Stream Databases – Apache Kafka & Apache Flink

INFO-F415 - Advanced Databases

Grégoire Jean-Nicolas (446638)
Installé Arthur (495303)
Vanderslagmolen Moïra (547486)
Xu Ze-xuan (541818)

December 2024

Table of contents

01

Stream databases

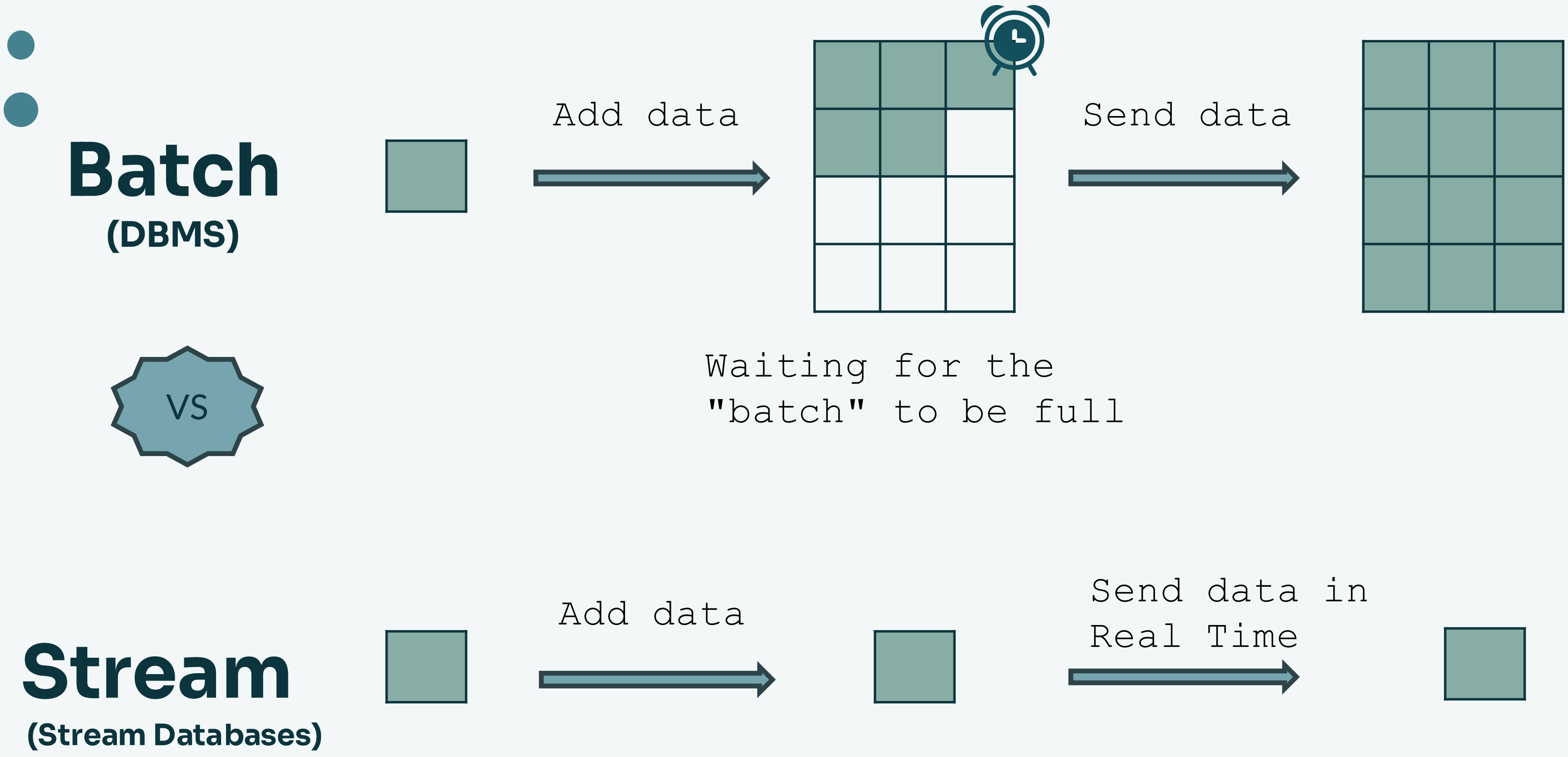
The technology

Traditional DBMS

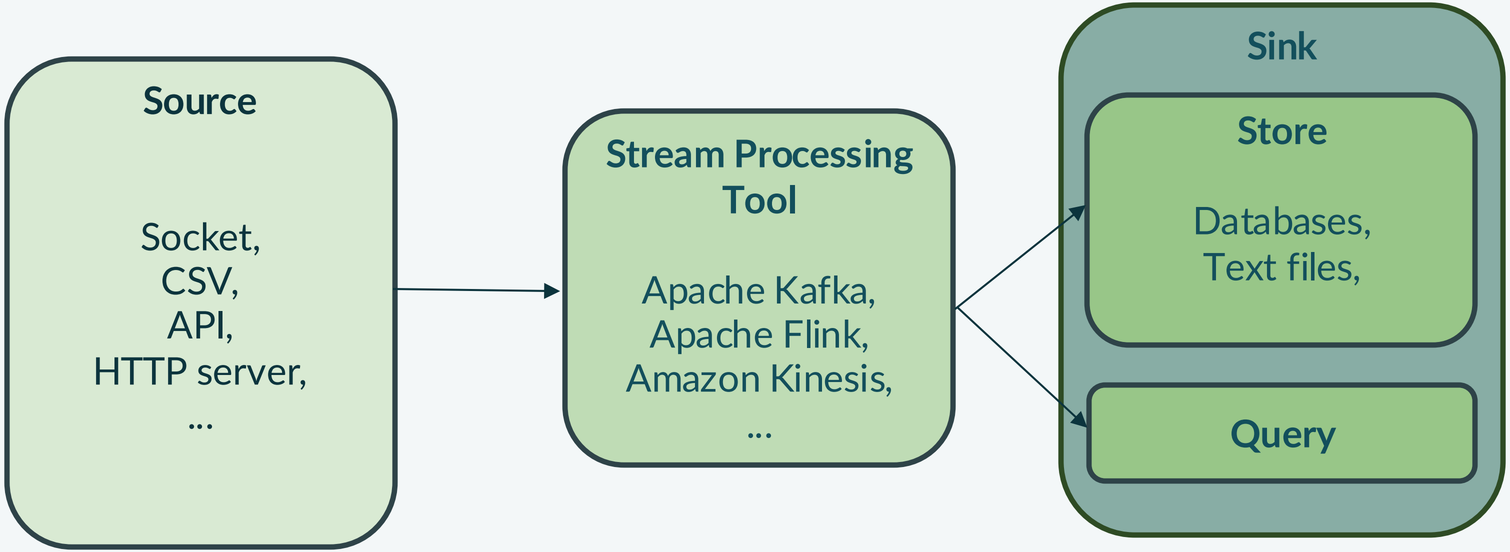
- **Widely used:** Portable across different database systems and well documented.
- **Centralized storage:** Data are stored in tables with fixed schema. Optimized for structured data with strong consistency.
- **Batch processing:** Data are processed in bulk (collection of data), and traditional database management systems are not designed for real time or continuous data flow.
- **Query Optimization:** Most DBMS systems implement index research which optimizes queries.
- **Concurrency Control:** Implements concurrency control to prevent data corruption and data inconsistency.
- **BackUps:** DBMS facilitates utilization of replications of the data.

