«Why Kafka Data Streaming is the Future»

Patrick P. Bucher

Summary

Apache Kafka is a horizontally scalable, fault-tolerant distributed platform for building real-time data pipelines and streaming applications (Apache Foundation, 2019). Kafka is more than an event stream or messaging system, even though it uses the same terminology. Its built-in distributed commit log keeps track of all messages being processed, turning Kafka into an *event ledger*, which allows going back to an earlier state. Kafka is commonly used for search indices, machine learning, buffered ingestion of big data sets, and common *ETL* (extract, transform, load) or *CDC* (change, data, capture) use cases.

A *record*, representing an event or a message, is the basic entity in Kafka. Every record consists of a key, a value and a timestamp. Records are immutable. Multiple records are grouped together to *partitions*, which are logically grouped to named *topics*. Kafka ledgers typically consist of multiple nodes. A *producer* sends a record for a topic to a *broker* (a node inside a Kafka cluster), and a *consumer* reads a record from a broker. Messages are *not* pushed to the consumers, but must be polled. Kafka applies *back pressure* to restrain producers from drowning nodes with too many messages. Kafka does not store the records in memory, but on the disk, harnessing fast disk caches and recent developments in storage hardware (solid state drives).

Records are ordered within a partition. Kafka replicates and balances data on the partition level from a *leader* node to multiple *followers*. Within a partition, every record has its unique *offset*. A consumer typically starts reading at offset zero. After an interruption, the consumer can continue at the offset where it left off. Multiple consumers can work together by splitting up the partition at specific offsets. Consumers within a *consumer group* are guaranteed to not process the same record twice. Different message delivery guarantees are being offered. Producers can send messages asynchronously (without delivery guarantee), or expect commit confirmation from either the leader or from the entire quorum (group of followers). Consumers can process messages at least once, at most once, or effectiverly once. (Ward, 2018)

Case Studies (C. Suter)

An insurance company began with the typical services for such a business: document management, policy calculation, data warehousing with batch processing and the like. The backend services only were available during business hours. The data warehouse processes data in the ETL manner. Later on, a customer portal was introduced, which required its own backend with 24/7 availability. An app was created, which also needed its own backend. One of its features was a step counter. Integrating all these services required one-to-one connections between the backend applications. Even though a single integration is a small problem, all those integrations taken together are hard to keep track of.

Apache Kafka was used to connect all those backend services to one another. The backends no longer were interconnected in different manners, but all had a similar interface to Kafka. Integrating new services thus became easy — even boring.

Another common use case for Kafka is credit card fraud detection. Kafka does not deal with the logic of detecting suspicious booking patterns itself. However, performing fraud detection needs a lot of data, and Kafka can provide those fast through a uniform interface.

Kafka also has its downsides. It is complex, because it consists of a lot of components (Zookeeper, schema registry etc.). Due to its central role in an integrated environment, it can become the single point of failure. Scaling and replication introduces parallelism into a system, which adds more complexity. Storing and replicating all the data in a centralised system is also a challenge for data governance and data protection.

Sources

- James Ward, Introduction to Apache Kafka, Devoxx Conference 2017, YouTube
- Kafka Website (Apache Foundation), kafka.apache.org
- Christoph Suter, Talk «Why Kafka Data Streaming is the Future» at HSLU, Rotkreuz 2019/03/20

Questions and Answers

1. Are there companies in Switzerland using Kafka productively?
   - Kafka is already heavily used by insurance companies and in the finance sector — companies that process a lot data and transactions.
2. What use cases is Kafka commonly used for?
   - Integration of backend services
   - Credit card fraud detection
   - Streaming in data from sensors (smartphone app with a step counter)
3. Is Kafka the ultimate interface for interconnecting microservices?
   - Kafka is overkill for small applications with only a few messages.
   - Its main purpose is asynchronuous communication between backend applications. Frontends should not be integrated directly over Kafka.
   - OAuth tokens and other security-sensitive information should not be put through Kafka. The applications involved (IDP, Proxy) should talk directly to one another instead.
4. Is Kafka suitable to process big media payloads (audio, video, pictures)?
   - Byte arrays can be streamed. But messages with a specific format are more common. Kafka's benefits don't serve that purpose very well.
5. Does Kafka work well in a Kubernetes environment with a network file system?
   - Kafka works well on OpenShift/Kubernetes. However, the storage used must be very fast, because Kafka persists all records on the disk.
   - The Operators Framework is a good option to run Kafka in OpenShift.