NACHRICHTEN VON KAFKA

Moderne Vernetzungsarchitektur, auch über Behördengrenzen hinweg

| von LASZLO LÜCK und DR. CHRISTIAN KIEHLE

„Von einem gewissen Punkt an gibt es keine Rückkehr mehr. Dieser Punkt ist zu erreichen.“ Franz Kafka

Neben einer kurzen Einführung in Apache Kafka werden in diesem Artikel die Vor- und Nachteile erörtert, die der Einsatz von Kafka bietet. IT-Entscheider bekommen so eine Grundlage, um den Einsatz von Apache Kafka für eigene IT-Vorhaben bewerten zu können.

Moderne IT-Verfahren basieren in immer höherem Maße auf verteilten Systembestandteilen. Mit der weithin geforderten und zunehmend auch realisierten Vernetzung von Behörden wächst die Bedeutung verteilter Systeme, auch über Behördengrenzen hinweg. Was in der Planungsphase auf einem hohen Abstraktionsniveau einfach und naheliegend aussieht, kann im Detail, insbesondere unter realen Betriebsbedingungen, durchaus hochkomplex und aufwendig sein. Mit der Verarbeitung ständig wachsender Datenmengen in immer kürzeren Zeitabständen, wie sie in vielen Bereichen der inneren Sicherheit (zum Beispiel der Terrorismusabwehr) anfallen, spielen sogenannte Messaging-Systeme eine immer größere Rolle. Auch kommt der Speicherung und Verarbeitung von Datenströmen über verteilte Plattformen eine wichtige Rolle zu. Aus der Vielzahl der kommerziellen und nichtkommerziellen Messaging-Systeme ist Apache Kafka[1] eines der bekanntesten Projekte. Apache Kafka ist ein Open-Source-Message-Broker, der sich in den letzten Jahren steigender Beliebtheit in geschäftskritischen Anwendungen der Industrie und der öffentlichen Verwaltung erfreut.[2]

MESSAGING-SYSTEM

An enterprise messaging system (EMS) or messaging system in brief is a set of published enterprise-wide standards that allows organizations to send semantically precise messages between computer systems. EMS systems promote loosely coupled architectures that allow changes in the formats of messages to have minimum impact on message subscribers. EMS systems are facilitated by the use of structured messages (such as using XML or JSON), and appropriate protocols, such as DDS, MSMQ, AMQP or SOAP with web services.[3]

Ein Enterprise-Messaging-System (EMS) oder kurz Messaging-System ist eine Sammlung veröffentlichter unternehmensweiter Standards, mit denen Organisationen semantisch präzise Nachrichten zwischen Computersystemen senden können. EMS-Systeme fördern lose gekoppelte Architekturen, bei denen Änderungen in den Nachrichtenformaten nur minimale Auswirkungen auf die Abonnenten von Nachrichten haben. EMS-Systeme werden durch die Verwendung strukturierter Nachrichten (wie XML oder JSON) und geeigneter Protokolle wie DDS, MSMQ, AMQP oder SOAP mit Webdiensten erleichtert.

WARUM SOLLTE EIN IT-VERANTWORTLICHER IN DER ÖFFENTLICHEN VERWALTUNG KAFKA KENNEN?

Kafka wurde 2011 von LinkedIn entwickelt. Nach dem Umbau vom Software-Monolithen zur verteilten Service-Architektur war der Plan, große Datenmengen aus unterschiedlichen Quellen zwischen den Services auszutauschen. Im Fokus stand dabei immer, eine skalierbare Plattform mit geringen Latenzen und hohem Durchsatz zu implementieren.

Genau diese Eigenschaften – hohe Skalierbarkeit, geringe Latenz – machen Apache Kafka für geschäftskritische Anwendungen im behördlichen Umfeld interessant. Hinzu kommt der Aspekt der freien Verfügbarkeit und der Wegfall von Lizenzkosten. Apache Kafka gilt als skalierbar und fehlertolerant und kommt insbesondere in Big-Data-Anwendungen zum Einsatz.