Strengths & Limitations

Strengths	Limitations
• Reliable and mature systems. • Rich ecosystem and tooling (e.g., PostgreSQL, MySQL). • Ideal for applications with predictable workloads.	• Latency • Scalability • Real-Time Analytics • **Schema Rigidity** (Limited flexibility to adapt to unstructured or semi-structured data.) • Long implementation


Batch
(DBMS)
Add data
Send data
Waiting for the
"batch" to be full
VS
Stream
(Stream Databases)
Add data
Send data in
Real Time

Stream databases



Use cases

Fraud Detection

IoT

Monitoring

Sensors

Chat

Transactions

Example of Use Case (ISS's velocity)

Source:

The ISS coordinates and timestamp

Processing:

I want to know the ISS's velocity in real time.

Every time I receive a new coordinate, I calculate the velocity.

Sink:

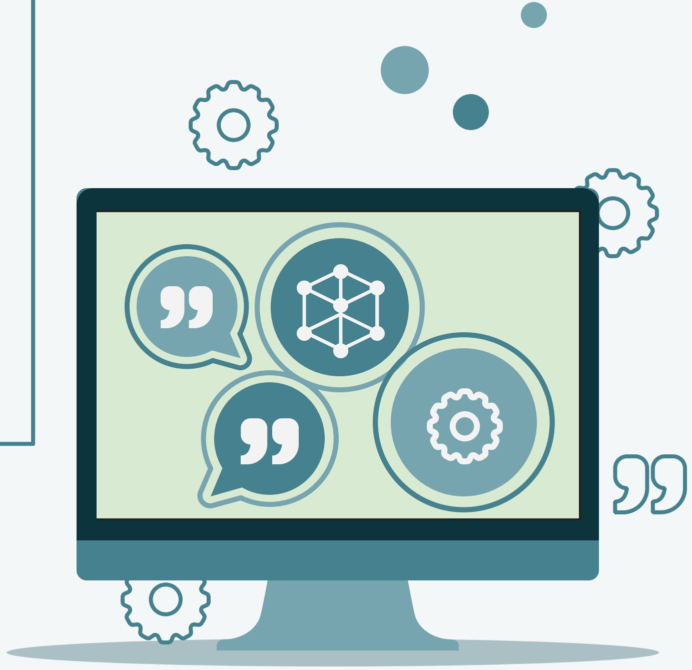
Prints the velocity

Strengths & Limitations

Strengths	Limitations
• Real-Time Analytics and processing • Low Latency • Support Event-driven architecture • Continuous Quering • Infinite Data	• No complex operations by default • Requires careful configuration for Data Reprocessing • Difficult to debug • Uses a lot of RAM

02

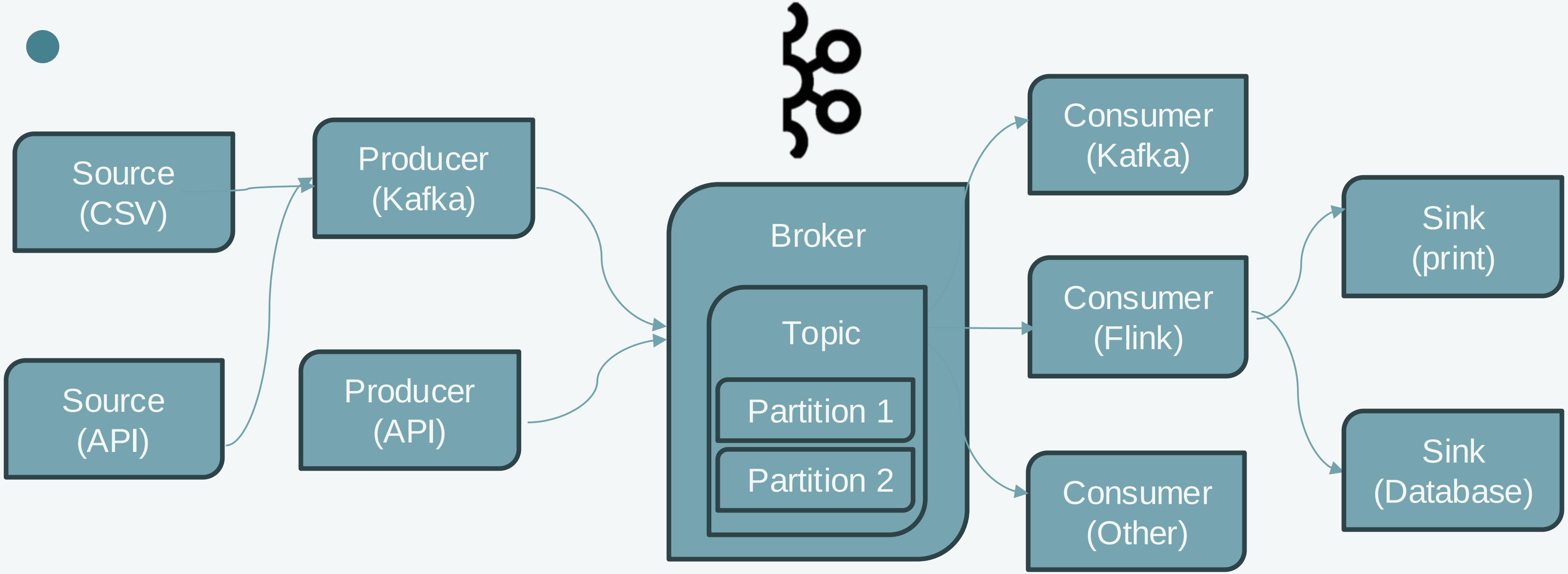
Implementation



Tools

1. Apache Kafka
2. Apache Flink
3. Apache Kafka & Apache Flink

Kafka Structure



Properties of Kafka

Advantages:

Permanent Storage

High-throughput and low-latency

Data consistency

Connect to almost anything

Disadvantages:

High resource consumption

Disk Space

Relies on Apache Zookeeper

Use cases of Kafka

HIGH THROUGHPUT

Deliver messages at network-limited speeds using a machine cluster, achieving latencies as low as 2 milliseconds.

SCALABLE

Scale production clusters to handle up to a thousand brokers, trillions of messages daily, petabytes of data, and hundreds of thousands of partitions, while supporting elastic expansion and contraction of storage and processing capacity.

