scandio
TECHNOLOGY & CONSULTING

Kafka

August 11th 2022

What is Kafka?

- Hybrid between Messaging Queue and database
- Comparable to RabbitMQ, AWS SQS
- Client-Server-Model
- More concretely: Publish/Subscribe model
- Can be used for stream-processing of data as well

Producers, consumers and topics

- Basic idea: there are producers and consumers in a system
- **Producers** publish data to the server (called a broker)
- **Consumers** consume that data from the broker
- Published messages are retained for a set amount of time even if a consumer has consumed them or until available memory is exceeded (!)

Producers, consumers and topics

- Producers publish their data into containers called **topics**
- Consumers can then choose which and how many topics to read from
- That way, different communication models can be implemented:
  - one-to-one
  - fan-out / broadcasting (one/many-to-many)
  - fan-in / aggregation (many-to-one)

Messages

- Messages have the following structure:
  - an optional message key
  - an optional partition id
  - a payload (also called value)
  - timestamp
  - … as many other metadata headers as you want, e.g. for hints about how to handle encrypted payloads or other important information
- The payload is binary data i.e. bytes
- Thus, it can be anything that can be serialized - Plaintext, XML, JSON representations of Kotlin objects, JPEGs in Base64 encoding…

Partitions

- Topics are not just large containers but are divided further
- These divisions are called **partitions**
- Each partition has its own **offset**, i.e. a message counter, meaning that message order is only guaranteed within partitions
- The broker decides which partition to put a message into by
  - the partition id specified by the producer
  - if partition id isn't given, it uses hash(message_key) % number_of_partitions
  - if a message key is also not set, it uses a round-robin strategy

Why do all of this?

- For communication between microservices, we could use HTTP APIs
- However, this creates a dependency on the receiving microservice: Its address needs to be known to the producer and changing to a different consumer requires development effort
- With Kafka, all you do is write to a topic - as long as the data format is known to consumers, it acts like an interface in a programming language and internals stop mattering

Why do all of this?

- HTTP APIs are good when we expect an answer (i.e. we send GET requests to a number of services)
- But a lot of microservices architectures are just pipelines that process data and then forward it to another microservice - no need to wait
- Kafka has built-in retry and failure detection mechanisms for crashing senders and receivers, which would be a mess to implement yourself
- Of course, services can have an additional HTTP API for manually database access etc.

Why do all of this?

- How do you determine when to scale HTTP services? Based on error messages?
- **Consumer groups** are groups of consumers that read from different partitions of the same topic
- Easy solution: Set up an autoscaler and configure it to add consumers to the group if messages in a topic or partition start to pile up faster than they can be consumed
- Maximal parallelism is achieved when there are as many consumers in a consumer group as there are partitions in the topic

Kafka in practice

- <demonstration>