Apache Kafka erfüllt zahlreiche Anforderungen, die eine IT-Architektur für verteilte, hoch skalierbare Anwendungen abdecken muss:

- Hohe Skalierbarkeit: Mit höheren Anforderungen können die Nachrichtenvermittler (sogenannte Broker) dynamisch erhöht werden. Dies ist auch nach der initialen Einrichtung möglich.
- Hohe Ausfallsicherheit: Durch die Möglichkeit einer (geografischen) Verteilung der Infrastruktur lässt sich die Wahrscheinlichkeit eines Datenverlusts oder Infrastrukturausfalls signifikant minimieren.
- Geringe Latenz: Die gesamte Messaging-Infrastruktur hat eine sehr niedrige Latenz, da sie sich unterschiedliche Lese- und Schreibstrategien des zugrunde liegenden Betriebssystems zunutze macht.
- Hohe Geschwindigkeit: Die durch Kafka implementierte Speicherstrategie (Segmentdateien mit konfigurierter Anzahl Nachrichten pro Datei) sowie die Verteilung der Daten in der Infrastruktur ermöglicht eine sehr hohe Gesamtperformance des Kafka-Verbunds.
- Sicherheit: Kafka nutzt ein dreistufiges Sicherheitssystem. So werden Sicherheitsmechanismen auf der Transportschicht (durch TLS/SSL), auf der Sitzungsschicht (SSL/SASL) und in der Anwendungsschicht (ACL) umgesetzt.

Eine nachhaltige Architekturentscheidung zugunsten einer Nutzung von Apache Kafka lässt sich jedoch nur treffen, wenn man ein ganzheitliches Bild der damit verbundenen Vor- und Nachteile sowie der Randbedingungen zeichnet.

> „Wir haben zunächst verschiedene benutzerdefinierte Daten-Pipelines für unsere verschiedenen Streaming- und Queuing-Daten entwickelt. Die Anwendungsfälle für diese [Pipeline-]Plattformen reichten von der Verfolgung von Site-Ereignissen (beispielsweise Seitenaufrufe) bis zum Sammeln von aggregierten Logs von anderen Diensten. Andere Pipelines stellten Warteschlangenfunktionen für unser InMail-Messagingsystem usw. bereit.
>
> Diese wurden für die Skalierung zusammen mit der Site benötigt. Anstatt jede Pipeline einzeln zu pflegen und und weiterzuentwickeln, haben wir in die Entwicklung einer einzelnen, verteilten Publisher-Subscriber-Plattform investiert. So wurde Kafka geboren."
>
> Joel Koshy, 2016[4]

DIE BESTANDTEILE VON APACHE KAFKA

Apache Kafka ist eine persistierende Nachrichten-Queue zur Speicherung und Verarbeitung von Datenströmen über eine verteilte Streaming-Plattform. Nachrichten werden in benannten Kanälen, den Topics, zusammengefasst. Dabei werden neue Nachrichten eines Topics an das Ende dieses Topics angehängt und ihnen wird eine hochzählende, nicht veränderbare (immutable) Nummer zugewiesen, die eine eindeutige Reihenfolge der Nachrichten im Topic sicherstellt. Eine Nachricht, die in eine Kafka-Nachrichtenschlange geschrieben wurde, wird dort so lange gespeichert, bis sie über konfigurierbare Eigenschaften ihres Topics wieder aus dieser Nachrichtenschlange entfernt wird.

KAFKA-CLUSTER UND KAFKA-NODES

Ein Apache-Kafka-Cluster besteht aus einer konfigurierbaren Anzahl Kafka-Nodes. Ein Kafka-Node speichert die Nachrichten auf einem Datenträger ab. Daher ist es wichtig, dass der Apache-Kafka-Node auf einen schnellen Datenspeicher zugreifen kann – am besten mithilfe von Solid State Drives (SSDs). Jede Kafka-Node hat zudem Zugriff auf ein weiteres System, den sogenannten Apache-Zookeeper.

