Protocol—how the data is transported (TCP, HTTP, REST, FTP, JDBC . . .)

Data format—how the data is parsed (binary, csv, json, avro)

Data schema & evolution—how the data is shaped and may change

Apache Kafka

Decoupling of data streams & systems

Integration with spark, flink, Hadoop,

Topics, partitions, and offsets

Topics-a particular stream of data, similar to a table in a database

You can have as many topics as you want

A topic is identified by its name

Topics are split in partitions

Partition 0, 1, 2

Each partition is ordered

Each message within a partition gets an incremental id, called offset

Offset only have a meaning for a specific partition.

Order is guaranteed only within a partition (not across partitions).

Data is kept only for a limited time.

Once the data is written to a partition, it can’t be changed (immutability)

Data is assigned randomly to a partition unless a key is provided.

Brokers and Topics

Kafka cluster—is composed of multiple brokers (servers)

Each broker is identified with its ID(integer)

Each broker contains certain topic partitions

Kafka is distributed

After connecting to any broker(called a bootstrap broker), you will be connected to the entire cluster

A good number to get started is 3 brokers

When you create a topic, it will be distributed across your brokers.

Topic Replication

Topics should have a replication factor greater than 1 (between 2 and 3)

That way, if a broker is down, another broker can serve the data

Other brokers have a copy of data, if you lose a broker

At any time only One broker can be a leader for a give partition

Only that leader can receive and serve data for a partition

The other brokers will synchronize the data

Each partition has one leader and multiple ISR (in-sync replica)

Producers and Keys

How do we get data into Kafka

Producers write data to topics (which is made of partitions)

Producers automatically know to which broker and partition to write to

In case of broker failures, producers will automatically recover

Producer sends data to broker 101, 102, 103. The load is balanced to many brokers thanks to the number of partitions. Round robin.

Producers can choose to receive acknowledgment of data writes (confirmation)

acks=0 Producers won’t wait for acknowledgment (possible data loss)

acks=1 is default, producer will wait for leader acknowledgment (limited data loss)

acks=all leader and replicas acknowledgment (no data loss)

Producers can choose to send a key with the message(string, number, etc.)

If key=null, data is sent round robin across partitions(broker 1010 then 102, then 103)

If a key is sent, then all messages for that key will always go to the same partition

A key is basically sent if you need message ordering for a specific field (ex: 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 partitions

Data is read in order within each partitions

Consumer Groups

Consumers read data in consumer groups

Each consumer within a group reads from exclusive partitions

In you have more consumers than partitions, some consumers will be inactive

If you have a high number of consumers, then you will need a high number of partitions

Consumer Offsets

Kafka stores the offsets at which a consumer group has been reading

Similar to Checkpointing or bookmarking

The offsets commited live ina 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. It will tell you where to start your read from.

**Delivery semantics for consumers**

Consumers choose when to commit offsets.

There are 3 delivery semantics.

At most once—offsets 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 (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)

Kafka cluster with 5 brokers, for example, kafka client connects to broker 101 (will do a metadata request, here’s a list of all brokers, it knows which broker it needs to connect with)

You only need to connect to 1 broker and not 100 brokers, for instance.

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 (new topic, broker dies, broker comes up, delete topics)

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 > v.0.10)

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 consumers can tolerate up to N-1 brokers being down

This is why a replication factor of 3 is a good idea

As long as the number of partitions remains constant for a topic, the same key will always go to the same partition

Q1.

Kafka topics . . . can have as many partitions as desired.

Question 2:

**Offsets are only relevant at the level of . . . the topic -partition.**

Question 3:

**Once sent to a topic, a message can be modified. False**

data is immutable and can't be modified !

Question 4:

**Brokers are identified by . . . an id number.**

Question 5:

**Every brokers . . . contain only a subset of the topics and the partitions.**

Question 6:

**If a topic has a replication factor of 3. Each partition will live on 3 different brokers.**

**If a topic has a replication factor of 3, what maximum number of brokers can be stopped without impacting the topic availability?**

2

**Each partition can only have 1 leader, and multiple replicas. True**

Question 9:

**To produce data to a topic, a producer must provide the Kafka client with...**

Very important: you only need to connect to one broker (any broker) and just provide the topic name you want to write to. Kafka Clients will route your data to the appropriate brokers and partitions for you!

**To get acknowledgement of writes to only the leader, we need to use the config...**

**acks=1**

**To read data from a topic, the following configuration is needed for the consumers**

Very important: you only need to connect to one broker (any broker) and just provide the topic name you want to read from. Kafka will route your calls to the appropriate brokers and partitions for you!

**Two consumers that have the same group.id (consumer group id) will read from mutually exclusive partitions**. True

**Kafka Consumer Offsets are stored in...**

Kafka this is the case since Kafka 0.9, in the topic \_\_consumer\_offsets

Kafka-topics CLI needs to connect to zookeeper.

**The kafka-console-producer CLI needs to connect to** Kafka. --broker-list (Kafka) is the right option

Question 3:

**If I produce to a topic that does not exist, by default It creates a warning (not an error)**

by default Kafka automatically creates topics

Question 4:

**When a topic is auto-created, how many partitions and replication factor does it have by default? Partition: 1 and replication factor: 1**

Question 5:

**kafka-console-consumer uses a random group id number.** try running kafka-consumer-groups --list to see!

Question 6:

**I should override the group.id for kafka-console-consumer using**

**--group mygroup**

Question 7:

**I perform operations on the consumer offsets using**

**kafka-consumer-groups**

kafka distributes partitions across the servers.

Horizontal scaling means more servers.

You will only read and write to/from the leader.