschäftsprozesse innerhalb einer Applikation abgebildet sind. Im Fall einer Web-Applikation geht das so weit, dass die Online-Zugriffe auf die Applikation, die Zugriffe auf eine Datenbank, das Umsetzen der Business-Logik und das Erstellen der Ausgabe als HTML-Seiten in derselben Applikation vonstatten gehen. Übrigens ist dies ein weit verbreiteter Weg, der seine Ursprünge in den Anfängen des World Wide Web hat. Er beschreibt die Umsetzung von z.B. diverser Content-Management-Systeme, Gästebücher und Foren, ist aber auch für andere Applikationen bis nach 2010 weit verbreitet. Warum ist das auf einmal "schlecht" und was soll generell besser sein?

Was sind Microservices?

Im Gegensatz zur monolithischen Architektur zeichnen sich die Service-Architekturen durch die Aufteilung der Geschäftsprozesse auf autarke Systeme aus. Man spricht von SOA als “Service Oriented Architecture”, wenn die Applikationsprozesse in einzelnen Services aufgeteilt und gekapselt sind. Hierbei ist es nicht zwingend notwendig, dass diese Services nur eine einzige, streng dedizierte Aufgabe abbilden dürfen. Wenn Applikationen in solch kleine Elementar-Services ausgeteilt werden, spricht man von "Microservices". Hier sind die einzelnen Services noch wesentlich kleiner – also gibt es also auch mehrere davon – und ebenso autonom nutzbar.

Die drei unterschiedlichen Architektur-Modelle im Überblick:



Vorteile eines Monolithen

Stellen Sie sich vor, Sie arbeiten in einer nicht-monolithischen Arbeitsumgebung an einem komplexen Excel-Sheet. Sie haben eine Tabelle erstellt und möchten nun das Ergebnis der Berechnung aus einer Formel einfügen. Dazu müssen sie zuerst ein weiteres Programm namens Excel-Functions (ein Service) öffnen und die beiden Programme “miteinander bekannt machen”. In Ihrem Excel-Sheet setzen sie nun einen Befehl ab, der dem Programm Excel-Functions die Attribute für Ihre Formel sendet. Nach

ein paar Millisekunden erhalten Sie eine Antwort mit dem Ergebnis aus der Berechnung. Im nächsten Schritt möchten Sie nun aus Ihrer Tabelle einen Graphen – sagen wir ein Kuchen-Diagramm – erstellen. Dafür müssen Sie das Programm Excel-Graphs öffnen, die beiden Programme miteinander bekannt machen, die Werte für das Diagramm an Excel-Graphs senden um dann nach ein paar Millisekunden (oder evtl. Sekunden, weil der Graph so groß ist) den fertigen Graphen zu erhalten.

So wollen Sie bestimmt nicht arbeiten. Sie wollen Excel öffnen und all die oben beschriebenen Funktionalitäten innerhalb des gleichen Programms ausführen – sie wollen einen Monolithen.

In der Software-Entwicklung verhält es sich sehr ähnlich wie dieses – zugegebenermaßen recht plakative – Beispiel beschreibt. Es ist nicht immer richtig, alle Funktionalitäten in eine Service-Architektur aufzuteilen. Ein Monolith bietet in den Bereichen Wartung und Deployment, aber auch für Fehlersuche viele Vorteile, da all dies an einer zentralen Stelle vonstatten gehen kann. Die Organisation des Quellcodes ist ebenfalls einfacher, da dieser in nur einem Code-Repository zu finden ist. Und letztlich ist auch das Monitoring einfacher, da nur wenige zentral verfügbare Metriken untersucht werden müssen. Und schlussendlich gibt es bei vielen Applikationen auch schlichtweg keinen Grund und keinen Bedarf, diese in Services aufzuteilen.

Gründe für eine Service-Architektur

Natürlich kommt es sehr darauf an, welche Art Applikation wir entwicklen wollen und welche Business-Anforderungen dieser zu Grunde liegen. Das obige Beispiel trifft wohl eher auf eine relativ einfache Applikation zu. Wird diese komplexer, zum Beispiel durch die Integration anderer Applikationen, die in unserer Plattform benötigt werden, ist es sinnvoll die Applikation in logische oder funktionale Blöcke zu schneiden.

Nehmen wir z.B. an, wir betreiben eine KFZ-Versicherungsplattform und wir sehen uns mit der extrem hohen Anzahl von Preisanfragen von bis zu 1500 pro Sekunde konfrontiert. Durch Performance-Tests stellen wir fest, dass die Technologie unser Applikation diesen Anforderungen nicht Rechnung tragen wird. Aufgrund des monolithischen Aufbaus können wir aber nicht einfach für dieses Teilsystem eine andere Technologie auswählen. Wir sollten uns in diesem Fall entscheiden, einen Pricing-Service zur Verfügung zu stellen, der nicht nur von unseren angeschlossenen Aggregatoren genutzt werden soll, sondern auch unsere Basis-Applikation soll diesen nutzen. Weiterhin bietet sich durch den Service die Möglichkeit, eine andere Technologie bzw. Programmiersprache einzusetzen, was zu weiteren Vorteilen führt. Und schon haben wir den ersten Service erdacht, erstellt und in Betrieb genommen.

