ENTAGO

Ereignisgetriebene (Event-Driven) Architekturen mit Kafka

Im vorliegenden Artikel werden die Motivation für eine ereignisgetriebene Architektur und verschiedene Vorgehensmodelle vorgestellt. Anschliessend wird Kafka als Messaging-System eingeführt und erläutert, wie sich eine der ereignisgetriebenen Architekturen mit Kafka umsetzen lässt. Abschliessend wird auf einige bekannte Herausforderungen eingegangen und ein Fazit gezogen.

Bei verteilten System kommt es dazu, dass ein System einem anderen System eine Zustandsänderung mitteilen möchte. Als Beispiel kann man sich ein Stammdatenmanagement-System vorstellen, welches Kundendaten verwaltet und ein weiteres System, welches von Support-Hotline-Mitarbeitern verwendet wird. Dieses Support-System hat eine Abhängigkeit zum Stammdatenmanagement-System - da die Support-Mitarbeiter auf Kundendaten zugreifen müssen.

Im einfachsten Fall greift das Support-System direkt via synchronem Aufruf auf das Stammdatenmanagement-System zu (Abbildung 1). Das hat allerdings einige Nachteile: Wenn das Stammdaten-System offline oder langsam ist, so wird auch das Support-System unbenutzbar. Auch Latenz im Netzwerk würde die Verwendung erschweren. Wenn sich die Schnittstelle des Stammdaten-Systems ändert, muss sich auch die Implementierung des Aufrufs auf Seite des Support-Services ändern.

Kommen noch weitere «Clients» neben dem Support-Service hinzu, die das Stammdaten-System abfragen wollen, muss dementsprechend skaliert werden.

Einfacher ist es, wenn das Stammdaten-System Zustandsänderungen als Events überträgt, die dann von Systemen, die an diesen Daten interessiert sind, konsumiert werden können. Wenn diese Events von einem hochverfügbaren, externen System übertragen werden, können dadurch die vorher genannten Nachteile und Herausforderungen gelöst werden: Das Support-System macht keine synchronen Aufrufe mehr zum Stammdaten-System und ist dadurch unabhängiger.

Zustandsänderungen werden per Event über ein externes System mitgeteilt - selbst wenn das Stammdaten-System komplett offline ist, funktioniert das Support-System weiterhin eigenständig.

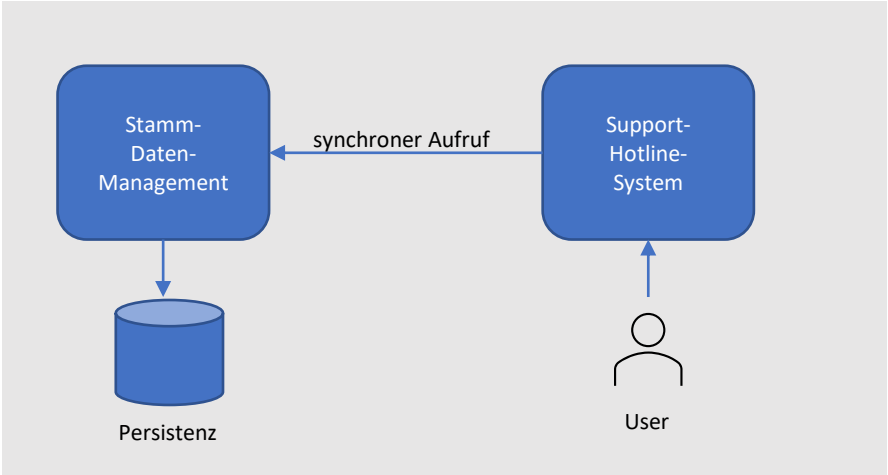

Abbildung 1: Synchroner Aufruf

Sollten weitere «Clients» dazukommen, können diese die Events über das externe System ebenfalls mitlesen. Somit ändert sich nichts für das Stammdaten-System - der Ersteller der Events muss nicht die Konsumenten der Events kennen. Wenn es mehr Konsumenten der Events gibt, entsteht dadurch auch keine höhere Last für den Ersteller des Events.

Diese architekturelle Änderung wird in Abbildung 2 gezeigt. Das zuvor erwähnte externe System für die Events wird ohne konkreten Technologie-Bezug dargestellt.

