What's the Difference Between Kafka and Redis?

What's the difference between Kafka and Redis?

Redis is an in-memory key-value data store, while Apache Kafka is a stream processing engine. However, you can compare the two technologies because you can use both to create a publish-subscribe (pub/sub) messaging system. In modern cloud architecture, applications are decoupled into smaller, independent building blocks called *services*. Pub/sub messaging provides instant event notifications for these distributed systems. Kafka supports a pull-based system where publishers and subscribers share a common message queue from which subscribers pull messages as needed. Redis supports a push-based system where the publisher distributes messages to all subscribers when an event occurs.

Read about Kafka »

How they work: Kafka vs. Redis pub/sub

Apache Kafka is an event streaming platform that allows multiple applications to stream data independently of each other. These applications, called *producers* and *consumers*, publish and subscribe information to and from certain data partitions called *topics*.

Meanwhile, Redis is designed as an in-memory database that supports low-latency data transfer between applications. It stores all the messages on RAM instead of a hard disk to reduce data read and write time. Like Kafka, multiple consumers can subscribe to a Redis stream to retrieve messages.

Though you can use both for pub/sub messaging, Kafka and Redis work differently.

Kafka workflow

Apache Kafka connects producers and consumers through compute clusters. Each cluster consists of several Kafka brokers that reside on different servers.

Kafka creates topics and partitions for these purposes:

- Topics to group similar data belonging to a subject of interest, such as email, payment, users, and purchase
- Partitions across different brokers for data replication and fault tolerance

Producers publish messages to the broker. When the broker receives a message, it categorizes the data into a topic and stores the data in a partition. Consumers connect to the relevant topic and extract data from its partition.

Redis workflow

Redis runs with a client-server architecture as a NoSQL database system. Producers and consumers are loosely coupled and don't have to know each other when sending messages.

Redis uses keys and primary-secondary nodes for these purposes:

- Keys to group similar messages. For example, “email” is a key that points to the data store that holds only email messages.
- Primary-secondary nodes for message replication.

When a producer sends a message to a specific node, Redis delivers the message to all the connected subscribers by checking the message key. The consumer must always initiate and maintain an active connection with the Redis server to receive messages. This is known as connected delivery semantics.

Read about pub/sub messaging »

Message handling: Kafka vs. Redis pub/sub

Apache Kafka provides developers with highly scalable distributed messaging systems. Meanwhile, Redis offers rich data structures that enable an application to push data to multiple nodes quickly. Both systems have several differences in their message queuing mechanisms.

Message size

Kafka and Redis work best when they send small-sized data packets between consumers and subscribers.

Redis, in particular, isn't designed to handle large data sizes without compromising throughput. It also can’t store large amounts of data, as RAM has a smaller capacity than disk storage.

Meanwhile, Kafka can support reasonably large messages despite not being specifically built to do so. Kafka can handle messages up to 1 GB if it compresses the message and you configure it for tiered storage. Instead of storing all messages in the local storage, it uses remote storage to store the completed log files.

Message delivery

Kafka consumers pull the data from the message queue. Each Kafka consumer keeps track of the message it has read with an offset, which it updates to retrieve the subsequent message. Consumers can detect and track duplicate messages.

On the other hand, Redis automatically pushes the message to the connected subscribers. Redis subscribers passively wait for incoming messages directed to them from the server. As it is an at-most-once delivery setup, Redis subscribers are not capable of detecting duplicate messages.

Message retention

Kafka retains messages after consumers read them. So, if a client application loses the retrieved data, it can request that data again from the partition it subscribes to. By setting the message retention policy, users can determine how long Kafka retains the data.

Conversely, Redis doesn't store messages after they are delivered. If no subscribers are connected to the stream, Redis discards the messages. Discarded messages cannot be recovered even if the subscriber connects to Redis later on.

Error handling

Both Kafka and Redis allow applications to mitigate unreliable message delivery, but they do so differently.

Error handling in Redis focuses on the interaction between the client application and the Redis services. With Redis, developers can address circumstances such as client timeouts, memory buffer exceeded, and maximum client limits. Because of its key-value pair database architecture, Redis cannot provide robust message error handling as Kafka does.

Kafka developers can store erroneous events in a dead letter queue, retry, or redirect them to allow consistent message delivery to client applications. Developers can also use the Kafka Connect API for automatically restarting connector tasks in certain errors.

Read about dead letter queues »