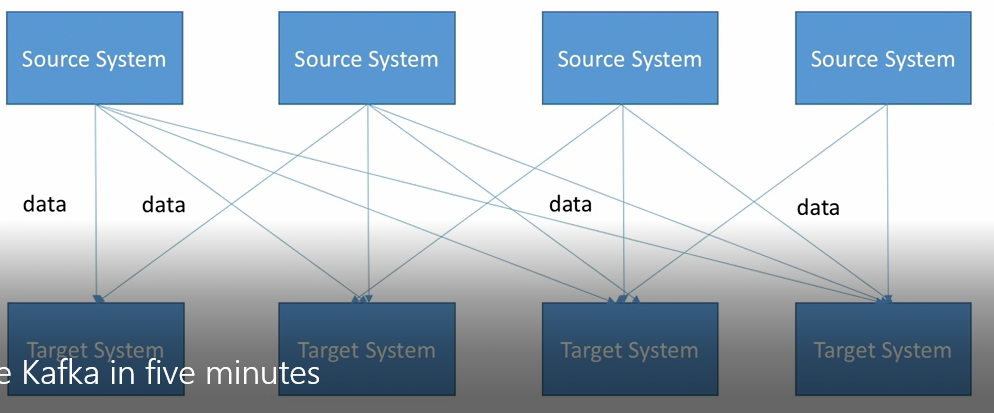
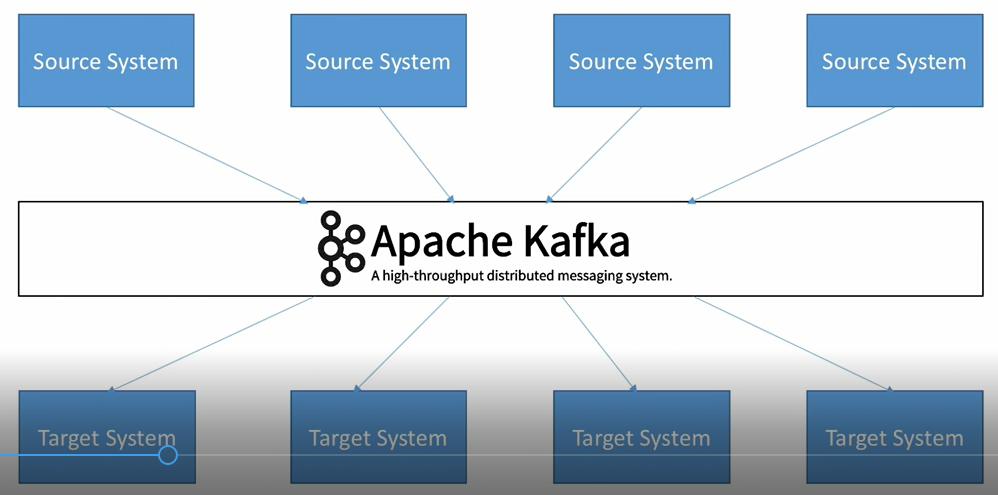
Traditional Mechanism



Suppose, we have 4 sources and 4 destinations between which data needs to be transmitted, then we will have total of 16 different connections.

With Kafka



We will establish just 8 connections.

Use Cases

1. Messaging System
2. Activity Tracking
3. Gathering metrics from many different locations
4. Application logs gathering
5. Stream processing
6. De-coupling of system dependencies
7. Integration with Spark, Flink, Storm, Hadoop and many other Big Data Technologies

Real time applications

1. Netflix uses Kafka to apply recommendations in real time while watching TV shows.
2. 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
3. Linkedin uses Kafka to prevent spam, collect user interactions to make better reommenations in real time