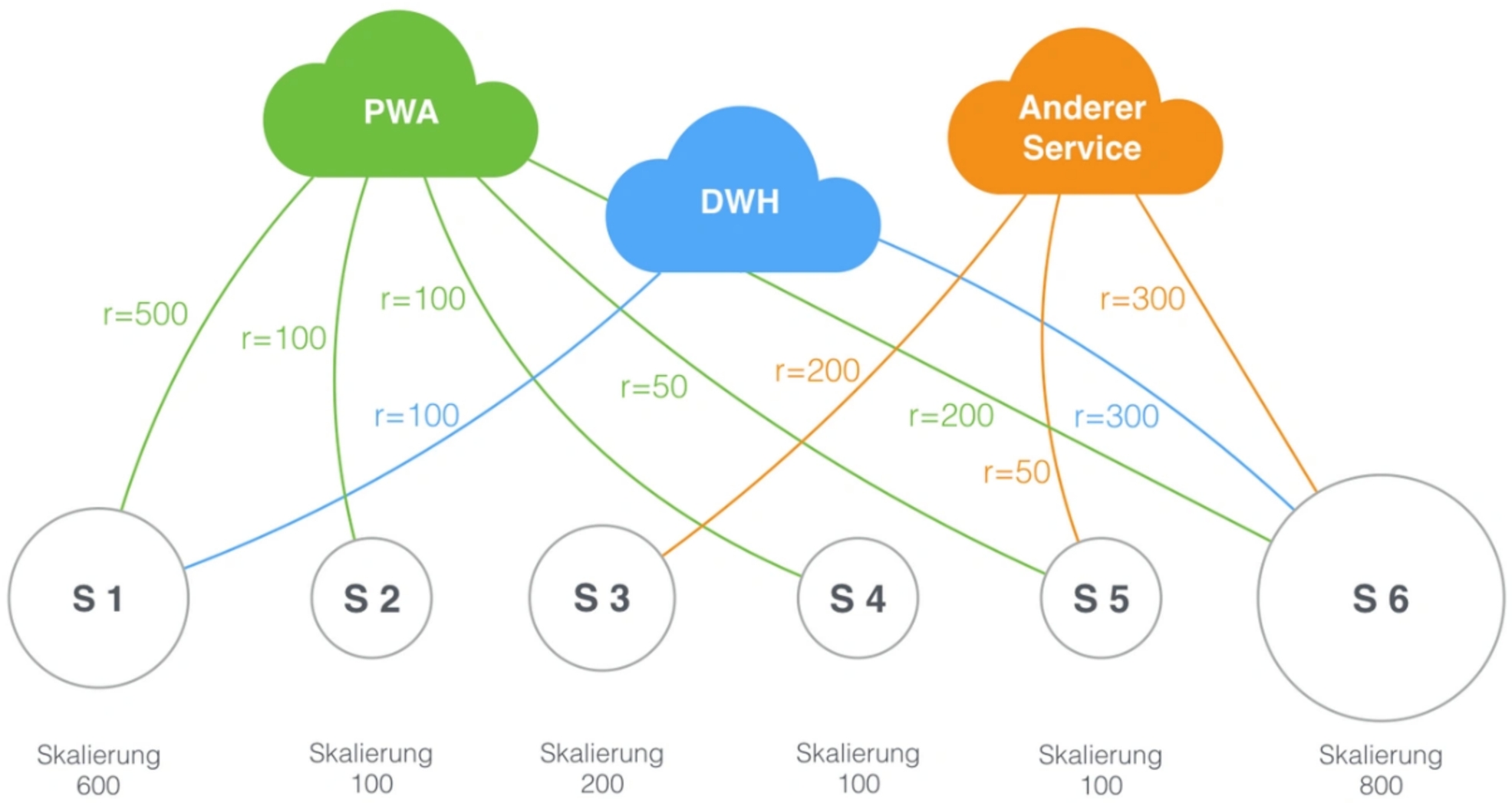
Diese Art von Service können wir nun für weitere Teile der Applikation adaptieren. Das können die Schadenaufnahme, die Dokumenterstellung, die Provisionsberechnung oder

das Nebenbuch sein, um nur einige Beispiele zu nennen. Durch diese Architektur sind wir in der Lage, die einzelnen Teile unserer Applikation – oder besser Plattform – dediziert skalieren zu können. Auch bei der Weiterentwicklung ergeben sich Vorteile, denn die Pricing-Engine ist ein autarkes Stück Software. Es kann getrennt von der Basis-Applikation konzipiert, entwickelt, getestet, skaliert und deployed werden.

Was hat es nun mit Microservices auf sich?

