Daten sind Macht

Event-Sourcing: Die Wichtigkeit der Datenanalyse

Tomasz Bieruta

Unternehmen müssen heute sehr schnell agieren und das Business und die IT befinden sich in einem ständigen Wandel. Die Time-to-Market, das heißt die Zeit von der Produktentwicklung bis zur Marktplatzierung, kann über „be or not to be" entscheiden und den Ausschlag geben, ob ein Produkt überhaupt lanciert wird. Unkontrollierte Systemausfälle führen sofort zu einem Imageverlust und langsame Antwortzeiten seitens des Unternehmens zu sinkender Kundenzufriedenheit und im schlimmsten Fall zu einem Kundenschwund. Für Unternehmen ist es daher von zentraler Wichtigkeit, dass bestimmte Daten korrekt und schnell ausgewertet werden, um auf die Kundenbedürfnisse besser und ohne Zeitverlust reagieren zu können. Mit statistischen Verfahren und/oder Machine-Learning-Algorithmen können eventuell weitere Erkenntnisse gewonnen werden und sich so neue Wachstumsmöglichkeiten eröffnen. Nun stellt sich die Frage: Können solche Systeme mit noch beherrschbarer Komplexität gebaut werden?

Die Rolle der Softwarearchitektur

Die Zerlegung eines Problems in kleinere Teile ist die Domäne der Softwarearchitekten, wie die nachfolgende kurze Definition der Softwarearchitektur von Helmut Balzert bestens beschreibt: *„eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Beziehungen"* [Bal08]. Was hierbei eine Systemkomponente ist, dazu gibt es viele Beschreibungen. Andreas Andersen gibt in seinem Buch [And03] eine gute Zusammenfassung. Komponenten der Enterprise-Architektur verbergen meist größere Applikationen (s. Abb. 1). Wie die russische Matrjoschka [Wikig] können diese Applikationen in weitere Sub-Komponenten aufgeteilt werden.

Monolithische versus Microservices-Strukturen

Monolithische Architekturen haben ihrerseits durchaus gewisse Vorteile: Applikationen können schneller entwickelt werden und deren Betrieb ist einfacher. Aber Business kann sich ändern und plötzlich müssen neue Anforderungen umgesetzt werden. Mit diesen neuen Anforderungen und daraus entstehenden Änderungen an den Applikationen werden oft mehr und mehr Abhängigkeiten eingebaut und die anfangs gut durchdachte, modulare Architektur beginnt auseinander zu driften. Als weiterer Erschwerungsfaktor kommt hinzu, dass die ursprünglichen Entwickler die Arbeitsstelle wechseln und dem kommenden Nachwuchs die notwendigen Kenntnisse bezüglich dieser Architekturstruktur fehlen. Die Wartungskosten steigen und die langen Release-Zyklen (und damit

