IKT-Monitoringbericht

Business 4.0 OWL

Ein Beitrag des Software Innovation Campus Paderborn – SICP im Rahmen des Projektes Business 4.0 OWL.

Vorwort

Im Rahmen des Projektes Business 4.0 engagiert sich der SICP – Software Innovation Campus Paderborn darum, Kompetenzen und Ressourcen zu IKT-Trends und Entwicklungen aus der Region zu bündeln.

In regelmäßigen Treffen werden daher mit Unternehmen aus der Region entsprechende Trends und Entwicklungen gesichtet und mit Blick auf ihre Nutzenpotentiale für und ihre Wirkungen auf die Unternehmen und deren Branchen diskutiert und bewertet. Ziel ist es auch, Transformationsprojekte und Einzelmaßnahmen zu motivieren und anzustoßen. Interessierte Unternehmen sind herzlich eingeladen an diesen Treffen teilzunehmen.

Zur Sicherung der Ergebnisse dieser Treffen werden Steckbriefe zu den einzelnen Themen im Rahmen dieses Berichts veröffentlicht. Der Bericht wird fortlaufend um die Themen weiterer Treffen ergänzt. Die aktuelle Version beinhaltet die Themen aus den ersten beiden Projektjahren.

Das Projekt Business 4.0 ist Bestandteil des integrierten Handlungskonzepts “OWL 4.0 – Industrie, Arbeit, Gesellschaft“ und wird von der Universität Paderborn und InnoZent OWL e.V. durchgeführt. Kooperationspartner sind die Industrie- und Handelskammern Ostwestfalen und Lippe, die Handwerkskammer Ostwestfalen-Lippe und das CPS.HUB NRW. Das Vorhaben wird vom Land NRW und aus Mitteln des Europäischen Fonds für regionale Entwicklung (EFRE) gefördert. Weitere Informationen finden Sie unter: www.owl-morgen.de

Inhaltsverzeichnis

Microservices: Wie sich dank innovativer Technologie Softwareprojekte besser managen lassen

1. Ausgangslage

Software wird heutzutage in nahezu allen Situationen des alltäglichen Lebens genutzt. Die kontinuierlich zunehmende Relevanz steigert gleichermaßen den Anspruch an Qualität und Stabilität. Eine zentrale Herausforderung: die Komplexität und Größe von Softwareprojekten, die es immer schwieriger machen, einen vollständigen Überblick über den geschriebenen Code zu behalten.

Aus diesem Grund haben sich über die Jahre verschiedene Komponententechnologien entwickelt, mit dem Ziel, die Wartbarkeit und Komplexität von Softwareprojekten zu verbessern. Trotz dieser Technologien stellt sich jedoch stets die Frage, ob es weitere Methoden gibt, die Faktoren wie Wartbarkeit und Übersichtlichkeit von Softwarelösungen improvisieren können. In diesem Kontext ist der Begriff „Microservices“ in den Fokus gerückt.

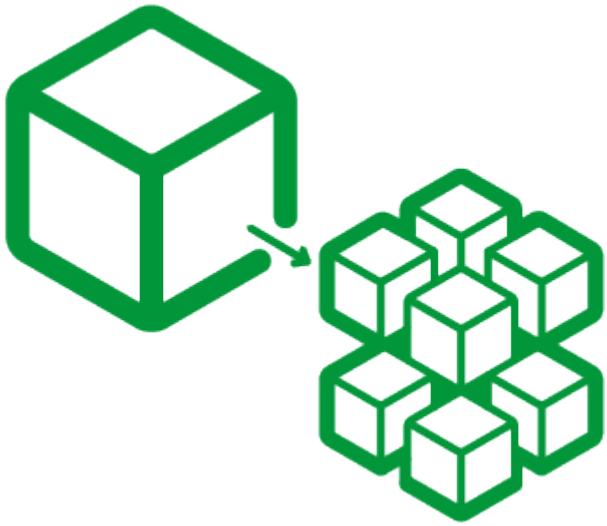

Abbildung 1: Skizze der Idee von Microservices (Quelle: https://www.nginx.com/solutions/microservices/)

2. Die Microservices-Technologie

Microservices sollen Software durch übersichtliche Architektur, bessere Wartbarkeit und Wiederverwertbarkeit einzelner Services verbessern. Im Kern lassen sich Microservices auf drei Kernelemente reduzieren:

1. Ein Programm (d. h. ein Microservice) soll nur eine Aufgabe erledigen, und das bestmöglich.
2. Die Programme sollen zusammenarbeiten können.
3. Es soll eine universelle Schnittstelle genutzt werden.

Aus diesen drei Kernelementen lassen sich verschiedene Kriterien definieren, welche den Begriff Microservices näher beschreiben:

- Microservices sind ein Modularisierungskonzept. Sie dienen dazu, ein großes Softwaresystem in kleinere abgekapselte Softwarekomponenten aufzuteilen. Dadurch beeinflussen sie die Organisation und die Software-Entwicklungsprozesse.
- Microservices können unabhängig voneinander eingesetzt werden. Änderungen in einem Microservice können somit unabhängig von Änderungen an anderen Microservices in Produktion gebracht werden.

- Microservices können in unterschiedlichen Technologien implementiert sein. Es gibt keine Einschränkung auf eine bestimmte Programmiersprache oder Plattform.
- Microservices haben einen eigenen Datenhaushalt, das heißt eine eigene Datenbank oder ein vollständig getrenntes Schema in einer gemeinsamen Datenbank.
- Microservices können eigene Unterstützungsdienste mitbringen, beispielsweise eine Suchmaschine oder eine spezielle Datenbank. Natürlich gibt es eine gemeinsame Basis für alle Microservices – beispielsweise die Ausführung virtueller Maschinen.
- Microservices sind eigenständige Prozesse oder virtuelle Maschinen, um auch eigene Unterstützungsdienste mitzubringen.
- Dementsprechend müssen Microservices über ein Netzwerk kommunizieren. Dazu nutzen Microservices Protokolle, die eine lose Kopplung („loose coupling“) unterstützen. Das können beispielsweise REST oder auch Messaging-Lösungen sein.

3. Vor- und Nachteile von Microservices

In der Literatur werden abhängig vom Kontext verschiedene Vorteile von Microservices genannt. Die folgende Auflistung ist somit als exemplarisch zu verstehen.