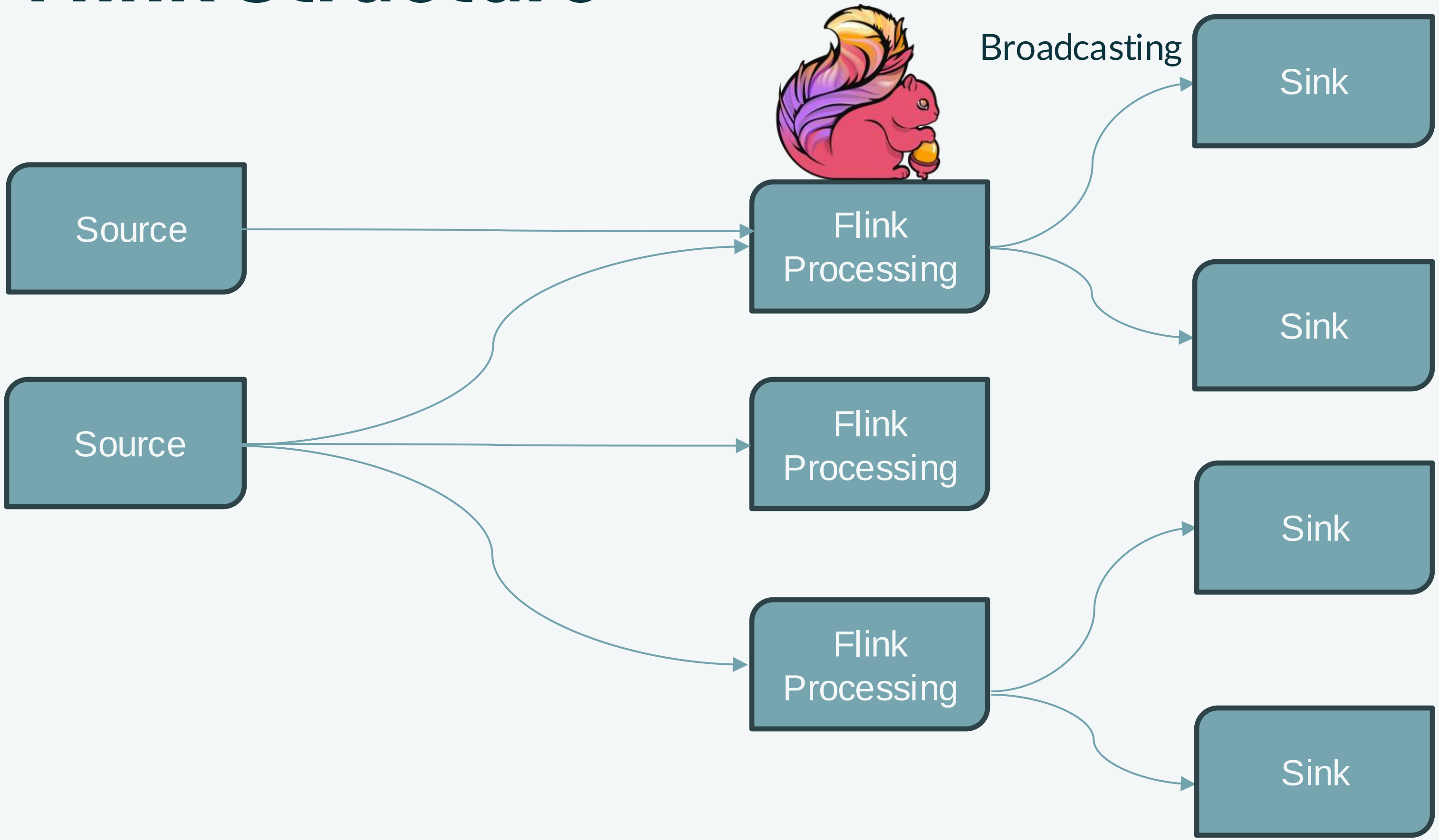
PERMANENT STORAGE

Ensure safe storage of data streams with a distributed, durable, and fault-tolerant cluster architecture.

HIGH AVAILABILITY

Efficiently span clusters across availability zones or connect separate clusters across different geographic regions.

Flink Structure



Properties of Flink

Stateful Stream Processing

Flink can group messages from sources based on tags, allowing it to apply the same code to each group independently.

Example: count the number of messages of each tag without requiring a dictionnary, as each groups is processed separently.

Timely Stream Processing

Flink can process message based on their **event** creation time.

This allows us to compare the current message with the previous one and decide whether to drop it or handle it differently if it is identified as a late message.

Exactly-Once Semantics

Flink ensure exactly-once state consistency even in case of failures. It guarantees that messages are neither duplicated or lost during processing.

Example: If a payment processing system crashes, Flink will ensure that no payment is couted twice or skipped.

Fault Tolerance

Flink ensures data consistency and recovery using checkpointing and state snapshots.

Use cases of Flink

Event Driven Applications

Stateful applications that process events from streams, triggering computations, state changes, or external actions in response.

