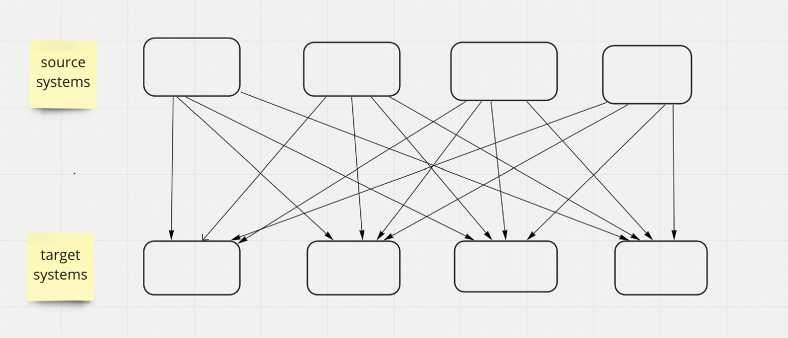
**Kafka — Architecture**

Let’s first try to understand the problems that are faced in an organisation —



complex system architecture

In the above diagram, there are 4 source systems and 4 target systems, i.e we require 16 integrations among them. In each integration there might be challenges —

a) Data transfer protocol (Http, Tcp, Ftp, Jdbc)

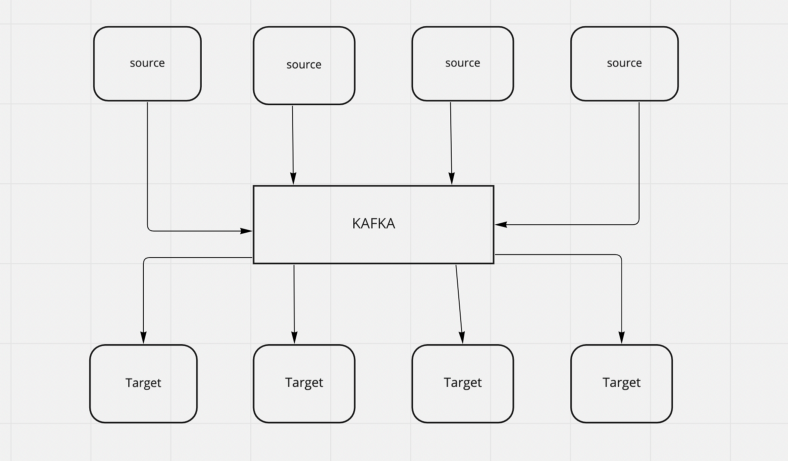
b) Data format (CSV, Json, Binary)

c) Data schema evolution and change.

Too Complex!!

*Note — Source systems can be click events, price data on an e-commerce platform, transactions or even user interaction. Target systems can be database, analytics engine, email system or audit system.*

**A better approach is to decouple our source systems and target systems using a message broker like KAFKA**.



kafka as a broker

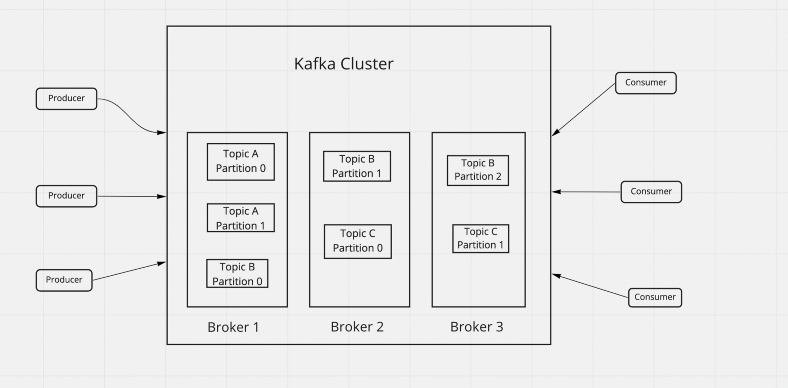
Now the question comes, why Kafka ? Well it has few features that makes it a better choice —

a) Distributed and resilient architecture

b) Horizontal scalability

c) High real time performance i.e. low latency

**Let’s take a look at the high level design of kafka —**



KAFKA Architecture

1. **Topic —**It is a logical data unit and identified by its name. It can have any type of message format. Topics are split into **partitions**. Messages with each partition are ordered. Each message with in a partition gets an incremental id called an **offset**.