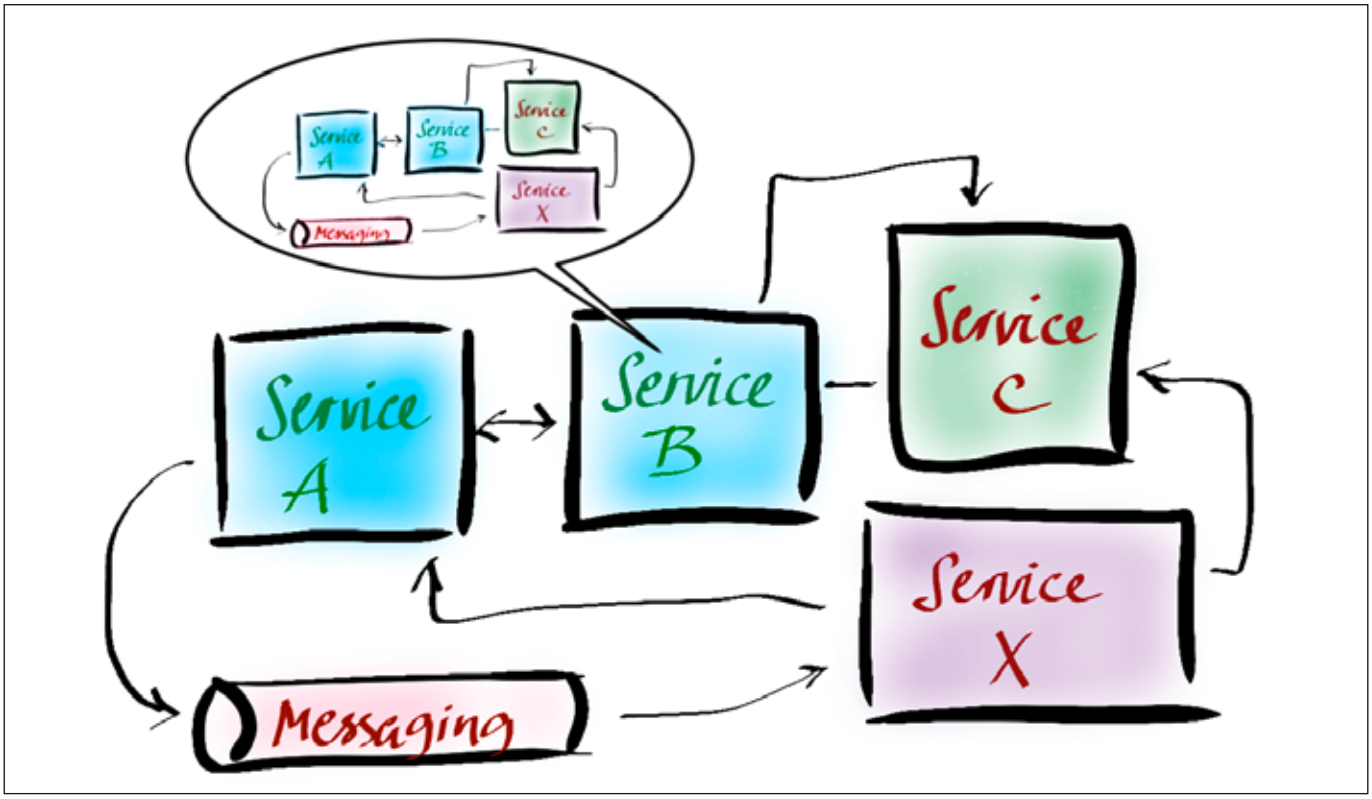

Abb. 1: Komponenten-Teilung (Quelle: adesso Schweiz AG)

Tomasz Bieruta ist Java-Teamleiter und Softwarearchitekt bei der adesso Schweiz AG. Er ist zertifiziert in TOGAF und CPSA-A (iSAQB) und hat jahrelange Erfahrung als Enterprise- und Softwarearchitekt. Seine Interessenbereiche sind vor allem: distributed event-based systems, data integration & data analytics.
E-Mail: tomasz.bieruta@adesso.ch

langsame Reaktion auf Marktveränderungen) führen zu Spannungen zwischen der Fach- und IT-Abteilung.

In der letzten Zeit wurden Microservices [Wikih] und Self-contained Systems [Wikii] immer populärer. Man hat also die Enterprise-Architektur quasi eine Ebene tiefer (auf die Applikationsebene) angewendet (vgl. Abb. 1). Während bei der Enterprise-Architektur jede Komponente/Applikation isoliert und autonom mit eigenen Daten und meist einer Datenbank entwickelt wurde, wird nun die Applikation selbst in mehrere autonome Microservices mit eigenen Daten und Datenbanken zerlegt.

