Selecting the Right streaming Engine for your Data Pipeline

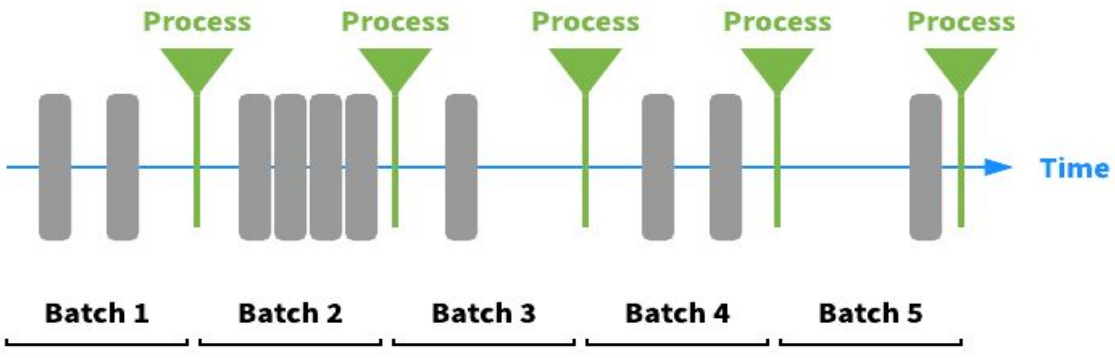
Streaming Data:
Until recently most of the data warehouses and data lakes were batch oriented where data was captured in file systems or/and databases and processed in batches. However, now the current technology advancement it is a competitive disadvantage to rely completely on batch-mode. Hence big-data systems are continuously evolving to be more stream oriented where data is processed as it arrives to get the competitive edge over peers.

Today, there are a number of Open source Streaming frameworks available in the market and almost all of them have been developed in the last few years. As everything is evolving quickly, it's so easy to get confused and swamped while selecting the right framework for your requirements. These systems need to process infinite data streams continuously and hence they need to be resilient, highly available and scalable as the volume of data grows.

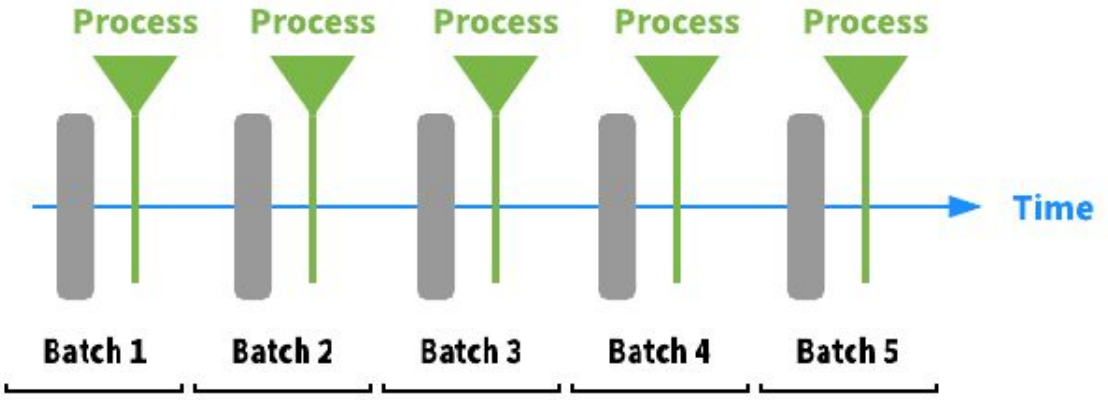
Let's briefly go through the various categories of streaming solutions available:

The streaming processing can be divided into 2 broad categories based on the way the incoming data in processed:

1) Micro batching:
   Micro batching is a practice of collecting data in small batches for processing to achieve some of the performance of traditional batch processing without actually increasing latency.
   Example: Spark Streaming



2) Native Streaming:
   In native stream processing we process each record as soon as it arrives without actually waiting for others.
   Example: Akka Streams, Flink, Apache Storm, etc.



The family of Streaming technologies can be broadly divided into 2 major categories:

1) Streaming Frameworks:
   Streaming frameworks run as a distributed service to which we have to submit jobs. They can be scaled across clusters easily. Integration with microservices usually requires that you run the framework separately from the microservices and exchange data through Kafka or other means.
   e.g. Apache Spark Structured Streaming, Apache Flink, Kafka connect, etc.