Zookeeper[5] erfüllt hauptsächlich Verwaltungsaufgaben innerhalb des Clusters, wie zum Beispiel die Verteilung der Partitionen, die Wahl des Leaders (der „führenden" Partition eines Clusters) und der Follower-Partitionen[6], das Speichern der Topics und Offset-Id für den Topic und vieles mehr.

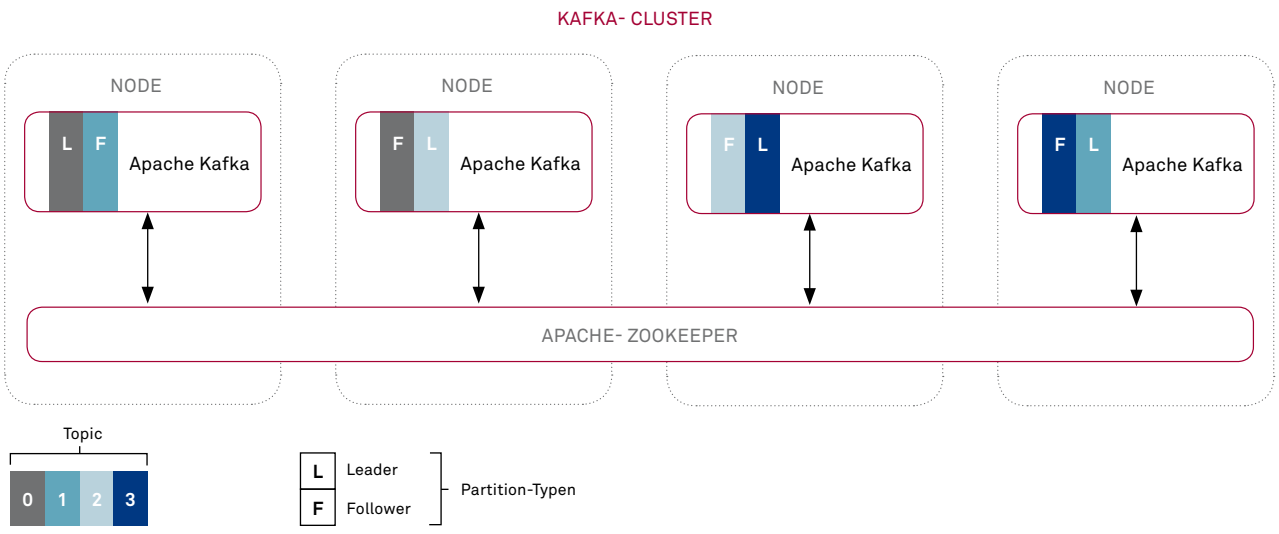

Abbildung 1: Aufbau eines Kafka-Clusters

KAFKA-MESSAGE-BROKER

Bei einer klassischen Message-Queue[7] werden die Nachrichten vom Sender in die Queue geschrieben und sequentiell zum Empfänger transportiert. Bis zur Verarbeitung der Nachricht durch den Empfänger sind die Nachrichten in der Message-Queue persistiert. Wenn die Nachricht durch einen Empfänger gelesen wurde, sendet der Empfänger eine Acknowlege-Message an den Broker. Dadurch wird sie aus der Queue entfernt und steht keinem anderen Empfänger mehr zur Verfügung. Eine Message-Queue wird immer dann verwendet, wenn gewährleistet sein muss, dass eine Nachricht genau einmal empfangen und verarbeitet wird. Kafka empfängt Nachrichten in sogenannten „Kanälen“ (Topics). Daten in einem Topic sind eine Sequenz fortlaufender Nachrichten. Diese Nachrichten werden auf Datenträgern gespeichert. Um die Latenz so gering wie möglich zu halten, werden die Nachrichten direkt vom Netzwerkstack auf den Datenträger, vorzugsweise eine Solid State Disk, geschrieben. Indem ein Topic immer auf mehrere Brokern (Nodes) verteilt (repliziert) wird, ist für Ausfallsicherheit gesorgt. Ein Broker ist typischerweise eine eigenständige Hardware-Instanz.