Treiben wir dies nun noch weiter, gelangen wir zu einer Microservice-Architektur. In dieser muss keine Basis-Applikation mehr bestehen. Sämtliche Funktionalitäten werden in kleine, autonome Services ausgelagert, die jeweils nur eine fachliche Bestimmung haben und diese ausführen. Die Kommunikation unter den einzelnen Services wird in den meisten Fällen über die Implementierung einer “Messaging-Lösung” bewerkstelligt. Diese kann synchron, aber auch asynchron erfolgen.

Die Vorteile einer Microservice-Architektur sind in der komplett getrennten Implementierung der Anforderungen zu finden. Sie sind oft nur wenige hundert Zeilen Quellcode groß und somit schnell zu überblicken, weiterzuentwicklen und sehr schnell zu deployen. Ausserdem ist die Skalierbarkeit der einzelnen Microservices ein sehr großer Vorteil. Aufgrund der Menge an Anfragen und der Häufigkeit der Einbindung des Microservices ergibt sich dann die Anforderung an die Skalierung dieses einzelnen Services. Die sehr vereinfachte Darstellung in der folgenden Grafik zeigt dies:



r = Request (Anfrage)
S = Service
PWA = Progressive Web App
DWH = Data Warehouse

Allerdings darf die Komplexität des Zusammenspiels der Services auf keinen Fall unterschätzt werden. Alleine zu wissen, welche Daten mit welchen Werten an jedem einzelnen Microservice zu einer bestimmten Zeit vorhanden sind, ist sehr komplex und erfordert ganz spezielle Mechaniken und letztlich auch Tools, um dies zu überwachen.

Was ist die "richtige" Lösung?

Es gibt keine “One-size-fits-all”-Lösung. Es sollte immer eine “Just-the-right-size”-Lösung sein. Was ist damit gemeint?

Nicht jede Lösung passt zu jeder Anforderung. Und jede dieser Lösungen hat ihre Vor-, aber auch Nachteile.

Bei sum.cumo sind alle drei Architektur-Modelle im Einsatz. Aus guten Gründen. Eine Applikation, die eher wenig “Traffic” im Durchschnitt vorzuweisen hat, muss nicht in eine komplexe Microservice-Architektur geschnitten werden. Die Komplexität bei der Erstellung und der administrative Aufwand wären wirtschaftlich nicht gerechtfertigt. Auch die Deployment-Zyklen sind in der Regel nicht so hoch, als dass auf eine Microservice-Architektur zurückgegriffen werden müsste. Bei diesen Applikationen geht es um Stabilität und einfache Wartung.
Bei einer Plattform mit hohen Zugriffszahlen und vielen unterschiedlichen Transaktionen sind Skalierung, schnelles Deployment und kurzfristig Applikations-Anpassungen ein Muss. Wer zum Beispiel schnell viele Mandanten mit einem Plattform-Setup betreiben will, kann die Services mehrfach nutzen können. Hier bietet sich ganz klar eine Microservice-Architektur an.

Aktuell zeigt die Erfahrung, dass die Richtung einer service-orientierten Architektur in den meisten Fällen für Versicherungsplattformen sehr gute Dienste leistet. Dabei ist wichtig, dass nicht von vornherein alles in Services ausgelagert werden muss. Es zeigt sich, dass im ersten Schritt die Organisation des Quellcodes in abgeschlossene Komponenten alle Möglichkeiten für eine spätere Extraktion bzw. Auslagerung in Services bietet. Dabei kann es sich um interne Services oder um Microservices handeln. Selbst eine Mischung ist denkbar, muss aber gut überlegt sein. Allerdings ist es egal, welche Architektur man wählt: der Quellcode muss immer in Komponenten geschnitten sein, denn sonst findet man einen nicht wartbaren Code-Klumpen vor oder den berühmten "Spaghetti-Code", bei dem der Code völlig wild verwoben ist und nur schwer verstanden werden kann.

Eine Lösung muss iterativ, qualitativ hochwertig, gut geschnitten sein und anforderungsgemäß entwickelt und weiterentwickelt werden. Die Anforderungen an das Geschäftsmodell bestimmen den Weg.

Auflösung

Es gibt die “One-size-fits-all”-Lösung nicht. Monolithen haben Vorteile gegenüber Microservices und Microservices gegenüber Monolithen. Die Auswahl der Software-Architektur ist nicht der Anfang eines Projekts sondern die genaue Planung des Geschäftsmodells und der Implikationen, die dieses im Laufe der Jahre auf die IT-Landschaft haben wird. Gute Entscheidungen folgen somit nicht blind den aktuellen Trends der Meinungsbildung, sondern basieren auf der genauen Abwägung aller Anforderungen.

AUTOR:IN

Andreas Wenk

#tech

Andy ist seit Januar 2014 bei sum.cumo, heute Mitglied der Geschäftsleitung und verantwortlich für den Fachbereich Backend & Betrieb. Er sorgt dafür, dass sich Kolleginnen und Kollegen in ihrer innovativen Kreativität ausleben können. Außerdem erarbeitet er mit seinem Team über die Fachbereichsgrenzen hinweg Lösungen für…

Alle Artikel von Andreas →