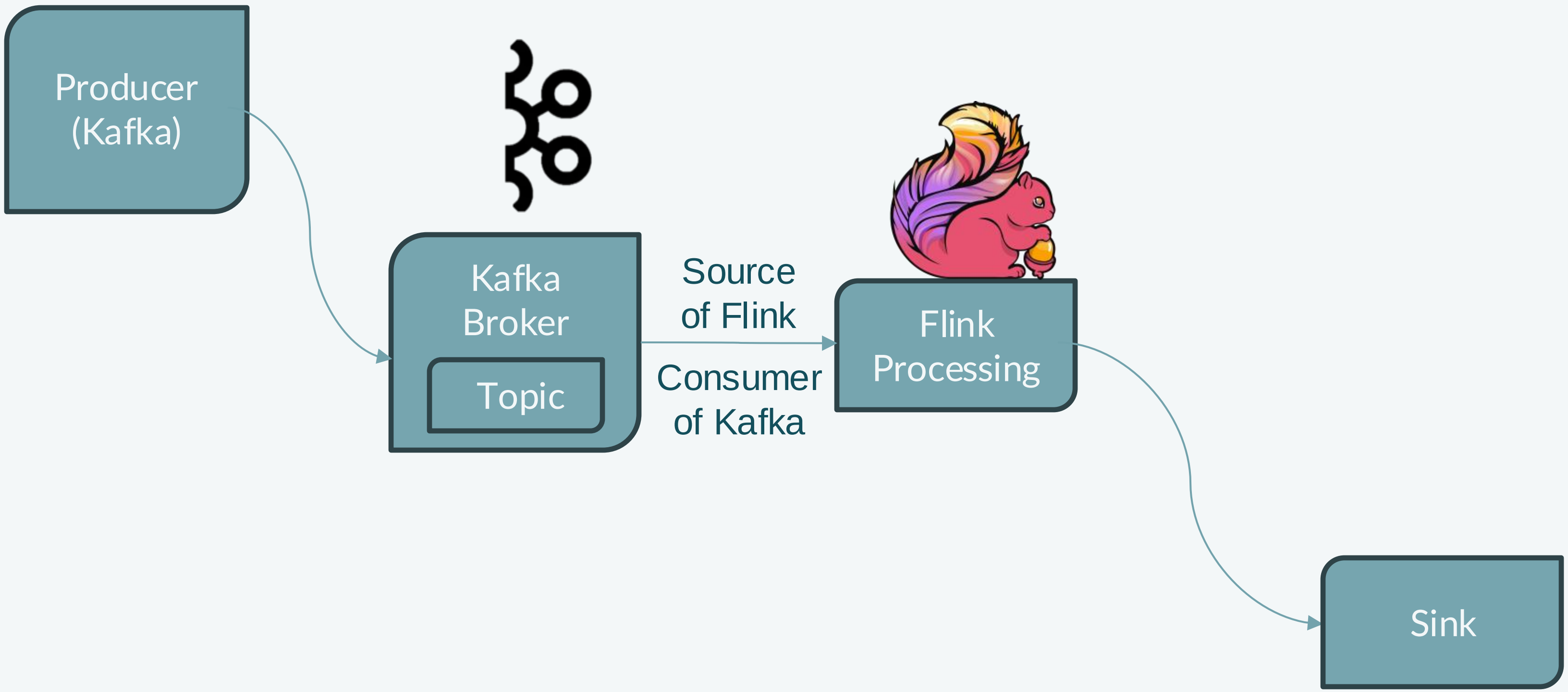
Stream & Batch Analytics

Analytical jobs extract information and insight from raw data. Apache Flink supports traditional batch queries on bounded data sets and real-time, continuous queries from unbounded, live data streams.

Data Pipelines & ETL

Extract-Transform-Load (ETL) is a common approach to convert and move data between storage systems.

Flink & Kafka Structure

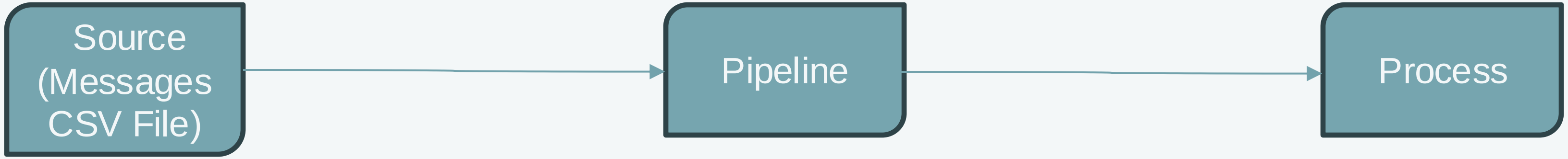


03

Application

Presentation of our Application

We developped an application that acts as a simulation of a "moderated" chat.
We have different versions of the application that use different ways to gets messages and use different ways to process the messages.



User1: Hey how are y'all doing ?
User2: You are a flower !
User3: Fine, how are you ?
User2: I am fine

User1: Hey how are y'all doing ?
User2: Message banned
User3 : Fine, how are you ?
User2: User banned

Dataset : Sentiment140.csv

Description:

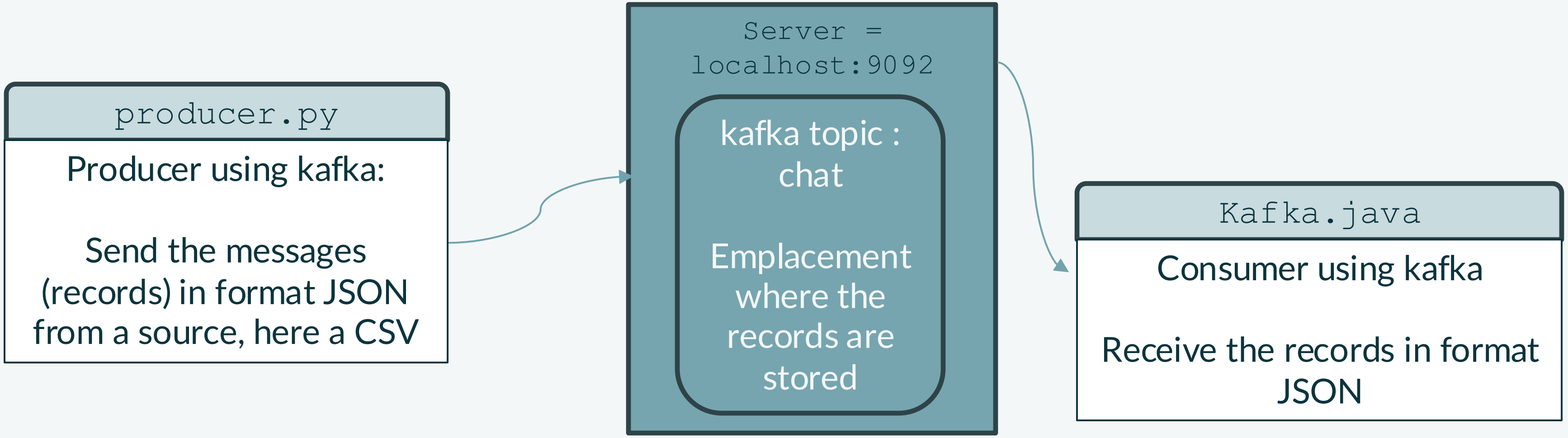
- Contains **1.6 million tweets** collected via the Twitter API.
- Used for **sentiment analysis** tasks.

Data Fields:

1. **Target**: Sentiment label (0 = negative -> "**:(**" or 4 = positive -> "**:) or nothing**").
2. **IDs**: Tweet IDs (e.g., 2087).
3. **Date**: Timestamp of the tweet (e.g., Sat May 16 23:58:44 UTC 2009).
4. **Flag**: Query term (or "NO_QUERY" if none).
5. **User**: Twitter username (e.g., robotickilldozr).
6. **Text**: The content of the tweet (e.g., "Lyx is cool").

Thanks to Marios Michailidis, Data Scientist at H2O ai, Volos, Greece.
Source: https://www.kaggle.com/datasets/kazanova/sentiment140

Apache Kafka



Producer using Kafka

producer.py

```
1. producer <- create kafka producer linked
   with broker at localhost:9092
2. data <- extract data from file
   cleaned_chat.csv
3. For each row in data:
4.     message <- row in json format
5.     producer send message to kafka topic
   "chat"
```

Kafka is able to act as a producer of a datastream which sends to a kafka topic that temporarily stores the message and waits until a consumer connects to the topic.
Each consumer will always receive all the messages retained in the topic from the start.

Kafka's Producer Steps

Step 1. Read one line of the CSV:

"0","1467810369","Mon Apr 06 22:19:45 PDT 2009","NO_QUERY","_TheSpecialOne_","Awww, lol, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D"

Step 2. Transform the CSV in a JSON record:

{"target": 0, "id": 1467810369, "date": Mon Apr 06 22:19:45 PDT 2009, "flag": NO_QUERY, "user": "_TheSpecialOne_", "text": "Awww, lol, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D"}

Step 3. The Producer send IMMEDIATELY the JSON record to the topic named chat

Consumer using Kafka

Kafka.java

```
Class Kafka:
    function connectToBroker():
        topic <- "chat"
        props <- property
        props set kafka broker to "localhost:9092"
        props set group ID to "kafka"
        props set how value are encoded and decoded
        consumer <-  kafka consumer with property props
        consumer link itself to topic
        return consumer
```

Kafka is also able to act as a consumer that will get datastreams from a topic and will process them before getting the next one.