Eine Nachricht steht für alle Consumer desselben Topics so lange zur Verfügung, bis die konfigurierbaren Eigenschaften des Topics eine Löschung der Nachricht erfordern. Die Nachrichten eines Topics unterliegen bestimmten Eigenschaften bezüglich ihres Verbleibs wie zum Beispiel der Lebensdauer (Time-to-live, TTL), des vorhandenen Speicherplatzes (Disk) oder einer Kombination von

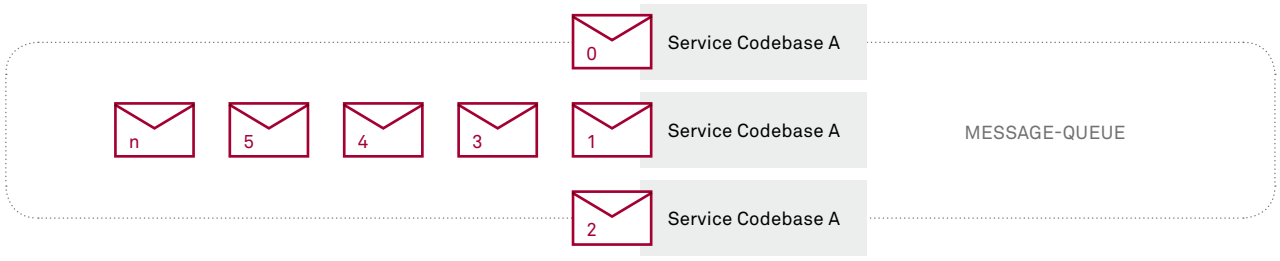

Abbildung 2: Einfache Message-Queue

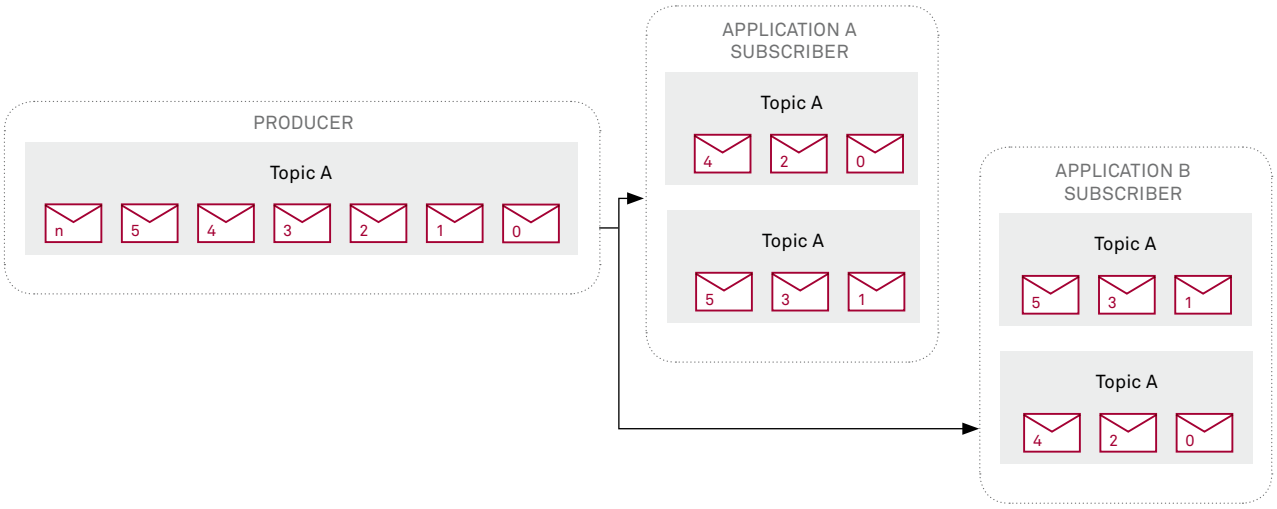

Abbildung 3: Kafka-Message-Queue mit Empfängern (Subscriber)

