3 ways to make your life easier with Apache Kafka

Jakub Scholz

DevConf.CZ, Brno, 25.1.2019
About me

Jakub Scholz

- Principal Software Engineer with the Red Hat AMQ Streams engineering team
- Long-term messaging specialist
- Former messaging team lead at Deutsche Börse

@scholzj

https://www.linkedin.com/in/scholzj/
What is Apache Kafka

A publish/subscribe messaging system

A streaming data platform

A distributed, horizontally-scalable, fault-tolerant, commit log
What is Apache Kafka

When I say Kafka, you say ... **scalable**!

But not everyone is LinkedIn or Netflix!

Luckily Kafka can do more than just scale.
What is Apache Kafka

- Developed at Linkedin back in 2010, open sourced in 2011
- Designed to be fast, scalable, durable and available
- Distributed by nature
- Data partitioning (sharding)
- High throughput / low latency
- Ability to handle huge number of consumers
Apache Kafka concepts

- Messages are sent to and received from a topic
  - Topics are split into one or more partitions (aka shards)
  - All actual work is done on partition level, topic is just a virtual object
- Each message is written only into a one selected partition
  - Partitioning is usually done based on the message key
  - Message ordering within the partition is fixed
- Retention
  - Based on size / message age
  - Compacted based on message key
- Replication
  - Each partition can exist in one or more Backup copies to achieve high availability in case of node failures
Apache Kafka concepts

Topics & Partitions
Apache Kafka concepts

Topics & Partitions

<table>
<thead>
<tr>
<th>Topic</th>
<th>Partition 0</th>
<th>Partition 1</th>
<th>Partition 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M M M M M M M</td>
<td>M M M M M M</td>
<td>M M M M M M</td>
</tr>
<tr>
<td>Offset</td>
<td>Offset 0</td>
<td>Offset 0</td>
<td>Offset 0</td>
</tr>
<tr>
<td></td>
<td>Offset 1</td>
<td>Offset 1</td>
<td>Offset 1</td>
</tr>
<tr>
<td></td>
<td>Offset 2</td>
<td>Offset 2</td>
<td>Offset 2</td>
</tr>
<tr>
<td></td>
<td>Offset 3</td>
<td>Offset 3</td>
<td>Offset 3</td>
</tr>
<tr>
<td></td>
<td>Offset 4</td>
<td>Offset 4</td>
<td>Offset 4</td>
</tr>
<tr>
<td></td>
<td>Offset 5</td>
<td>Offset 5</td>
<td>Offset 5</td>
</tr>
<tr>
<td></td>
<td>Offset 6</td>
<td>Offset 6</td>
<td>Offset 6</td>
</tr>
<tr>
<td></td>
<td>Offset 7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Consumer

Older

Newer
Apache Kafka concepts

High Availability

Leaders and followers spread across the cluster.
Clients always connect only to the leaders.
Apache Kafka concepts

High Availability

If broker with leader partition goes down, new partition leader is elected on different node.
Apache Kafka concepts

Consumer Groups

- Consumer Group
  o Grouping multiple consumers
  o Each consumer reads from a “unique” subset of partition → max consumers = num partitions
  o They are “competing” consumers on the topic, each message delivered to one consumer
  o Messages with same “key” delivered to same consumer
- More consumer groups
  o Allows publish/subscribe
  o Same messages delivered to different consumers in different consumer groups
Apache Kafka concepts

Consumer Groups
Apache Kafka concepts

Consumer Groups

- Topic
  - Partition 0
  - Partition 1
  - Partition 2
  - Partition 3

- Consumer Group
  - Consumer
  - Consumer
  - Consumer
  - Consumer
Message replay
Message Replay

How does it work?
Message Replay

Use cases

- **Loading, transforming or enriching messages**
  - Replay the data in case of data loss or bug in processing
- **Training and testing of algorithms and ML models**
  - Use different models and algorithms on the same data sets again and again
- **Event Sourcing**
  - Storing of state as a series of events / state changes
  - Events can be replayed to reconstruct the state
Adding consumers
Adding consumers

How does it work?
Adding Consumers

Use cases

- **Integration**
  - Have multiple independent consumers without any complicated setup
  - Each of them can consume on their own pace

- **A/B testing**
  - Run multiple implementation in parallel to compare / verify results

- **Feeding production data to test systems**
  - Easily connect your test / acceptance environments to production to run them using production data
Change Data Capture
Change Data Capture
Without CDC

- Multiple connections to different services
- What do you do when one of them is not available?
- How do you make sure no data are lost?
Change Data Capture
With CDC

- Decouples the messaging layer form your application
- Make it easier to handle disruptions
- Demo is using Debezium ([https://debezium.io/](https://debezium.io/))
Conclusion
Demo source codes

- The demos from this talk can be found here:

- The demos in this talk were run on OpenShift using a project called **Strimzi**
  - Interested to hear more?
  - **Tomorrow (26th January)**
  - **10:30am**
  - **Room G202 (same room as this talk)**
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHatNews
# Apache Kafka vs. traditional messaging

<table>
<thead>
<tr>
<th></th>
<th>Apache Kafka</th>
<th>Traditional messaging</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td>“Dumb broker, smart clients”</td>
<td>“Smart broker, dumb clients”</td>
</tr>
<tr>
<td><strong>Durability</strong></td>
<td>Durable storage</td>
<td>Volatile or durable storage</td>
</tr>
<tr>
<td><strong>Storage duration</strong></td>
<td>Potential long-term storage of messages</td>
<td>Temporary storage of messages</td>
</tr>
<tr>
<td><strong>Message retention</strong></td>
<td>Retained until expired or compacted</td>
<td>Retained until consumed</td>
</tr>
<tr>
<td><strong>Consumer state</strong></td>
<td>Client managed (can be stored in broker)</td>
<td>Broker managed</td>
</tr>
<tr>
<td><strong>Selectors</strong></td>
<td>No</td>
<td>Yes, per consumer</td>
</tr>
<tr>
<td><strong>Stream replay</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>High-availability</strong></td>
<td>Replication</td>
<td>Replication</td>
</tr>
<tr>
<td><strong>Protocols</strong></td>
<td>Kafka protocol</td>
<td>Standardized protocols (AMQP, MQTT)</td>
</tr>
<tr>
<td><strong>Delivery guarantees</strong></td>
<td>Best-effort or guaranteed</td>
<td>Best-effort or guaranteed</td>
</tr>
</tbody>
</table>