`ConnectToBroker` configure a consumer which is connected to the topic "chat" at the server at the ip `localhost:9092` and will output the key and the value as two separeted string.

Kafka's Consumer Steps

Step 1. Read the Record received on the topic chat:

{"target": 0, "id": 1467810369, "date": Mon Apr 06 22:19:45 PDT 2009, "flag": NO_QUERY, "user": "_TheSpecialOne_", "text": "Awww, lol, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D"}

Step 2. Check if user is banned or if text contains illegal words:

TheSpecialOne is not banned
lol is an illegal word

Step 3. Outputs the message on the terminal

Consumer using Kafka

Kafka.java

```
Class Kafka:
    function main():
        file <- "../../banlist.txt"
        bannedWords <- extract words from file
        consumer = connectToBroker()
        while (true):
            Messages <- get messages from consumer
            For each message in Messages:
                Process record with Utils given bannedWords
```

Our consumer extract all the banned words from our `banlist.txt` and then try to extract new datastreams from the topic. Finaly, we call `processMessage` to process the message with the banned words list.

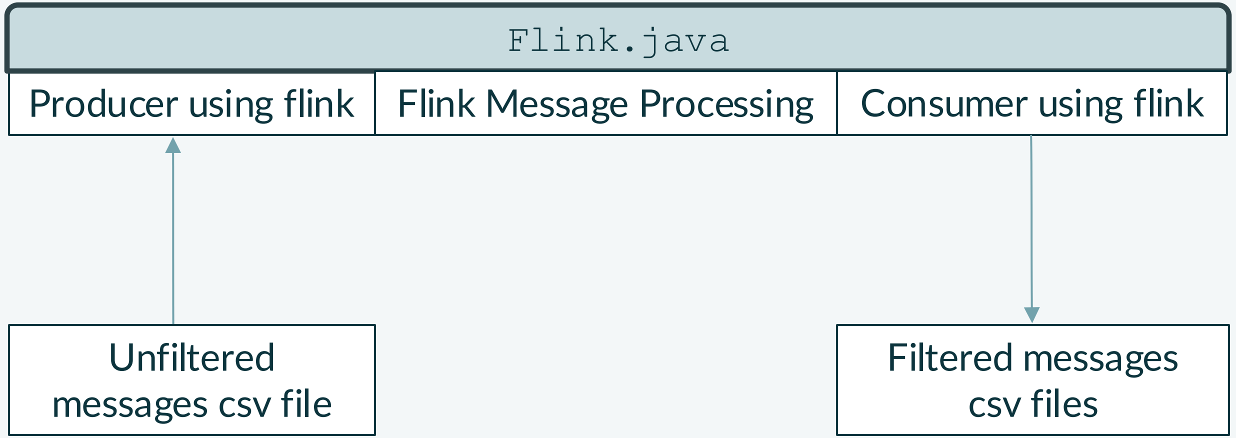
Consumer using Kafka

Utils.java

```
Class Utils:
    bannedUsers <- list
    function processMessage(message, bannedWords):
        dict <- convert message into dictionnary
        user <- get user of the message from dict
        if (is user in bannedUsers)):
            show the message in red with the text part hidden
            stop the function
        if (does the message contains banned words):
            ban the user
            show the message in red with the text part hidden
            stop the function
        show the user with the message
```

`ProcessMessage` checks if the user should be banned and check if the message contains banned words. If the message does not contains any banned words, it normally displays the message with the username in front. It only displays "`Message contains banned word`".

Apache Flink



Source in Flink

Flink.java

```
Class Flink:
    function main():
      env <- create a flink environment
        banList <- extract words from banlist.txt
        banListStream <- create datastream from Banlist
        descriptor <- create a state descriptor using banlist
        broadcastBanList <- set the banListStream with descriptor
        messages <- create datastream from chat.csv
```

The Flink's producer gets each row of a csv file as a data stream. It will be used by the consumer part.

Processing and Sink in Flink

Flink.java

```
ProcessFunction <- function BroadcastProcessFunction():
    banList
    banUserList

    @Override
    function processElement(message,output):
            columns // split message into user, date, message, ...
            if checkIsUserBanned
                    continue;
            if checkMessageIsBanned
                    continue;
            print(message)
    @Override
    function processBroadcastElement(bannedWords, ctx, output):
            banList <- bannedWords
            banUserList <- create empty set
```

Flink's Steps

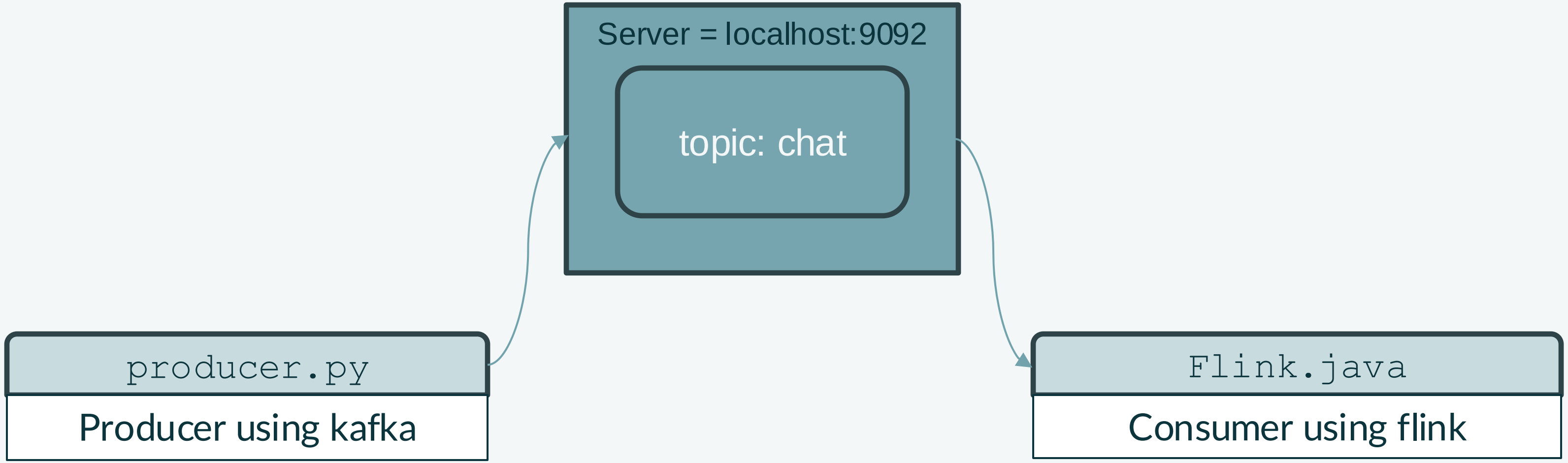
Step 1. Transform CSV into a Datastream

Step 2. Process each line of the CSV (through Broadcast ProcessFunction)

Step 3. Check if the user is banned or if the message contains an illegal word

Step 4. Outputs the message on the terminal

Apache Kafka & Apache Flink



For creating the moderated chat using mixed kafka-flink, we use kafka for the producer and flink for the consumer and a kafka topic to setup the simulated chat.

