

# Kafka origins and use cases

---

# Apache Kafka Origins

- Started as a publish-subscribe based fault tolerant messaging system => **a continuous time-ordered log => a log = an append-only, totally-ordered sequence of records (ordered by time).**
- originated at LinkedIn (Jay Kreps) and later became an open source Apache project in 2011, then First-class Apache project in 2012;
- **started out as a way to make data ingest to Hadoop easier;**
- **2014: the original team that launched Kafka formed Confluent.io (main driver of Kafka and commercial distribution provider)**
- High throughput (100's of k's messages/s);
- Written in Scala and Java;
- fast, scalable and distributed by design;
- Initially was positioned as an alternative to Flume, but with an ambitious roadmap.

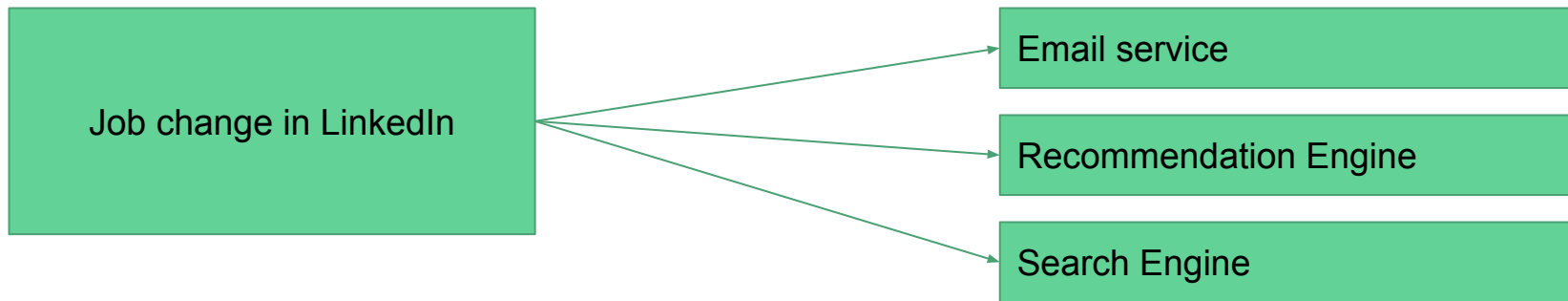
When there are multiple data sources and destinations involved, writing a separate data pipeline for each source and destination pairing quickly evolves to an unmaintainable mess. **Kafka helped LinkedIn standardize the data pipelines** and allowed getting data out of each system once and into each system once, significantly reducing the pipeline complexity and cost of operation.

# LinkedIn Realizations

**Events are more interesting/important than the states.** State can be very static, while events will capture all the time new information.

State = I work at Confluent

Event = I changed my job from Confluent to LinkedIn => you can derive more insights and more timely