Martin Fowler hat in seinem Blog-Beitrag [1] und Konferenz-Beitrag [2] vier verschiedene Vorgehensweisen für die Erstellung ereignisgetriebene Architekturen identifiziert. Drei davon werden nachfolgend näher beschrieben:

Ereignisgetriebene Architekturen

Event Notification: Zustandsänderungen werden per Event informiert. Hierbei wird allerdings nur informiert, dass sich etwas geändert hat, aber nicht, was genau. Beispielsweise informiert das Stammdaten-System, dass sich die Adresse des Kunden mit der ID 412 geändert hat. Systeme, die diese Events konsumieren, können dann entscheiden, ob diese Änderung für sie relevant ist und müssen dementsprechend direkt die Daten vom Stammdatensystem abfragen. Dabei bleiben aber einige der erwähnten Nachteile bestehen: Synchrone Abhängigkeit zwischen Support-Service und Stammdaten-Service, Netzwerk-Latenz, Stammdaten-Service muss skalieren, wenn mehrere Services anfragen.

Event-Carried State Transfer: Zusätzlich zu Event Notification wird auch der aktuelle Zustand des Objekts via Event publiziert. Somit müssen Systeme, die diese Events lesen, nicht mehr das Stammdatenmanagement-System abfragen, da die relevanten Informationen schon Teil des Events selbst sind. Lesende Systeme legen die relevanten Informationen lokal ab.

Hierbei kann zum Beispiel nur die Änderung mit dem Event transportiert werden, dann muss das konsumierende System aber den Zustand vor der Änderung kennen. Alternativ kann der komplette Datensatz transportiert werden. Somit müsste in dem Beispiel das Support-System einfach nur die neue Version des Kunden speichern, der im Event transportiert wird.

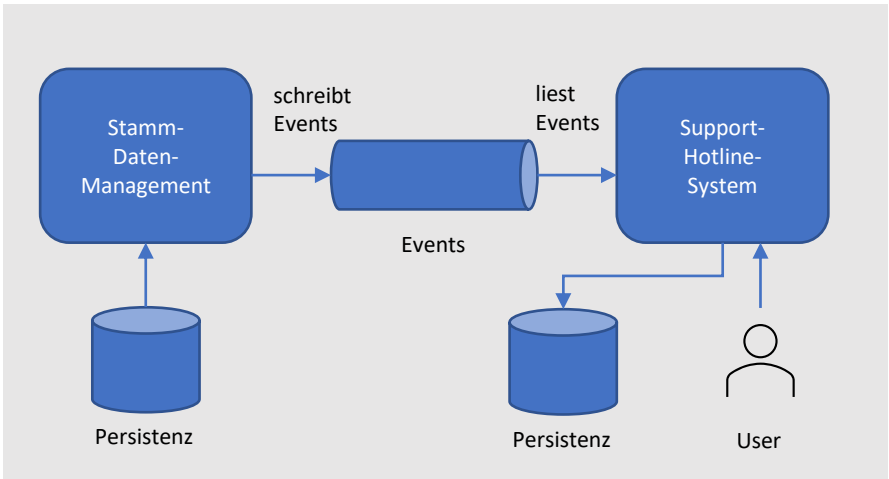

Abbildung 2: Kundendaten via Events

Event Sourcing: Der Zustand einer Entität kann über das erneute Abspielen aller Änderungen wiederhergestellt werden. Ein prominentes Beispiel dafür ist GIT. Der aktuelle Zustand des Sourcecodes in einem Git-Repository kann wiederhergestellt werden, in dem alle Commits abgespielt werden. Ein weiteres Beispiel, welches für die Allgemeinheit eher zugänglich ist, wäre ein Bankkonto. Der Zustand (Saldo) ergibt sich aus allen Transaktionen, die stattgefunden haben. Wenn es sehr viele Events gibt, können sogenannten «Snapshots» erstellt werden, um den Zustand zu einem bestimmten Zeitpunkt festzuhalten. Beim Beispiel des Kontos wären das monatliche und jährliche Abschlüsse.

Nachfolgend wird Kafka vorgestellt und daraufhin, wie Event-Sourcing mit Kafka umgesetzt werden kann.

Kafka

Kafka wurde von LinkedIn als Messaging System entwickelt. Es ist organisiert in Cluster und Broker. Die Queues, auf den Nachrichten versendet werden, werden in Kafka Topics genannt. Die Applikation, die eine Nachricht auf ein Topic schreibt, wird Publisher genannt, die Applikation, die eine Nachricht liest, wird Consumer genannt. Ein Consumer liest eine Nachricht aus einem Topic und teilt Kafka die gelesene Position (also das «Offset») mit. Somit weiss Kafka, bis wohin welcher Consumer in einem Topic gelesen hat. Das hat den Vorteil, dass sich ein Consumer nicht sein Offset merken muss - Kafka verwaltet dies zentral. Das Offset kann auch wieder zurückgesetzt werden, somit würde der Consumer noch einmal alle Nachrichten aus dem Topic verarbeiten.

