Axon und Kafka

Wie schneidet Axon im Vergleich zu Apache Kafka ab?

Vijay Nair

AxonIQ

Einleitung

Einer der häufigsten Diskussionspunkte, die regelmäßig in Interaktionen mit Kunden/Interessenten oder in Foren auftauchen, ist die Frage, wie Axon im Vergleich zu Apache Kafka abschneidet. Kann das eine die Aufgabe des anderen übernehmen? Sind sie komplementär zueinander? Können sie zusammenarbeiten? Bietet Axon irgendwelche Funktionen für die Zusammenarbeit mit Kafka?

Axon ist eine Plattform, die speziell für die Implementierung von CQRS- und Event-Sourcing-basierten Architekturen entwickelt wurde - eine Architektur, die das Design und die Entwicklung von Anwendungen als ein *System of Events* und nicht als ein *System of State* behandelt.

Mit Event Sourcing als zentralem Diskussionsthema beabsichtigt dieser Blogbeitrag, klare Antworten auf die oben genannten Fragen zu liefern. Sehen wir uns an, wie wir das angehen wollen.

Was werden Sie in den nächsten Minuten zu lesen bekommen?

- **Was ist Event Sourcing und wie passt es in das Event-Driven Architecture Ökosystem?**
  Wir erläutern die Prinzipien des Event Sourcing, das die Implementierung von Konzepten aus den Bereichen Domain-Driven Design, CQRS (Command Query Responsibility Segregation), Event Storage und Event Processing erfordert. Anschließend gehen wir auf die Infrastrukturkomponenten ein, die für die Implementierung einer Event-Sourcing-basierten Architektur erforderlich sind.
- **Die Implementierungsaspekte dieser Komponenten**
  Zunächst mit Kafka, gefolgt von Axon. Am Ende dieser Aufgabe sollten wir in der Lage sein, die Eignung der beiden Plattformen für die Implementierung von Event Sourcing klar zu bewerten, was uns im Wesentlichen hilft, die meisten der oben gestellten Fragen zu beantworten.
- **Ein Beweis für die komplementäre Nutzung der beiden Plattformen**
  Und wir liefern Details darüber, wie die beiden Plattformen durch die Nutzung von Axons Unterstützung für Kafka kombiniert werden können

Schnappen Sie sich Ihren Kaffee (oder Tee) und lassen Sie uns beginnen!

Event-Driven Architectures

Das Paradigma der Event-Driven Architecture (EDA) befürwortet den Aufbau von Anwendungen, in deren Mittelpunkt die *Erzeugung* und *das Handling von Events* steht.

Obwohl es diese Technologie schon lange gibt, hat sie in letzter Zeit eine Renaissance erlebt, was auf groß angelegte Innovationen in diesem Bereich zurückzuführen ist. Dies wurde vor allem durch die neue Klasse moderner Anwendungen angetrieben, die *reaktiv in Echtzeit*, *verteilt* und *skalierbar* sein müssen. Dies hat zu neuen Patterns/Frameworks und Plattformen geführt, die bei der Erstellung solcher Anwendungen helfen. Eines dieser Patterns, das in letzter Zeit an Bedeutung gewonnen hat, ist das Event-Sourcing-Pattern.

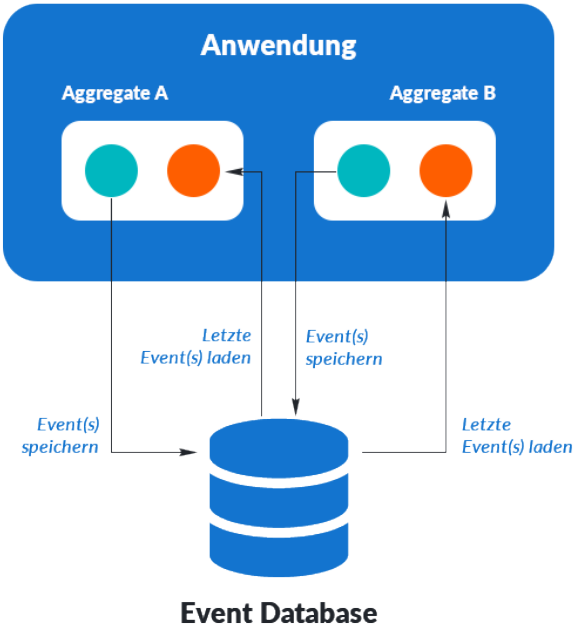
Event Sourcing

Das aus der Welt des Domain-Driven Design (DDD) stammende *Event Sourcing* befürwortet das Design und die Entwicklung von Anwendungen, indem es diese als ein *System of Events* und nicht als ein *System of State* behandelt.

