Introduction

How do they compare against each other and which one to use?
Background: Pub/Sub Systems

Distributed interaction paradigm

1: Decoupling the publishers and subscribers. Dimensions:

- Entity decoupling
- Time decoupling
- Synchronization decoupling

Figure 2. Pub/Sub system scheme [3]
2: **Routing logic**: Decides if and where a packet that is coming from a producer will end up at a consumer. Types:

- **Topic-based subscription**
- **Content-based subscription**

...Setting standards for the assistance dog industry since 1987...

Figure 2. Pub/Sub system scheme [3]
Background: Quality-of-Service Guarantees

- Correctness
- Efficiency
- Transactions
- Scalability
- Availability

Quality of Service Guarantees

- Delivery Guarantees
  - at most once
  - at least once
  - exactly once
  - no ordering
  - partitioned ordering
  - global order

- Ordering Guarantees
  - no ordering
  - global order

- Latency
  - time

- Throughput
  - BW

Best Performance

Loss might occur
duplication might occur
Description

Kafka 0.10 vs RabbitMQ 3.5
High-level Description: Apache Kafka

- Built at LinkedIn to handle:
  - Billions of messages
  - Consumers of the same stream, reading at different speeds
- Scalable system
- Distributed commit log (records)

Figure 3. Records in Kafka[4]
High-level Description: Apache Kafka

Figure 4. Apache Kafka architecture[1]
High-level Description: RabbitMQ

- Scalable implementation of the AMQP (Advanced Message Queuing Protocol):
  - Interoperability, performance, scalability and reliability
  - Modular approach: Exchanges and queues

- RabbitMQ:
  - Efficient acknowledgment mechanism for the publishers
  - Better defined transactional behavior
  - Better support for asynchronous batch transfer
  - Degree of coupling between producers and consumers (i.e. the flow control)

Figure 5. RabbitMQ architecture[5]
Qualitative Comparison

*Kafka 0.10 vs RabbitMQ 3.5*
# Qualitative comparison

<table>
<thead>
<tr>
<th></th>
<th>RabbitMQ</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time decoupling</strong></td>
<td>DRAM as long as possible</td>
<td>Designed with the various consumption rates</td>
</tr>
<tr>
<td><em>Buffer a large amount of messages</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Routing logic</strong></td>
<td>Topic based</td>
<td>Topic based</td>
</tr>
<tr>
<td><em>Different exchange types</em></td>
<td>Content based</td>
<td></td>
</tr>
<tr>
<td></td>
<td>API to create others</td>
<td></td>
</tr>
</tbody>
</table>
# Qualitative comparison

<table>
<thead>
<tr>
<th>Delivery Guarantees</th>
<th>RabbitMQ</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgment behavior</td>
<td>t3: ACK to producer, and producer can delete the message&lt;br&gt;t4: consumer gets the message&lt;br&gt;t5: the consumer sends an ACK, and the broker can delete the message</td>
<td>t5: has no way of understanding ownership moved to the consumer. It will keep the message until a configured timeout expires</td>
</tr>
</tbody>
</table>
Qualitative comparison

<table>
<thead>
<tr>
<th></th>
<th>RabbitMQ</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordering Guarantees</td>
<td>Flows</td>
<td>Partitions</td>
</tr>
<tr>
<td></td>
<td>Reorders retransmitted packets inside its queue</td>
<td>No inter-batch order</td>
</tr>
<tr>
<td>Availability</td>
<td>Replication</td>
<td>Replication</td>
</tr>
<tr>
<td></td>
<td>Mirrored queues ares not automatically created</td>
<td></td>
</tr>
</tbody>
</table>
## Qualitative comparison