Topics, Offsets and Partitions

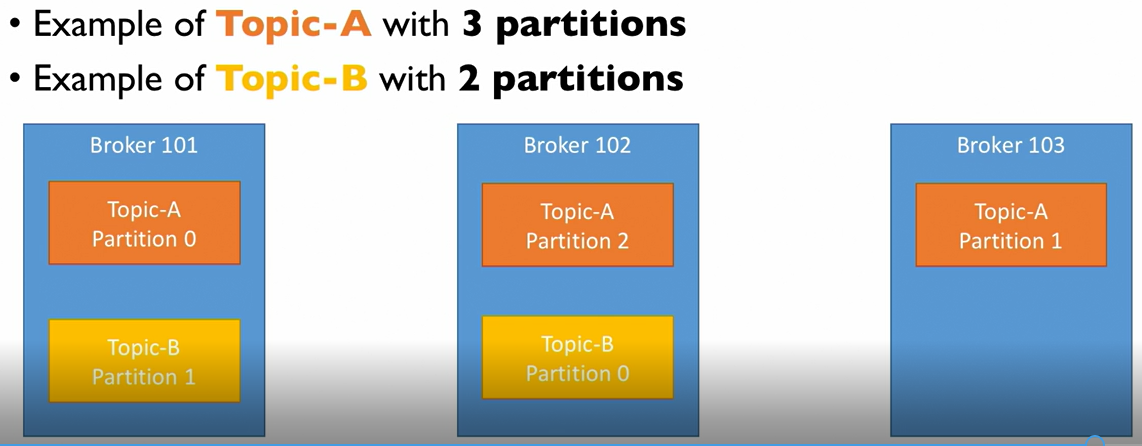
1. Stream of data
2. We can create as many topics as we want
3. A topic is iedntified by its name
4. Topics are split into partitions
5. Each message within a message gets incremental id called offset
6. Data in topic gets deleted after a limited time (default one week)
7. One data is written to a partition, it cannot be changed (immutability)
8. Data is assigned randomly to a partition unless a key is provided



Brokers

1. Kafka cluster is made up of multiple brokers (servers)
2. Each broker is identified by ID(integer)
3. Each broker contains certain topic partitions
4. After connecting to certain broker, we will be connected to entire cluster
5. A good number to start is 3 brokers