Über die Vor- und Nachteile der Microservice-Architektur wurde bereits viel diskutiert. Eberhard Wolff hat eine gute Zusammenfassung [Wol16] gegeben. Eins muss jedoch deutlich gesagt werden, aus einem Monolith wird ein verteiltes System gebaut und man wird automatisch mit den gleichen Problemen konfrontiert, die die Enterprise-Architekten beschäftigen: Security, Kommunikation (Schnittstellen), Abhängigkeiten zwischen den Komponenten, Datenkonsistenz, Datenmigration, Datenintegration usw. Spätestens bei der Datenintegration muss man sich fragen, wie die Daten in ein Datawarehouse oder andere Reporting Datenbanken überführt werden. Man will ja schließlich auch Analysen und Auswertungen machen können oder auch Daten an die Kunden liefern.

Wie in Abbildung 2 dargestellt, werden die Daten aus verschiedenen Quellen ausgelesen und in die Zielsysteme repliziert. Die Integration kann mit ETL-Tools im Batch-Modus oder mit Change Data Capture [Wikia] in Echtzeit erfolgen. Diese Art der Datenintegration ist in vielen Unternehmen zu finden. Bei der Microservice-Architektur ist die Integration umfangreicher, da nun die Daten zwischen mehreren Systemen ausgetauscht werden müssen. Aufgrund der starken Kopplung zu den Datenbanken und deren Schemata sind die Microservices nicht mehr so autonom und isoliert. Bei einer Schema-Änderung müssen demnach auch andere Systemkomponenten entsprechend angepasst werden.

Die Services leben also nicht isoliert und müssen auf Änderungen/Ereignisse reagieren können [Mom16]. Um die Kopplung möglichst klein zu halten, setzt man häufig Messaging, zum Beispiel Java Message Service (JMS), ein. Nachdem die Daten in der Datenbank gespeichert wurden, werden entsprechende Nachrichten an die Message Broker gesendet. Natürlich muss garantiert werden, dass bei Fehlern die Daten konsistent bleiben. Typischerweise werden hierfür verteilte Transaktionen benutzt (2-Phase-Commit, [Wikic]). Doch dieses 2-Phase-Commit hat auch Nachteile: Ein Ausfall des Transaktionsmanagers (Transaktionskoordinator) kann zu Deadlocks führen und die Verfügbarkeit des ganzen Systems gefährden.

Dem Kunden den Wunsch von den Augen ablesen

Softwareapplikationen werden grundsätzlich gebaut, um Geschäftsprozesse zu optimieren. In einer Domäne, wie zum Beispiel dem Verkauf, kann der Umsatz dank einer Online-Applikation gesteigert werden. Während des Prozesses werden laufend Daten gesammelt (Bestellungseingang, Lieferung, ...), um auf eventuelle Kundenanfragen reagieren zu können. Mit entsprechenden Reports können die relevanten Prozesse analysiert und weiter optimiert werden.

Ein Wunschszenario jedes Unternehmens wäre es doch, wenn man Kundenwünsche erahnen könnte, bevor sie der Kunde äußert.

