4/6/2021

Bipin Kumar

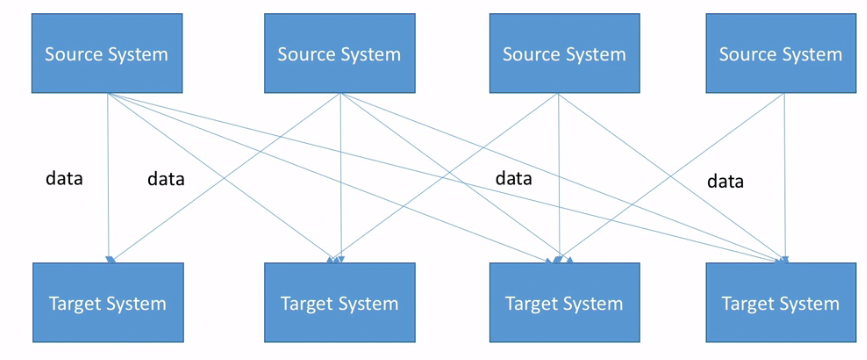
shael.dhn88@gmail.com

**KAFKA**

Kafka notes

When we start writing any application we generally have source system and target system as given in below diagram.

But after time system become complex and more complicated as shown in below diagram.



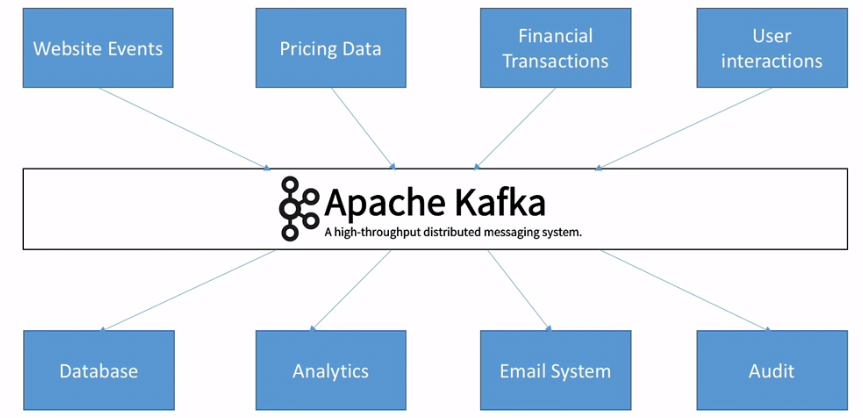
If we take above example and suppose if we have 4 source system and 4 target system then we need to write 16 integration code.

Each integration will come with some difficulties.

* Protocol: which protocol it gone use (HTTP, FTP, TCP, JDBC etc.)
* Data format: How the data is parsed (csv, json, xml etc.)
* Data schema and evolution- How the data id shaped and may change.

Each source system will have an increase load from the connections.

To resolve these issue Apache Kafka came in picture as shown in below diagram.



So Now all the source system send data to Kafka and target system will use these data as per there needs. So due to this we achieve decoupling of system.

Why Kafka?

* Created by linkedin, now open source project mainly maintained by confluent.
* Distributed, resilient architecture, fault tolerant.
* Horizontal scalability:
* Can scale to 100s of brokers
* Can scale to millions of messages per second.
* High performance (latency of less than 10ms) real time
* Used by the 2000+ firms, 35% of the fortune 500: e.g. Airbnb, linkedin, uber,Netflix etc

Use cases:

* Messaging system
* Activity tracking
* Gather metrics from many different locations
* Application logs gathering
* Stream processing (With the kafka streams API or Spark for example)
* De-coupling of system dependencies
* Integration with spark, Flink, storm, Hadoop and many other big data technologies.

e.g. Netflix uses kafka to apply recommendation in real time while you’re watching TV shows

Uber uses kafka to gather user, taxi and trip data in real time to compute and forecast demand and compute surge pricing in real time.

LinkedIn uses kafka to prevent spam, collect user interaction to make better connection recommendations in real time.

Kafka is only used as a transportation mechanism.

Kafka Theory:

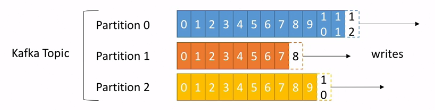
1. Topics, Partitions and offsets:

Topics: a particular stream of data

* Similar to a table in a database (without all the constraints)
* You can have as many topics as you want
* A topic is identified by its name

Topics are split in partitions

* Each partition is ordered.
* Each message within a partition gets an incremental id, called offset.