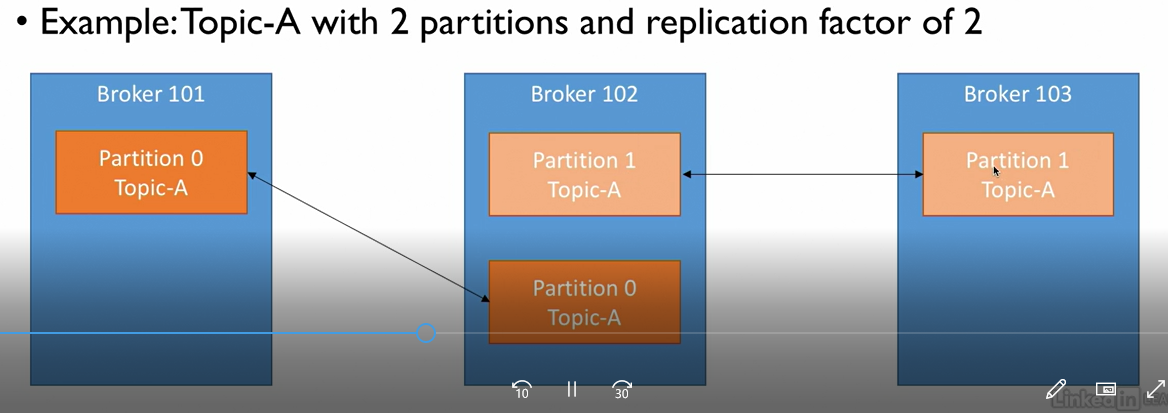
Brokers and Topic

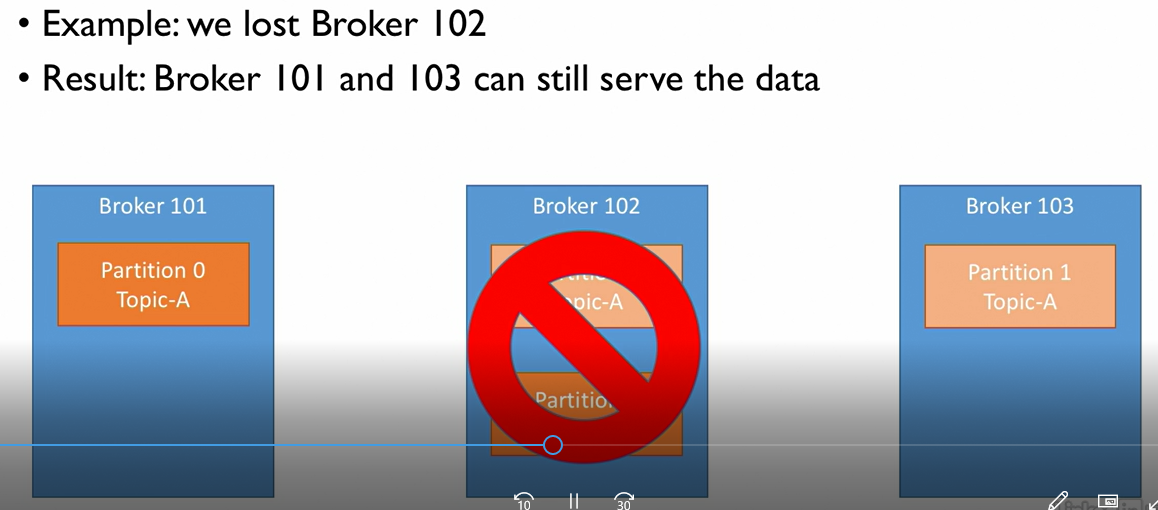


Topic Replication Factor

Replication factor of 2 means we have 2 copies of each data.

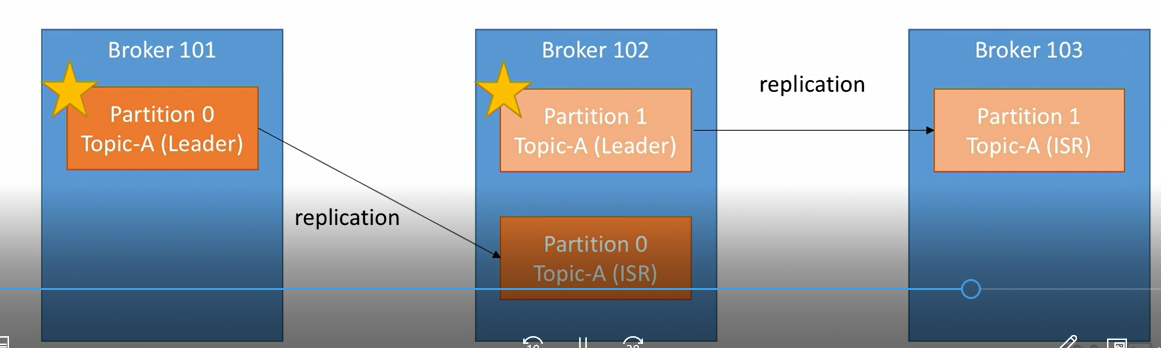
1. Topics should have replication factor>1 (usually between 2 and 3)
2. This way if a broker is down, another broker can serve the data





Leader for a partition

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



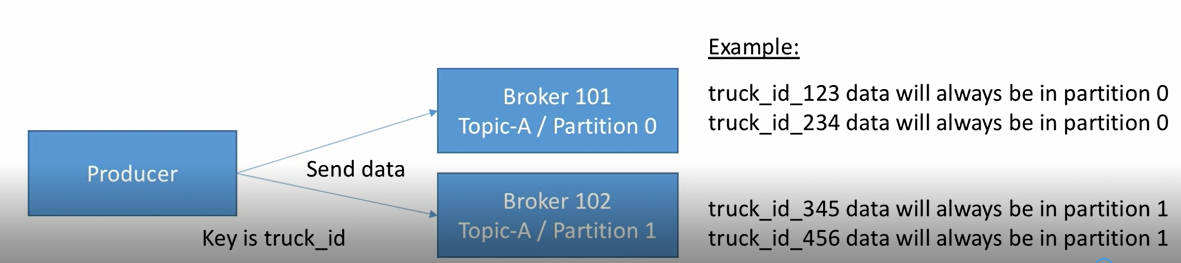
When broker 101 goes down, broker 2 will become leader for partition 0 and when broker 101 is up, it will again try to become leader for partition 0.

Producers

1. Producers write data to topics ( which is made of parititions)
2. Producers automatically knowto which broker and partition to write to
3. In case of broker failures, broker will automatically recover
4. Producers can choose to receive acknowledge of data writes:
   1. acks=0: Producers will not wait for acknowledgement
   2. acks=1: Producers will wait for acknowledgement from leader (limited data loss)
   3. acks=all: Producres will wait for acknowledgement from leader and replicas (no data loss)

