Cloud Computing and Fundamentals | COL733
Benchmarking: RabbitMQ and Kafka

Dipen Kumar (2018CS50098)
Manoj Kumar (2018CS50411)
Sanjali Agarwal (2018CS50419)

Project Proposal:

Which systems are you picking? Why are they competitors?

In this new age of evolving architectures, events have always been consistent. We use events in one form or another while processing messages. With multiple open-source platforms, it is difficult to choose one over the other.

In this project, we are going to understand the trade-offs between RabbitMQ and Kafka, which can help us in choosing the right messaging platform within given event-driven architecture.

At a high level, Kafka and RabbitMQ have some common use cases. For example, both can be used as part of a microservices architecture that connects producing and consuming apps. Both can also be used as a message buffer, providing a location to temporarily store messages when consuming apps are unavailable or smoothing out spikes in messages generated by producers.

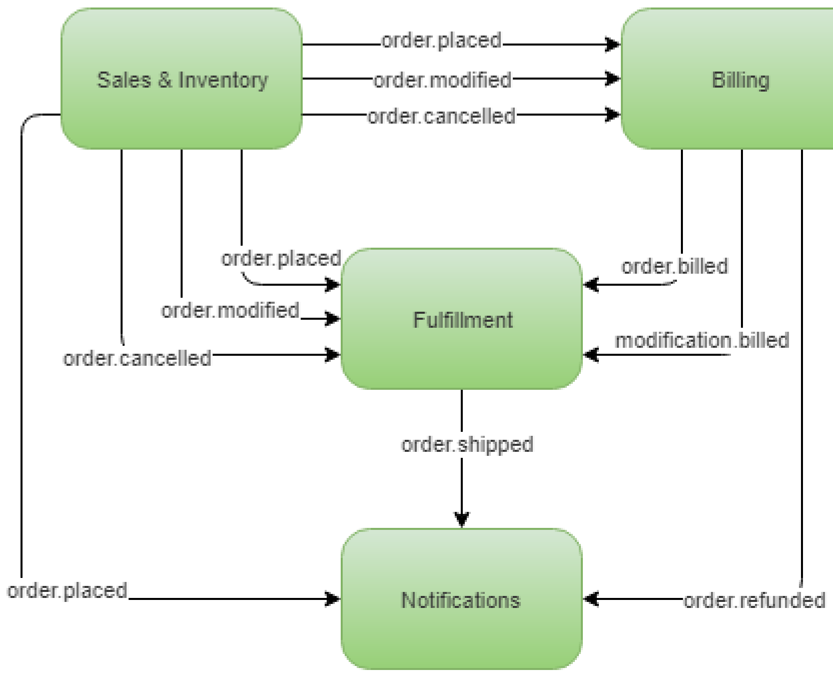
Both can also handle very large amounts of messages. But because they handle those messages in different ways, each is best suited for subtly different use cases.

The Agenda goes like this:

1. Initially, we'll be looking at the features of RabbitMQ and comparing them with the features of Kafka.
2. Then we'll look at when can we use each of these platforms.
3. We will overlay this whole RabbitMQ vs Kafka with a simple case study of microservices-based architecture.
4. Finally, we will look at the limitations and the challenges between these two platforms.

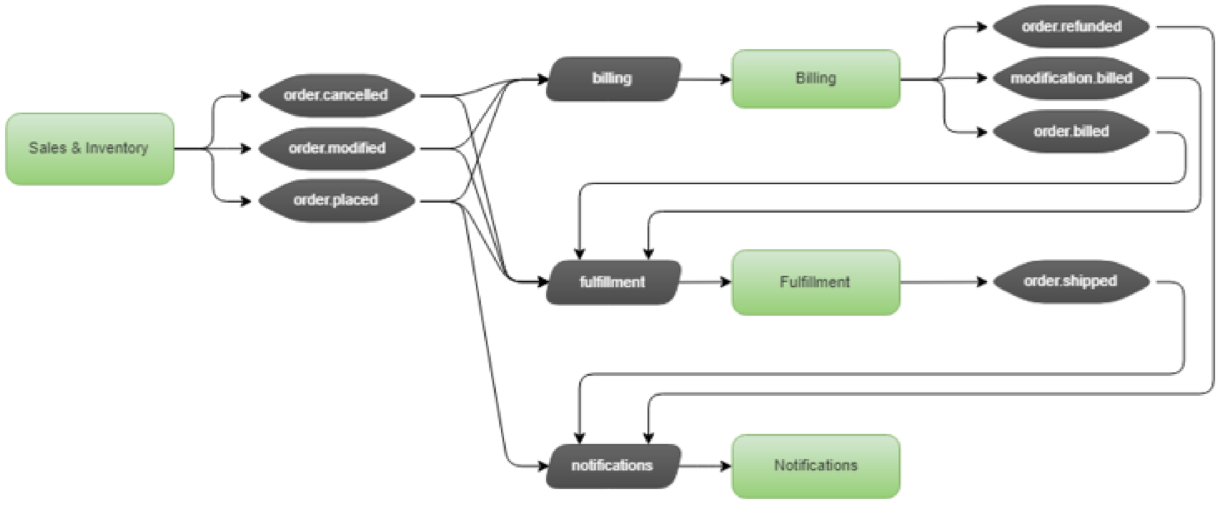
Sample Case Study
https://jack-vanlightly.com/blog/2018/5/21/event-driven-architectures-queue-vs-log-case-study

