High-level Design of Kafka

Learn about the basic building blocks and programming interface of Kafka.

We'll cover the following

Data flow

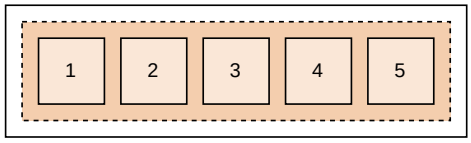
Let's start with the basic units of Kafka and build up the concepts of the components in which they are saved.

Message

The basic unit of data in Kafka is called a **message**. It can be thought of as a row in a CSV file or a record in a database. Messages contain a payload of bytes. It can also have a payload of metadata, referred to as a key. Keys can be hashed to write messages to certain partitions in a topic in an organized way. This assures us that the messages with similar keys exist in the same partition.

Batch

Messages are exported to Kafka in batches to increase its throughput. A **batch** is a set of messages that exist in the same partition of a topic.

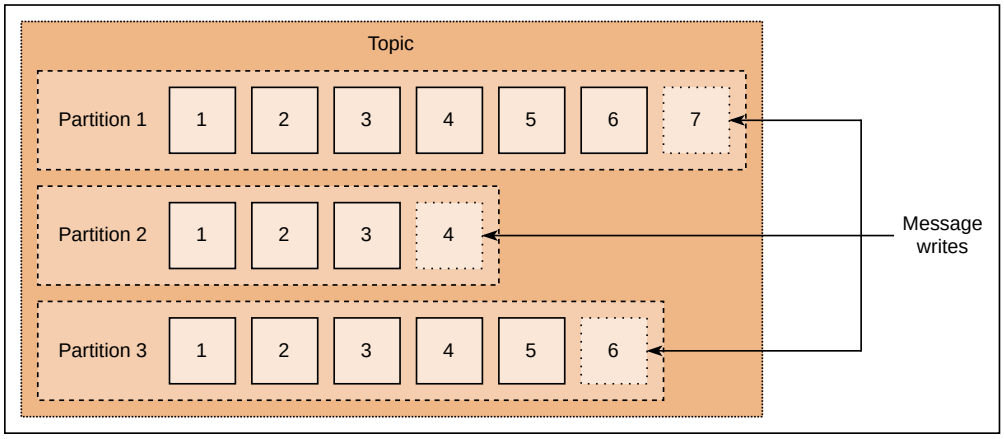


Batch of messages

Batching messages eliminates the need for each message to require a full TCP/IP roundtrip, which was detrimental to the throughput requirements of Kafka.

Topic

Messages are categorized into a particular type, and those particular types of messages collectively are called a **topic**. We can think of topics as a table in a database or a folder in a filesystem. Topics are composed of partitions. Messages can only be appended to partitions of a topic, which are read from beginning to end. Messages from a single partition are read in an orderly fashion. However, because there are multiple partitions in a topic, there is no guarantee of ordered messages across the whole topic.



Partitions in a topic (the numbering indicates the writing sequence within a partition)

A topic provides scalability to Kafka because its partitions can be saved in different servers, making it horizontally scalable across several servers. The term **stream** is often used when data processing systems like Kafka are mentioned. A single topic can be referred to as a stream that is moved from producers to consumers.

> **Note:** A topic is the primary abstraction provided by Kafka. Producers post their messages to a topic, and consumers get their messages from their intended topic. You can think of a topic as a mailbox in which one party puts a message that the intended recipient can later fetch.

Architecture

Kafka has three main components. Each component is described as follows.

Producer

The producers perform the following tasks:

- They publish messages to the brokers in the form of topics.

- They can publish multiple messages (batch) in a single publish request.
- Any producer can interact with multiple brokers.

Broker

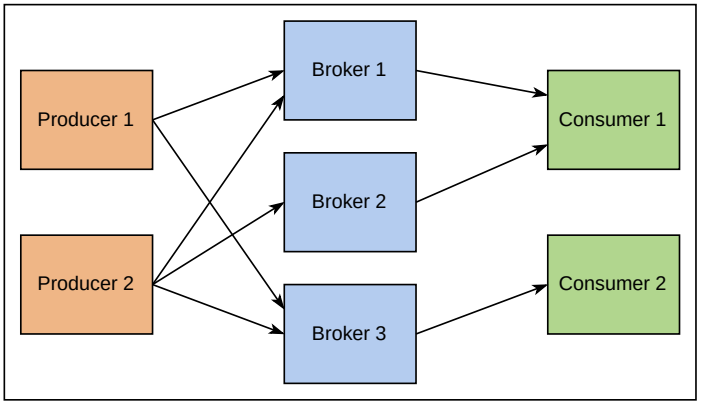
The brokers perform the following roles:

- They are a set of servers.
- They store the messages published by the producers.

Consumer

The consumers perform the following roles:

- They subscribe to a set of messages in the brokers.
- They consume the subscribed set of messages from the brokers.
- Any consumer can interact with multiple brokers.



Kafka's workflow

Kafka API

Messaging is quite a simple concept. A sender sends a message through a channel, and then there is a receiver. The API design of Kafka described below is equally simple.

Producer API

- First, a producer is defined.
- Messages are defined, each containing a payload of bytes, and then they are put in a set.
- The set of messages is sent.

```
1  P = new Producer();
2  M = new Message("Hello world".getBytes());
3  S = new MessageSet(M);
4  P.send("topic",S);
```

Broker API

Consumer API

- The consumer creates one or more message streams for a topic for subscribing to it.
- The messages in that topic will be uniformly distributed in substreams.
- Each stream of messages provides an iterator, and the consumer uses that iterator to iterate over all the messages of the continual stream of messages being received and then process them.
- Unlike conventional iterators, the iterator in consumers does not terminate when there are no more messages to consume. It stops iterating until new messages are published.
- Kafka supports both the point-to-point and publish-subscribe delivery methods.

```
streams[] = Consumer.createMessageStreams("topic",1)
for (M: streams[0]){
  bytes = M.payload();
  //perform operations on received bytes
```

Consumer API

In this lesson, we learned how the data is structured in Kafka, which components help move and store the data, and how they do it.

Back

Introduction to Kafka

Next

Detailed Design of Kaf...

Mark as Completed