Event Sourcing schreibt vor, dass der Zustand der Anwendung **nicht explizit in einer Datenbank gespeichert** werden soll, sondern **als eine Reihe von zustandsändernden Events**. In DDD wird der Zustand einer Anwendung durch den Zustand ihrer verschiedenen *Aggregate* repräsentiert, d. h. durch Geschäftseinheiten, die die Kerngeschäftslogik einer Anwendung modellieren. Jede Operation auf einem Aggregat führt zu einem Event, das die Zustandsänderung des Aggregats

beschreibt. Dieses Event wird in einer Datenbank gespeichert und an die Menge der Events angehängt, die möglicherweise bereits für dieses Aggregat aufgetreten sind. Der aktuelle Zustand eines *Aggregats* wird dann rekonstruiert, indem die gesamte Historie der Events aus der Datenbank geladen und wiedergegeben wird.

Das Konzept des Event Sourcing ist im Folgenden dargestellt.



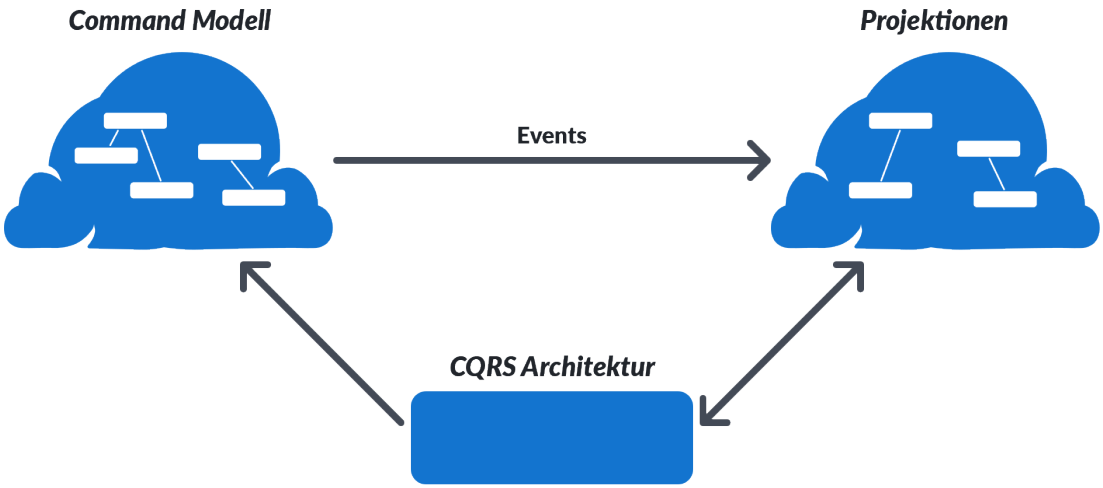
Im Folgenden finden Sie eine Zusammenfassung der Vorteile von Event Sourcing,

- Etablierter Audit Pfad
- Data Mining / Analytics
- Design Flexibilität
- *Temporäre Reports*

Event Sourcing wird fast immer mit einem anderen Pattern verwendet: CQRS (Command Query Responsibility Seperation). Außerhalb des Kontexts von Event Sourcing beschreibt CQRS im Wesentlichen das Konzept von zwei verschiedenen Modellen, dem Command Modell, das Anweisungen (d. h. *Commands*) zur Änderung des Anwendungszustands empfängt, und dem Query (or Read) model, das Anweisungen (d. h. *Queries*) zum Abrufen einer bestimmten Darstellung des

aktuellen Zustands empfängt. Event Sourcing stützt sich auf das Command Modell, um die Events zu verarbeiten und zu speichern, wenn sie auf einem Aggregat auftreten. Die Events werden an das Query-Modell weitergeleitet, das kontinuierlich aktualisiert wird, um eine bestimmte Darstellung des aktuellen Zustands basierend auf denselben Events zu erhalten. Kurz gesagt, CQRS ermöglicht Event Sourcing.