*Note — Kafka topics are immutable, so once the data is written to a partition, it can’t be modified. The order of messages is guaranteed with in a partition only.*

2. **Producers —**They write data to the topics. They know on which partition they have to write the data. They can also send a **key**along with the message.

*Note — If the key is null, then the messages go to partitions in a round robin approach. If the key is not null, then all the messages with a particular key go to the defined partitioned.*

**3. Consumers —**They read data from a topic. They pull messages. A consumer can read from more than one partition in a topic. Data is read from a topic in the order they are in it..

*Note — We need to mention the format of messages at the consumers.*

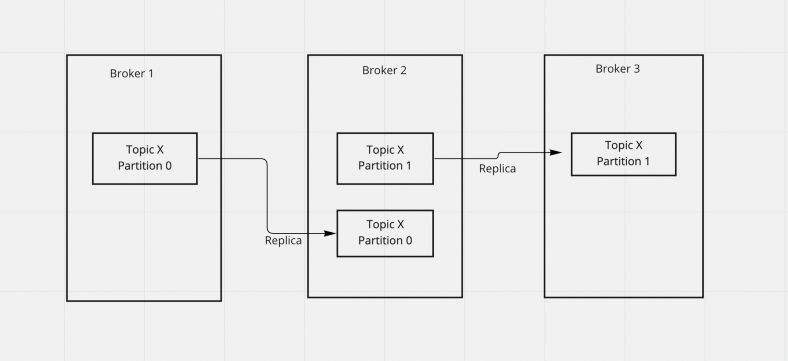
4. **Consumer group —**When there are more than one consumer in our application, they read data as a group which is called consumer group. It is completely possible that we have more numbers of consumers than we have partitions. In such scenarios those extra consumers will sit idle.

**Message Delivery semantics for consumers —**

1. At least once — Offsets are committed after the message is processed. If any exception occurs while reading a message, it will be read again. This may lead to duplicate message processing, so the processing of messages should be idempotent.
2. At most once — Offsets are committed as soon as the messages are received. Here, message may lost if something goes wrong while processing.
3. Exactly once — Offsets are committed after the message is processed once.

***Note — A kafka cluster is made up of more than one kafka brokers or servers. Each server has its own ID and fixed number of topic partitions.***

**Topic Replication —**Topics should have a replication factor greater than 1 (usually between 2 & 3). If a broker is down, another broker can serve the data for the same topic.



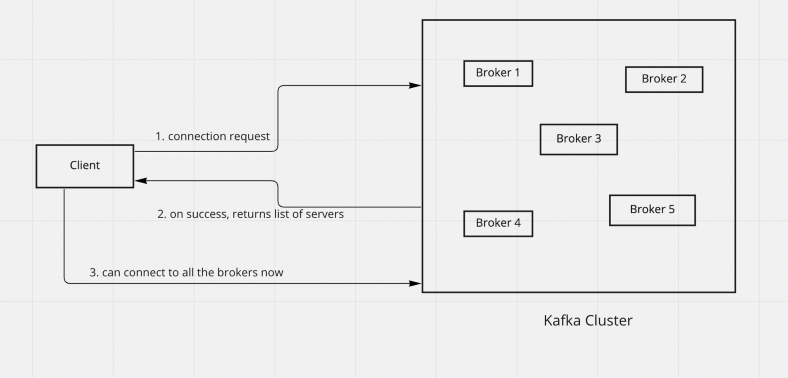
Replication

**Leader Concept —**At any point of time, only one broker can be a leader for a particular partition. Producers can send data to leaders only, in other words, data can only go to the broker that has the leader partition.

**Producer Acknowledgement —**Kafka brokers can send write acknowledgements to the producers.

1. acks = 0 (Producers will not wait for the acknowledgements, resulting in a risk of data loss)
2. acks = 1 (Producer waits for acknowledgement from the leader only)
3. acks = all (Producer waits for acknowledgement from the leader as well as the replicas)

**Kafka Broker Discovery —**Every kafka broker is also known as a “bootstrap server” i.e we only need to connect with one broker and the kafka client will know how to connect to the entire cluster.



Kafka Broker Discovery

**References —**