# LinkedIn Realizations

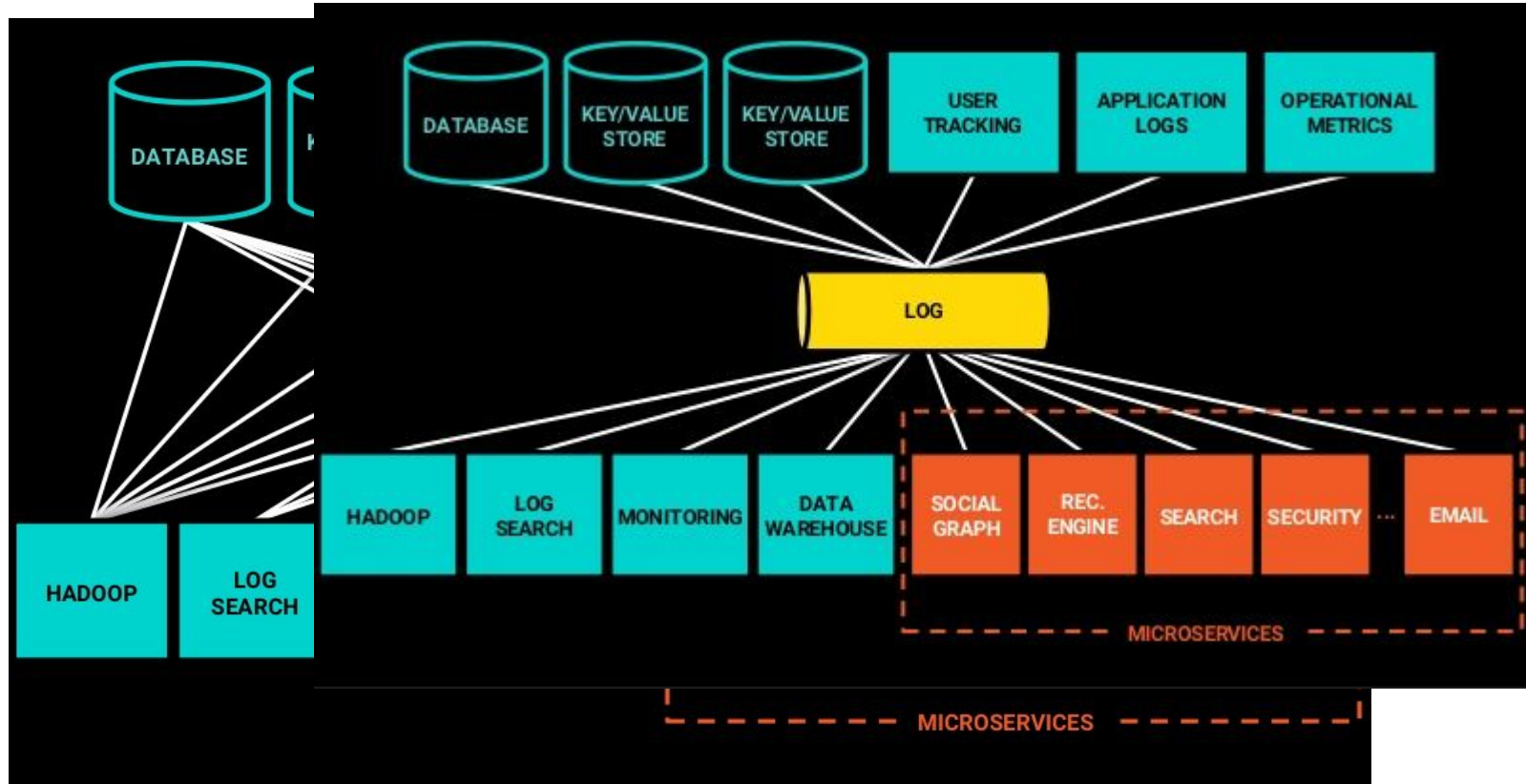
**Leverage all digitized events:**

0.1%	99.9%
Transactional data (e.g. job change, connections to others)	Non Transactional data (e.g. clicks from your mobile device in the app)

## **Database centric architecture - mismatched**

- Databases are build for states not events, but a lot of applications want the events not the states
- Databases are mostly for transactional data not necessarily for non transactional data

# LinkedIn Architecture



# Kafka most popular generic use cases

## Messaging

- Messaging bus
- RabbitMQ alternative, but distributed
- Kafka for its low end-to-end latency and strong durability guarantees.

## Website Activity Tracking

- Original use case: **rebuild a user activity tracking pipeline as a set of real-time publish-subscribe feeds. This means site activity (page views, searches, or other actions users may take) is published to central topics with one topic per activity type. These feeds are available for subscription for a range of use cases including real-time processing, real-time monitoring, and loading into Hadoop or offline data warehousing systems for offline processing and reporting.**

## Log aggregation

- collects physical log files off servers and puts them in a central place (a file server or HDFS perhaps) for processing.

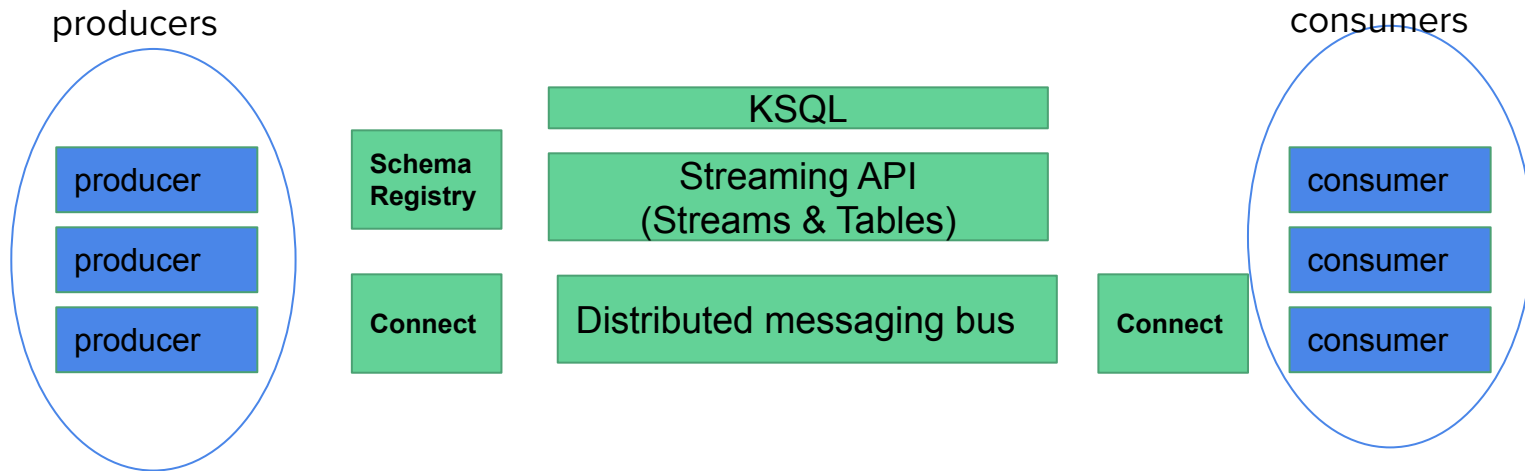
## Stream processing

- a processing pipeline for recommending news articles might crawl article content from RSS feeds and publish it to an "articles" topic; further processing might normalize or de-duplicate this content and published the cleansed article content to a new topic; a final processing stage might attempt to recommend this content to users.

## Event Sourcing

- changes are logged as a time-ordered sequence of records. Kafka's support for very large stored log data makes it an excellent backend for an application built in this style.