Das CQRS-Pattern ist im Folgenden abgebildet.



Zusammenfassend lässt sich sagen, dass die Implementierung von Event Sourcing die Umsetzung einer Reihe von Aspekten rund um die Event Storage und Domain-Driven Design und CQRS beinhaltet. Die Implementierung dieser Funktionen in einer Event Sourced Architecture erfolgt durch die Kombination einer

- Physische Infrastruktur, d. h. ein Event Store, der als Datenbank für die Speicherung von Events dient
- Logische Infrastruktur, d. h. ein Application Framework, das eine API zur Modellierung von Aggregaten, zur Handhabung von commands/queries und zur Durchführung von Event-Sourcing-Operationen bereitstellt

Lassen Sie uns diese Möglichkeiten im Detail untersuchen.

Event-Store-Funktionen

Der Event Store muss sich an bestimmte Grundprinzipien halten, die im Folgenden beschrieben werden:

Append Events

Der Event Store muss in der Lage sein, die Events in einer „Append-Only“ Weise zu speichern, wie sie in den verschiedenen Aggregaten auftreten.

In Bezug auf die Append-Funktionen muss der Event Store Folgendes vorsehen:

- Konsistenz - Wenn wir Events für ein Aggregat an den Event Store anfügen, muss dieser validieren und sicherstellen, dass die Sequenznummer für die Events inkrementell und ohne Duplikate gespeichert wird. Die Sequenzierung ist entscheidend, da die korrekte Konstruktion des Zustands des Aggregats ein strukturiertes Lesen seiner Events erfordert.
- Atomarität - Eine Operation auf einem Aggregat kann zu mehreren Events führen. Der Event Store muss sicherstellen, dass entweder alle Events zusammen oder gar keine geschrieben werden.
- Dauerhaftigkeit - Festgeschriebene Events im Event Store müssen gegen Datenverlust geschützt werden.
- Snapshots - Funktion zum Erstellen und Speichern von Snapshots für eine Reihe von Events eines Aggregats zur Optimierung der Zustandskonstruktionsphase.

Read Events

Der Event Store muss die Funktionen zum Lesen der gespeicherten Events bereitstellen. In Bezug auf die Lesefunktionen muss der Event Store Folgendes bieten:

- Alles zu einem Aggregat - Möglichkeit, alle Events zu lesen, die für ein bestimmtes Aggregat gespeichert wurden, einschließlich eventueller Snapshots.

- Alles seit einem Zeitpunkt - Möglichkeit, alle Events zu lesen, die seit einem bestimmten Zeitpunkt gespeichert wurden, um unsere query/read Modelle zu erstellen.
- Ad-hoc-Queries - Funktion zur Durchführung von Ad-hoc-Queries im Event Store.
- Isolation - Sicherstellen, dass gespeicherte Events erst dann gelesen werden können, wenn die Transaktion in den Speicher übertragen wurde.
- Optimierung - Für das Lesen neuerer Events optimiert.

Route Events

Der Event Store ist für das Routing der von Aggregaten ausgesendeten Events an die beteiligten Empfänger verantwortlich. Im Wesentlichen ähnelt diese Fähigkeit der eines Event-Busses mit Unterstützung der erforderlichen Broker-Funktionen - Sync/Async-Modi und garantierte Übermittlungen.

Skalierbarkeit

Der Event Store muss eine konstante und vorhersagbare Leistung für das Anfügen, Lesen und Routen von Events bieten, während der Event Store auf möglicherweise Milliarden von Events anwächst.

Verfügbarkeit & Verlässlichkeit

Der Event Store muss immer verfügbar sein und in einem Clustermodus mit Funktionen für Lastausgleich, automatisches/schnelles Failover und Wiederherstellung arbeiten können.

Application Framework API Funktionen

Die API, die von einem Application Framework in einer Event-Sourcing-basierten Architektur bereitgestellt wird, bietet die notwendigen Funktionen für Client-

Applikationen, um mit dem Event Store zu kommunizieren und Operationen auszuführen. Die API muss extrem performant sein und den extremen Anforderungen an die Skalierbarkeit gerecht werden.