Stärken und Chancen von Microservices

Die Nutzung und Entwicklung von Microservices bietet viele Vorteile wie etwa:

1. Microservices sind ein „starkes“ Modularisierungskonzept. Damit ist gemeint, dass Microservices über explizite Schnittstellen kommunizieren, die mit Mechanismen wie Messages oder REST umgesetzt sind. Dadurch ergibt sich eine hohe Entkopplung hinsichtlich der übergeordneten Funktionalitäten. Das wiederum führt schlussendlich zu einer Reduktion unerwünschter Abhängigkeiten.
2. Microservices können leichter ersetzt werden als komplette Softwaresysteme. Neue Microservices müssen als Ersatz weder die Code-Basis noch die Technologien des alten Microservice übernehmen. Das reduziert zusätzliche Kosten aufgrund von Fehlentscheidungen, weil der Einsatz von neuen Technologien jederzeit angepasst werden kann.
3. Microservices binden sich gut in ein Time-to-market-Konzept ein. Denn Microservices können einzeln und je nach Bedarf in Produktion gebracht werden. Das zuständige Team kann ohne großen Koordinationsaufwand mit anderen Teams die nötigen Features entwickeln und bereitstellen („deployment“). Das erhöht die Möglichkeiten der parallelen Arbeit und fördert agile Prozesse.
4. Jeder Microservice kann unabhängig von anderen Services skaliert werden. So kann ein Microservice bei einer erhöhten Nutzerlast ausgetauscht oder mit mehr Ressourcen ausgestattet werden, ohne dass sich diese Veränderung auf andere Microservices auswirkt.
5. Microservices unterstützten das Konzept des „continuous delivery“: Da Microservices klein und unabhängig voneinander sind, können diese einfacher live gehen bzw. bereitgestellt werden. Zusätzlich verringert dieses Konzept das Risiko: Die Bereitstellung („Deployment“) vieler kleiner voneinander unabhängiger Komponenten ist deutlich weniger riskant und fehleranfällig als beispielsweise das Deployment eines großen Monolithen.
6. Microservices unterstützen das so genannte „End-to-end-ownership“-Modell im Entwicklungsprozess und erhöhen die Teamproduktivität. In der Regel kümmert sich jeweils ein Entwicklerteam um alle Phasen der Entwicklung eines Microservices. Das vereinfacht die interne Kommunikation und steigert die Vielfalt während der Entwicklung.

Risiken und Herausforderungen von Microservices

Die Nutzung und Entwicklung von Microservices birgt jedoch auch einige Risiken und Herausforderungen:

1. Die Architektur des Systems besteht aus den Beziehungen der Services. Dies kann zu Schwierigkeiten in der Organisation von Microservices führen, da nicht direkt ersichtlich ist, welcher Microservice dem jeweils anderen zugeordnet ist.
2. Eine Refaktorisierung (die manuelle oder automatisierte Strukturverbesserung von Quelltexten), bei der die Funktionalitäten zwischen Microservices verschoben werden, sind schwer umsetzbar.
3. Die Aufteilung des Systems in Microservices ist nachträglich nur schwer zu ändern.
4. Die initiale Aufteilung des Systems in fachliche oder logische Microservices ist von entscheidender Bedeutung. Anfängliche Fehler bei der Aufteilung können schwerwiegende Folgen nach sich ziehen.
5. Durch die hohe Anzahl an Microservices im „Live-Betrieb“ steigt die Komplexität an die Betriebsinfrastruktur. Microservices erzwingen eine Automatisierung der Betriebsprozesse um den Aufwand gering zu halten.
6. Eine Herausforderung stellt die Sicherstellung der richtigen Service-Granularität dar.
7. Auch die Sicherstellung eines transparenten Monitorings und Reportings bezüglich Microservices verbunden mit einer konsistenten und transparenten Fehlerbehandlung ist wichtig, um Risiken zu minimieren oder auszuschließen.
8. Wichtig ist zudem die Sicherstellung eines konsistenten Transaktions-Handlings über Service-Grenzen hinweg.
9. Zudem sollte ein geeignetes Governance und Service-LifeCycle-Management sichergestellt sein.

4. Weiterführende Literatur

Online Ressourcen

1. https://www.informatik-aktuell.de/entwicklung/methoden/microservice-architekturen-nicht-nur-fuer-agile-projekte.html
2. https://www.informatik-aktuell.de/entwicklung/methoden/von-monolithen-und-microservices.html
3. http://www.handelskraft.de/2016/08/services-die-zukunft-der-softwarearchitektur-liegt-in-der-unabhaengigkeit/
4. https://www.internetworld.de/technik/microservices/microservices-shops-bausteinen-1146827.html

Bücher

1. Eberhard Wolff. *Microservices: Grundlagen flexibler Softwarearchitekturen*. dpunkt, 2015, ISBN 978-3-86490-313-7
2. Mike Amundsen, Matt Mclarty. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O’Reilly Media, 2016, ISBN 978-1-4919-5625-0

Reactive Design: robuste, skalierbare und anpassungsfähige Softwarearchitektur ermöglichen

1. Ausgangslage

Die heutigen Erwartungen an Software unterliegen dem rapiden Wachstum der Nutzung von Software im technischen wie auch im sozialen Sinne. Der Wandel hin zu einer digitalen Gesellschaft und der weltweiten Vernetzung durch das Internet machen Software allgegenwärtig. Die Art und Weise, wie moderne Systeme implementiert werden, unterliegt ebenfalls diesem Wandel. Um den heutigen Ansprüchen zu genügen, müssen Softwaresysteme heute deutlich robuster, skalierbarer und anpassungsfähiger sein.

2. Das Reactive Manifesto: technologieübergreifende Definitionen für reaktive Systeme

Gegenwärtig werden immer größere Datenmengen und zudem Millionen von Zugriffen durch Softwaresysteme verarbeitet. Auch die „Machine-to-Machine-Kommunikation“ spielt eine immer größere Rolle. Zudem steigt mit dem „Internet of Things“ (IoT) die Anzahl der „Nutzer” von Software erheblich.

Bonér et al. postulieren in ihrem „Reactive Manifesto”, dass moderne Softwarearchitektur den Anforderungen (robuster, skalierbarer, anpassungsfähiger) gewachsen ist. Ein reaktives System besitzt die Eigenschaften „responsive” (antwortbereit), „resilient” (widerstandsfähig), „elastic” (elastisch/flexibel). Um das zu erreichen, muss ein System message-driven (nachrichtenorientiert) sein (siehe Grafik). Man bezeichnet diese Anwendungen dann als reaktive Anwendungen im Sinne des Manifests.
Das Reactive Manifesto beschreibt demnach keine neuen Konzepte oder Paradigmen, vielmehr formalisiert es Begriffe und bietet technologieübergreifende Definitionen für reaktive Systeme.

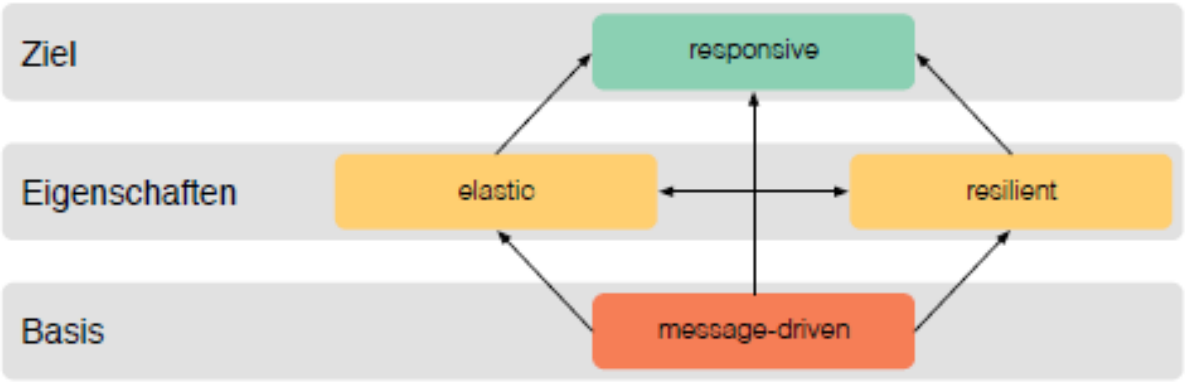

Abbildung 2: Die vier Merkmale einer reaktiven Anwendung (Quelle: https://github.com/alexvetter/thesis-reactive-design-patterns/releases/tag/2016-06-19_release, S. 12):

Aus der oben beschriebenen Definition lassen sich Eigenschaften für reaktive Anwendungen ableiten.

