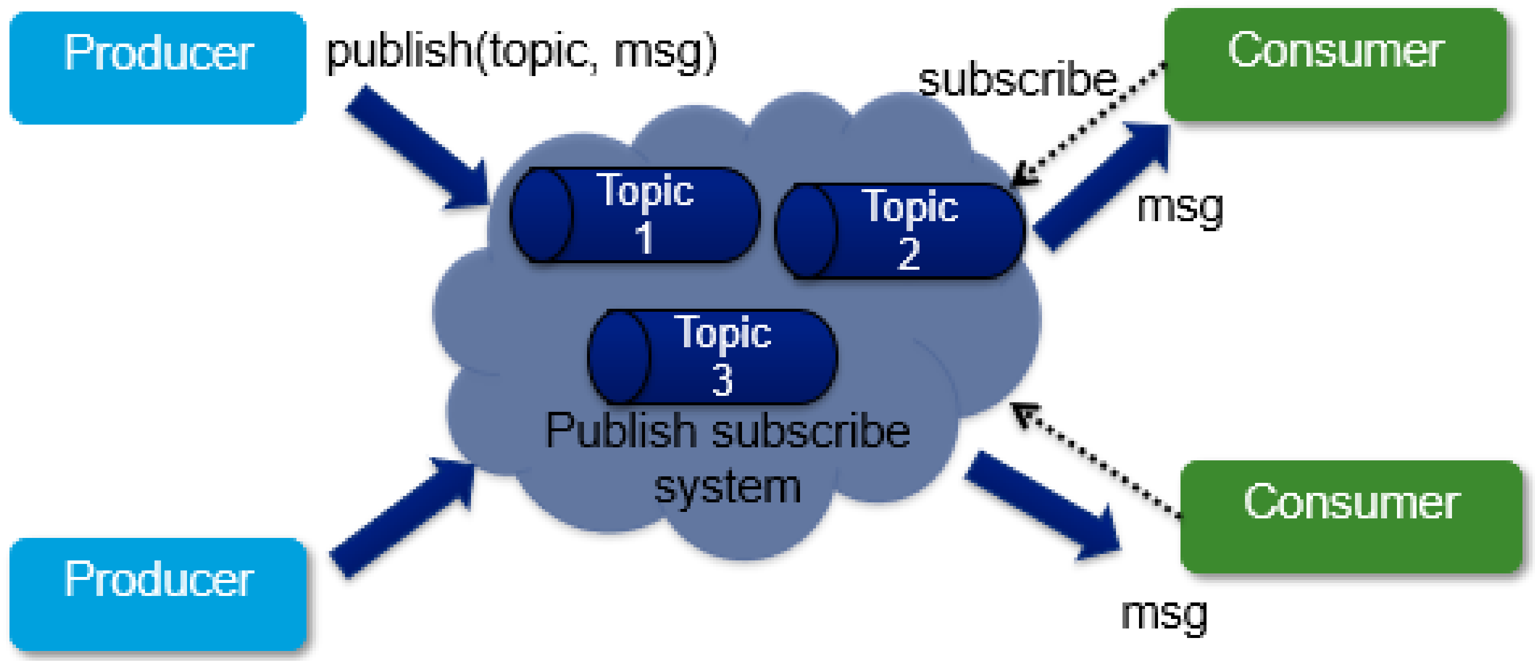
Kafka

Need for streaming based system in Big Data

+ A unified platform for handling all the real-time data feeds a large company might have.
+ High throughput to support high volume event feeds.
+ Support real-time processing of these feeds to create new AND derived feeds.
+ Support large data backlogs to handle periodic ingestion from offline systems.
+ Support low-latency delivery to handle more traditional messaging use cases. Guarantee fault-tolerance in the presence of machine failures.

What is Kafka

Kafka is Distributed,Persistent,Reliable and High throughput Pub Sub based, event based messaging system.



Kafka Concepts

There are one or more servers available in Apache Kafka cluster, basically, these servers (each) are what we call a broker. Brokers are also responsible for maintaining general state information of the system, leader election, etc