Diese Funktionen werden im Folgenden beschrieben:

Aggregat-Modellierung

Da sich Event Sourcing in erster Linie mit Aggregaten befasst, muss die API Funktionen zur Modellierung von Aggregat-Stereotypen bereitstellen, einschließlich der Fähigkeit, Befehle zu verarbeiten, Events zu generieren und den Zustand eines Aggregats basierend auf seinen vergangenen Events zu rekonstruieren.

Event Publishing

Die API muss Funktionen zum Übertragen von Events in den Event Store bereitstellen, so dass das Aggregat als Ursprung erkannt wird, um später diese Aggregat-Instanz als Event-Source nutzen zu können.

Event Handling

Die API muss Funktionen zum Empfangen und Verarbeiten von veröffentlichten Events bereitstellen.

Event Wiedergabe

Die API muss Funktionen zur Wiedergabe von Events aus dem Event Store bereitstellen, damit eine Projektion vom Beginn des Streams oder einem beliebigen Punkt im Stream erstellt bzw. wiederhergestellt werden kann. Die Reihenfolge, in der Events gestreamt werden, sollte bei Wiederholungen konsistent sein.

Event Versionierung

Die API muss die Möglichkeit bieten, verschiedene Versionen von Events zu verarbeiten, die sich während des Lifecycles einer Anwendung entwickeln, ohne dass die Domänenlogik der Anwendung jede verfügbare Version des Events kennen muss.

Exception Handling

Die API muss für jede Event-Sourcing-Operation geeignete Funktionen zum Exception-Handling bereitstellen.

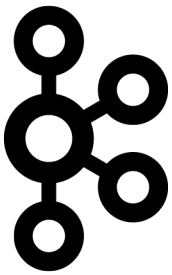
Eine zusammenfassende Darstellung aller Fälle ist unten abgebildet:

Damit ist dieser Abschnitt komplett, in dem die Funktionen, die eine Event-Sourcing-Infrastruktur bieten muss, detailliert beschrieben wurden. Im nächsten Teil des Blogs geht es um die Implementierung dieser Infrastruktur mit Kafka und Axon.

Implementierung von Event Sourcing

Kafka - Event Streaming, somit kein Event Sourcing

Ursprünglich bei LinkedIn entwickelt und derzeit unter der Apache Foundation, leistete Kafka Pionierarbeit bei der Einführung des Konzepts des Event Streaming, das sich im Wesentlichen um 3 Hauptfunktionen dreht:



- Veröffentlichen / Abonnieren von Event Streams
- Events speichern
- Event Streams in Echtzeit verarbeiten

Die Implementierung von Event Sourcing mit Kafka erfordert ein optimales Design von Kafka Topics, die als Event Store für das Anfügen und Lesen von Aggregat-Events dienen. Wir könnten mit einem grundlegenden Design mit einem einzelnen partitionierten Topic pro Aggregat-Instanz beginnen. Dies würde geordnete Appends für die Events garantieren und das Lesen des Zustands könnte so einfach sein wie das Lesen des Topics ab Offset 0. Es wird schnell klar, dass dies zu Skalierungsproblemen führen würde, sobald wir Millionen von Aggregate-Instanzen haben, da es einfach zu viele Topics geben wird. Ein anderes Design könnte darin bestehen, ein einziges partitioniertes Topic für alle Aggregat-Typen zu haben. Dies führt dazu, dass das Lesen von Events extrem langsam ist, da der gesamte Datensatz der Aggregat-Typen und Instanzen abgefragt werden muss.

Wir könnten eine Datenbank zum Mix hinzufügen, um einige dieser Funktionen bereitzustellen, aber dann würden wir auf das Problem der verteilten Transaktionen stoßen, um Schreibvorgänge über Ressourcen hinweg zu garantieren. Um dieses Problem zu lösen, könnten wir ein anderes Tool der Plattform verwenden - Kafka

Streams. Streams bietet das Konzept von State Stores, die den Event-Stream des Aggregats als Snapshots speichern könnten. Dies löst zwar das Problem der verteilten Speicherung sowie der Lesevorgänge für die Konstruktion eines Aggregatzustands, da es aber nur den aktuellen Snapshot speichert, kann es nicht zum Aufbau Ihrer Lesemodelle verwendet werden.

