Apache Kafka

Apache Kafka is an open-source stream processing platform developed by the Apache Software Foundation written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.

1)What are the advantages of using Apache Kafka?

Answer) The Advantages of using Apache Kafka are as follows:

High Throughput:The design of Kafka enables the platform to process messages at very fast speed. The processing rates in Kafka can exceed beyond 100k/seconds. The data is processed in a partitioned and ordered fashion.

Scalability:The scalability can be achieved in Kafka at various levels. Multiple producers can write to the same topic. Topics can be partitioned. Consumers can be grouped to consume individual partitions.

Fault Tolerance:Kafka is a distributed architecture which means there are several nodes running together to serve the cluster. Topics inside Kafka are replicated. Users can choose the number of replicas for each topic to be safe in case of a node failure. Node failure in cluster won't impact. Integration with Zookeeper provides producers and consumers accurate information about the cluster. Internally each topic has its own leader which takes care of the writes. Failure of node ensures new leader election.

Durability:Kafka offers data durability as well. The message written in Kafka can be persisted. The persistence can be configured. This ensures re-processing, if required, can be performed.

2)Which are the elements of Kafka?

Answer)The most important elements of Kafka:
Topic – It is the bunch of similar kind of messages
Producer – using this one can issue communications to the topic
Consumer – it endures to a variety of topics and takes data from brokers.
Brokers – this is the place where the issued messages are stored

3)What is Kafka Logs?

Answer)An important concept for Apache Kafka is "log". This is not related to application log or system log. This is a log of the data. It creates a loose structure of the data which is consumed by Kafka. The notion of "log" is an ordered, append-only sequence of data. The data can be anything because for Kafka it will be just an array of bytes.

4)Explain What Is Zookeeper In Kafka? Can We Use Kafka Without Zookeeper?

Answer)Zookeeper is an open source, high-performance co-ordination service used for distributed applications adapted by Kafka.

No, it is not possible to bye-pass Zookeeper and connect straight to the Kafka broker. Once the Zookeeper is down, it cannot serve client request.

Zookeeper is basically used to communicate between different nodes in a cluster

In Kafka, it is used to commit offset, so if node fails in any case it can be retrieved from the previously committed offset

Apart from this it also does other activities like leader detection, distributed synchronization, configuration management, identifies when a new node leaves or joins, the cluster, node status in real time, etc.

5)Why Replication Is Required In Kafka?

Answer)Replication of message in Kafka ensures that any published message does not lose and can be consumed in case of machine error, program error or more common software upgrades.

6)What is the role of offset in Kafka?

Answer)Offset is nothing but an unique id that is assigned to the partitions. The messages are contained in this partitions. The important aspect or use of offset is that it identifies every message with the id which is available within the partition.

7)What is a consumer group?

Answer)A consumer group is nothing but an exclusive concept of Kafka.

Within each and every Kafka consumer group, we will have one or more consumers who actually consume subscribed topics.

8)What is the core API in Kafka?

Answer)They are four main core API's:
1. Producer API
2. Consumer API
3. Streams API

4. Connector API

9)Explain the functionality of producer API in Kafka?

Answer)The producer API is responsible where it will allow the application to push a stream of records to one of the Kafka topics.

10)Explain the functionality of Consumer API in Kafka?

Answer)The Consumer API is responsible where it allows the application to receive one or more topics and at the same time process the stream of data that is produced.

11)Explain the functionality of Streams API in Kafka?

Answer)The Streams API is responsible where it allows the application to act as a processor and within the process, it will be effectively transforming the input streams to output streams.

12)Explain the functionality of Connector API in Kafka?

Answer)The Connector API is responsible where it allows the application to stay connected and keeping a track of all the changes that happen within the system. For this to happen, we will be using reusable producers and consumers which stays connected to the Kafka topics.

13)Explain what is a topic?

Answer)A topic is nothing but a category classification or it can be a feed name out of which the records are actually published. Topics are always classified, the multi subscriber.

14)What is the purpose of retention period in Kafka cluster?

Answer)Within the Kafka cluster, it retains all the published records. It doesn't check whether they have been consumed or not. Using a configuration setting for the retention period, the records can be discarded. The main reason to discard the records from the Kafka cluster is that it can free up some space.