beidem. Eine weitere Eigenschaft ist das sogenannte „Compacted Topic". Hierbei wird jeweils nur die neueste Nachricht einer Nachrichten-ID gespeichert. Ältere Nachrichten mit derselben Id werden entfernt. Schreibt ein Publisher Nachrichten in ein Topic, werden sie von den Empfängern (Subscribers), die diesen Kanal abonniert haben, gelesen und verarbeitet. Hat der Empfänger die Nachricht gelesen beziehungsweise verarbeitet, wird das gegenüber Apache Kafka quittiert (Acknowledge).

Werden Nachrichten schneller in die Kafka-Queue geschrieben, als sie von den verarbeitenden Systemen gelesen beziehungsweise quittiert werden, kommt es zu einem sogenannten „Lag". Ein Lag beschreibt die Differenz zwischen der Nummer der in die Queue geschriebenen und der Nummer der durch das verarbeitende System verarbeiteten Nachricht. Ein zu großer Lag kann zu Dateninkonsistenzen beziehungsweise Datenverlusten führen. Daher wird der Lag in geeignete Monitoring-Mechanismen aufgenommen.

DATENMODELLE ALS KLEINSTER GEMEINSAMER NENNER ZWISCHEN PUBLISHER, SUBSCRIBER UND VERSIONIERUNG

Die Nachricht eines Topics kann in Apache Kafka von verschiedenen Konsumenten gelesen und verarbeitet werden. Beim Design des Datenmodells zu einer Nachricht ist es deshalb unumgänglich, dass sich alle Konsumenten auf ein definiertes Datenmodell einigen (kleinster gemeinsamer Nenner). Um Änderungen am Datenmodell eines Topics zuzulassen, kann man geeignete Mechanismen einsetzen, beispielsweise Versi-

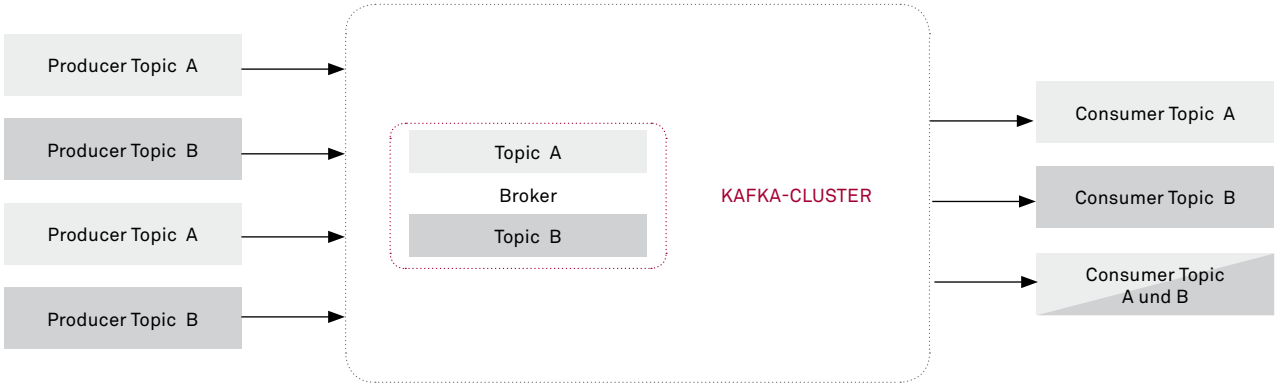

Abbildung 4: Kafka-Cluster mit Publisher und Subscriber

VORTEILE	
Open Source	Keine Lizenzkosten, unabhängig von der Größe des Kafka-Clusters
Persistenz	Ausgleich heterogener Nachrichtenperformance: Bei schnellen Producern und langsamen Consumern werden die Daten in Kafka gespeichert.
Performance & Skalierbarkeit	Kafka ist auch in kleiner Konfiguration sehr performant. Bei Bedarf lässt sich die Performance des Clusters durch weitere Broker leicht skalieren.
Latenz	Die gewählte Lese- und Speicher-Strategie sorgt für geringe Latenzen beim Schreiben oder Abrufen der Nachrichten.
Sicherheit	Umsetzung zeitgemäßer Sicherheitsstandards zur Authentifizierung, Autorisierung und Nachrichten- sowie Transportverschlüsselung