Die von Kafka bereitgestellte API ist lediglich auf eine bestimmte Auswahl von Operationen beschränkt, die für Event Sourcing erforderlich sind (Publishing von Events / Handling von Events). Es wird ziemlich schnell deutlich, dass Kafka uns nicht die volle Bandbreite an Funktionen bietet, um eine effiziente, robuste und skalierbare Event-Sourcing-Infrastruktur zu implementieren.

Axon - Event Sourcing und mehr...

Axon ist die führende Plattform, die sich exklusiv mit der Implementierung von Event-Driven-Architekturen befasst und erkannt hat, dass eine solche Architektur mehr als nur Events benötigt.

Axon bietet nicht nur einen hoch skalierbaren Event Store, sondern erweitert das Konzept traditioneller CQRS/Event Sourcing-Architekturen, indem es jede Operation innerhalb einer Anwendung (Commands/Queries und Events) als Nachrichten behandelt. Jeder dieser Nachrichtentypen erfordert eine andere Routing-Strategie, die Axon unterstützt.

Axon bietet zwei Hauptkomponenten

- **Axon Server** - Ein hoch skalierbarer, verbreiteter und zweckmäßiger Event Store und Message Router der ohne Konfiguration auskommt. Er leitet Nachrichten auf Basis der von jeder Anwendung bei der Verbindung bereitgestellten Funktionen weiter, wobei die spezifischen Routing-Anforderungen jedes Nachrichtentyps berücksichtigt werden. Events, die einen Wert für einen langen Zeitraum tragen, werden für den Zweck des Event Sourcing gespeichert und sofort für das Event Streaming zur Verfügung gestellt.
- **Axon Framework** - Implementiert die gesamte Palette an API-Funktionen, die

für Event Sourcing / Message Routing-Vorgänge erforderlich sind. Es bietet die erforderlichen Bausteine, um alle nicht-funktionalen Anforderungen zu erfüllen, so dass sich die Entwickler stattdessen auf die funktionalen Aspekte ihrer Applikation konzentrieren können.

Axon hat Kunden aus vielen verschiedenen Branchen, die damit eine Event-Driven Microservices-Infrastruktur auf Enterprise-Niveau ausrollen.

Kombination von Axon und Kafka

Wie wir oben gesehen haben, ist die Implementierung von Event Sourcing von Grund auf komplex und erfordert eine speziell entwickelte Plattform auf Enterprise-Niveau. Auf den ersten Blick sieht die Kafka-Plattform nach der idealen Lösung für Event Sourcing aus, und Kunden, die sie derzeit als Infrastruktur für die Event-Verarbeitung nutzen, sehen sie als eine sinnvolle Erweiterung, um auch eine Event-Sourcing-Infrastruktur zu unterstützen und auszurollen. Sobald jedoch die verschiedenen Event Sourcing-Anforderungen implementiert werden, wird klar, dass Kafka dies nicht von Haus aus bietet oder es einfach nicht unterstützt. Ist dies eine Beschränkung von Kafka? Absolut nicht, Kafka wurde mit dem Ziel entwickelt, eine Event-Streaming-Plattform zu sein, und in dieser Rolle glänzt es.

Wir haben festgestellt, dass Unternehmen den Weg gehen, Kafka zu erweitern, um den Zweck einer Event-Sourcing-Infrastruktur zu erfüllen. Das führt häufig dazu, dass die nicht unterstützten Anforderungen um Kafka herum aufgebaut werden, was in der Regel keine sehr gute Idee ist. Diese Anforderungen sind recht komplex und erfordern viel Aufwand, Zeit und Wissen. Unternehmen sind besser dran, wenn sie eine speziell entwickelte Plattform wie Axon für Event Sourcing verwenden. Es ist so, als würden Sie Kafka als Ihre Event-Infrastruktur nutzen, ohne dass Sie Ihre eigene ausrollen müssen!