2) Streaming Libraries:
   Streaming libraries can be embedded in your microservices, providing greater flexibility in how you integrate analytics with other processing, with very low latency and lower overhead than streaming frameworks.
   E.g. Akka streams, Kafka Streams, AlpAkka, etc.

Let's look into the features to consider while selecting your Streaming Engine:

1) What is the acceptable Latency?
   a) Do you need to process records within milliseconds? (E.g. Payment processing, Fraud detection, etc)
   b) Is latency in the range of minutes acceptable? (Logs analysis)

2) What is the data volume to be processed per unit time?

Data Pipelines nowadays needs to process Billions of records everyday and if it can happen one record at a time over a long period of time then any solution can be made to work however if we are processing Twitter firehouse then we surely need to look towards scaling through distributed processing.

3) What kind of Data processing is needed:

If you are simply lifting and shifting your data or running simple ETL logic (filtering and record transformation) then there are many options to go with. If you need to perform complex joins or group by

queries then it will constrain your choices.

4) Preferred Application architecture:

Libraries can be easily integrated into your microservices providing greater flexibility and control. Frameworks provide very rich analytics and easy scalability as they run jobs on distributed services.

5) How strong and dynamic the team around the project is:

Adding new technology into the tech stack is always challenging. How easily people contributing to the project can adapt to the tool/technology and availability of CI/CD processes weighs a lot in decision making.

Let's explore some of the best breed streaming solutions available in market based on the above criteria:

1. Streaming Frameworks:
   Pick streaming frameworks when your dataset is large enough to require distributed processing. These frameworks handle very high volume of data and can do complex transformations with relatively low latency. However integrating these frameworks with microservices requires exchanging data through some messaging service like Kafka/MQ as these are standalone services.

   a. Apache Spark Streaming (Structured streaming):
      i. Use it if you are already using it for your batch processing. This transition to spark streaming from spark batch processing is easy.
      ii. Use it if your transformational logic is mainly in SQL or you want to integrate it with Machine learning systems.

   b. Apache Flink:
      i. Use it if you are not on Spark and need low latency with all the high level APIs that Spark offers. Apache Flink is fairly new in the game and although there is a rise in adoption, its open source community is not as big as Spark.
   c. Apache Storm:
      i. Use it for streaming analytics, especially when you are using languages like R which are not supported by Spark. Storm is highly scalable and provides lower latency than Spark Streaming.

   d. Kafka Connect:
      i. Use it for data streaming between Kafka and other systems. It comes with out of the box offset management and auto-recovery features and connectors to a variety of systems.

2. Streaming Libraries:

Streaming libraries can be directly integrated within your microservices and hence provides low latency. Processing is always faster if your transformation is happening in the same process rather than in separate service. It removes the extra overhead of going to Kafka or other messaging services. Libraries provide

better control and flexibility in how applications are deployed and managed.

Pick them when data partitioning is not required (Volume is manageable within standalone application) or when you want to continue with your microservices based development pattern rather than supporting heterogeneous environments.

a. Akka Streams:

i. Use it when you want fine-grained control over your data. Akka Streams follows Reactive Manifesto which defines core principles for Reactive programming (https://www.reactivemanifesto.org/). Alpakka is an integration library for Akka streams and provides a set of connectors for various databases, event bus, distributed file system, etc. You can read more about it here. If you want to learn more about Akka Streams and learn how we are able to use it to build a reactive, stable and fast data ingestion framework then please follow this link.

b. Kafka Streams:

Use it when your source and target system is Kafka or you want to run SQL on data available in Kafka.

Note: Of course, you should also consider how these tools fit into your workloads, infrastructure, and requirements, you may find that the ideal solution is a mixture of the above suggestions.

References:

- https://stackoverflow.com/questions/53727872/alpakka-kafka-vs-kafka-streams
- https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b
- https://stackoverflow.com/questions/37738102/using-apache-flink-for-data-streaming
- https://www.upsolver.com/blog/batch-stream-a-cheat-sheet
- https://www.lightbend.com/blog/cloud-native-streaming-data-with-akka-streams-kafka-steams