# Apache Kafka - evolved purpose, evolved use cases



# Apache Kafka Releases

**Nov 2012 - 0.7.0 (Incubating)**

**Nov 2015 - 0.9.0**

**Oct 2016 - 0.10.1.0**

- 0.10.1.1 (Oct 2016)

**Jun 2017 - 0.11.0.0**

- 0.11.0.3 (July 2018)

**Oct 2017 - 1.0.0**

- Release Plan 1.0.1

- 1.0.2 (July 2018)

.....

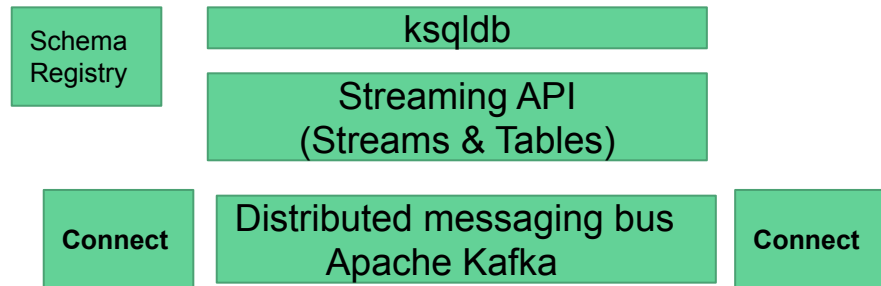
**Oct 2019 - 2.4.0**

**June 2020 - 2.6.0**

**November 2020 - 2.7.0**

**April 2021 - 2.8.0 (KIP 500 - no Zookeeper testing)**

**July 2021 - 3.0 (planned date)**





# Confluent Open Source

A developer-optimized distribution of Apache Kafka®. 100% open source.

- Adds schema management, connectors, clients, and the highest level of testing in the industry on top of Apache Kafka;
- Confluent CLI;
- Connectors: HDFS sink for HDFS and Hive, JDBC, Elasticsearch, Amazon S3;
- clients that allow your Kafka cluster to talk to applications written in Java, C/C++, .NET and Python;
- Schema Registry (Avro format)
- REST Proxy
- KSQL + Kafka Streams

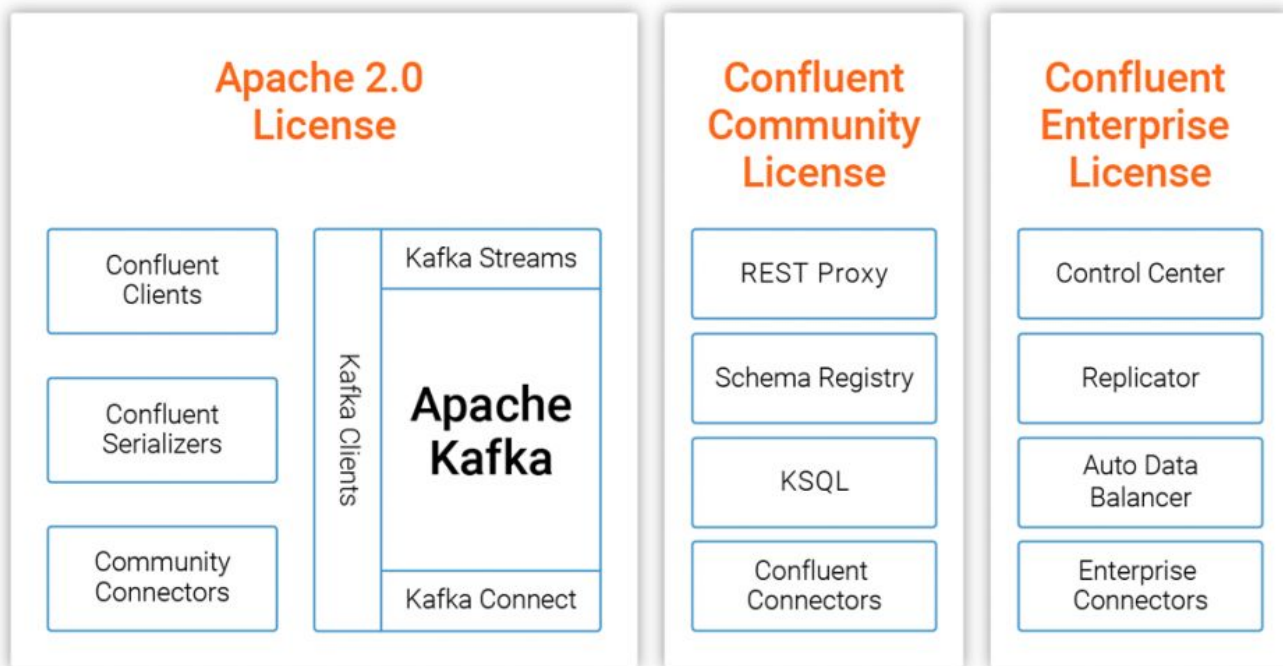
# Confluent Enterprise

- **Confluent Control Center** (monitoring and management system for both developers and operations teams.)
- **Confluent Replicator** (Multi-datacenter replication for disaster recovery and distributed data pipelines)
- **Auto Data Balancer** (Rack-aware rebalance algorithm that optimizes for disk space utilization. Partition movements are executed with minimal impact to your traffic.)
- **JMS connectors** (Packaged connectors to easily integrate data from any JMS broker, including IBM MQ and Apache ActiveMQ.)
- All the components from Open Source version for use in production