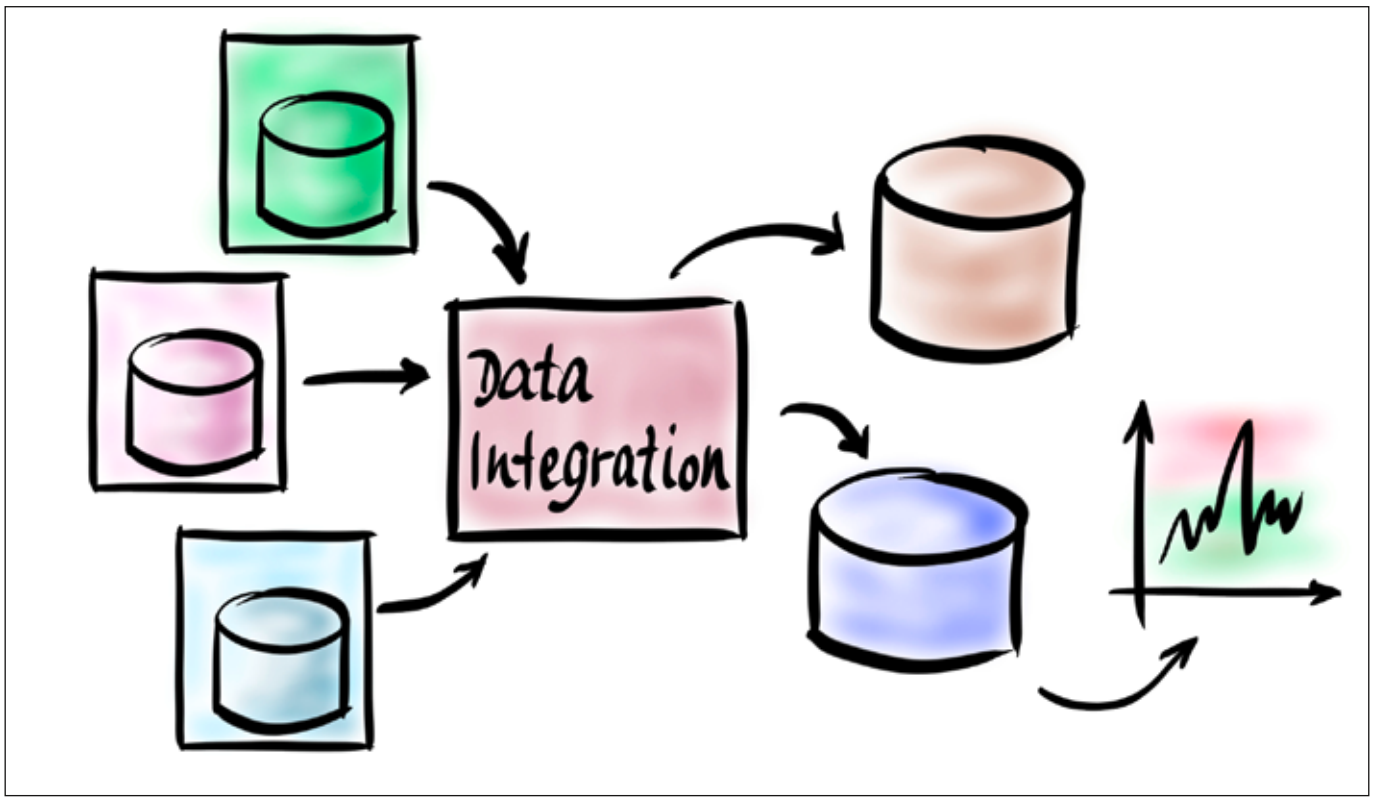

Abb. 2: Datenintegration (Quelle: adesso Schweiz AG)

Haben Sie sich beispielsweise schon einmal gefragt, woher denn Amazon weiß oder zumindest zu wissen glaubt, welche Themen oder Bücher Sie interessieren könnten? Längst tragen statistische Verfahren und Machine-Learning-Algorithmen [Wikif] zum besseren Verständnis der Kundenbedürfnisse bei.

Neben den typischen Domänen wie dem Verkauf gibt es Unternehmen, die Rohdaten gezielt sammeln, diese anreichern, auswerten und die so gewonnene Information weiterverkaufen (z. B. aus Wetterdaten werden Wetterprognosen erstellt oder anhand der besuchten Internetseiten lassen sich Benutzerprofile anfertigen). Diese Datensammlungen können somit wichtige Erkenntnisse liefern und sind eine wertvolle Informationsquelle.

Es ist allerdings erstaunlich, wie viele Unternehmen immer noch bewusst oder unbewusst Datenverluste in Kauf nehmen. Nehmen wir als Beispiel folgende Kundenbestellung: Die Bestellung hat ein Attribut **status** mit dem Anfangswert „pending". Später kommen neue Prozessschritte hinzu und der Status bekommt einen neuen Wert, zum Beispiel „accepted", „canceled" oder „closed".