Kafka & Flink's Steps

Step 1 (Kafka). Read one line of the CSV:

Step 2 (Kafka). Transform the CSV in a JSON record:

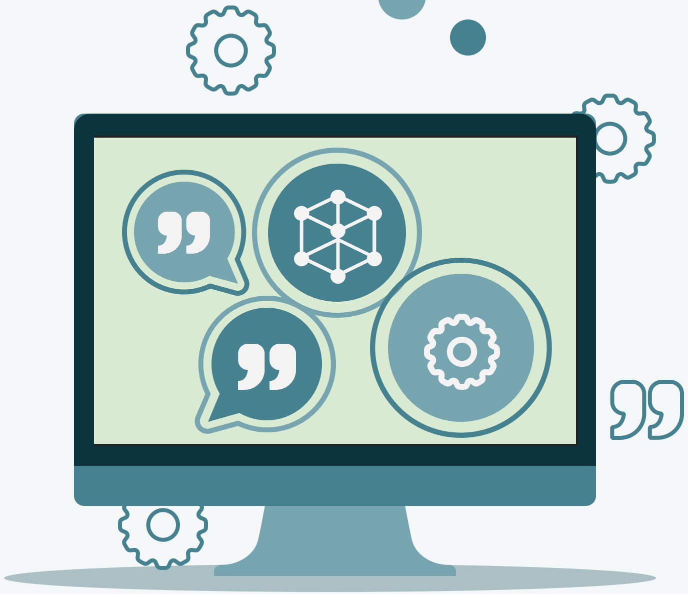
Step 3 (Kafka). The Producer send IMMEDIATELY the JSON record to the topic named chat

Step 1 (Flink). Connect to the topic of Kafka

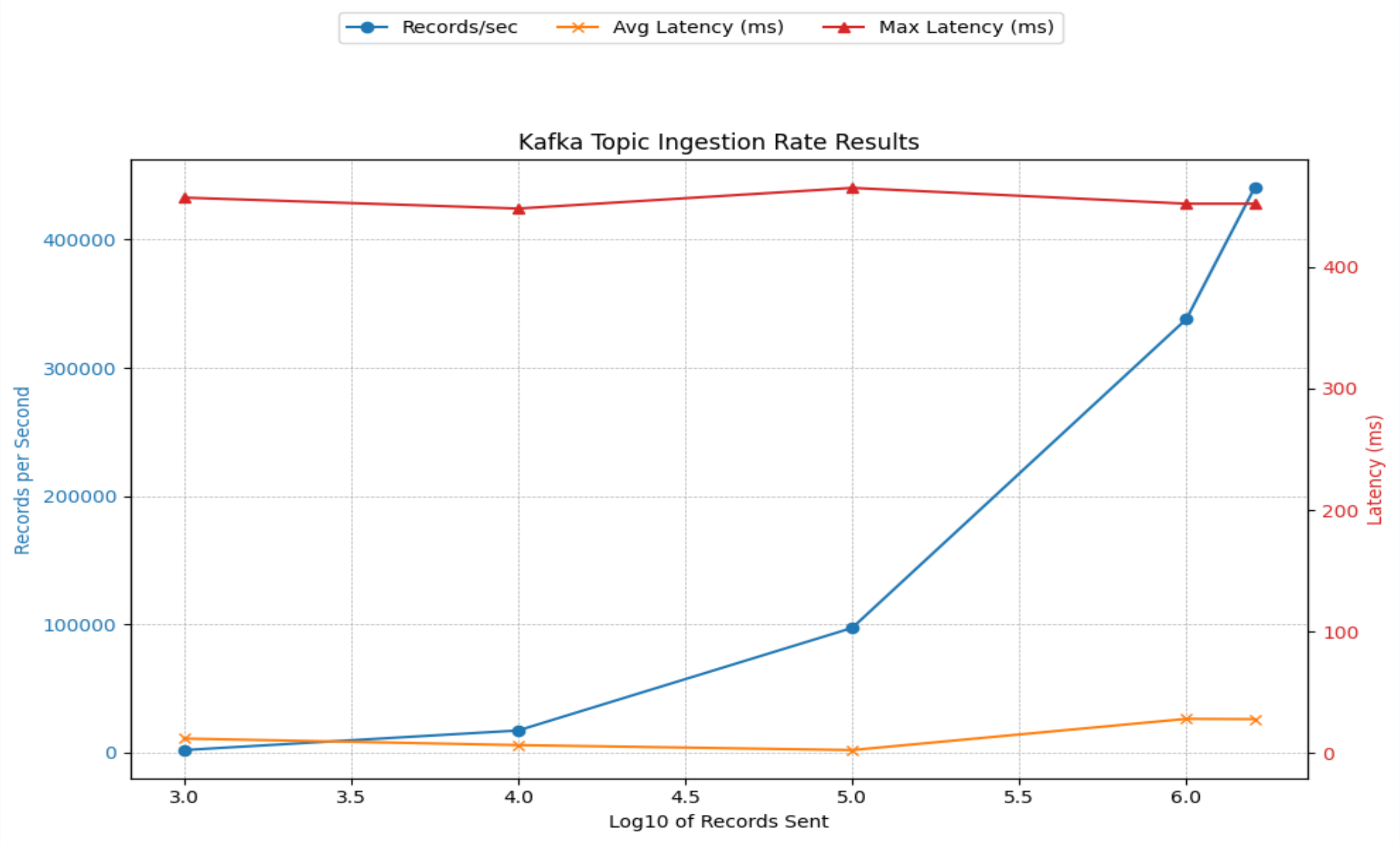
Step 2 (Flink). Create a sink that when invoked, process the record received