# Commercial licensing

---

# Commercial licensing



Source: <https://www.confluent.io/blog/license-changes-confluent-platform/>

# Possible Installations

---

# Installations/Distributions

- Apache Kafka (premises, cloud,...)
- Confluent Platform (premises, cloud,...)
- Cloudera/Hortonworks/ installations have already Kafka included and integrated in their eco-system
  - If you use it then you must check for versions supported
  - Cloudera Kafka source (0.11.0)
  - Hortonworks Kafka source
- Stratio Kafka source for ubuntu
- IBM Message Hub -Kafka as-a-service in a IBM's Bluemix PaaS
- Strimzi (<http://strimzi.io/>) - Apache Kafka Operator for Kubernetes and Openshift.
- TIBCO Messaging - Apache Kafka Distribution

# Use cases

---

# Who uses Kafka

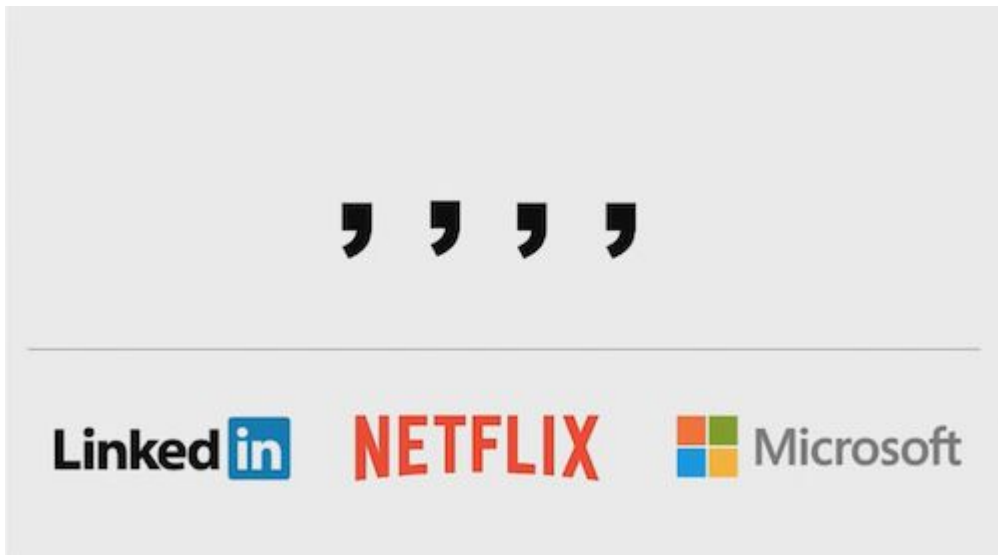


Source: <https://www.youtube.com/watch?v=XMXCZSJR1iM>



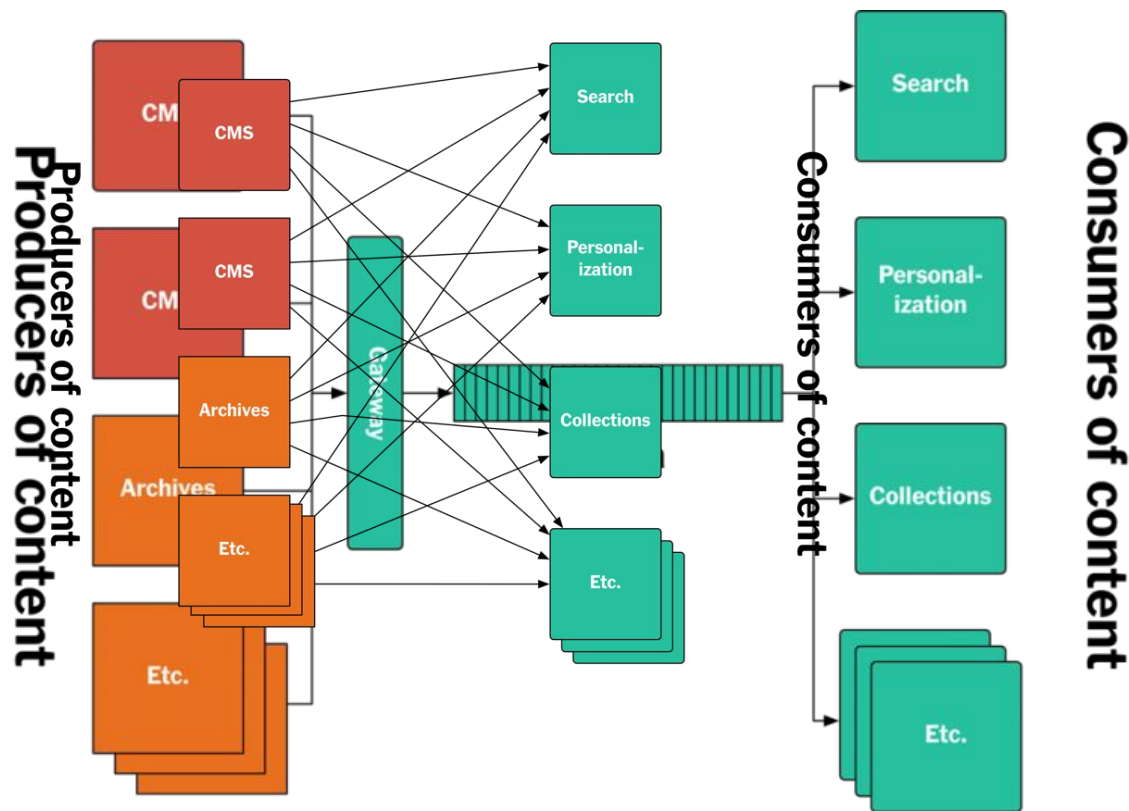
# Who uses Kafka: the 4 comma club

(companies processing over 1 trillion messages / day)



Source: <https://www.youtube.com/watch?v=XMXCZSJr1iM>

# Publishing Pipeline with Apache Kafka @ NY Times



“We need the log to retain all events forever, otherwise it is not possible to recreate a data store from scratch.

We need log consumption to be ordered. If events with causal relationships are processed out of order, the result will be wrong.”

Source:

<https://www.confluent.io/blog/publishing-apache-kafka-new-york-times/>

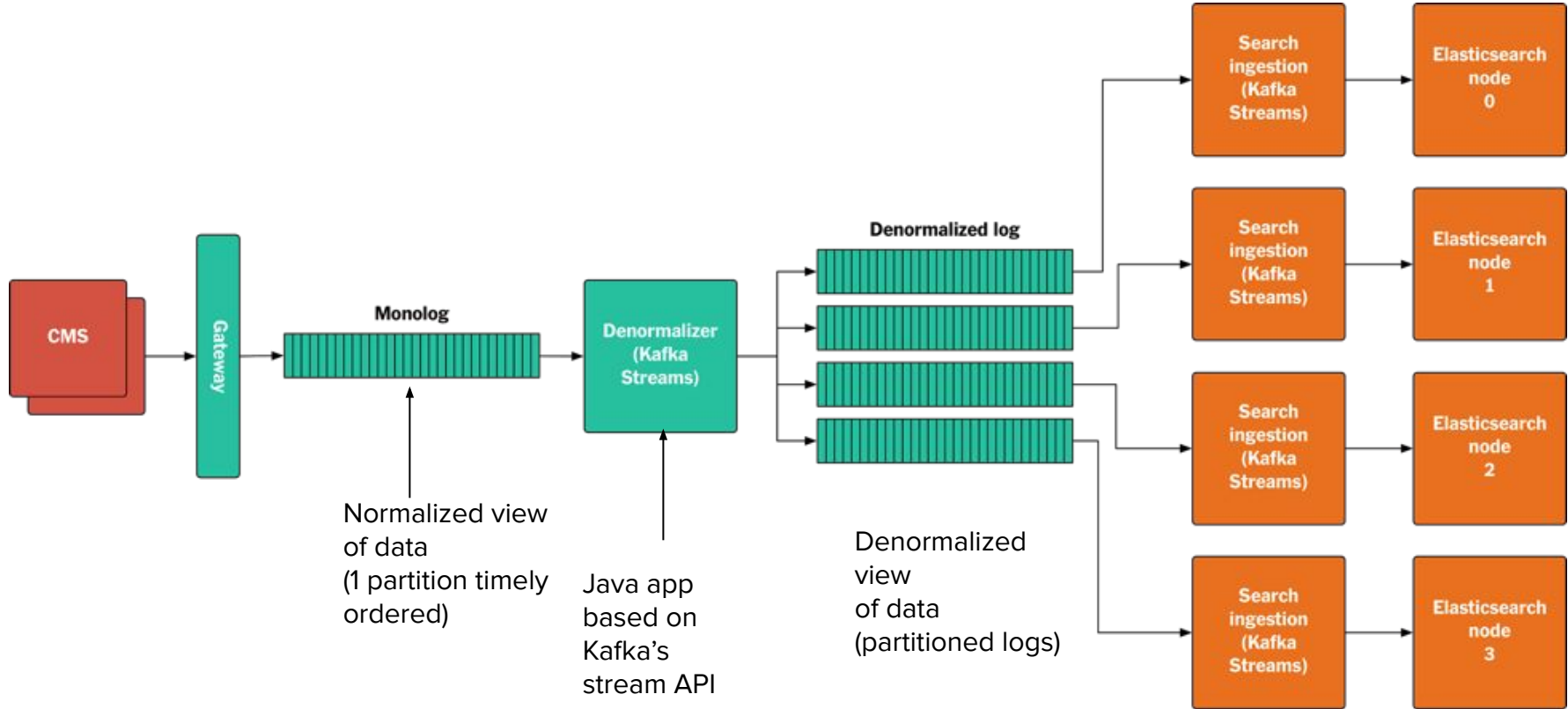
## Publishing Pipeline with Apache Kafka @ NY Times - requirements

- => A service that provides **live content for the web site and the native applications**, needs to make assets available immediately after they are published, **but it only ever needs the latest version of each asset**.
- => Different **services that provide lists of content, some are manually curated, some are query-based**. For the query-based lists, whenever an asset is published that matches the query, requests for that list need to include the new asset. We also have to support changes to the query itself, and the creation of new lists, which requires accessing previously published content to (re)generate the lists.
- => We have an **Elasticsearch cluster powering site search**. Here the latency requirements are less severe — if it takes a minute or two after an asset is published before it can be found by a search it is usually not a big deal. However, the search engine needs easy access to previously published content, since we need to reindex everything whenever the Elasticsearch schema definition changes, or when we alter the search ingestion pipeline.
- => We have **personalization systems** that only care about recent content, but that need to reprocess this content whenever the personalization algorithms change.