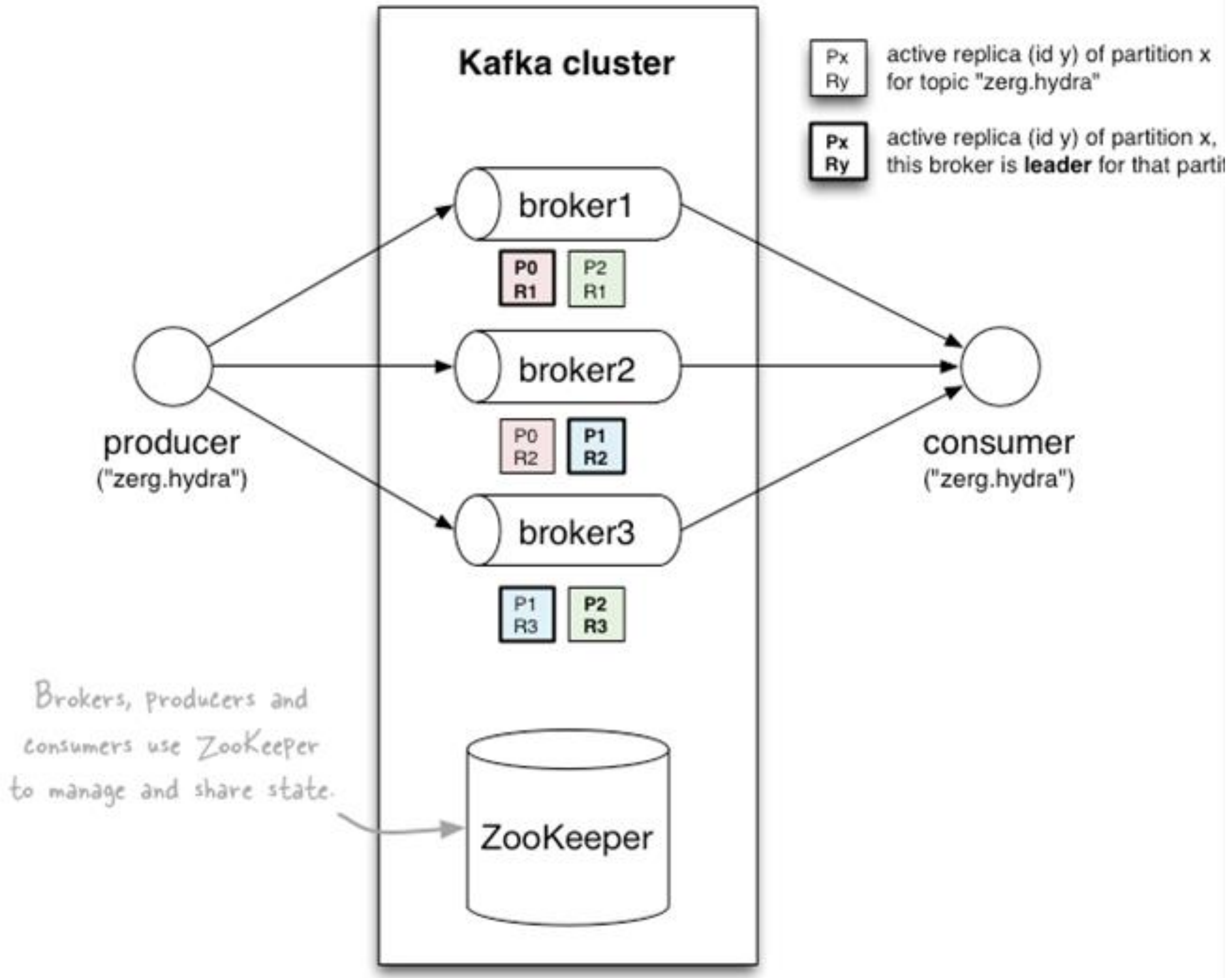
+Producers write data to brokers.

+Consumers read data from brokers.

+All this is distributed.

+Data is stored in topics.

+Topics are split into partitions, which are replicated.


Kafka cluster
Px Ry
active replica (id y) of partition x for topic "zerg.hydra"
Px Ry
active replica (id y) of partition x, this broker is **leader** for that partit
broker1
P0 R1
P2 R1
broker2
P0 R2
P1 R2
broker3
P1 R3
P2 R3
producer ("zerg.hydra")
consumer ("zerg.hydra")
Brokers, producers and consumers use ZooKeeper to manage and share state.
ZooKeeper

More Concepts

- **Partitions**: A topic consists of partitions.Partition: ordered + immutable sequence of messages that is continually appended to – a commit log. Partitions of a topic is configurable. Partitions determines max consumer (group) parallelism.
- **Kafka Log**: A log is nothing different but another way to view a partition. Basically, a data source writes messages to the log. Further, one or more consumers read that data from the log at any time they want.
- **Offset**: messages in the partitions are each assigned a unique (per partition) and sequential id called the offset Consumers track their pointers via (offset, partition, topic) tuples
- **Replicas**: “backups” of a partition They exist solely to prevent data loss. Replicas are never read from, never written to. They do NOT help to increase producer or consumer parallelism! Kafka tolerates (numReplicas – 1) dead brokers before losing data
- **Consumer Group:** Kafka can have multiple consumer process/instance running. Basically, one consumer group will have one unique group-id. Moreover, exactly one consumer instance reads the data from one partition in one consumer group, at the time of reading. Since, there is more than one consumer group, in that case, one instance from each of these groups can read from one single partition. However, there will be some inactive consumers, if the number of consumers exceeds the number of partitions. Let’s understand it with an example if there are 8 consumers and 6 partitions in a single consumer group, that means there will be 2 inactive consumers.