Nun kommt in unserem Beispiel ein neuer Prozessverantwortlicher ins Unternehmen und möchte wissen, wie viel Zeit im Durchschnitt die einzelnen Prozessschritte in Anspruch nehmen. Eine solche Auswertung ist hier aber leider nicht möglich, da der Prozessstatus bei jeder Aktualisierung überschrieben wurde (Tabellen-Update) und jeweils nur der aktuelle Status der Bestellung abgefragt werden kann. Natürlich wird in diesem Unternehmen entsprechend gehandelt und ab sofort werden alle Daten gesammelt. Der entscheidende Punkt ist aber: Wieso nicht von Anfang an die Änderungen an den Rohdaten unterbinden?

Das Potenzial von Event-Sourcing

Eine interessante Möglichkeit der Datenhaltung bietet Event-Sourcing [Wikid] an. Die Methode ist schnell erklärt: Der Zustand der Applikation wird als Folge von serialisierten Events persistiert. Es wird quasi ein Logbuch geführt, was im System passiert ist. Am Beispiel unserer Bestellung würden Statusänderungen als Nachrichten (OrderStatusChanged mit Attributen, wie: OrderId, OrderStatus, Timestamp) sequenziell gespeichert werden. Der Status der Bestellung kann somit zu jedem Zeitpunkt aus den Events hergestellt werden.

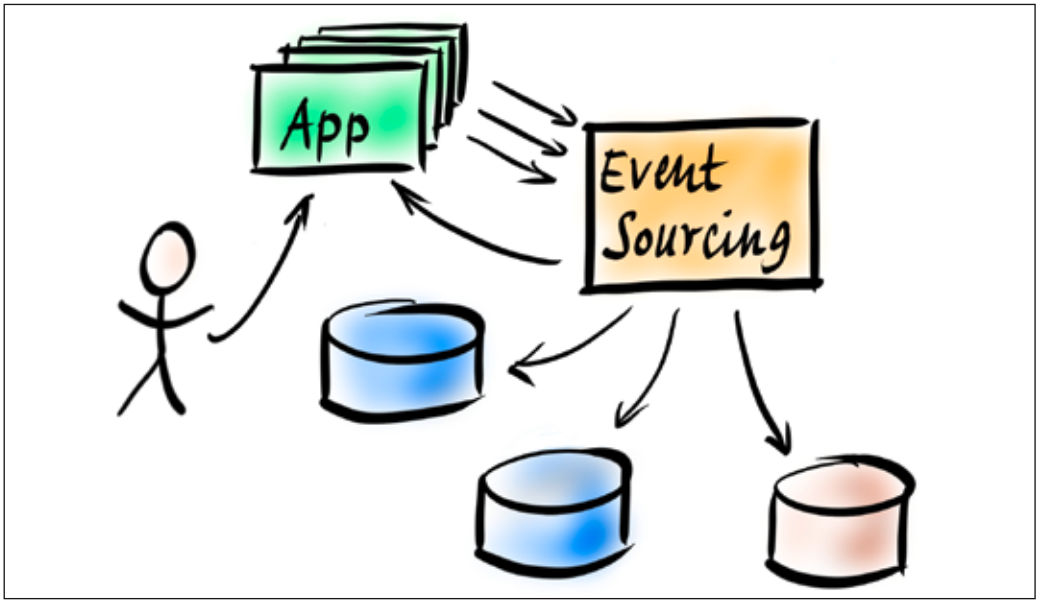

Abb. 3: Event-Sourcing (Quelle: adesso Schweiz AG)

Die serialisierten Events sind jedoch nicht für komplexe SQL-Abfragen geeignet. Zu diesem Zweck werden die Events ausgelesen und weiterverarbeitet. Aus den Events können je nach Bedarf verschiedene Views generiert werden (s. Abb. 3). Am Beispiel unserer Bestellung könnte ein Microservice die Statistiken für die Events OrderStatusChanged berechnen.

Verglichen mit der Infrastruktur wie in Abbildung 2, wird der Applikationszustand nicht verteilt (in Microservices) gehalten und anschließend zentralisiert, sondern in Form von Nachrichten direkt an ein verteiltes Messaging-System mit permanenter Persistenz gesendet. Die Datenintegration ist explizit und ein wichtiger Bestandteil des Gesamtsystems.