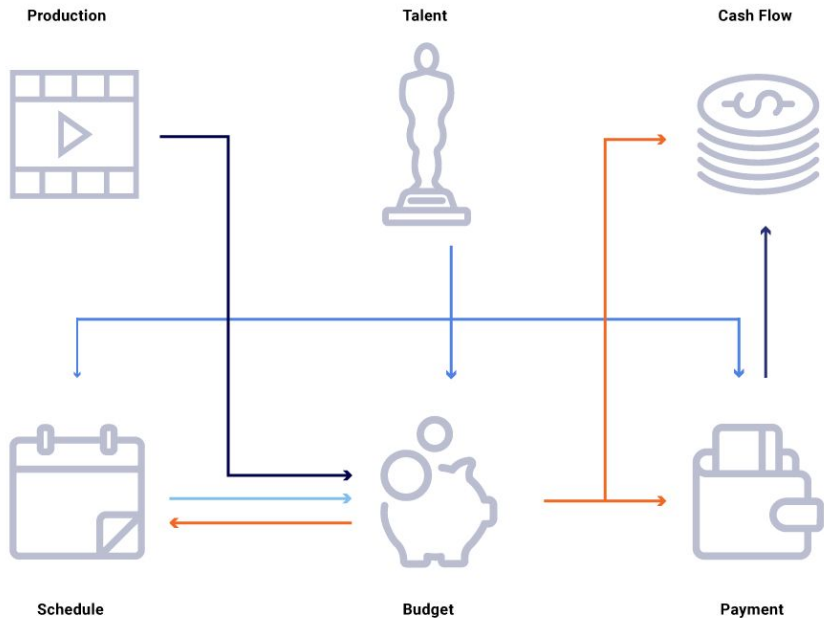
## Publishing Pipeline with Apache Kafka @ NY Times - solution

- **Not a solution based on database:** a database typically stores the result of some event, while the log stores the event itself — the log therefore becomes an ordered representation of all events that happened in the system.
- **Using a log, you can then create any number of custom data stores.** These stores become materialized views of the log — they contain derived, not original, content. If you want to change the schema in such a data store, you can just create a new one, have it consume the log from the beginning until it catches up, and then just throw away the old one.
- **a log-based architecture simplifies accessing streams of content.** In a traditional data store, accessing a full dump (i.e., as a snapshot) and accessing “live” data (i.e., as a feed) are distinct ways of operating. In our solution you start consuming the log at some specific offset — this can be the beginning, the end, or any point in-between — and then just keep going. This means that **if you want to recreate a data store, you simply start consuming the log at the beginning of time.** At some point you will catch up with live traffic, but this is transparent to the consumer of the log.