Role of Zookeeper in Kafka

Apache Zookeeper serves as the coordination interface between the Kafka brokers and consumers.

+ Also, we can say it is a distributed configuration and synchronization service.
+ Basically, ZooKeeper cluster shares the information with the Kafka servers.
+ Moreover, Kafka stores basic metadata information in ZooKeeper Kafka, such as topics, brokers, consumer offsets (queue readers) and so on.
+ In addition, failure of Kafka Zookeeper/broker does not affect the Kafka cluster. It is because the critical information which is stored in the ZooKeeper is replicated across its ensembles. Then Kafka restores the state as ZooKeeper restarts, leading to zero downtime for Kafka.

Producer and Consumer Kafka

+You use Kafka “producers” to write data to Kafka brokers.

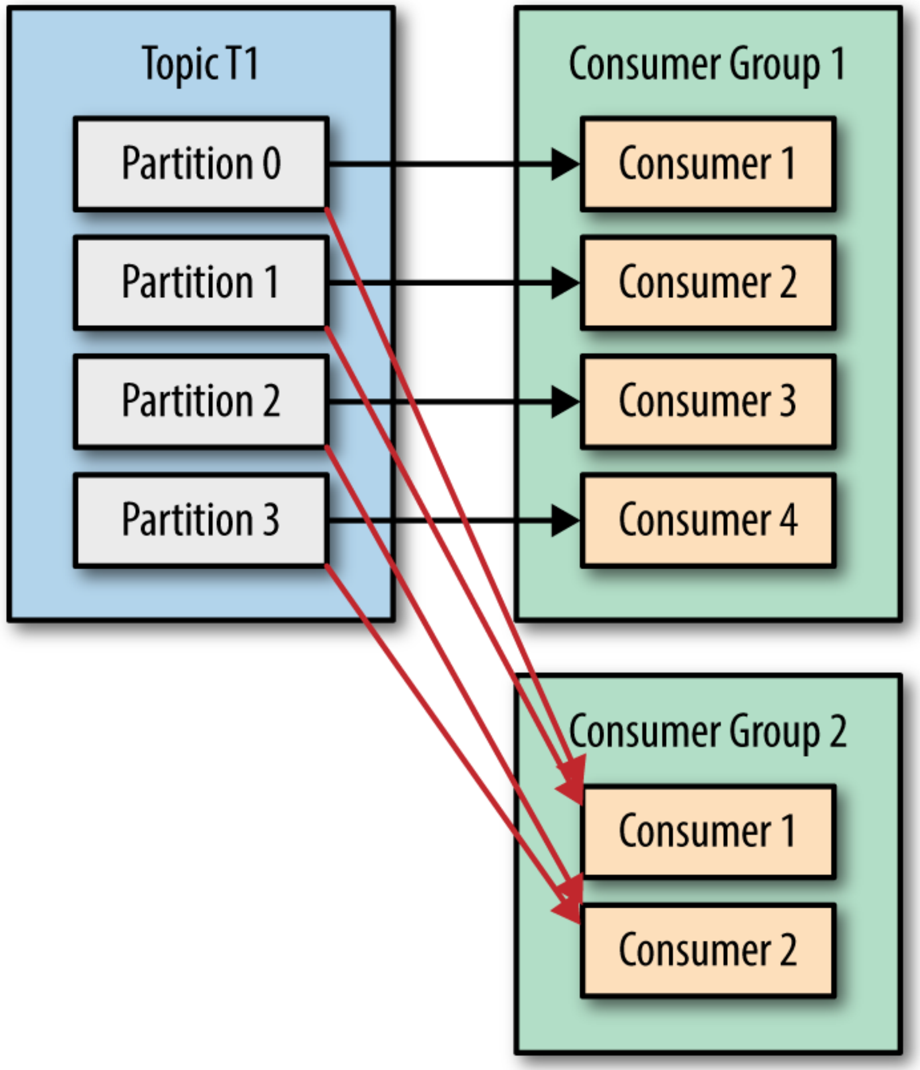
+Available for JVM (Java, Scala), C/C++, Python, Ruby, etc.