NACHTEILE	
Komplexität während der Migration	Werden bestehende Nachrichtenaustauschformate zwischen Systemen hin zu Kafka migriert, entsteht während der Migrationsphase sowohl beim Betrieb als auch innerhalb der Applikationsarchitektur eine erhöhte Komplexität.

Tabelle 1: Vor- und Nachteile von Kafka auf einen Blick

onierungen von Topics, um den Konsumenten die Möglichkeit zu geben, nicht sofort die neueste Version eines Topics einsetzen zu müssen.

STRATEGIEN ZUR AUSFALLSICHERHEIT

Ein Ausfall des Kafka-Clusters führt ohne geeignete Strategie zur Ausfallsicherheit unweigerlich zum Ausfall des Gesamtsystems. Daher müssen Strategien geplant werden, die im Betrieb (Geo-Redundanz) beziehungsweise in den Applikationen (Nachrichtenaustausch ohne Apache Kafka) die Ausfallsicherheit gewährleisten oder zumindest abfangen. Gleichzeitig erhöhen diese Strategien jedoch auch die Systemkomplexität, so dass hier zwischen Systemkomplexität und Ausfallsicherheit abzuwägen ist.

FAZIT

Apache Kafka bietet eine beliebig skalierbare, schnelle und für hohen Datendurchsatz ausgelegte Messaging-Plattform, bei der die Daten konfigurierbar persistiert werden. Das System ist auch bei hoher Belastung resilient und kann mit kurzen Latenzen aufwarten. Dabei werden die Daten und Zugriffe nach modernen Standards abgesichert.

Der Einsatz von Kafka kann in vielen Szenarien sinnvoll sein und zum Kernelement einer zukunfts- und leistungsfähigen Messaging-Infrastruktur werden.

Aufgrund der Komplexität der Ablösung und der Kritikalität innerhalb der Gesamtarchitektur muss eine Migration von einem bestehenden Messaging-System in ein auf Kafka basierendes System durch eine ausreichend tiefgehende Analyse- und Designphase vorbereitet werden.

Solange die Gesamtarchitektur in der Migrationsphase mit mehreren Kommunikationskanälen funktionieren muss, erhöht sich mit dem Einsatz von Kafka die Komplexität aufgrund der Neukonzipierung der Daten- und Verarbeitungsmodelle. Hier empfiehlt sich eine stufenweise Ablösung bestehender Kommunikationsarchitekturen. ●

1 https://kafka.apache.org/ (abgerufen am 08.08.2019).
2 https://kafka.apache.org/powered-by (abgerufen am 08.08.2019).
3 https://en.wikipedia.org/wiki/Enterprise_messaging_system (abgerufen am 19.11.2019).
4 https://insidebigdata.com/2016/04/28/a-brief-history-of-kafka-linkedins-messaging-platform/ (abgerufen am 08.08.2019, eigene Übersetzung).
5 https://www.cloudkarafka.com/blog/2018-07-04-cloudkarafka_what_is_zookeeper.html (abgerufen am 09.10.2019).
6 https://cloud.ibm.com/docs/services/EventStreams?topic=eventstreams-partition_leadership&locale=de (abgerufen am 09.10.2019).
7 Beispiele für Message-Queues im Bereich der Open-Source-Lösungen sind ActiveMQ (https://activemq.apache.org, abgerufen am 09.10.2019) oder RabbitMQ (https://www.rabbitmq.com, abgerufen am 09.10.2019).