Topic example: truck\_gps

* Say we have a fleet of trucks; each truck reports its GPS position to Kafka.
* You can have a topic trucks\_gps that contains the position of all trucks.
* Each truck will send a message to kafka every 20 seconds, each message will contain the truck ID and the truck position (latitude and longitude)
* We choose to create that topic with 10 partitions (arbitrary number)

Offset only have a meaning for a specific partition. E.g. offset 3 in partition 0 doesn’t represent the same data as offset 3 in partition 1.

* Order is guaranteed only within a partition (not across partitions)
* Data is kept only for a limited time (default is one week)
* Once the data is written to a partition, it can’t be changed.
* Data is assigned randomly to a partition unless a key is provided.

**Broker:**

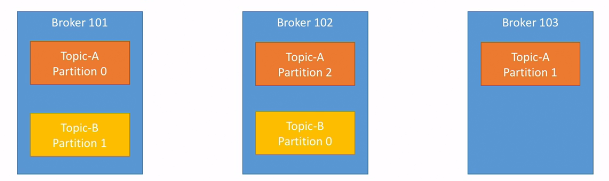
* A Kafka Cluster is composed of multiple brokers(servers)
* Each broker is identified with its ID (integer)
* Each broker contains certain topic partitions.
* After connecting to any broker (called a bootstrap broker), we will be connected to the entire cluster.
* A good number to get started is 3 brokers, but some big clusters have over 100 brokers.
* In these examples we choose to number brokers starting at 100 (arbitrary)



Example of Topic-A with 3 partitions:

Then Topic-A with partition 0 is in 101, partition 2 in 102 and partition 1 in 103.

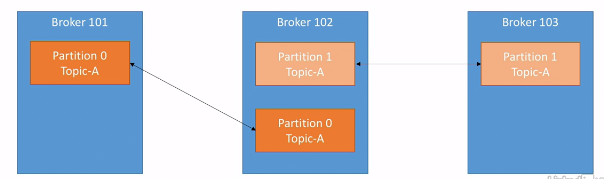
Now suppose we have Topic-B with 2 partitions then may be broker 101 will have partition 1, 102 will have partition 0 as shown in below diagram.



Note: Data is distributed and broker 103 doesn’t have any topic B data.

**Topic replication factor:**

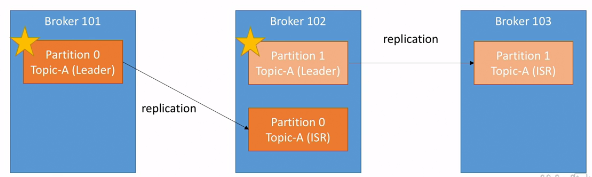
* Topic should have a replication factor > 1 (usually between 2 and 3)
* This way if a broker is down, another broker can serve the data.
* Example Topic-A with 2 partitions and replication factor of 2.



Suppose we lost broker 102. Result broker 101 and 103 still serve the data.

Concept of leader for a partition.

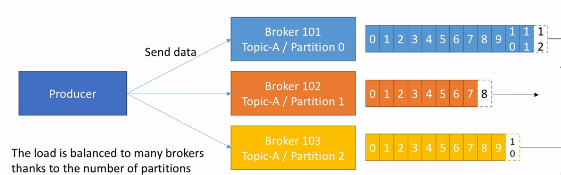
* At any time only one broker can be a leader for a given partition.
* Only that leader can receive and serve data for a partition.
* The other brokers will synchronize the data.
* Therefore, each partition has one leader and multiple ISR (in-sync replica)



Leader and ISR decide by zookeeper.

Producers:

* Producer write data to topics (which is made of partitions)
* Producer automatically know to which broker and partition to write to.
* In case of broker failures, Producers will automatically recover.



If we send data without key then data will be distributed between brokers in round robin fashion.

Producers can choose to receive acknowledgment of data writes:

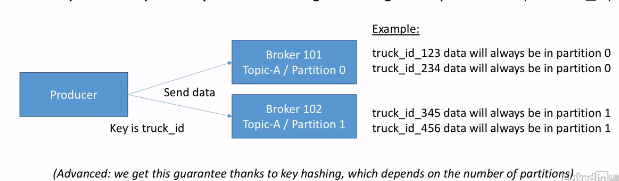
Acks=0: producer won’t wait for acknowledgment (possible data loss)

Acks=1: producer will wait for leader acknowledgment (limited data loss)

Acks=all: leader + replicas acknowledgment (no data loss)