Eventuell ist es Ihnen auch schon einmal passiert, dass Sie beim Helpdesk eines Unternehmens erfahren mussten, dass Sie immer noch unter Ihrer alten Adresse geführt werden. Wenn man dann den Satz hört: „... Moment, ich schaue noch in einem anderen System nach ...", dann wissen Sie auch, warum – die Daten werden an dezentralen Orten gepflegt, was natürlich eine Fehlerquelle darstellt.

Event-Sourcing hat noch eine weitere wichtige Eigenschaft. Dank der sequenziellen Speicherung der Events sind verteilte Transaktionen nicht mehr notwendig, um die Datenkonsistenz zu gewährleisten [Kle12]. Verschiedene Systeme können unabhängig voneinander die Events lesen, eigene Daten aktualisieren und mit der Zeit einen konsistenten Zustand erreichen (Eventual Consistency, [Wikie]).

Auch der IT-Betrieb kann von Event-Sourcing profitieren. Zum Beispiel wenn eine neue Datenbank (oder DB-Version) ausgerollt werden muss. Ein Consumer kann die Datenbank ohne Unterbrechung und vor allem in laufendem Betrieb aktualisieren. Irgendwann werden die Daten up to date sein und die Datenbank kann einfach freigeschaltet werden.

Welche Frameworks empfehlen sich?

Es gibt bereits einige gute Frameworks auf dem Markt (geteventstore [EventSt], Eventuate [Eventuate], AxonFramework [Axon]), die bei der Implementierung der CQRS (Command Query Responsibility Segregation, [Wikib]) und Event-Sourcing-Architektur sehr hilfreich sein können.

Zunehmend setzt sich Kafka [Kafka] als Event-Store durch. Ursprünglich wurde Kafka als sehr effizientes, verteiltes Messaging System bei LinkedIn entwickelt. Die Datenmengen, die Kafka verarbeiten kann, ist schon erstaunlich:

„... At the busiest times of day, we are receiving over 13 million messages per second, or 2.75 gigabytes of data per second. To handle all these messages, LinkedIn runs over 1100 Kafka brokers organized into more than 60 clusters ..." [Pal15]

Kafka speichert die Nachrichten auf der Festplatte für einen konfigurierbaren Zeitraum (oder Datenmenge). Diese Konfiguration kann auch ausgeschaltet werden und damit eine permanente Speicherung der Daten erreichen.

Consumer lesen die Daten aus Topics und committen die gelesene Position (Offset). Dank der Persistenz der Nachrichten ist die Performanz der Event-Producer nicht von der Performanz der Consumer beeinträchtigt. Die Consumer müssen auch die Nachrichten nicht in Echtzeit verarbeiten. Fällt ein Consumer aus, kann er nach dem Restart seine Arbeit ausgehend von der letzten Position fortsetzen.

Neben dem Client-API (Producer und Consumer) bietet Kafka die Programmierschnittstellen Kafka Connect und Kafka Streams für Event-Stream-Verarbeitung. Mit Connect kann Kafka Daten zwischen den Datenbanken replizieren (wie ETL-Tool). Kafka Streams [KafkaStreams] sind sehr gut für Daten-Aggregation und -Transformation geeignet. Dabei werden die Daten aus Kafka gelesen und in Kafka geschrieben.

In vielen Fällen benötigt die Stream-Verarbeitung einen Zustand. Dieser Zustand wird lokal gespeichert (state store). Mit der Einführung der „Interactive Queries“ [Confl] ist es möglich, den aktuellen Zustand der Streams abzufragen. Dies hat einen enormen Vorteil, denn so lässt sich eine Microservice-Architektur deutlich einfacher umsetzen. Ein Microservice kann Nachrichten an Kafka Cluster schicken und mit Kafka Streams kann der aktuelle Applikationszustand lokal reproduziert werden (Materialized Views).