Taken from the above source, we present here an example and analogy using event-driven architecture by comparing queues with logs, basically, queues are nothing but RabbitMQ and logs are events log from Kafka. The architecture for the case study is shown below:

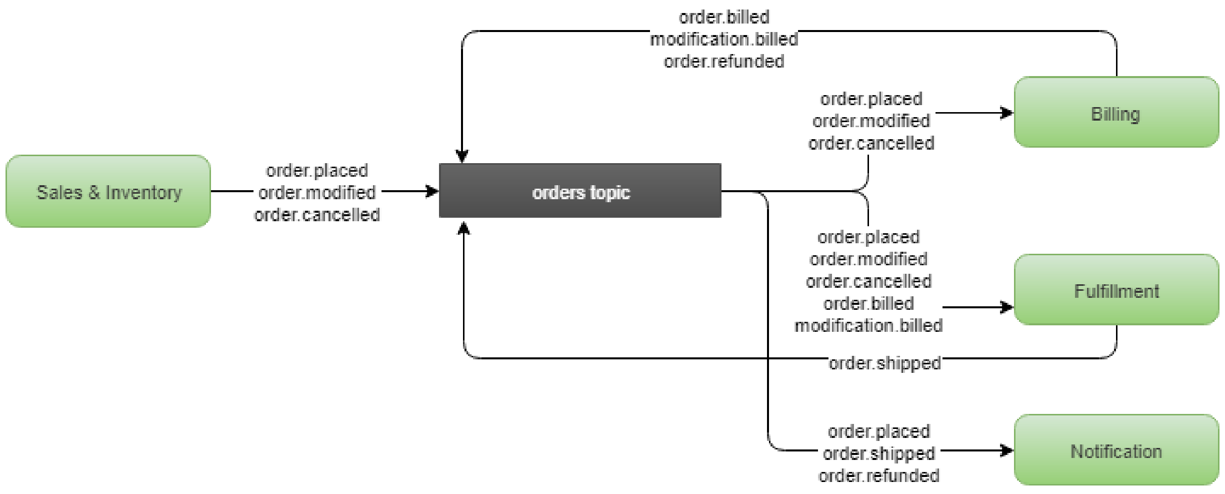


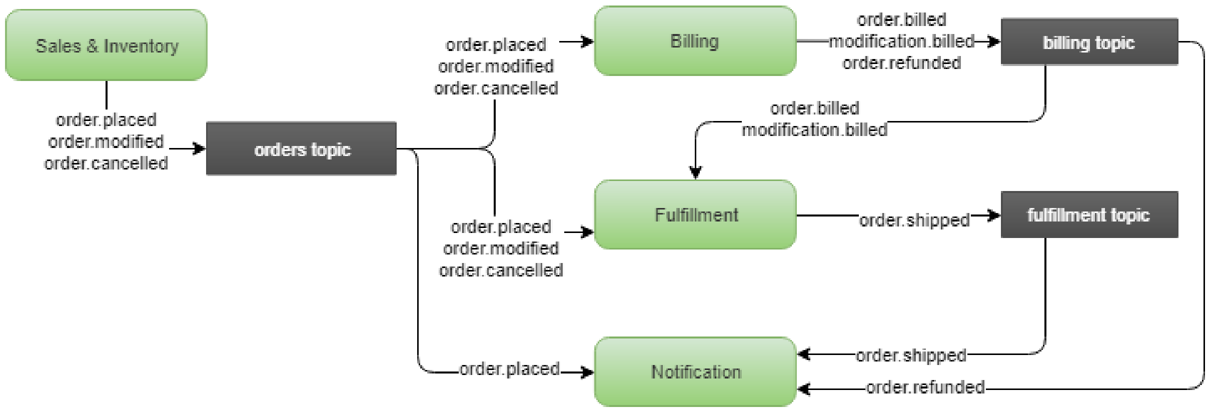
The edges here are events and the rectangular green boxes are the 4 different microservices which interact with each other in terms of events.

If we overlay this with a RabbitMQ kind of use case with Queue, we end up with the design shown below:



If we overlay this with a Kafka kind of use case with log, we end up with following two designs shown below:





What difference do you expect to see between the two systems? Why does this difference matters in the real world?

Kafka should work best for streaming A to B without resorting to complex routing, but with maximum throughput.
RabbitMQ can perform complex routing to consumers and integrate multiple applications and services with non-trivial routing logic.
Kafka would have storage overhead because it stores messages for a long period of time to provide message retention for replays.

What benchmarks will you run? How will these benchmarks quantify these differences?

With above mentioned case study or similar or say it example microservices-based architecture, we would measure performance as messages processed per unit time, end-to-end latency for a message to traverse the pipeline from the producer through the system to the consumer, trade offs between latency with durability and availability. We would see use case of acknowledgment based message retention in RabiitMQ v/s policy based (e.g. 30 days) message retention in kafka which adds storage overhead to kafka. With our example we will try to quantify and infer all possible aspects of RabittMQ v/s Kafka and provide a detailed report with results and data with visuals.