04

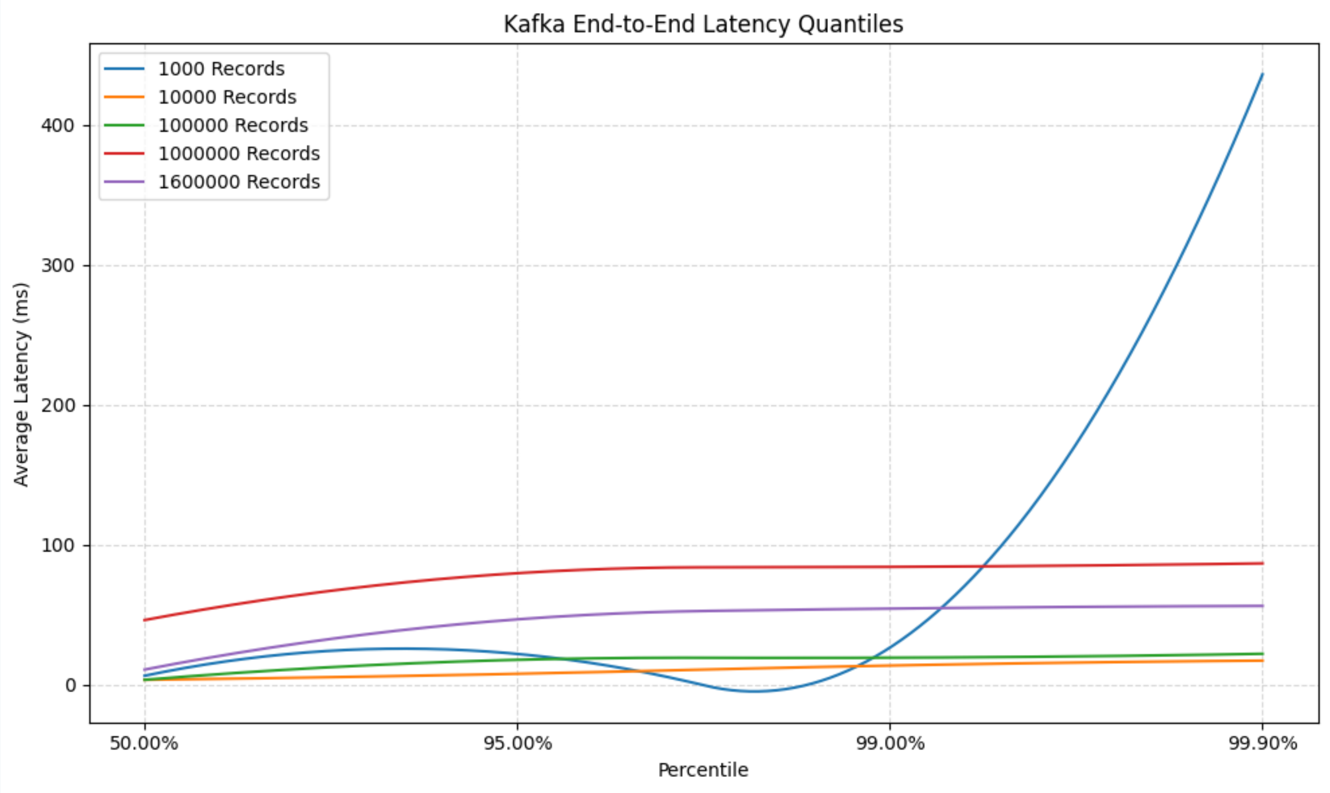
Benchmark



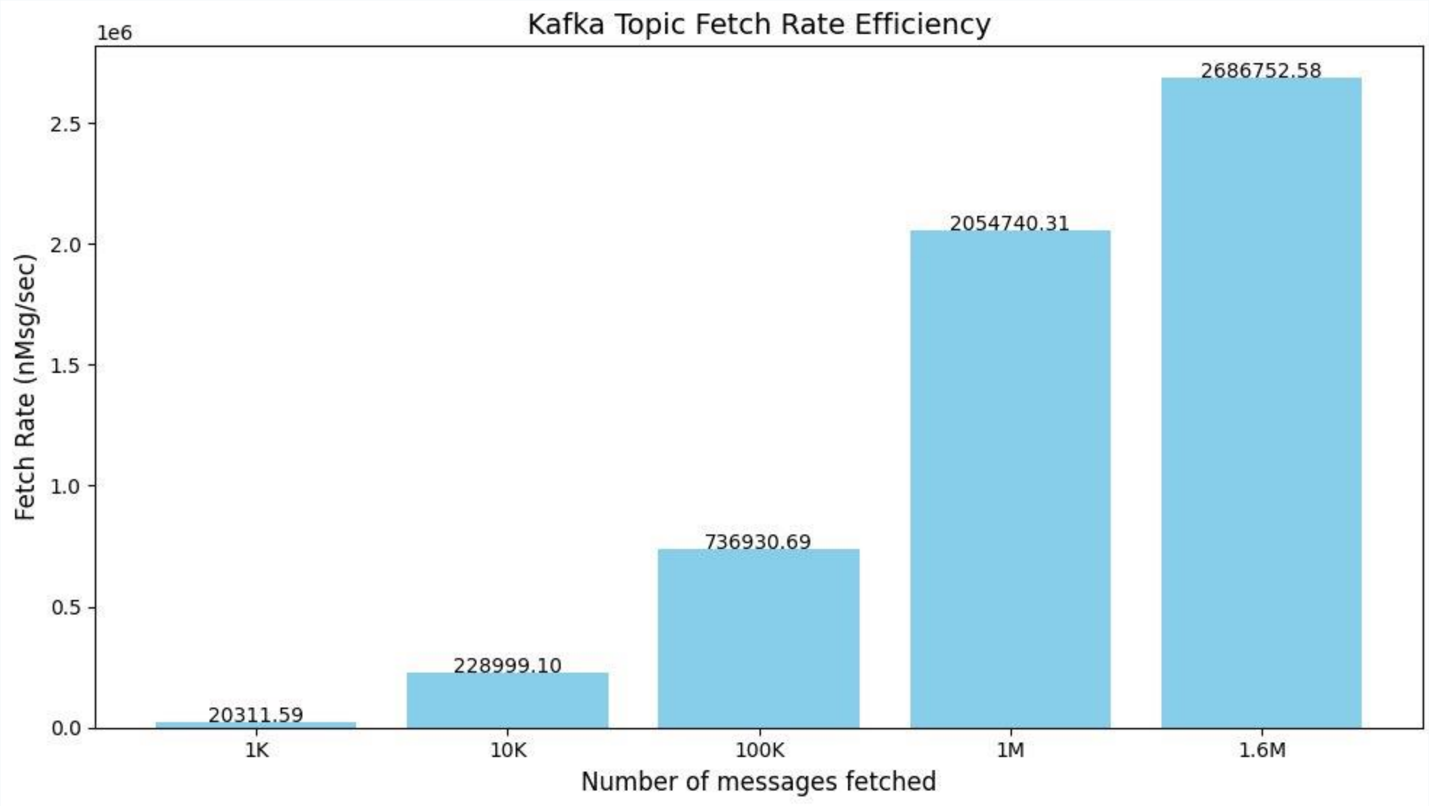
Kafka topic ingestion rate benchmark



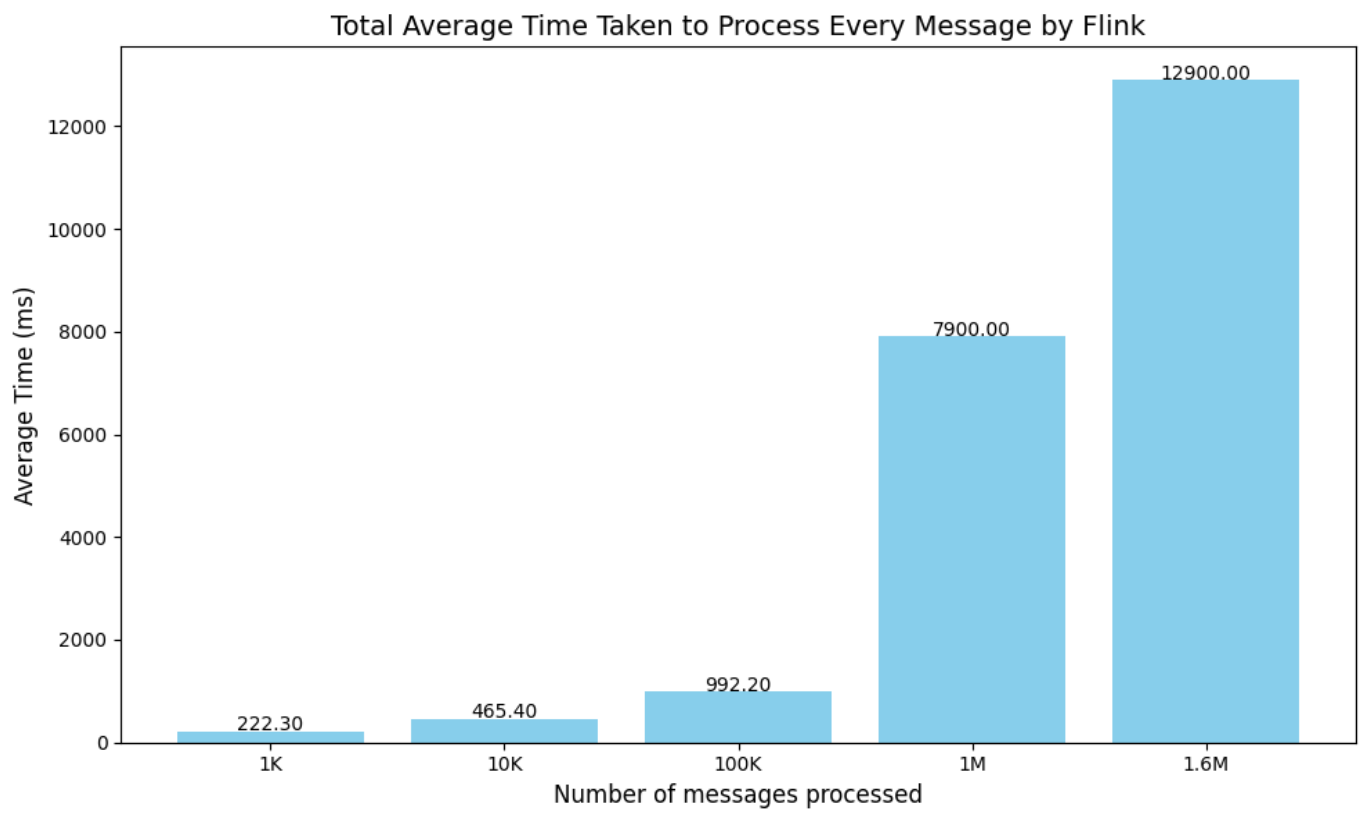