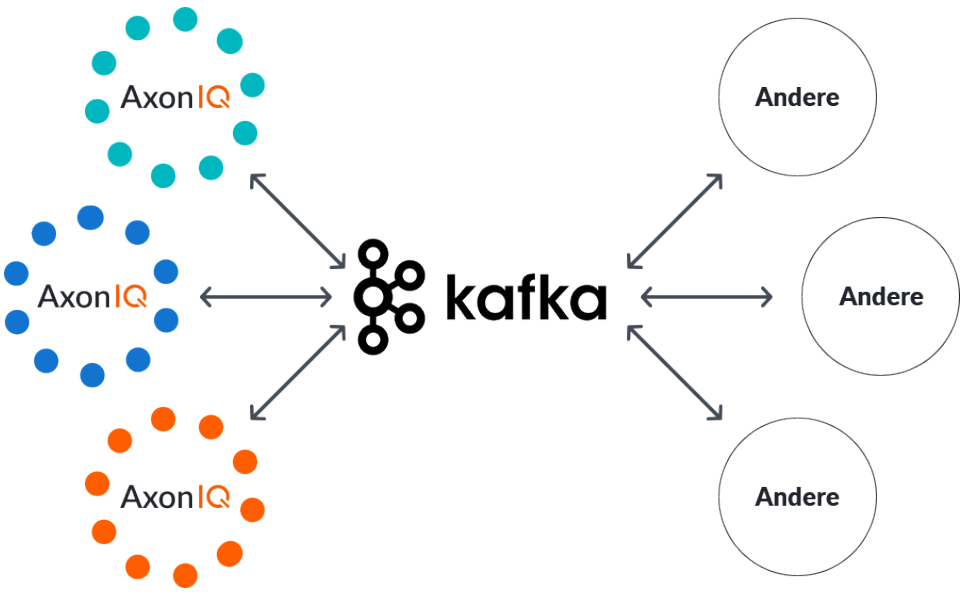
Erwarten wir also, dass Sie einfach Ihre Kafka-Infrastruktur abschaffen und sie durch Axon ersetzen? Definitiv nicht! Und das empfehlen wir auch nicht. Kunden verlassen sich auf Kafka als ihren zentralen Event Hub und haben ein Ökosystem von Anwendungen darum aufgebaut, das von Analytics bis zu Integration Services reicht. Es besteht kein Zweifel daran, dass die Kafka-Plattform heute die

dominierende Event Processing Infrastruktur ist.

Die Stärke von Axon liegt im Anwendungslevel-Messaging, wo Services ihre Aktivitäten auf einer detaillierteren Ebene koordinieren müssen - zum Beispiel Services, die im Bereich Order Fulfillment zusammenarbeiten. Diese Vorgänge führen zu Events, von denen einige wichtiger sind als andere. Diese so genannten Milestone-Events sind es typischerweise wert, über diese kooperierenden Services hinaus veröffentlicht zu werden. Genau hier liegt die Stärke von Kafka.

Um die Integration mit Kafka zu unterstützen, bietet Axon einen Kafka-Connector als Teil des Axon Frameworks (https://docs.Axoniq.io/reference-guide/extensions/kafka). Der Zweck des Connectors ist es, das Beste aus beiden Welten zu nutzen, d. h. Axon für seine zweckmäßigen Event-Sourcing-Funktionen und die Verarbeitung von Command/Query Messages zu verwenden und gleichzeitig die Verantwortung für die Event-Zustellung an nachgeschaltete Systeme zu übertragen, die Kafka nutzen.

Die Darstellung der Integration ist unten zu sehen, wobei mehrere Axon-Anwendungen für die Bereitstellung von Events an nachgeschaltete Systeme auf Kafka angewiesen sind



Axon und Kafka - zwei unterschiedliche Einsatzgebiete

Abschließend lässt sich sagen, dass Axon und Kafka zwei unterschiedliche Zwecke im Bereich der Event-Driven Architecture erfüllen - Axon bietet die Unterstützung auf Anwendungsebene für die Domain Modeling und Event Sourcing sowie das Routing von Commands, Events und Queries, während Kafka als Event-Streaming-Plattform glänzt. Die Kombination aus beidem stellt ein überzeugendes Angebot dar, da es Kunden hilft, ihre bestehenden Investitionen in Kafka beizubehalten und gleichzeitig Axon zu nutzen, um eine robuste, Event-Driven Microservices-Infrastruktur einzuführen

AxonIQ