15)Mention what is the maximum size of the message does Kafka server can receive?

Answer)The maximum size of the message that Kafka server can receive is 1000000 bytes.

16)Explain how you can improve the throughput of a remote consumer?

Answer)If the consumer is located in a different data center from the broker, you may require to tune the socket buffer size to amortize the long network latency.

17)Is it possible to get the message offset after producing?

Answer)You cannot do that from a class that behaves as a producer like in most queue systems, its role is to fire and forget the messages. The broker will do the rest of the work like appropriate metadata handling with id's, offsets, etc.

As a consumer of the message, you can get the offset from a Kafka broker. If you gaze in the SimpleConsumer class, you will notice it fetches MultiFetchResponse objects that include offsets as a list. In addition to that, when you iterate the Kafka Message, you will have MessageAndOffset objects that include both, the offset and the message sent.

18)Explain the concept of Leader and Follower?

Answer)Every partition in Kafka has one server which plays the role of a Leader, and none or more servers that act as Followers. The Leader performs the task of all read and write requests for the partition, while the role of the Followers is to passively replicate the leader. In the event of the Leader failing, one of the Followers will take on the role of the Leader. This ensures load balancing of the server.

19)If a Replica stays out of the ISR for a long time, what does it signify?

Answer)It means that the Follower is unable to fetch data as fast as data accumulated by the Leader.

20)How do you define a Partitioning Key?

Answer)Within the Producer, the role of a Partitioning Key is to indicate the destination partition of the message. By default, a hashing-based Partitioner is used to determine the partition ID given the key. Alternatively, users can also use customized Partitions.

21)In the Producer, when does QueueFullException occur?

Answer)QueueFullException typically occurs when the Producer attempts to send messages at a pace that the Broker cannot handle. Since the Producer doesn't block, users will need to add enough brokers to collaboratively handle the increased load.

22)Kafka Stream application failed to start, with the a rocksDB exception raised as "java.lang.ExceptionInInitializerError.. Unable to load the RocksDB shared libraryjava". How to resolve this?

Answer)Streams API uses RocksDB as the default local persistent key-value store. And RocksDB JNI would try to statically load the sharedlibs into java.io.tmpdir. On Unix-like platforms, the default value of this system environment property is typically /tmp, or /var/tmp; On Microsoft Windows systems the property is typically C:\\WINNT\\TEMP.

If your application does not have permission to access these directories (or for Unix-like platforms if the pointed location is not mounted), the above error would be thrown. To fix this, you can either grant the permission to access this directory to your applications, or change this property when executing your application like java -Djava.io.tmpdir=[].

23)Have you encountered Kafka Stream application's memory usage keeps increasing when running until it hits an OOM. Why any specific reason?

Answer)The most common cause of this scenario is that you did not close an iterator from the state stores after completed using it. For persistent stores like RocksDB, an iterator is usually backed by some physical resources like open file handlers and in-memory caches. Not closing these iterators would effectively causing resource leaks.

24)Extracted timestamp value is negative, which is not allowed. What does this mean in Kafka Streaming?

Answer)This error means that the timestamp extractor of your Kafka Streams application failed to extract a valid timestamp from a record. Typically, this points to a problem with the record (e.g., the record does not contain a timestamp at all), but it could also indicate a problem or bug in the timestamp extractor used by the application.

25)How to scale a Streams app,that is increase number of input partitions?

Answer)Basically, Kafka Streams does not allow to change the number of input topic partitions during its life time. If you stop a running Kafka Streams application, change the number of input

topic partitions, and restart your app it will most likely break with an exception as described in FAQ "What does exception "Store someStoreName's change log (someStoreName-changelog) does not contain partition someNumber mean? It is tricky to fix this for production use cases and it is highly recommended to not change the number of input topic partitions (cf. comment below). For POC/demos it's not difficult to fix though.

In order to fix this, you should reset your application using Kafka's application reset tool: Kafka Streams Application Reset Tool.

26)I get a locking exception similar to "Caused by: java.io.IOException: Failed to lock the state directory: /tmp/kafka-streams/app-id/0_0". How can I resolve this?