Kafka End-to-End Latency Quantiles



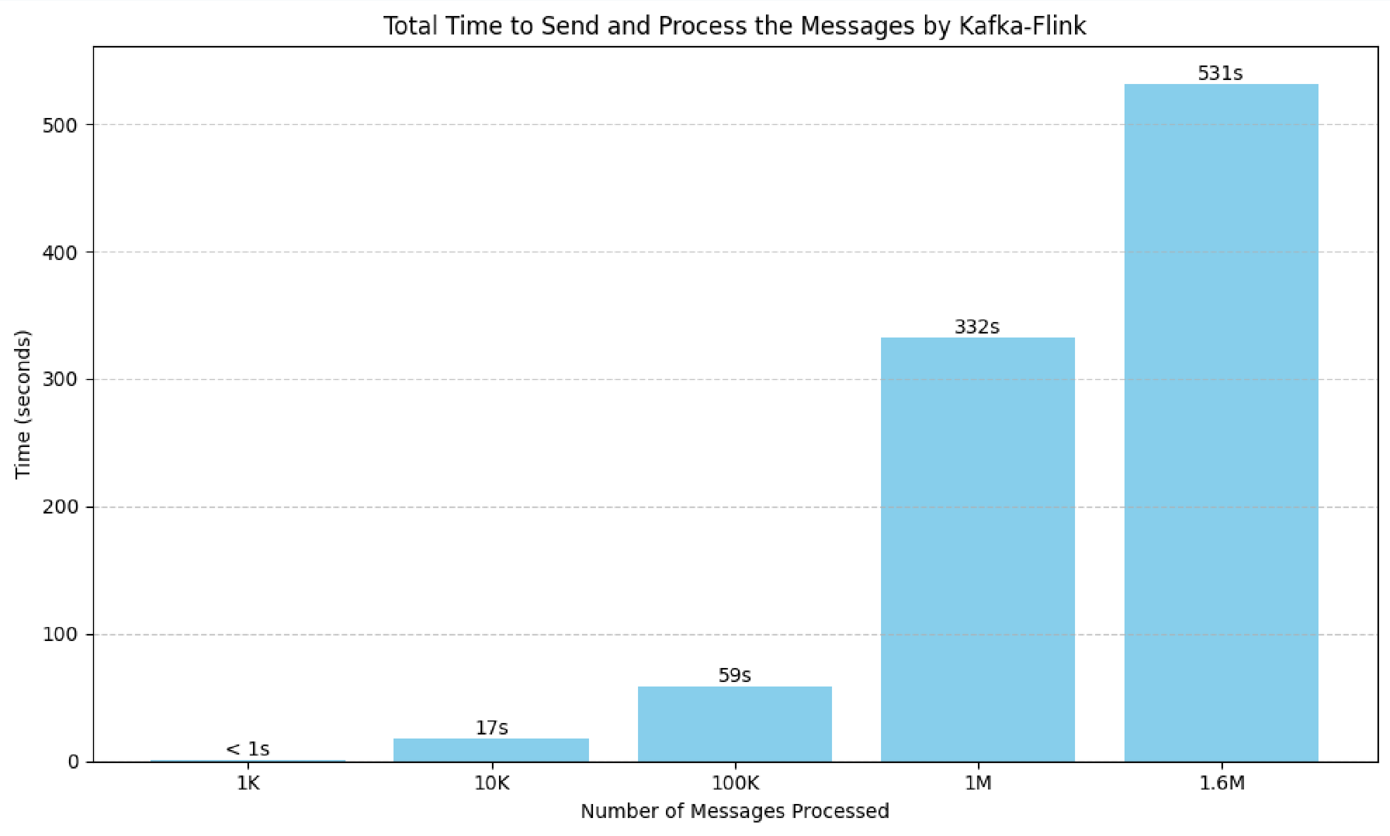
Kafka Topic fetch rate Benchmark



Benchmark for Flink



Benchmark for Kafka-Flink



Thank you for your attention ! :D