1. Eine reaktive Anwendung muss auf variable Belastung reagieren. Das System ist automatisch in der Lage, dynamische Skalierung durchzuführen und nicht benötigte Ressourcen wieder freizugeben.
2. Eine reaktive Anwendung muss auf Fehler reagieren. Die Software ist von Grund auf widerstandsfähig gegenüber Fehlerzuständen. Die Wiederherstellung des Normalzustands erfolgt automatisch.
3. Eine reaktive Anwendung muss auf Nutzer oder Komponenten reagieren. Jede Anfrage wird in der geforderten Antwortzeit oder schneller bearbeitet.

4. Eine reaktive Anwendung muss auf Nachrichten reagieren. Das System verwendet asynchrone Nachrichtenübertragung zur Kommunikation zwischen Komponenten.

Konsequenterweise werden Prinzipien und Design Patterns (Entwurfsmuster) benötigt, die die Eigenschaften des Reactive Manifestos erfüllen, um eine reaktive Anwendung zu entwerfen. Design Patterns sind hierbei bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme in der Softwareentwicklung. In den Links der weiterführenden Literatur finden sich Quellen, die spezifische Patterns für ein reaktives Design vorschlagen.

3. Vor- und Nachteile

In der Literatur werden abhängig vom Kontext verschiedene Vorteile und Nachteile eines Reactive Designs genannt. Die folgende Auflistung ist somit als exemplarisch zu verstehen.

Vorteile des Reactive Designs

- Ein Reaktives Design bezieht inhärent mögliche Fehler die entstehen könnten ins Design ein (sogenanntes „embrace failure“), da nachrichtenbasierte Kommunikation von Natur aus fehleranfällig ist. Dies erhöht konsequenterweise die Widerstandsfähigkeit des Systems.
- Reaktive Systeme eignen sich unter anderem für Webanwendungen mit beträchtlicher Interaktion und bestimmten Echtzeitanforderungen. Diese müssen mit variabler Last und stetigen Anfragen bzw. Events umgehen können.
- Eine moderne (Web-) Anwendung besteht meist vollständig aus nutzerabhängigen und dynamischen Daten. Der Anteil der Nutzerinteraktion mit einer Webanwendung ist sehr hoch. Jede Interaktion ist einem Event gleichzusetzen und muss entsprechend verarbeitet werden. Auch möchten die Nutzer beispielsweise über Änderungen durch andere Nutzer so schnell wie möglich informiert werden – nicht erst, wenn die Seite neu geladen wird. Die nachrichtenbasierte Kommunikation von reaktiver Software kann hier ideal eingesetzt werden.
- Auch in weiteren Kontexten nutzbar: App-Programmierung, Internet of Things (IoT) – überall da, wo die drei Eigenschaften (antwortbereit, flexibel, widerstandsfähig) hohe Relevanz für das Produkt oder System genießen.
- Schlussendlich führt Reactive (Web-) Design zu einer optimalen Bereitstellung von Inhalten für die Endnutzer und somit zu einem besseren Nutzererlebnis.
- Durch die Anwendbarkeit von Design Patterns können viele Standardprobleme schnell gelöst werden.

Risiken und Herausforderungen des Reactive Designs

- Der Fokus auf nachrichtenbasierte Kommunikation sorgt, bei entsprechende Nutzerskalierung, für einen hohen Aufwand in der Abarbeitung von Nachrichten.
- Die Einstieghürde auf Programmierer-Ebene ist hoch, da ein Reactive Design einen Wechsel des Design- und Programmierparadigmas erfordert.
- Das Prinzip wird kaum unterrichtet in der Ausbildung. Ergo muss gegebenenfalls Personal geschult werden.
- Ein komplettes Re-Design vorhandener Software ist häufig einfacher als ein Design-Change von bestehender Software.
- Dementsprechend ist auch das Einbinden von alter Software in ein neues, reaktives System mit hohem Aufwand verbunden.

4. Weiterführende Literatur

Online Ressourcen

1. https://www.reactivedesignpatterns.com/
2. https://www.lightbend.com/blog/architect-reactive-design-patterns
3. https://www.creativebloq.com/web-design/reactive-design-101413185
4. https://infopark.com/de/blog/reactive-design

Bücher

1. Roland Kuhn, Jamie Allen. *Reactive Design Patterns*. Manning, 2016, ISBN 978-1617291807

Angular: JavaScript für anspruchsvolle Benutzererfahrungen

1. Ausgangslage

Die digitale Transformation verändert viele Aspekte des Alltags in der IT-Welt. Begrifflichkeiten wie Cloud-Computing, Social Media, Big Data, Digital Analytics, Internet-of-Things (IoT) sind direkt oder indirekt Teil des digitalen Wandels. Aufgrund der digital veränderten Welt hat sich die Kundeninteraktion grundsätzlich verändert. Traditionelle Benutzeroberflächen werden durch schnelle und interaktive Benutzerschnittstellen mit einer Vielzahl von digitalen Mediaartefakten ersetzt, die einen Hauptfokus auf Benutzerfreundlichkeit legen. Des Weiteren sorgt der schnelle Anstieg von mobilen Geräten dafür, dass Nutzer eine optimale User Experience (UX) erwarten, egal ob sie Inhalte über Handy, Tablet oder Rechner abrufen. Um dieser Herausforderung zu begegnen, wechseln viele alte, „traditionelle“ Webseiten und Webanwendung auf Technologien wie „Responsive Webdesign“ und „Client-Site-Applications“.

In den letzten Jahren sind viele verschiedene Frameworks erschienen. Unter anderem haben verschiedene JavaScript-Ableger für Furore gesorgt. Aber sind sie auch geeignet für komplexe und anspruchsvolle Applikationen? Wie steht es um Wartbarkeit und Erweiterbarkeit?

In diesem Artikel wollen wir das bekannte Open-Source Framework Angular von Google, dass seit der Version 2 eine vielversprechende Basis für anspruchsvolle Applikationen liefert, vorstellen und unter diesen Aspekten beleuchten.