Für komplexere Datenanalysen in Echtzeit kann Apache Spark [Spark] mit integrierter Unterstützung für Machine-Learning-Algorithmen verwendet werden.

Fazit

Mit der Kafka Messaging Plattform kann nicht nur eine event-basierte Architektur umgesetzt werden, sondern es können vor allem die Daten – eines der wichtigsten Güter eines Unternehmens – zentralisiert und als eine globale, zuverlässige Informationsquelle (wie z. B. bei der New York Times, [Svi17]) verwaltet werden. Neue Anforderungen können schneller umgesetzt und sogar im laufenden Betrieb freigeschaltet werden.

In Event-Sourcing wird der aktuelle Applikationszustand aus allen historischen Events abgeleitet. Sollte eine Entity sehr viele Events haben, kann dies zu langsameren Antwortzeiten führen. Oft bieten verschiedene Frameworks Optimierungen - sogenannte Snapshots – an. Dank der Kafka „Interactiven Queries“ ist der aktuelle Zustand in Echtzeit verfügbar.

Eindeutig lassen sich mithilfe von Microservices mit Kafka als Event-Sourcing flexible, skalierbare Architekturen umsetzen.

Literatur und Links

[And03] A. Andersen, Komponentenbasierte Softwareentwicklung, Hanser Fachbuch, 2003

[Axon] http://www.axonframework.org/

[Bal08] H. Balzert u.a., Lehrbuch der Softwaretechnik: Softwaremanagement, Spektrum Akademischer Verlag, 2008

[Confl] http://docs.confluent.io/current/streams/developer-guide.html#streams-developer-guide-interactive-queries

[EventSt] https://geteventstore.com/

[Eventuate] http://eventuate.io/

[Gün17a] M. Günther, Skalierfähige, asynchrone Nachrichtenverarbeitung mit Apache Kafka, in JavaSPEKTRUM, 03/2017

[Gün17b] M. Günther, Streaming-Applikationen mit Kafka Streams, in: JavaSPEKTRUM, 04/2017

[Kafka] https://kafka.apache.org/

[KafkaStreams] https://kafka.apache.org/documentation/streams

[Kle12] M. Kleppmann, Staying in Sync: from Transactions to Streams, InfoQ, 20.5.2016, https://www.infoq.com/presentations/event-streams-kafka

[Mom16] L. Momot, Microservices – Kommunikation, SDX AG, 16.11.2016, https://www.sdx-ag.de/2016/11/microservices-kommunikation/

[Pal15] T. Palino, Running Kafka At Scale, LinkedIn, 20.5.2015, https://engineering.linkedin.com/kafka/running-kafka-scale

[Spark] https://spark.apache.org/

[Svi17] B. Svingen, The Source of Truth: Why The New York Times Stores Every Piece of Content of Ever Published in Kafka, Kafka Summit 2017, https://kafka-summit.org/sessions/source-truth-new-york-times-stores-every-piece-content-ever-published-kafka/

[Wikia] https://en.wikipedia.org/wiki/Change_data_capture

[Wikib] https://de.wikipedia.org/wiki/Command-Query-Responsibility-Segregation

[Wikic] https://de.wikipedia.org/wiki/Commit-Protokoll

[Wikid] https://de.wikipedia.org/wiki/Event_Sourcing

[Wikie] https://en.wikipedia.org/wiki/Eventual_consistency

[Wikif] https://en.wikipedia.org/wiki/Machine_learning

[Wikig] https://de.wikipedia.org/wiki/Matrjoschka

[Wikih] https://de.wikipedia.org/wiki/Microservices

[Wikii] https://de.wikipedia.org/wiki/Self-contained_Systems

[Wol16] W. Wolff, Microservices – eine Bestandsaufnahme, InnoQ, 18.10.2016, https://www.innoq.com/de/articles/2016/10/microservices-eine-bestandsaufnahme/