# Publishing Pipeline with Apache Kafka @ NY Times - solution



# Kafka @ Netflix



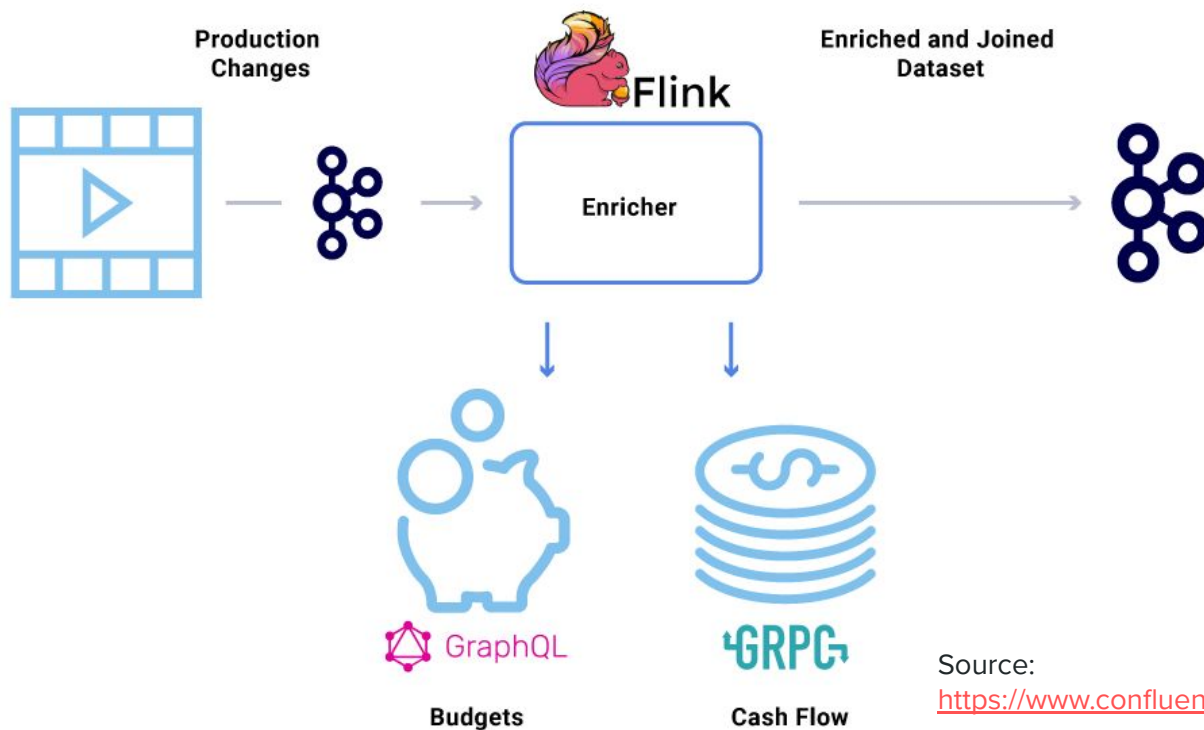
Change of a show date - chain of events

**“Events are much more than triggers and updates. They become the immutable stream from which we can reconstruct the entire system state.”**

Source:

<https://www.confluent.io/blog/how-kafka-is-used-by-netflix/>

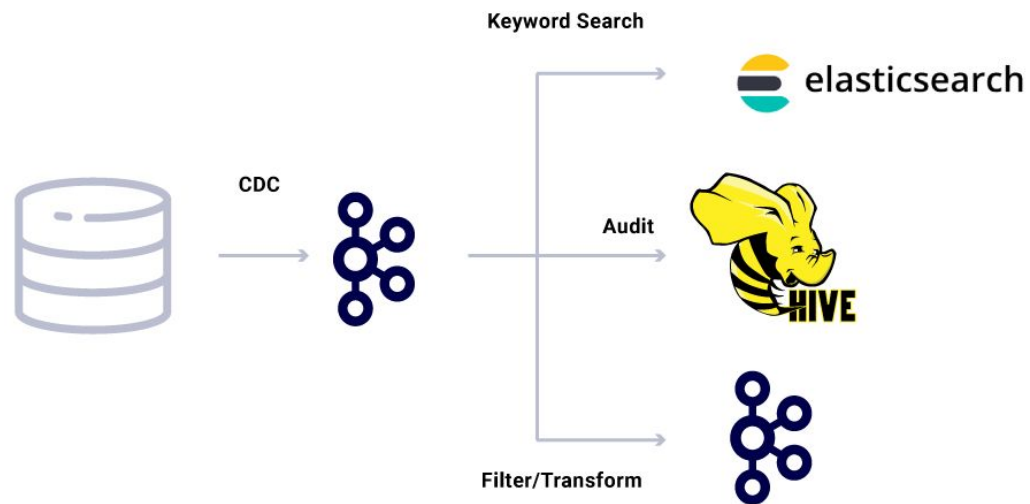
# Kafka @ Netflix



Source:

<https://www.confluent.io/blog/how-kafka-is-used-by-netflix/>

# Kafka @ Netflix



Source:

<https://www.confluent.io/blog/how-kafka-is-used-by-netflix/>



# Kafka @ New Relic - real time events processing

## **Kafka as a change log / Change Data Capture**

- A changelog is data contained on a topic that is intended to be read through from start to finish, most likely when an application starts up. The goal is to rebuild the state by replaying history. The service may continue to listen for updates.

## **Kafka as a state cache**

- “We use Kafka topics to store and reload state that has been snapshotted. This is what I call, for lack of a better term, a “Durable Cache,” which means using Kafka topics to store and reload snapshotted state.”

The production Kafka cluster at New Relic processes more than 15 million messages per second for an aggregate data rate approaching 1 Tbps.

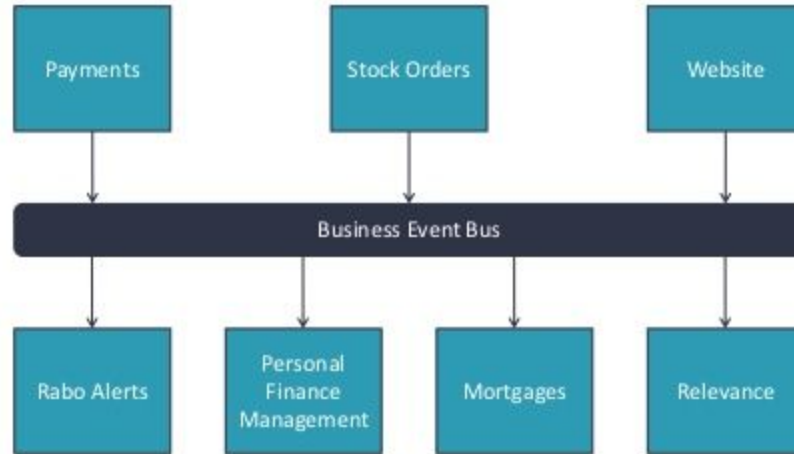
# Rabobank Alerts based on Kafka Streams

Rabobank is based in the Netherlands with over 900 locations worldwide, 48,000 employees, and €681B in assets.