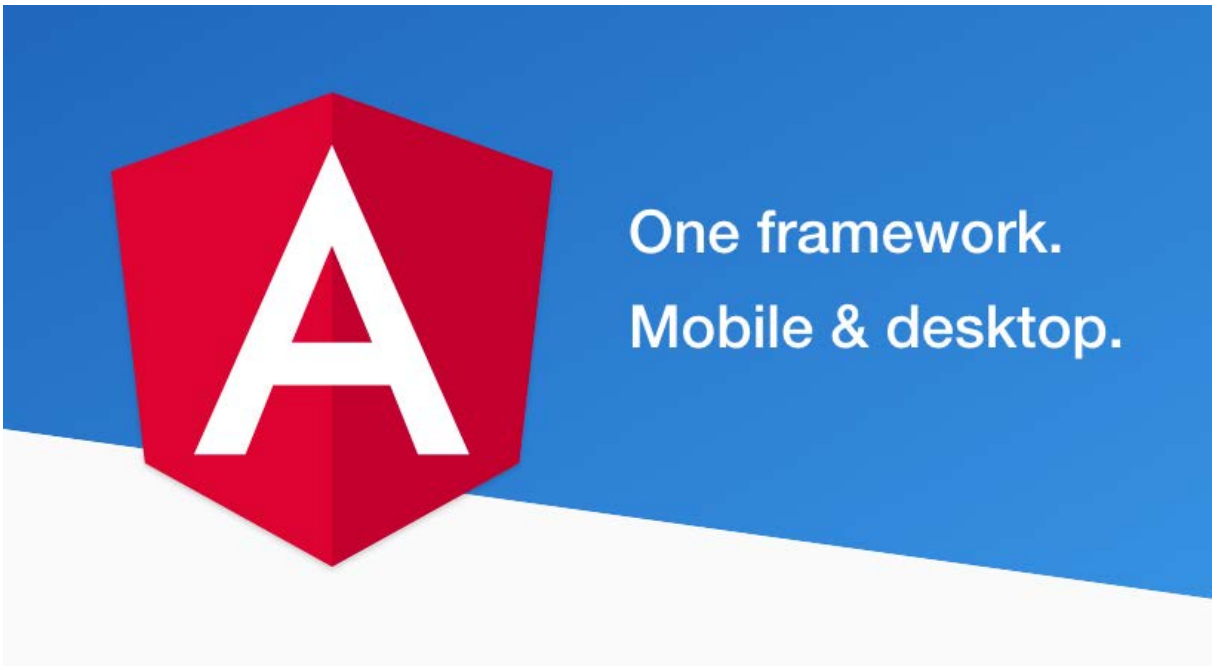

Abbildung 3: Angular-Logo (Quelle: https://angular.io/)

2. Die Angular-Technologie

Durch die Erscheinung von AngularJS (auch Angular.js oder Angular 1.x) in 2010, und insbesondere durch die Erscheinung des Release von Angular 2 in 2016, hat sich die Entwicklung von Client-Side Apps (bzw. Webseiten) stark vereinfacht. Angular ist dabei ein von Google entwickeltes Open-Source Front-End-

Webapplikationsframework, dass alle Aspekte des Entwickler-Workflows bei der Entwicklung der komplexen Web-Anwendungen adressiert.

Google ist dabei nicht nur Haupttreiber und Hauptentwickler von Angular, sondern wendet das Framework auch für seine eigenen Produkte an. Zudem wird Angular auch durch eine Community aus Einzelpersonen und Unternehmen unterstützt. Mittlerweile ist Angular mit Abstand das populärste Front-End Framework unter Entwicklern. Technisch gesehen basiert es seit der Version Angular 2 auf Microsofts TypeScript, einer Erweiterung von Javascript. Aus Benutzerperspektive bringen durch Angular geschriebene Apps und Webseiten insbesondere große Verbesserungen bezüglich Geschwindigkeit und Benutzerfreundlichkeit.

Ein wichtiger Aspekt von Angular sind Single-Page-Applications, die dafür sorgen, dass auf leistungsfähigen Browsern und mobilen Endgeräten anspruchsvolle Benutzererlebnisse flüssig wiedergegeben werden können. Durch moderne Browser wird es möglich, dass sich die Logik einer Webseite auf der Seite des Endgeräts befindet. Dadurch können Entwickler nützliche Interaktionen durch die Anwendung von Formularen, Navigationen und Slidern anbieten. Weiterhin wird die Benutzerfreundlichkeit durch sogenannte Progressive Web Applications (PWAs) unterstützt. Dadurch ist es möglich, dass viele Features, die normalerweise nur native Anwendungen unterstützen (im Sinne von lokal installierte Programme), genutzt werden können. Beispiele hierfür sind: Installierbarkeit, Offlinefähigkeit oder Push-Benachrichtigungen.

Eine gesamte Angular-Anwendung besteht aus einer Hierarchie von Komponenten und Modulen, die je nach Bedarf ausgetauscht und kombiniert werden können. Der Vorteil aus Entwicklerperspektive ist, dass man sich nur auf einen bestimmten Teil der Anwendung konzentrieren kann, ohne die Arbeit der anderen Entwickler zu behindern. Dementsprechend kann das Team unabhängiger arbeiten, die ausgelagerten Module können später einfacher integriert werden und Projekt-Manager können Sprints einfacher planen.

3. Vor- und Nachteile

In Folgendem erklären wir die Vorteile von Angular, sowohl für aus Entwickler-, als auch aus der Endbenutzerperspektive.

Stärken und Chancen

- Skalierbarkeit: Die neue Version von Angular benutzt intensiv die Funktionen der neusten JavaScript-Version (ECMAScript 6). Dadurch wird die objektorientierte Programmierung (OOP) und Aufteilung einer Anwendung in verschiedene wiederverwendbare Module für die Entwickler stark vereinfacht.
- Geschwindigkeit: Durch die Angular Single-Page-Applications-Framework sind Webseiten schneller als traditionelle auf dem Server verarbeitete Webseiten. Der Unterschied zu einer nativen Smartphone-Anwendung ist kaum noch bemerkbar.
- Im Hinblick auf agile Softwareentwicklung können Sprints in Angular leicht geplant werden, da sich Entwickler aufgrund der Hierarchie von Komponenten und Modulen auf einen bestimmten Teil der Software konzentrieren können, ohne die Arbeit der anderen Entwickler zu behindern.
- Aus Lizenzierungsaspekten haben Entwickler viel Freiheit mit dem Code, da Angular eine MIT-Lizenz verwendet, die die Wiederverwendung in proprietärer Software erlaubt und GPL-kompatibel (General Public License) ist.
- Angular besitzt integrierte Testbibliotheken, mit denen sich Benutzerverhalten in automatisierten Aufgaben (z. B. durch Klicken auf Schaltflächen) nachahmen lässt.

Risiken und Herausforderungen

- Große Angular-Projekte sollten gut geplant werden, da sich ohne eine entsprechende Code-Organisierung im Verlauf des Projekts viele Herausforderungen ergeben.

- Debugging ist, abhängig von der Größe der Software, ein durchaus komplexes Unterfangen. Es gibt einen entsprechenden Tool-Support, der aber eine gewisse Code-Organisierung voraussetzt.
- Wie bei jedem Framework ist die Lernkurve am Anfang steil, da gewisse Programmierstile und -paradigmen erst erlernt werden müssen.