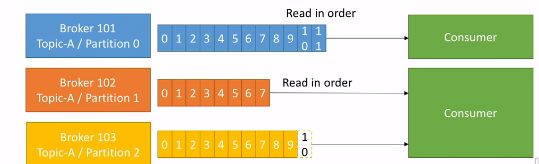
Producers: Message keys

* Producers can choose to send a key with the message (string, number etc.)
* If key=null, data is sent round robin.
* If a key is sent, then all message for that key will always go to the same partition.
* A key is basically sent if you need message ordering for a specific field (e.g.: truck\_id)



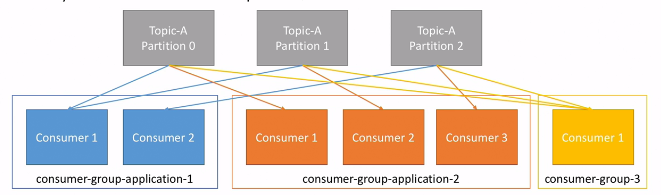
Consumers:

* Consumers read data from a topic (identified by name)
* Consumers know which broker to read from
* In case of broker failures, consumers know how to recover
* Data is read in order within each partition



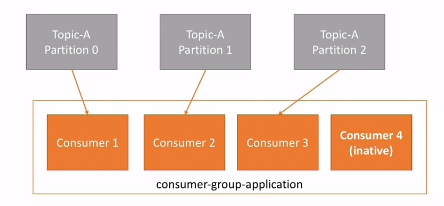
Consumer groups:

* Consumer read data in consumer groups.
* Each consumer within a group reads from exclusive partitions
* If you have more consumers than partitions, some consumers will be inactive.



Note: Consumers will automatically use a Group Coordinator and a consumer coordinator to assign a consumer to a partition.

If we have more consumers than partitions, some consumers will be inactive.



**Consumer Offsets:**

* Kafka stores the offsets at which a consumer group has been reading.
* The offsets committed live in a kafka topic named \_\_consumer\_offsets
* When a consumer in a group has processed data received from Kafka,it should be committing the offsets.
* If a consumer dies, it will be able to read back from where it left off thanks to the committed consumer offsets.



* Consumers choose when to commit offsets.
* There are 3 delivery semantics:
* At most once:

- offset are committed as soon as the message is received.

- if the processing goes wrong the message will be lost (it won’t be read again)

* At least once (Usually preferred):

- offsets are committed after the message is processed.

- If the processing goes wrong, the message will be read again.

- This can result in duplicate processing of messages. Make sure your processing is idempotent (i.e. processing again the messages won’t impact your systems)

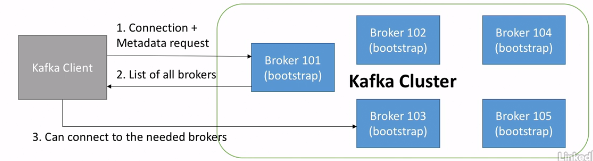
* Exactly once:

- Can be achieved for Kafka => Kafka workflows using Kafka streams API

- For Kafka => External system workflows, use an idempotent consumer.

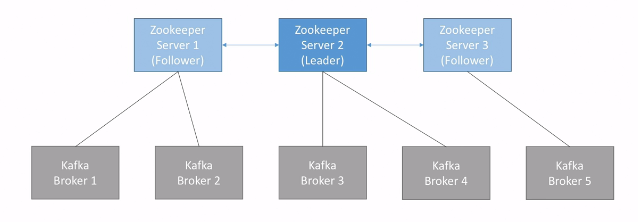
**Kafka Broker Discovery:**

* Every Kafka broker is also called a “Bootstrap server”
* That means that you only need to connect to one broker, and you will be connected to the entire cluster.
* Each broker knows about all brokers, topics and partitions (metadata).



Zookeeper:

* Zookeeper manages brokers (keeps a list of them)
* Zookeeper helps in performing leader election for partitions
* Zookeeper sends notifications to Kafka in case of changes (e.g. new topic, broker dies, broker comes up, delete topics etc.
* Kafka can’t work without zookeeper
* Zookeeper by design operates with an odd number of servers (3, 5, 7 …)
* Zookeeper has a leader (handle writes) the rest of the servers are followers (handle reads)
* Zookeeper does not store consumer offsets with Kafka > v0.10



Kafka Guarantees:

* Messages are appended to a topic-partition in the order they are sent.
* Consumers read messages in the order stored in a topic- partition
* With a replication factor of N, Producers and consumer can tolerate up to N-1 brokers being down.
* This is why a replication factor of 3 is good idea. Allows for one broker to be taken down for maintenance, and for another broker to be taken down unexpectedly.
* As long as the number of partitions remains constant for a topic ( no new partitions), the same key will always go to the same partition.