Answer)You can apply the following workaround:

switch to single threaded execution

if you want to scale your app, start multiple instances (instead of going multi-threaded with one instance)

if you start multiple instances on the same host, use a different state directory (state.dir config parameter) for each instance (to "isolate" the instances from each other)

It might also be necessary, to delete the state directory manually before starting the application. This will not result in data loss – the state will be recreated from the underlying changelog topic.0

27)How should I set metadata.broker.list?

Answer)The broker list provided to the producer is only used for fetching metadata. Once the metadata response is received, the producer will send produce requests to the broker hosting the corresponding topic/partition directly, using the ip/port the broker registered in ZK. Any broker can serve metadata requests. The client is responsible for making sure that at least one of the brokers in metadata.broker.list is accessible. One way to achieve this is to use a VIP in a load balancer. If brokers change in a cluster, one can just update the hosts associated with the VIP.

28)Why do I get QueueFullException in my producer when running in async mode?

Answer)This typically happens when the producer is trying to send messages quicker than the broker can handle. If the producer can't block, one will have to add enough brokers so that they jointly can handle the load. If the producer can block, one can set queue.enqueueTimeout.ms in producer config to -1. This way, if the queue is full, the producer will block instead of dropping messages.

29)Why are my brokers not receiving producer sent messages?

Answer)This happened when I tried to enable gzip compression by setting compression.codec to 1. With the code change, not a single message was received by the brokers even though I had called producer.send() 1 million times. No error printed by producer and no error could be found in broker's kafka-request.log. By adding log4j.properties to my producer's classpath and switching the log level to DEBUG, I captured the java.lang.NoClassDefFoundError: org/xerial/snappy/SnappyInputStream thrown at the producer side. Now I can see this error can be resolved by adding snappy jar to the producer's classpath.

30)Is it possible to delete a topic?

Answer)Deleting a topic is supported since 0.8.2.x. You will need to enable topic deletion (setting delete.topic.enable to true) on all brokers first.

31)Why does Kafka consumer never get any data?

Answer)By default, when a consumer is started for the very first time, it ignores all existing data in a topic and will only consume new data coming in after the consumer is started. If this is the case, try sending some more data after the consumer is started. Alternatively, you can configure the consumer by setting auto.offset.reset to earliest for the new consumer in 0.9 and smallest for the old consumer.

32)Why does Kafka consumer get InvalidMessageSizeException?

Answer)This typically means that the fetch size of the consumer is too small. Each time the consumer pulls data from the broker, it reads bytes up to a configured limit. If that limit is smaller than the largest single message stored in Kafka, the consumer can't decode the message properly and will throw an InvalidMessageSizeException. To fix this, increase the limit by setting the property fetch.size (0.7) / fetch.message.max.bytes (0.8) properly in config/consumer.properties. The default fetch.size is 300,000 bytes. For the new consumer in 0.9, the property to adjust is max.partition.fetch.bytes, and the default is 1MB.

33)Should I choose multiple group ids or a single one for the consumers?

Answer)If all consumers use the same group id, messages in a topic are distributed among those consumers. In other words, each consumer will get a non-overlapping subset of the messages. Having more consumers in the same group increases the degree of parallelism and the overall throughput of consumption. See the next question for the choice of the number of consumer instances. On the other hand, if each consumer is in its own group, each consumer will get a full copy of all messages.

34)Why some of the consumers in a consumer group never receive any message?

Answer)Currently, a topic partition is the smallest unit that we distribute messages among consumers in the same consumer group. So, if the number of consumers is larger than the total number of partitions in a Kafka cluster (across all brokers), some consumers will never get any data. The solution is to increase the number of partitions on the broker.

35)Why are there many rebalances in Kafka consumer log?

Answer)A typical reason for many rebalances is the consumer side GC. If so, you will see Zookeeper session expirations in the consumer log (grep for Expired). Occasional rebalances are fine. Too many rebalances can slow down the consumption and one will need to tune the java GC setting.

36)Why messages are delayed in kafka consumer?

Answer)This could be a general throughput issue. If so, you can use more consumer streams (may need to increase # partitions) or make the consumption logic more efficient.

