Apache Kafka
Skalierbare Nachrichtenverarbeitung und mehr!

Guido Schmutz

JAVA FORUM
stuttgart

Trivadis makes IT easier.

BASEL • BERN • BRUGG • DÜSSELDORF • FRANKFURT A.M. • FREIBURG I.BR. • GENEVA
HAMBURG • COPENHAGEN • LAUSANNE • MUNICH • STUTTGART • VIENNA • ZURICH
Guido Schmutz

Working for Trivadis for more than 19 years
Oracle ACE Director for Fusion Middleware and SOA
Co-Author of different books
Consultant, Trainer Software Architect for Java, Oracle, SOA and Big Data / Fast Data
Member of Trivadis Architecture Board
Technology Manager @ Trivadis

More than 25 years of software development experience

Contact: guido.schmutz@trivadis.com
Blog: http://guidoschmutz.wordpress.com
Twitter: gschmutz
Unser Unternehmen.

Trivadis ist führend bei der IT-Beratung, der Systemintegration, dem Solution Engineering und der Erbringung von IT-Services mit Fokussierung auf Oracle- und -Tech-Fertigkeiten in der Schweiz, Deutschland, Österreich und Dänemark. Trivadis erbringt ihre Leistungen aus den strategischen Geschäftsfeldern:

Trivadis Services übernimmt den korrespondierenden Betrieb Ihrer IT Systeme.
Mit über 600 IT- und Fachexperten bei Ihnen vor Ort.

- 14 Trivadis Niederlassungen mit über 600 Mitarbeitenden.
- Über 200 Service Level Agreements.
- Mehr als 4'000 Trainingsteilnehmer.
- Forschungs- und Entwicklungsbudget: CHF 5.0 Mio.
- Finanziell unabhängig und nachhaltig profitabel.
- Erfahrung aus mehr als 1'900 Projekten pro Jahr bei über 800 Kunden.
Agenda

1. Introduction & Motivation
2. Kafka Core
3. Kafka Connect
4. Kafka Streams
5. Kafka and ”Big Data” / ”Fast Data” Ecosystem
6. Confluent Data Platform
7. Kafka in Architecture
8. Summary
Introduction & Motivation
A little story of a “real-life” customer situation

Traditional system interact with its clients and does its work
Implemented using legacy technologies (i.e. PL/SQL)

New requirement:
• Offer notification service to notify customer when goods are shipped
• Subscription and inform over different channels
• Existing technology doesn’t fit
A little story of a “real-life” customer situation

Events are “owned” by traditional application (as well as the channels they are transported over)

Implement notification as a new Java-based application/system

But we need the events ! => so let’s integrate
A little story of a “real-life” customer situation

integrate in order to get the information! Oracle Service Bus was already there

Rule Engine implemented in Java and invoked from OSB message flow
Notification system informed via queue
Higher Latency introduced (good enough in this case)
A little story of a “real-life” customer situation

Treat events as first-class citizens

Events belong to the “enterprise” and not an individual system => Catalog of Events similar to Catalog of Services/APIs !!

Event (stream) processing can be introduced and by that latency reduced!
Treat Events as Events and share them!
Treat Events as Events, share and make use of them!
Kafka Stream Data Platform

Source: Confluent
Kafka Core
Apache Kafka - Overview

Distributed publish-subscribe messaging system

Designed for processing of real time activity stream data (logs, metrics collections, social media streams, …)

Initially developed at LinkedIn, now part of Apache

Does not use JMS API and standards

Kafka maintains feeds of messages in topics
Apache Kafka - Motivation

LinkedIn’s motivation for Kafka was:

• “A unified platform for handling all the real-time data feeds a large company might have.”