4. Literatur

Homepages

1. Angular-Homepage: https://angular.io
    1.1 Quickstart: https://angular.io/guide/quickstart
    1.2 Tutorial: https://angular.io/tutorial
2. Angular-Blog: https://blog.angular.io
3. Angular University: https://blog.angular-university.io/
4. Angular Material: https://material.angular.io

Bücher

1. Matt Frisbie. *Angular 2 Cookbook*. 2017, Packt Publishing, ISBN 978-1785881923.
2. Chandermani Arora and Kevin Hennessy. (2004). *Angular 2 By Example*. 2016, Packt Publishing, ISBN: 978-1785887192.
3. Nathan Murray, Felipe Coury, Ari Lerner, and Carlos Taborda. *ng-book: The Complete Book on Angular 4*. 2017, CreateSpace Independent Publishing Platform. ISBN: 978-1546376231

Event Sourcing und CQRS: Wie sich mit Software experimentieren lässt

1. Ausgangslage

Die Welt in der wir leben, und damit auch die Art wie wir Software schreiben, wird immer dynamischer. Wir agieren in immer ungewisseren Kontexten, die Welt ändert sich deutlich schneller und die Systeme werden immer komplexer – so komplex, dass ein einzelner Mensch oder eine kleine Gruppe nicht mehr in der Lage sind, das gesamte System in Gänze zu verstehen.

Dadurch sind wir heutzutage immer mehr darauf angewiesen, unsere Softwaresysteme skalierbar zu schreiben, sie schnell austauschbar zu halten, robust gegenüber Fehlern zu gestalten und mehrere Versionen simultan zu unterstützen. Die große Frage, die sich uns allen dabei stellt, ist: Wie können wir dies bewerkstelligen? Zwei mögliche unterstützende Architekturmuster, um der Herausforderung Herr zu werden, sind Event Sourcing und CQRS.

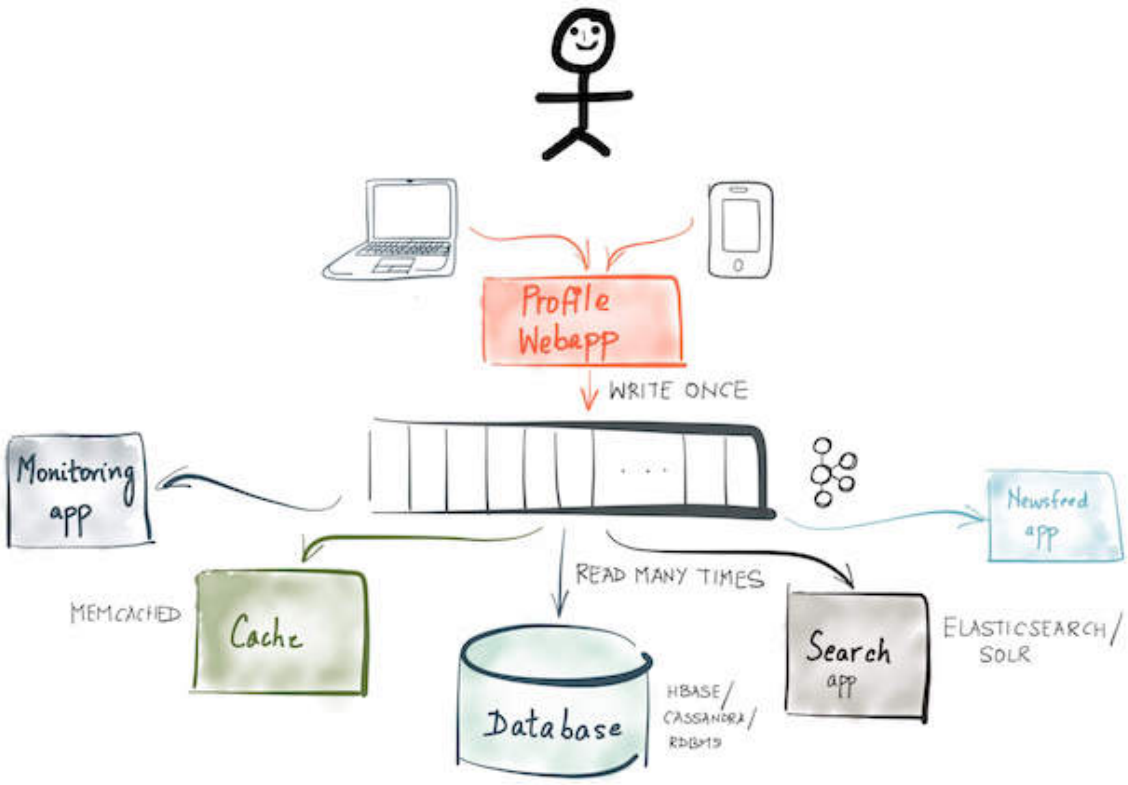

Abbildung 4: EventSourcing basierte Architektur (Quelle: https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection/)

2. Die Event Sourcing- & CQRS-Muster

Event Sourcing ist ein Verfahren, bei dem alle Veränderungen des Zustands einer Softwareanwendung als Ereignisse (Events) abgebildet und aufgezeichnet werden. Diese Aufzeichnung erhält dabei die Reihenfolge der Ereignisse über die komplette Lebenszeit der Anwendung – es entsteht ein Protokoll über alle Veränderungen, die sich in den Daten widerspiegeln.

Daraus ergeben sich einige Möglichkeiten:

1) Man kann den momentanen Anwendungszustand komplett verwerfen und mit Hilfe des Ereignisprotokolls und des initialen Zustands komplett wiederherstellen, in dem die Ereignisse in der gleichen Reihenfolge auf den initialen Zustand angewandt werden.
2) Wir können den Anwendungsstatus zu jedem vergangenen Zeitpunkt bestimmen (sog. Temporal Query), in dem wir auf dem initialen Anwendungszustand die Events nur bis zu einem bestimmten Zeitpunkt anwenden.
3) Falls im Nachgang festgestellt wird, dass ein Ereignis zu einem fehlerhaften Zustand führt (im Sinne, dass die Auswirkungen unerwünscht sind), kann dieses Ereignis über ein sogenanntes retroaktives Ereignis in seinen Auswirkungen neutralisiert werden.

Gängige Beispielanwendungen die Event Sourcing nutzen, sind Versionskontrollsysteme (z. B. Subversion) und Auditprotokolle im Finanzsektor. Das wohl bekannteste Beispiel für Event Sourcing bleibt das Sparbuch, indem nur die Transaktionen gespeichert werden.

CQRS (Abkürzung für Command-Query-Responsibility-Segregation – übersetzbar mit Kommando-Abfrage-Zuständigkeits-Trennung) ist eine Weiterentwicklung des CQS-Prinzips. Das CQS-Prinzip besagt, dass eine Methode entweder als Abfrage (die lediglich Daten zurückliefert und keine Nebeneffekte hat) oder als Kommando (dass Nebeneffekte auf den Anwendungszustand hat, aber keine Daten liefert) implementiert werden soll. CQRS erweitert dieses Prinzip auf die Gesamtarchitektur von Systemen (anstatt auf Methoden).