+ In order to send messages asynchronously to a topic, KafkaProducer class provides send method. So, the signature of send() is

+ After creating a Kafka Producer to send messages to Apache Kafka cluster. Now, we are creating a Kafka Consumer to consume messages from the Kafka cluster.

+ Kafka Consumer subscribes to one or more topics in the Kafka cluster then further feeds on tokens or messages from the Kafka Topics. In addition, using Heartbeat we can know the connectivity of Consumer to Kafka Cluster. However, let’s define Heartbeat. It is set up at Consumer to let Zookeeper or Broker Coordinator know if the Consumer is still connected to the Cluster. So, Kafka Consumer is no longer connected to the Cluster, if the heartbeat is absent. In that case, the Broker Coordinator has to re-balance the load. Moreover, Heartbeat is an overhead to the cluster. Also, by keeping the data throughput and overhead in consideration, we can configure the interval at which the heartbeat is at Consumer.

Partition to Consumer Group Mapping



Within same group: **NO**

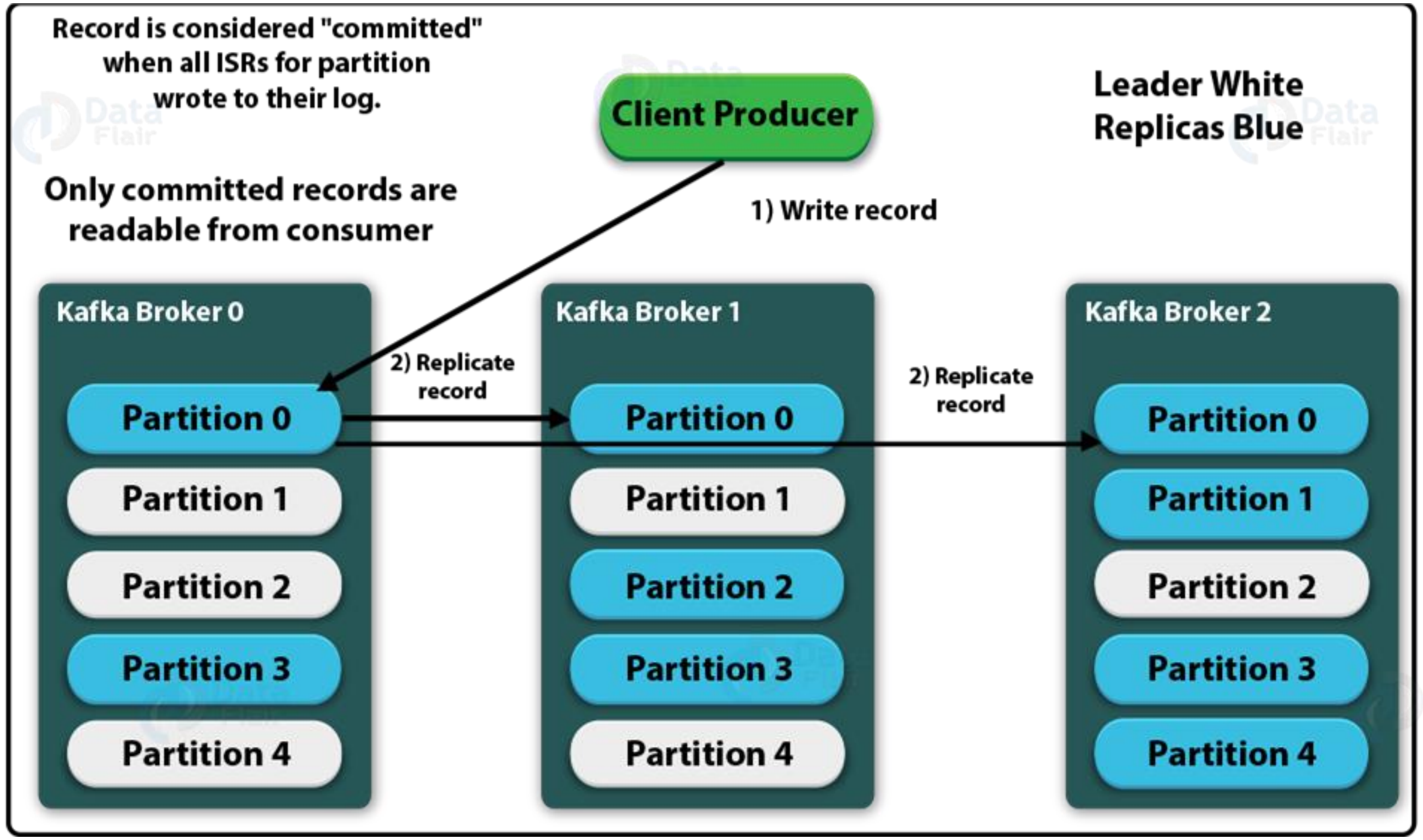
- Two consumers (*Consumer 1, 2*) within the same group (*Group 1*) **CAN NOT** consume the same message from partition (*Partition 0*).