Must haves

• High throughput to support **high volume event feeds**.
• Support real-time processing of these feeds to create **new, derived feeds**.
• Support large data backlogs to handle periodic ingestion from **offline systems**.
• Support low-latency delivery to handle more traditional **messaging use cases**.
• Guarantee **fault-tolerance** in the presence of machine failures.
Kafka High Level Architecture

The who is who

- **Producers** write data to **brokers**.
- **Consumers** read data from **brokers**.
- All this is distributed.

The data

- Data is stored in **topics**.
- **Topics** are split into **partitions**, which are **replicated**.
Apache Kafka - Architecture

Kafka Broker

Movement Topic
1 2 3 4 5 6

Movement Processor

Engine-Metrics Topic
1 2 3 4 5 6

Engine Processor

Truck
Apache Kafka - Architecture

- Write Ahead Log / Commit Log
- Producers always append to tail
- think append to file
# Durability Guarantees

Producer can configure acknowledgements

<table>
<thead>
<tr>
<th>Value</th>
<th>Impact</th>
<th>Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>• Producer doesn’t wait for leader</td>
<td>weak</td>
</tr>
</tbody>
</table>
| 1 (default) | • Producer waits for leader  
• Leader sends ack when message written to log  
• No wait for followers | medium     |
| all    | • Producer waits for leader  
• Leader sends ack when all In-Sync Replica have acknowledged | strong     |
Apache Kafka - Partition offsets

Offset: messages in the partitions are each assigned a unique (per partition) and sequential id called the offset

- Consumers track their pointers via \(\text{offset, partition, topic}\) tuples
**Data Retention – 3 options**

1. **Never**

2. **Time based (TTL)**
   
   log.retention.{ms | minutes | hours}

3. **Size based**
   
   log.retention.bytes

4. **Log compaction based (entries with same key are removed)**
   
   kafka-topics.sh --zookeeper localhost:2181 \  
   --create --topic customers \  
   --replication-factor 1 --partitions 1 \  
   --config cleanup.policy=compact
Apache Kafka – Some numbers

Kafka at LinkedIn => over 1800+ broker machines / 79K+ Topics
https://engineering.linkedin.com/kafka/running-kafka-scale

<table>
<thead>
<tr>
<th>1.3 Trillion messages per day</th>
<th>330 Terabytes in/day</th>
<th>1.2 Petabytes out/day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak load for a single cluster</td>
<td>2 million messages/sec</td>
<td>4.7 Gigabits/sec inbound</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 Gigabits/sec outbound</td>
</tr>
</tbody>
</table>

Kafka Performance at our own infrastructure => 6 brokers (VM) / 1 cluster

- 445'622 messages/second
- 31 MB / second
- 3.0405 ms average latency between producer / consumer
Kafka Connect Architecture

Data Sources

- SQL Database
- Connector
- Connector
- Connector

Kafka Connect

Apache Kafka

Data Sinks

- SQL Database
- Connector
- Connector
- Connector

...
## Kafka Connector Hub – Certified Connectors

<table>
<thead>
<tr>
<th>CONNECTOR</th>
<th>TAGS</th>
<th>DEVELOPER</th>
<th>SUPPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS (Sink)</td>
<td>HDFS, Hadoop, Hive</td>
<td>Confluent</td>
<td>Confluent</td>
</tr>
<tr>
<td>JDBC (Source)</td>
<td>JDBC, MySQL</td>
<td>Confluent</td>
<td>Confluent</td>
</tr>
<tr>
<td>Attunity (Sink)</td>
<td>CDC, Oracle</td>
<td>Attunity</td>
<td>Attunity</td>
</tr>
<tr>
<td>Couchbase (Source)</td>
<td>Couchbase, NoSQL</td>
<td>Couchbase</td>
<td>Couchbase</td>
</tr>
<tr>
<td>JustOne (Sink)</td>
<td>Postgress</td>
<td>JustOne</td>
<td>JustOne</td>
</tr>
<tr>
<td>Striim (Source)</td>
<td>CDC, Oracle, MS SQLServer</td>
<td>Striim</td>
<td>Striim</td>
</tr>
<tr>
<td>Syncsort DMX (Source)</td>
<td>DB2, IMS, VSAM, CICS</td>
<td>Syncsort</td>
<td>Syncsort</td>
</tr>
<tr>
<td>Syncsort DMX (Sink)</td>
<td>DB2, IMS, VSAM, CICS</td>
<td>Syncsort</td>
<td>Syncsort</td>
</tr>
<tr>
<td>Vertica (Source)</td>
<td>Vertica</td>
<td>HP Enterprise</td>
<td>HP Enterprise</td>
</tr>
<tr>
<td>Vertica (Sink)</td>
<td>Vertica</td>
<td>HP Enterprise</td>
<td>HP Enterprise</td>
</tr>
</tbody>
</table>