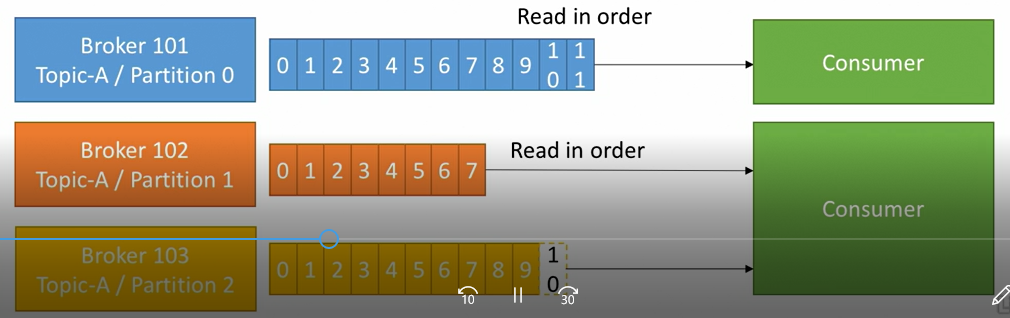
Producers: Message Key

1. Producers can send key along with message
2. Key can be anything like string or integers or something else
3. If key=null, messages produced by producer will be sent to brokers in round robin fashion (broker 101 then 102 then 103…)
4. If key is sent, all messages with same key will always go to same partition



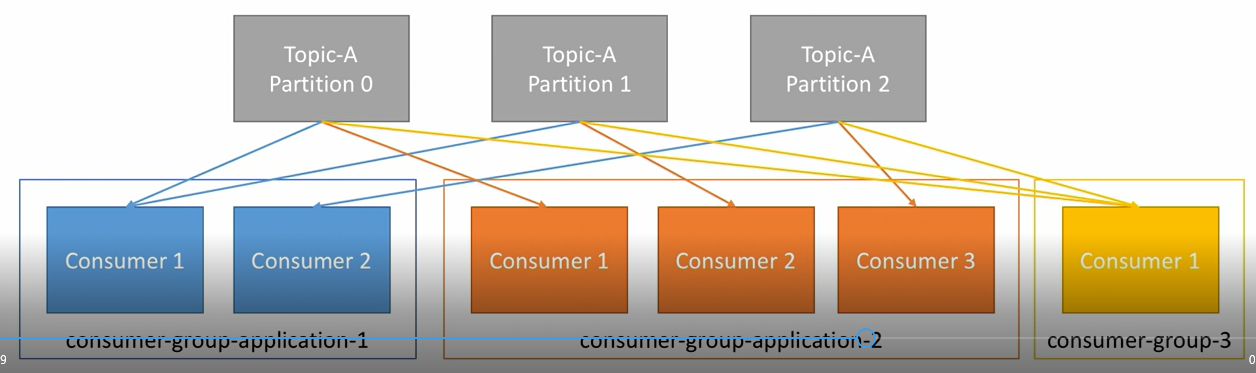
Consumers

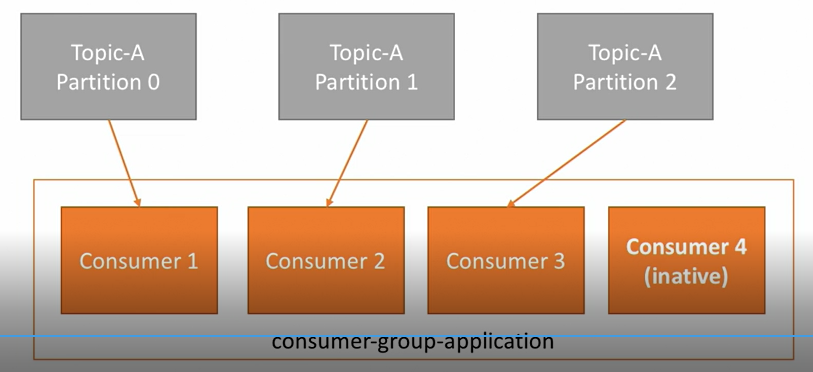
1. Consumers read data from topic
2. Consumers know which broker to read from
3. In case of broker failures, consumers know how to recover
4. Data is read in order within each partitions



Consumer Group

1. Consumers read data in consuer groups
2. Each consumers within a group read from exclusive partitions
3. If we have more consumers then partitions, some consumers will be inactive.
4. Consumers will automatically use a GroupCoordinator and a ConsumerCoordinator to assign a consumers to a partitions