Across different groups: **YES**

- Two consumers in two groups (*Consumer 1* from *Group 1*, *Consumer 1* from *Group 2*) **CAN** consume the same message from partition (*Partition 0*).

Kafka Topic Partition Replication

- For the purpose of fault tolerance, Kafka can perform replication of partitions across a configurable number of Kafka servers. Basically, there is a leader server and zero or more follower servers in each partition. Also, for a partition, leaders are those who handle all read and write requests.
- However, if the leader dies, the followers replicate leaders and take over. Additionally, for parallel consumer handling within a group, Kafka uses also uses partitions.
- he broker which has the partition leader handles all reads and writes of records. Moreover, to the leader partition to followers (node/partition pair), Kafka replicates writes. On defining the term ISR, a follower which is in-sync is what we call an ISR (in-sync replica). Although, Kafka chooses a new *ISR* as the new leader if a partition leader fails.



Acks

- In Kafka, a message is considered committed when “any required” ISR (in-sync replicas) for that partition have applied it to their data log.
- Message acking is about conveying this “Yes, committed!” information back from the brokers to the producer client.
- Exact meaning of “any required” is defined by request.required.acks.
- Only producers must configure acking Exact behavior is configured via request.required.acks, which determines when a produce request is considered completed. Allows you to trade latency (speed) <-> durability (data safety).
- Consumers: Acking and how you configured it on the side of producers do not matter to consumers because only committed messages are ever given out to consumers. They don’t need to worry about potentially seeing a message that could be lost if the leader fails.
- Typical values of **request.required.acks** 0: producer never waits for an ack from the broker.Gives the lowest latency but the weakest durability guarantees.
- 1: producer gets an ack after the leader replica has received the data. Gives better durability as the we wait until the lead broker acks the request. Only msgs that were written to the now-dead leader but not yet replicated will be lost.
- -1: producer gets an ack after all ISR have received the data. Gives the best durability as Kafka guarantees that no data will be lost as long as at least one ISR remains.
- Beware of interplay with request.timeout.ms!
- “The amount of time the broker will wait trying to meet the `request.required.acks` requirement before sending back an error to the client.”

Producer and Consumer Kafka Console Utils

```
cd /usr/hdp/current/kafka-broker
```

List Topics

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Create Topic

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic ineurontopic
```

Producer to a topic

```
bin/kafka-console-producer.sh --broker-list 172.18.0.2:6667 --topic ineurontopic
```

Consumer from a Topic

```
bin/kafka-console-consumer.sh --bootstrap-server 172.18.0.2:6667 --topic ineurontopic --from-beginning
```

Documentation :*https://gerardnico.com/dit/kafka/kafka-console-consumer*

Class Notes

/FileStore/newdata.csv

/databricks-datasets/

%sh

mv /dbfs/FileStore/newdata.csv /dbfs/databricks-datasets/

reduce - pull the entire dataset down into a single location because it is reducing to one final value.

[1,2,3,4]

rdd.reduce(somefunction) ---> N1,N2,N3....---> Single python array result into Driver

[(1,2),(2,3),(2,4),(1,2).....] ->N1,N2,N3....-->[(1,4),(2,7)]

pairedrdd.reduceByKey(somefunction) - one value for each key. And since this action can be run on each machine locally first then it can remain an RDD and have further transformations done on its dataset.

Enterprise Data Architecture

Airflow

- executor mem
- driver mem
- num

Big.

Sources → Ingestion tools → Processing (ETL) → Consumers.

Hortonworks
Cloudera
Emr | GCP.

Cloudera
DA

Structured
- CSV
- RDBMS
- "
- Erp data

Batch
10 mins

SQOOP — High volume, M/R, Dir.

Spark JDBC — Less volume.

spark.read.jdbc(...)
connection string

Spark — SQL, DF, RDD, stream, ML.

Hive.

M/R — Legacy
- very unstructured data

Tableau
① PowerBI
② files Extract
③ Data Scien/ M.L. Python/R.

semi/Unstructured
- audio
- video
- IoT
- NoSQL
- json
- xml ...

NRT
5-10 mins

Realtime
<5 mins

Kafka

Hive ? <<<
① Legacy
② OLAP — Analytical processing on structured data.

Ease of Memory Utilization
Better YARN utilization