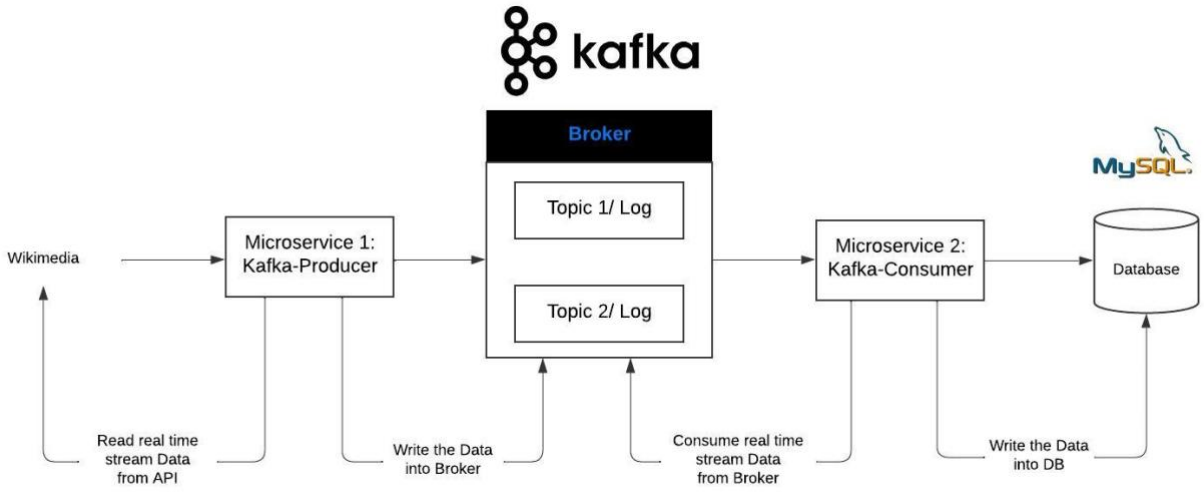
LEIDWEIN, Alex UND AL AGELE, Ismail
SFR-Software Frameworks
Wien 22.04.2024
se23m057@technikum-wien.at
se23m017@technikum-wien.at



Project Overview with Execution Details:

Our project sets up a real-time data processing pipeline using Apache Kafka to funnel data from Wikimedia's recent changes API into a MySQL database. This system is composed of two primary Spring microservices:

Kafka-Producer Microservice**:** This service uses Spring with Kafka integration to pull live updates from the Wikimedia API and push them into designated Kafka topics.

Kafka-Consumer Microservice**:** Also built with Spring, this service listens to Kafka topics and commits the incoming data to a MySQL database, ensuring the data is stored persistently.

Apache Kafka managed by Zookeeper, facilitates the reliable messaging queue. Kafka Brokers ensure messages are properly maintained and distributed for the consumer microservice.

To run the application, follow these general steps:

Set up Kafka**:** Run Zookeeper and Kafka Broker instances.

```
bin/zookeeper-server-start.sh config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
```

Configure MySQL**:** Set up a MySQL database instance, define the schema, and ensure it's accessible to your consumer microservice. Apply necessary indexes to support your query patterns.

Deploy Microservices (Optional)**:**

For the Kafka-Producer, configure it with the endpoint of Wikimedia API and Kafka Broker details. Then run the Spring application, typically using a command like java -jar kafka-producer-service.jar.

For the Kafka-Consumer, ensure it has the Kafka topic and MySQL database credentials configured. Deploy it similarly with a command such as java -jar kafka-consumer-service.jar.

Monitoring (Optional)**:** Use Spring Actuator to monitor the health and metrics of your microservices, and Kafka's own monitoring tools to keep an eye on the brokers and topics.

Error Handling (Optional)**:** Implement error handling strategies in your microservices for scenarios like API rate limiting, Kafka disconnections, or database downtimes.

LEIDWEIN, Alex UND AL AGELE, Ismail
SFR-Software Frameworks
Wien 22.04.2024
se23m057@technikum-wien.at
se23m017@technikum-wien.at

What happens if the microservice goes down and cannot process the messages from Kafka anymore?

If the Kafka-Producer microservice goes down, new messages from Wikimedia will not be published to Kafka. For the Kafka-Consumer, if it goes down, it will stop processing new messages from Kafka. Kafka will retain the messages until the consumer can process them, thanks to its retention policy.

What happens if the microservice consumes the messages and goes down while processing the message?

If the Kafka-Consumer microservice crashes during message processing, the message remains unacknowledged. Kafka's offset management ensures that the message can be reprocessed after the consumer restarts. It won't commit the offset of that message, so the message will be read again upon restart.

What happens when the microservice consumes the message but cannot write the event into the database as it is unavailable?

If the microservice cannot write to the database, it typically retries based on a defined policy. The message will remain uncommitted in Kafka, ensuring that it doesn't lose its place in the stream. After the database recovers, the microservice can attempt to write again.

Can you span a transaction across reading from Kafka and writing to the database?

Directly spanning a transaction across Kafka and a database isn't straightforward due to their distinct transaction management systems. However, we can use the "exactly-once" semantics in Kafka in conjunction with a database transaction to achieve a similar result. This usually involves an idempotent design or two-phase commit protocol to ensure consistency across the systems.

Why did you decide on the given database model (SQL, NoSQL like document store, column store, or graph database)?

The SQL database model, specifically MySQL, was chosen for its robustness in handling structured data with clear schema requirements. It supports complex queries, ACID (Atomicity, Consistency, Isolation, Durability) transactions, and relationships between different data entities, which is essential for processing and querying the structured and relational data typically produced by Wikimedia's recent changes API.

What guarantees does the database give you if you want to join different entities?

MySQL guarantees referential integrity and supports powerful join operations, enabling it to link tables through indexed keys efficiently. When different entities are joined, MySQL ensures data consistency through foreign key constraints and provides transactional support to maintain data accuracy across related entities.

Describe how the database scales (leader/follower, sharding/partitioning,) horizontally to multiple instances?

MySQL can scale horizontally using replication in a leader/follower configuration where one primary server (leader) handles writes and multiple secondary servers (followers) serve read operations. For write-scaling, MySQL supports sharding or partitioning data across different instances, which can be based on specific keys or ranges. This allows the distribution of the workload and data across several servers, improving performance and fault tolerance. MySQL also offers clustering and cloud solutions for full horizontal scaling and high availability.