Consumer Offsets

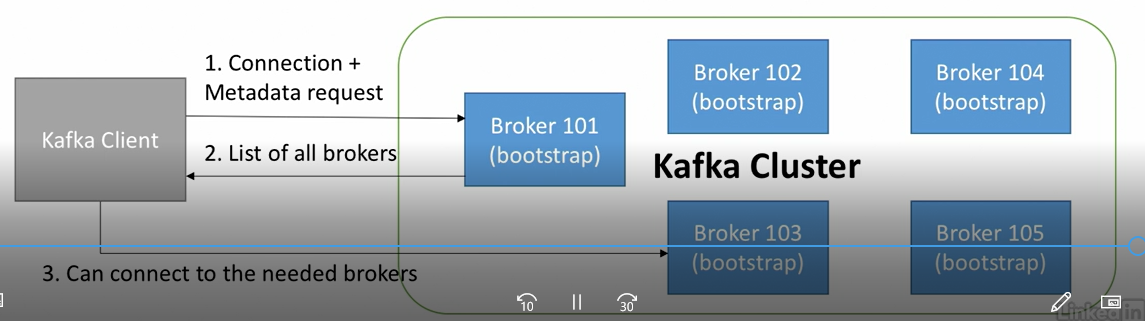
1. Kafka stores offsets at which consumer group has been reading
2. The offsets committed live in a Kafka topic named \_\_consumer\_\_ offsets
3. When a consumer in a group has processed data received from Kafka, it should be committing the offsets.
4. If a consumer dies, it will be able to read back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

Delivery Schemantics for consumers

1. Consumers choose when to commit offset.
2. There are 3 delivery schemantics:
   1. At most once:
      1. Offsets are committed as soon as the messages is received
      2. If processing goes wrong, messages will be lost ( it won’t be read again)
   2. At least once (usually preferred):
      1. Offsets are committed after messages are processed
      2. If processing goes wrong, messages are read again
      3. This can result in duplicate processing of messages. Make sure your processing is idempotent (i.e. processing messages again won’t impact your system)
   3. Extactly once
      1. Can be achieved for Kafka-> Kafka workflows using Kafka Streams API
      2. For Kafka -> External System workflows, use a idempotent consumer

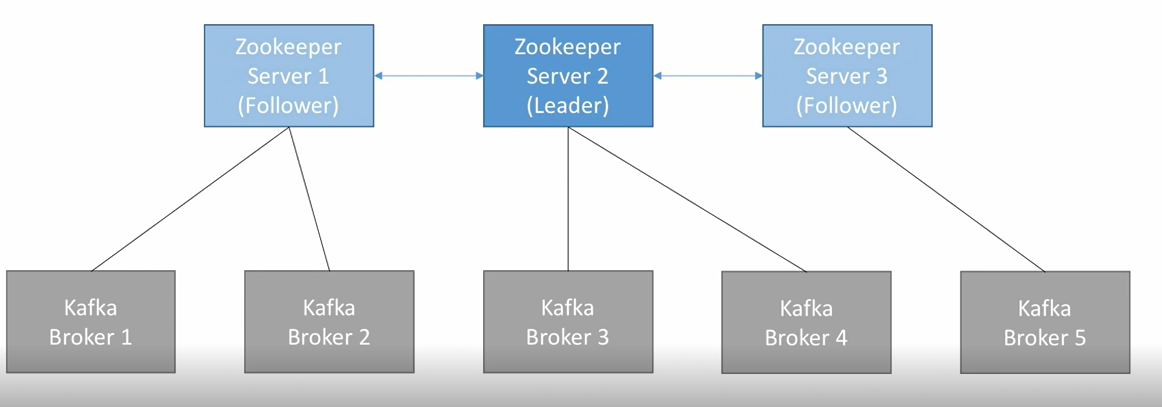
Kafka Broker Discovery

1. Every kafka broker is also called a “bootstrap-server”
2. That means we will only need to connect to one broker and we will be connected to entire cluster.
3. Each broker knows about all brokers, topics and partitions (metadata)



Zookeeper

1. It manages broker (keeps list of them)
2. It hels in performing leader election for partitions
3. Zookeper send notification to Kafka in case of changes (eg. New topic, broker dies, broker comes up, delete topics, etc)
4. Kafka cannot work without zookeeper
5. Zookeeper operates with an odd number odd number os servers(1,3,5,7…)
6. Zookeeper has a leader (handle writes). Rest of the servers are followers (handle reads)
7. Zookeper does not store consumer offsets with Kafka> V0.10



Kafka Guarantees

1. Messages are appended to a topic-partition in the order they are sent
2. Consumers read messages in the order stored in a topic-partition
3. With a replication factor of N, producers and consumers can tolerate upto N-1 brokers being down
4. As long as number of partitions remains constant for a topic, same key will always go to tha same partition.

Kafka Concepts