Source: [http://www.confluent.io/product/connectors](http://www.confluent.io/product/connectors)
# Kafka Connector Hub – Additional Connectors

<table>
<thead>
<tr>
<th>CONNECTOR</th>
<th>TAGS</th>
<th>DEVELOPER</th>
<th>SUPPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ignite (Source)</td>
<td>File System</td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>Apache Ignite (Sink)</td>
<td>File System</td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>Bloomberg Ticker (Source)</td>
<td></td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>Cassandra (Source)</td>
<td>Cassandra, Datastax</td>
<td>Community</td>
<td>Community 1 2</td>
</tr>
<tr>
<td>Cassandra (Sink)</td>
<td>Cassandra, Datastax</td>
<td>Community</td>
<td>Community 1 2</td>
</tr>
<tr>
<td>Elastic Search (Sink)</td>
<td>search, elastic, log, analytics</td>
<td>Community</td>
<td>Community 1 2 3</td>
</tr>
<tr>
<td>HBase Sink</td>
<td>HBase, NoSQL</td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>Kudu (Sink)</td>
<td>Kudu</td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>Mixpanel (Source)</td>
<td>analytics</td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>MongoDB (Source)</td>
<td>Mongo, MongoDB, NoSQL</td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>MQTT (Source)</td>
<td>MQTT, messaging</td>
<td>Community</td>
<td>Community</td>
</tr>
<tr>
<td>MySQL CDC-Debezium (Source)</td>
<td>MySQL, CDC, Oracle</td>
<td>Community</td>
<td>Community</td>
</tr>
</tbody>
</table>

Source: [http://www.confluent.io/product/connectors](http://www.confluent.io/product/connectors)
Kafka Streams
**Kafka Streams**

- Designed as a **simple and lightweight library in Apache Kafka**
- no external dependencies on systems other than Apache Kafka
- Leverages **Kafka as its internal messaging layer**
- agnostic to resource management and configuration tools
- Supports **fault-tolerant local state**
- Event-at-a-time processing (not microbatch) with millisecond latency
- Windowing with out-of-order data using a Google DataFlow-like model
Kafka Streams Architecture
Kafka and "Big Data" / "Fast Data" Ecosystem
Kafka and the Big Data / Fast Data ecosystem

Kafka integrates with many popular products / frameworks

- Apache Spark Streaming
- Apache Flink
- Apache Storm

- Apache NiFi
- Streamsets
- Apache Flume
- Oracle Stream Analytics
- Oracle Service Bus
- Oracle GoldenGate
- Spring Integration Kafka Support

Storm built-in Kafka Spout to consume events from Kafka
Confluent Platform
Confluent Data Platform 3.0
Kafka in Architecture
Customer Event Hub – taking Velocity into account
Customer Event Hub – mapping of technologies
Summary

- Kafka can scale to millions of messages per second, and more
- Easy to start with for a PoC
- A bit more to invest to setup production environment
- Monitoring is key
- Vibrant community and ecosystem
- Fast pace technology
- Confluent provides Kafka Distribution
Guido Schmutz
Technology Manager

guido.schmutz@trivadis.com