Die Kombination aus Event Sourcing und CQRS versetzen uns in die Lage, unsere Anwendungsteile unabhängig voneinander zu entwickeln, indem wir parallele Modelle durch Synchronisierung und Dateneigentümerschaft ermöglichen. Zudem erhöhen diese Muster unsere Widerstandsfähigkeit gegenüber Fehlern, da wir Änderungen nachvollziehen können und bei einem Fehler in einer Komponente die anderen nicht automatisch mitabstürzen. Zudem können so innerhalb einer Microservice-Architektur mehrere Varianten parallel betrieben werden. Dadurch kann auch im Produktivsystem experimentiert und bei Fehlern blitzschnell auf eine Rückfallebene zurückgegangen werden. Des Weiteren erhalten wir mit dem Event Store für Event Sourcing die Möglichkeit, Nutzerinteraktionen zu speichern und später auszuwerten.

3. Vor- und Nachteile

Im Folgenden sollen die Vor- und Nachteile der beiden Technologien und ihrer Kombination diskutiert werden.

Stärken und Chancen

- Wie bereits beschrieben, erlaubt die Kombination von Event Sourcing und CQRS es, einfacher mit Software zu experimentieren.
- Beide Muster haben prinzipiell eine hohe Skalierbarkeit (gesehen auf die Größe der Anwendung/der Architektur). Darüber hinaus sorgt CQRS dafür, dass Lese- und Schreibzugriffe auf Datensätze unabhängig voneinander skalierbar sind.
- EventSourcing erhöht die Transparenz der Anwendung, da jede Veränderung aufgezeichnet wird. Dadurch lassen sich beim Experimentieren Fehler deterministisch(er) finden.
- Des Weiteren ist CQRS nicht nur ein natürliches Entwurfsmuster für SOA (Service Oriented Architecture) und Cloud-Anwendungen, sondern erlaubt es auch, dass der Entwicklungsprozess zwischen mehreren Entwicklerteams aufgeteilt werden kann.

Risiken und Herausforderungen

Für Event Sourcing ergeben sich folgende Nachteile:

- Es entsteht ein hoher Entwicklungsaufwand. Insbesondere, wenn im Nachgang Veränderungen gemacht werden sollen, die Einfluss auf das Ereignisprotokoll haben.
- Das Paradigma, dass jede Anwendungsänderung einem Event entspricht (und entsprechend implementiert werden muss), ist eine große Umstellung auf Programmiererseite.
- Bei entsprechender Anwendungsgröße bzw. Interaktivität erzeugt Event Sourcing einen enormen Speicheraufwand.

Bei CQRS existieren folgende Nachteile:

- Wie bei EventSourcing ergibt sich ein höherer Entwicklungsaufwand.
- Der Fokus auf das Verhalten der Anwendung führt gerade bei einfachen Systemen zu größeren Aufwänden bei Analyse und Design, als eine reine Implementierung von Datenstrukturen.
- Es wird eine Infrastruktur gebraucht, die zu CQRS passt.
- Transaktionen sind eine Herausforderung, da nur „eventual consistency“ gewährleistet wird

4. Weiterführende Literatur

Online Ressourcen

1. https://www.heise.de/developer/artikel/CQRS-neues-Architekturprinzip-zur-Trennung-von-Befehlen-und-Abfragen-1797489.html?seite=all
2. http://codebetter.com/gregyoung/2010/02/13/cqrs-and-event-sourcing/
3. https://martinfowler.com/eaaDev/EventSourcing.html
4. https://martinfowler.com/bliki/CQRS.html
5. https://github.com/heynickc/awesome-ddd

Bücher

1. Dominic Betts, Julian Dominguez, Grigori Melnik, Fernando Simonazzi, Mani Subramanian. *Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure*. 2013, Microsoft patterns & practices, ISBN: 978-1621140160.
2. Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. 2003, Pearson Professional, ISBN: 978-0321125217.

Big Data Analytics: Innovation aus dem Gold des 21. Jahrhunderts

1. Ausgangslage

Big Data Analytics ist ein Sammelbegriff für die Untersuchung und Analyse großer Datenmengen unterschiedlichster Arten (Big Data). Es ist das angestrebte Ziel, in den Daten versteckte Muster, unbekannte Korrelationen und andere nützliche Informationen zu entdecken, um diese in Wettbewerbsvorteile gegenüber Konkurrenten umzuwandeln bzw. andere geschäftliche Vorteile, zum Beispiel effektiveres Marketing oder höhere Umsätze, daraus zu generieren.

Das Hauptziel von Big Data Analytics besteht somit darin, Unternehmen zu besseren geschäftlichen Entscheidungen zu verhelfen. Dazu werden Anwender in die Lage versetzt, riesige Mengen Transaktionsdaten sowie andere Datenquellen auszuwerten. Diese Daten werden von konventionellen Programmen für Business Intelligence in der Regel nicht berücksichtigt. Beispiele für diese Quellen sind Protokolle von Webservern und Internet-„Clickstreams“, Berichte über Aktivität in sozialen Medien (inklusive Bilder und Videos), Einzelverbindungsnachweise für Mobiltelefone oder von Sensoren erfasste Informationen. Abhängig vom Kontext wird Big Data ausschließlich mit unstrukturierten Daten dieser Arten in Verbindung gebracht, generell sind die Methoden von Big Data Analytics aber auch für herkömmliche Datenquellen nutzbar.

Da das Feld Big Data Analytics sehr breit gefächert ist und momentan stark beforscht wird, stellt dieser Artikel keine explizite Methode oder Technologie vor, sondern erklärt die grundlegende Methodik.

Abbildung 5: Die verschiedenen Quellen von Big Data (Quelle: https://www.ness.com/breaking-the-silo-in-big-data-analytics-2/)

2. Die Big Data Analytics-Methodik

Big Data Analytics lässt sich in einzelne Teilgebiete gliedern, die jedoch eng miteinander verknüpft sind. Die wichtigsten davon sind:

- Datenbeschaffung aus verschiedenen Quellen (z.B. mithilfe von Suchabfragen)
- Verknüpfung und Auswertung der gewonnenen Daten

- Aufbereitung und Präsentation der Ergebnisse

Die erste Herausforderung bei Big Data Analytics ist die Aufgabe, riesige Datenmengen mit unterschiedlichen Formaten und Inhalten aus verschiedenen Quellen zu erfassen und für die weitere Bearbeitung aufzuarbeiten. Dabei besteht oft die Schwierigkeit, dass die Datenmengen nicht nur groß (im Sinne von Anzahl und/oder Dateigröße) sind, sondern in der Regel auch nicht strukturiert sind und in verschiedenen Formaten vorliegen. Solche unstrukturierten Informationen sind mit herkömmlicher Datenbanksoftware kaum zu erfassen und so kommen bei Big Data Analytics aufwändige Prozesse zur Extraktion, Erfassung und Transaktion der Daten zum Einsatz. Diese Verfahren arbeiten oft in „Echtzeit" – können also die Daten verarbeiten, während sie gesammelt werden.