<table>
<thead>
<tr>
<th>Transactions</th>
<th>RabbitMQ</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If any message fails, the producer gets the chance to republish these messages, and RabbitMQ will insert them in the queue in order</td>
<td>Does not support transactions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multicast</th>
<th>RabbitMQ</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same information to multiple destinations</td>
<td>Providing a dedicated queue per individual consume</td>
<td>Handled at the consumer side</td>
</tr>
</tbody>
</table>
## Qualitative comparison

<table>
<thead>
<tr>
<th>Dynamic Scaling</th>
<th>RabbitMQ</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding additional nodes to running clusters or removing a node from a cluster</td>
<td>Additional nodes will be able to become master for newly created queues</td>
<td>The user can decide to move existing partitions to the new node</td>
</tr>
<tr>
<td></td>
<td>Cannot be used to redistribute master queue assignments of existing queues</td>
<td>Not transparent for consumers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quantitative Comparison

Kafka 0.10.0.1 vs RabbitMQ 3.5.3
Quantitative Comparison

Measured in terms of:

- Latency
- Throughput

Dimensions:

- Delivery Guarantees
- Availability
Quantitative Comparison: Latency

At Most Once Mode

Serial Pipeline handling a packet

Latency results

Three Erlang Process

With and without replication:
- Mean: 1 - 4 ms
- Max: 2 - 17 ms

Latency results

At Least Once Mode

Latency results

Not really impacted

Kafka

Storage access latency

<table>
<thead>
<tr>
<th></th>
<th>50 percentile</th>
<th>99.9 percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>without replication</td>
<td>1 ms</td>
<td>15 ms</td>
</tr>
<tr>
<td>with replication</td>
<td>1 ms</td>
<td>30 ms</td>
</tr>
</tbody>
</table>

Increases in case of replication
Quantitative Comparison: Throughput

At Most Once Mode

Important factors

Record size

RabbitMQ

Kafka

Record size, topic count and partition count

At Least Once Mode

Results

Throughput drops by 50%
Express performance in packets per unit of time

Throughput decreases by 50% to 75%
Express performance in bytes per unit of time
Distinct Features

Kafka 0.10 vs RabbitMQ 3.5
Distinct features

**RabbitMQ**

- Standardized Protocol
- Multi-protocol
- Comprehensive Management and Monitoring Tools
- Multi-tenancy and Isolation
- Consumer Tracking
- Disk-less
- Publisher Flow Control
- Queue Size Limits
- Message TTL

**Kafka**

- Long Term Message Storage
- Message Replay
- Kafka Connect
- Log Compaction
Preferred Use Cases

Kafka 0.10 vs RabbitMQ 3.5
<table>
<thead>
<tr>
<th>Preferred Use Cases</th>
<th>RabbitMQ</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pub/Sub Messaging</td>
<td>![RabbitMQ]</td>
<td>![Kafka]</td>
</tr>
<tr>
<td>Request-Response Messaging</td>
<td>![RabbitMQ]</td>
<td>![Kafka]</td>
</tr>
<tr>
<td>Operational Metrics Tracking</td>
<td>![RabbitMQ]</td>
<td>![Kafka]</td>
</tr>
<tr>
<td>Underlying Layer for IoT Applications Platform</td>
<td>![RabbitMQ]</td>
<td>![Kafka]</td>
</tr>
<tr>
<td>Information-centric Networking</td>
<td>![RabbitMQ]</td>
<td>![Kafka]</td>
</tr>
</tbody>
</table>

- RabbitMQ, followed by Kafka
- Kafka, followed by RabbitMQ
- Working in parallel
- Pub/Sub Messaging
- Scalable Ingestion System
- Data-Layer Infrastructure
- Capturing Change Feeds
- Stream Processing
Conclusions

Kafka 0.10 vs RabbitMQ 3.5
References


[2] https://trends.google.com/trends/explore?date=today%205-y&q=%2Fm%2F0bhc0tk,%2Fm%2F0zmynvd, 10/12/2019


[4] https://www.youtube.com/watch?v=ElilYxUOjOQ, 10/12/2019

Thank you!

Any Question?