Rabo Alerts is a service that allows Rabobank customers to be alerted whenever interesting financial events occur. A simple example of an event is **when a certain amount was debited from or credited to your account, but more complex events also exist**. Alerts can be configured by the customer based on their preferences and sent via three channels: email, SMS and mobile push notifications. It's noteworthy to mention that Rabo Alerts is not a new or piloted service. It has been in production for over ten years and is available to millions of account holders only its implementations had several shortcomings (e.g. old implementation took 5 minutes up to 4-5 hours to deliver alerts to customers) and its architecture required a redesign.

# Rabobank - Business Event Bus

“Sharing events” caught on quickly



High Confidentiality, Medium Integrity, High Availability




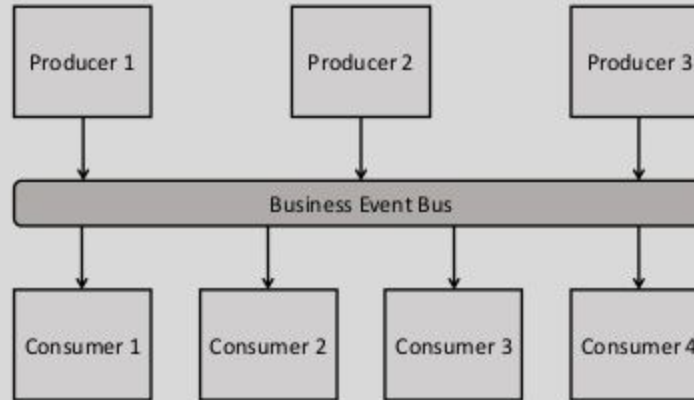
Source: <https://www.slideshare.net/ConfluentInc/financial-event-sourcing-at-enterprise-scale>

# Rabobank Business Event Bus

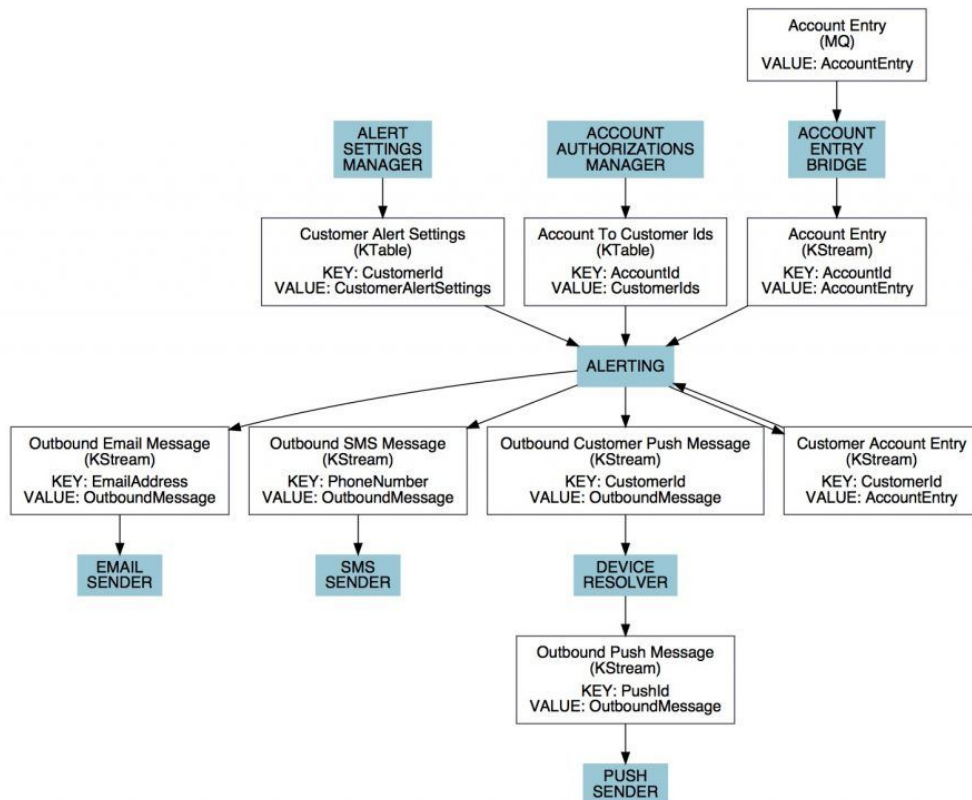
Before B

## Using Business Event Bus

Dizzit & 



# Rabobank alerts



**The whole round trip from payment order confirmation to the alert arriving on a mobile device takes only one to two seconds, more often being around one rather than two.**

**The whole alerting chain—from an account entry posted on Kafka up to senders pushing out messages to customers—is typically executed within 120 milliseconds.**

Link source:

<https://www.confluent.io/blog/real-time-financial-alerts-rabobank-apache-kafkas-streams-api/>

# Walmart - near real time search platform & scalable data processing systems.



“Our Kafka implementation is processing billions of updates per day mostly driven by **Pricing & Inventory adjustments**. We built **operational tools for tracking flows, SLA metrics, message send/receive latencies for producers and consumers, alerting on backlogs, latency and throughput**.

The nice thing of capturing all the updates in Kafka is that we can process the same data for Reprocessing of the catalog, sharing data between environments, A/B Testing, Analytics & Data warehouse.