Bei der anschließenden Datenanalyse kommen Methoden aus dem Bereich des maschinellen Lernens zum Einsatz. Mittels spezieller Software können versteckte Muster und Zusammenhänge identifiziert werden. Die dabei gewonnenen Erkenntnisse bilden eine wichtige Grundlage für ökonomisch sinnvolle Entscheidungen auf Managementebene. Des Weiteren lassen sich mittels „Predictive Analytics" auf Basis historischer Daten außerdem zukünftige Ereignisse vorhersagen. Dabei wird zunächst aus den historischen Daten ein Modell erstellt, welches in der Vergangenheit aufgetretene Trends und Zusammenhänge erfasst, und diese anschließend in die Zukunft projiziert. Als Beispiel für die eingesetzten Techniken seien die Ausreißer-Analyse, die Clusteranalyse und die Regressionsanalyse genannt.

Die finale Aufbereitung und Visualisierung der gewonnenen Ergebnisse kann in der Regel mit Standardsoftware gelöst werden. Sämtliche Software, die in einem der beschriebenen Schritte angewandt wird, kann unter dem Begriff Big Data Analytics mit einbezogen werden.

Technisch gesehen entstehen durch die Größe der zu verarbeitenden Datenmenge und die häufig erforderliche Echtzeitfähigkeit Anforderungen, die den Einsatz spezieller Software erfordern. Die unstrukturierten Datenquellen lassen sich nicht immer in traditionellen (zentralen) Datenbanken verwenden, und auch mit den Anforderungen bei der Verarbeitung sind diese bisweilen überfordert. Als Folge davon ist eine neue Klasse von Big-Data-Technologien entstanden, die intensiv eingesetzt wird. Zu ihnen zählen NoSQL-Datenbanken, Apache Hadoop (ein Framework für skalierbare, verteilt arbeitende Software), MapReduce (ein von Google entwickeltes Programmiermodell für nebenläufige Berechnungen über mehrere Petabyte große Datenmengen) und Apache Spark (ein Framework für Cluster Computing). Diese Technologien bilden den Kern eines quelloffenen Software-Frameworks, das die Verarbeitung von großen Datenbeständen über mehrere Server hinweg (sogenannte Cluster-Systeme) unterstützt.

3. Vor- und Nachteile

Im Folgenden sollen einige der Vor- und Nachteile von Big Data Analytics und auch Big Data im Allgemeinen vorgestellt werden.

Stärken und Chancen

- Big Data liefert einen handfesten Wettbewerbsvorteil bei Entscheidungsprozessen. Die Auswertung von IT-gestützten Daten hilft im Tagesgeschäft, schnellere und vor allem bessere Entscheidungen zu treffen.
- Big Data Analytics kann als Prognose- und Frühwarnsystemen genutzt werden, dass potenziell Konjunkturzyklen und Volatilitäten im Markt vorhersehen kann.
- Ein absolutes Muss stellt Big Data für das Marketing dar, denn hier sind detaillierte Informationen der Schlüssel zu den Wünschen der Kunden.
- Richtig eingesetzt kann Big Data eine Komponente der Modernisierung von Management-Ansätzen und Geschäftsmodellen sein, die Grundlage für die Sicherung unseres wirtschaftlichen Status Quo ist.

Risiken und Herausforderungen

- Jedes Big Data Projekt muss und sollte juristisch genauestens unter die Lupe genommen werden, da Erhebung, Speicherung und Weiterverarbeitung von Daten in Deutschland strengen Regelungen unterliegt.
- Die meisten Unternehmen haben keine interne Big-Data-Verwaltungs- und Analytik-Kompetenz und es entstehen hohe Kosten für die Beschäftigung von erfahrenen Analytik-Profis.
- Große Datenmengen mit betriebsinternen Zusammenhängen sind ein lohnendes Ziel für Betriebsspionage durch Hacker. Nicht jedes Unternehmen hat die Ressourcen, die Daten entsprechend zu schützen.
- Big Data Analytics funktioniert nur dann, wenn Datensätze so vollständig und korrekt wie möglich sind.
- Es ist zu betonen, dass die durch Big Data Analytics getroffene Prognosen Unsicherheiten unterliegen, da von der Vergangenheit auf die Zukunft geschlossen wird.

4. Weiterführende Literatur

Online Ressourcen

1. https://www2.deloitte.com/content/dam/Deloitte/de/Documents/Mittelstand/studie-data-analytics-im-mittelstand-deloitte-juni-2014.pdf
2. https://www-935.ibm.com/services/de/gbs/thoughtleadership/GBE03519-DEDE-00.pdf
3. https://www.intel.de/content/dam/www/public/emea/de/de/pdf/unstructured-data-analytics-paper.pdf

Bücher

1. Wolfgang Becker, Patrick Ulrich, Tim Botzkowski. *Data Analytics im Mittelstand*. Gabler Verlag, 2016, ISBN 978-3-658-06563-8
2. Michael Minelli, Michele Chambers, Ambiga Dhiraj. *Big Data, Big Analytics*. John Wiley & Sons, 2013, ISBN 978-1-118-14760-3

Resilienz: Robustheit gegenüber Systemfehlern

1. Ausgangslage

Der Begriff Resilienz bezeichnet im Bereich der Rechnernetze die Fähigkeit, ein akzeptables Servicelevel bereitzustellen (und aufrecht zu erhalten), während im Netzwerk Fehler auftreten. Diese Fehler (allgemeiner gesprochen „Risiken und Herausforderungen“) können unterschiedlicher Natur sein: Zum Beispiel Fehlkonfigurationen im Netzwerk, großflächige (Natur-) Katastrophen oder gezielte Angriffe. Dabei ist es im Generellen für das Konzept der Resilienz nicht im Detail wichtig, was genau ein Service (bzw. Servicelevel) ist. Gängige Beispielszenarien für eben solche Services sind Kommunikationsdienste (Videokonferenzen, Instant Messaging), Kollaborationsdienste (z. B. Google Docs, Microsoft Sharepoint) und Dienste zum verteilten Speichern von Daten (z. B. Dropbox, Google Drive). Konsequenterweise ist Resilienz kein isoliertes Themengebiet, sondern berührt eine große Anzahl von IT-Themen und Technologien, unter anderem: Security, Ausfallsicherheit und Redundanz.

Um die Resilienz eines Netzwerkes zu erhöhen, gibt es vielfältige Methoden und Techniken. Diese sind abhängig von den tatsächlichen Risiken und finanziellen Rahmenbedingungen der Risikobereitschaft des Unternehmens (bzw. Netzwerkbetreibers). Im Folgenden soll auf die Resilienz-Test-Methodik der „Simian Army“ eingegangen werden, welche sehr erfolgreich weltweit von Netflix eingesetzt wird.

Abbildung 6: Visualisierung der Simian Army (Quelle: https://devops.com/netflix-the-simian-army-and-the-culture-of-freedom-and-responsibility/)

2. Die Simian Army

Die Idee der Simian Army steht im engen Zusammenhang mit dem Wechsel von Netflix auf die Cloud-Architektur von Amazon (Amazon Web Services – AWS). Im Rahmen dieser Umstellung wurde ein Konzept für die Softwarearchitektur der einzelnen Netflix-Services und ein entsprechendes (möglichst realistisches) Testsystem entwickelt.