ENTAGO

Event Sourcing mit Kafka

Standardmässig behalten Messaging-Systeme Nachrichten nur für eine bestimmte Zeit oder so lange, bis sie verarbeitet wurden. In Kafka lässt sich genau einstellen, wie lange eine Nachricht in einem Topic verbleibt, dabei spricht man von «retention». Falls Topics so konfiguriert werden, dass Nachrichten nie entfernt werden, spricht man von «retentionless».
Diese Einstellung wird benötigt, wenn Kafka als Event Store für Event Sourcing verwendet wird. Im Beispiel des Kontos würde ein publizierendes System alle Änderungen eines Kontos als einzelne Nachricht auf ein Kafka-Topic schreiben - dieses Topic ist «retentionless» konfiguriert. Alle lesenden Systeme erhalten den Saldo, in dem alle Nachrichten auf dem Topic verarbeitet werden.

Auch neue Systeme, beziehungsweise Systeme, die erst in Zukunft hinzukommen, können auf dem Topic lesen und die vorhandenen Daten verarbeiten.

Herausforderungen

Bei der Umsetzung von Event Sourcing mit Kafka gibt es auch einige Herausforderungen, welche gelöst werden müssen.

Schema Evolution: Wenn Nachrichten auf einem Topic versendet werden, entsprechen diese einem bestimmten Schema. Im Fall von Kontobewegungen kann ein simples Schema beispielsweise folgende Attribute enthalten: **Kontonummer** und **Betrag**. Was passiert nun, wenn ein Attribut hinzukommt, wegfällt oder sich verändert? Tools wie «Avro» unterstützen die Definition und die Evolution von Schemata. Dort können Strategien mit den Nachrichten versandt werden, sodass lesende Systeme mit Schema-Änderungen umgehen können.

Externe Abhängigkeiten: Eine weitere Herausforderung sind externe Abhängigkeiten beim Abspielen von Nachrichten, zum Beispiel wenn der aktuelle Preis für ein Produkt erst von einem externen System erfragt werden muss. Wenn man zwei Wochen später diese Nachrichten nochmal abspielt, kann man das externe System nicht fragen «Was war der Preis vor zwei Wochen?». Somit müssen alle Antworten von externen Systemen ebenfalls gespeichert werden und in einem Event Store verfügbar gemacht werden.

Asynchronität: Diese Herausforderung entsteht, sobald es mehr als einen Publisher für einen Event Store gibt. Als Beispiel dafür kann man sich Git vorstellen, wenn Benutzer lokale Commit pushen möchten - sich aber der remote branch bereits geändert hat. In diesem Fall wird dieser Konflikt manuell gelöst. Bei automatisierten Systemen müssen dafür aber Strategien entwickelt werden, wie mit solchen potentiellen Konflikten umgegangen wird. Falls es nur einen Publisher pro Event Store gibt, entsteht dieses Problem allerdings nicht.

Fazit

Bei der Umsetzung von Event Sourcing mit Kafka können Systeme entkoppelt und entlastet werden. Ein weiterer Vorteil ist, dass die komplette Historie von Entitäten wie Kontodaten jederzeit verfügbar und vor allem dadurch auch auditiert werden kann. Darüber hinaus kann die Historie bei Fehleranalysen hilfreich sein – falls sich beispielsweise ein Attribut einer Entität fälschlicherweise geändert haben sollte.

Da alle Änderungen zu einer Entität in einem Eventstore (bei Kafka in einem Topic) vorliegen, können auch zukünftige Systeme alle relevanten Daten direkt laden, indem die Nachrichten des Eventstores verarbeitet werden. Aufwände für «Initial-Load» Implementierungen fallen somit weg.

Abschliessend kann festgehalten werden, dass Event-Sourcing und ereignisgetriebene Architekturen nicht der heilige Gral der Software-Entwicklung sind - es ist ein Werkzeug im Werkzeugkasten der Software-Ingenieure und kann bei passenden Anwendungsfällen in Betracht gezogen werden.

Falls für Sie ereignisgetriebene Vorgehensweisen oder Kafka ein Thema ist, kontaktieren Sie uns gerne für ein Beratungsgespräch.

Quellen

[1] https://martinfowler.com/articles/201701-event-driven.html Abruf 14.09.2021

[2] https://www.youtube.com/watch?v=STKCRSUsyP0 Abruf 14.09.2021

Moritz Eberhard
Senior Software Engineer

ENTAGO

Entago AG
Buckhauserstrasse 34
CH-8048 Zurich

info@entago.ch

www.entago.ch