Die Softwarearchitektur wird von Netflix selbst auch als „Rambo Architektur“ bezeichnet, da jede Service-Teilkomponente unabhängig von den anderen Komponenten einwandfrei funktionieren muss. Jeder Service ist somit darauf vorbereitet, dass andere Services, zu denen eine Abhängigkeit besteht, ausfallen könnten. Sollte beispielsweise das Bewertungssystem ausfallen, verschlechtert sich zwar die Qualität der Antworten auf Filmanfragen, aber es wird dennoch eine Antwort geben. In dem Fall, dass die Suchfunktion ausfällt, muss das Streaming der Filme trotzdem einwandfrei funktionieren.

Die Netflix Simian Army ist das entsprechende (Open Source) Testkonzept, um die Rambo Architektur auf Herz und Nieren zu testen. Grundidee ist hier, dass einzelne autonome Softwaretestprozesse (genannt Monkeys) versuchen, spezielle Aspekte der Architektur lahm zu legen. Zum Standpunkt der Erstellung des Artikels gibt es eine Vielzahlt an Monkeys mit unterschiedlichen Aufgabengebieten. Im Folgenden werden einige wichtige Monkeys im Detail vorgestellt, um einen besseren Einblick zu bekommen.

Der *Chaos Monkey* hat die Aufgabe, zufällig Instanzen und Services innerhalb der Architektur zu zerstören. Er überprüft somit konkret die Resilienz des Systems gegenüber einzelnen zufälligen Ausfällen. Somit sollten Komponenten der Architektur, die nicht vom Chaos Monkey angegriffen werden, normal funktionieren.

Der *Chaos Gorilla* ist eine Erweiterung des Chaos Monkeys, der nicht nur einzelne Services angreift, sondern ganze Verfügbarkeitszonen in AWS. Diese Zonen bestehen aus Services der gesamten Architektur, die für eine Region (beispielsweise Europa) zuständig sind. Wird die Zone ausgeschaltet, so sollte eine andere Zone die Funktionalität übernehmen.

Der *Latency Monkey* erzeugt künstliche Verzögerungen bei der Kommunikation der einzelnen Komponenten und simuliert somit einen Leistungsabfall des Netzwerks. Die Verzögerung kann hierbei variiert werden. So entspricht beispielsweise ein sehr hoher Wert einem kompletten Netzwerkausfall für eine Komponente.

Der *Security Monkey* findet bekannte Sicherheitslücken und Schwachstellen in falsch konfigurierten Teilen der Architektur. Sollte so eine Schwachstelle gefunden sein, wird der entsprechende Architekturteil angeschaltet. Dazu stellt er sicher, dass Sicherheitszertifikate gültig sind.

Des Weiteren gibt es Monkeys, die eher eine überwachende Funktion haben. So schaut der *Janitor Monkey,* ob Ressourcen unnötig belegt werden und gibt diese frei. Der *Doctor Monkey* überprüft einzelne Komponenten auf ihren internen Gesundheitszustand (im Sinne von Rechner- und/oder Speicherauslastung).

Es bleibt zu erwähnen, dass die Simian Army für das Testen einer Cloud-Architektur entwickelt wurde und in diesem Kontext momentan eingesetzt wird. Nichtdestotrotz können viele der grundsätzlichen Ideen zum Testen von Resilienz im allgemeinen Fall genutzt werden.

3. Vor- und Nachteile

Im Folgenden sollen einige der Vor- und Nachteile von Resilienz im allgemeinen und der Nutzung der Simian Army im Speziellen diskutiert werden.

Stärken und Chancen

- Resilienz geht in vielerlei Hinsicht über das Thema IT-Sicherheit hinaus und bietet eine ganzheitliche Betrachtungsweise auf Verfügbarkeit.
- Insbesondere in Bereichen, in denen Benutzer an hohe Verfügbarkeit von vielen Services gewöhnt sind (wie z. B. Video-Streaming) ist es sehr sinnvoll, Resilienz als einen Kernaspekt des Unternehmens zu adressieren.
- Die Idee einzelne Komponenten einer (resilienten) Architektur durch verschiedene Monkeys zu testen, ist vielfältig einsetzbar.
- Die Simian Army ist ein Baukastensystem. Man kann nur die Monkeys einsetzen, die tatsächlich gebraucht werden, oder aus bestehenden Monkeys einen Neuen erzeugen. Der Individualisierung sind prinzipiell keine Grenzen gesetzt.
- Das Konzept der Simian Army skaliert mit der Größe der Netzwerk- und Service-Architektur, da Monkeys unabhängig voneinander agieren.

Risiken und Herausforderungen

- Eine hohe Resilienz ist in der Regel an höhere Kosten (für Hard- und Software) geknüpft. Es muss somit ein Trade-Off gefunden werden, der im konkreten Fall den höchsten Nutzen verspricht.
- Erst ab einer gewissen Netzwerkgröße (die häufig mit einer entsprechenden Unternehmensgröße einhergeht) ist es sinnvoll, sich mit dem Thema Resilienz ganzheitlich auseinander zu setzen. Für kleine Szenarien sind häufig punktuelle Ansätze (Datenbackups, Cloud-basierte Frontends etc.) ausreichend.
- Die Grundidee der Simian Army ist am besten anwendbar, wenn die eigene Architektur auf Microservices und cloudbasierten Komponenten beruht.
- Der Einsatz einzelner Monkeys sollte eng verknüpft sein mit der unternehmenseigenen Resilienzstrategie. Es sollte vorab gut überlegt werden, gegen welche Fehler man sich absichern möchte und wie man diese in spezielle Monkeys implementiert.
- Netflix setzt die Simian Army im Produktivsystem ein. Man sollte in seiner eigenen Unternehmensarchitektur somit vorab überprüfen, ob ein solcher Einsatz auch möglich ist.

4. Weiterführende Literatur

Online Ressourcen

4. https://www.crisp-research.com/netflix-der-chaos-monkey-und-die-simian-army-das-vorbild-fur-eine-gute-cloud-systemarchitektur/
5. https://medium.com/netflix-techblog/tagged/simian-army
6. https://de.slideshare.net/betclicTech/betclic-mini-trainingnetflixsimianarmy
7. http://www.exforsys.com/tutorials/testing-types/monkey-testing.html

Bücher

1. Ilya Gertsbakh, Yoseph Shpungin. *Network Reliability and Resilience*. Springer, 2011, ISBN 978-3-642-22374-7

Impressum

Herausgeber ist das SI-Lab – Software Innovation Lab der Universität Paderborn.

Inhaltliche Verantwortlichkeit: Dr. Thim Frederik Strothmann

Geschäftsführung: Dr. Stefan Sauer und Dr. Gunnar Schomaker

Anschrift:
Universität Paderborn
Fakultät für Elektrotechnik, Informatik und Mathematik
SI-Lab – Software Innovation Lab
Zukunftsmeile 1
33102 Paderborn