Another potential issue is when multiple topics are consumed in the same consumer connector. Internally, we have an in-memory queue for each topic, which feed the consumer iterators. We have a single fetcher thread per broker that issues multi-fetch requests for all topics. The fetcher thread iterates the fetched data and tries to put the data for different topics into its own in-memory queue. If one of the consumer is slow, eventually its corresponding in-memory queue will be full. As a result, the fetcher thread will block on putting data into that queue. Until that queue has more space, no data will be put into the queue for other topics. Therefore, those other topics, even if they have less volume, their consumption will be delayed because of that. To address this issue, either making sure that all consumers can keep up, or using separate consumer connectors for different topics.

37)How to improve the throughput of a remote consumer?

Answer)If the consumer is in a different data center from the broker, you may need to tune the socket buffer size to amortize the long network latency. Specifically, for Kafka 0.7, you can increase socket.receive.buffer in the broker, and socket.buffersize and fetch.size in the consumer. For Kafka 0.8, the consumer properties are socket.receive.buffer.bytes and fetch.message.max.bytes.

38)How to consume large messages?

Answer)First you need to make sure these large messages can be accepted at Kafka brokers. The broker property message.max.bytes controls the maximum size of a message that can be accepted at the broker, and any single message (including the wrapper message for compressed message set) whose size is larger than this value will be rejected for producing. Then you need to make sure consumers can fetch such large messages from brokers. For the old consumer, you should use the property fetch.message.max.bytes, which controls the maximum number of bytes a consumer issues in one fetch. If it is less than a message's size, the fetching will be blocked on that message keep retrying. The property for the new consumer is max.partition.fetch.bytes.

39)How does Kafka depend on Zookeeper?

Answer)Starting from 0.9, we are removing all the Zookeeper dependency from the clients (for details one can check this page). However, the brokers will continue to be heavily depend on Zookeeper for:
Server failure detection.
Data partitioning.
In-sync data replication.

Once the Zookeeper quorum is down, brokers could result in a bad state and could not normally serve client requests, etc. Although when Zookeeper quorum recovers, the Kafka brokers should be able to resume to normal state automatically, there are still a few corner cases the they cannot and a hard kill-and-recovery is required to bring it back to normal. Hence it is recommended to closely monitor your zookeeper cluster and provision it so that it is performant.

Also note that if Zookeeper was hard killed previously, upon restart it may not successfully load all the data and update their creation timestamp. To resolve this you can clean-up the data directory of the Zookeeper before restarting (if you have critical metadata such as consumer offsets you would need to export / import them before / after you cleanup the Zookeeper data and restart the server).

40)Why can't Kafka consumers/producers connect to the brokers? What could be the reason?

Answer)When a broker starts up, it registers its ip/port in ZK. You need to make sure the registered ip is consistent with what's listed in metadata.broker.list in the producer config. By default, the registered ip is given by InetAddress.getLocalHost.getHostAddress. Typically, this should return the real ip of the host. However, sometimes (e.g., in EC2), the returned ip is an internal one and can't be connected to from outside. The solution is to explicitly set the host ip to be registered in ZK by setting the "hostname" property in server.properties. In another rare case where the binding host/port is different from the host/port for client connection, you can set advertised.host.name and advertised.port for client connection.

41)How many topics can I have?

Answer)Unlike many messaging systems Kafka topics are meant to scale up arbitrarily. Hence we encourage fewer large topics rather than many small topics. So for example if we were storing notifications for users we would encourage a design with a single notifications topic partitioned by user id rather than a separate topic per user.

The actual scalability is for the most part determined by the number of total partitions across all topics not the number of topics itself (see the question below for details).

42)How do I choose the number of partitions for a topic?

Answer)There isn't really a right answer, we expose this as an option because it is a tradeoff. The simple answer is that the partition count determines the maximum consumer parallelism and so you should set a partition count based on the maximum consumer parallelism you would expect to need (i.e. over-provision). Clusters with up to 10k total partitions are quite workable. Beyond that we don't aggressively test (it should work, but we can't guarantee it).

Here is a more complete list of tradeoffs to consider:

A partition is basically a directory of log files.

Each partition must fit entirely on one machine. So if you have only one partition in your topic you cannot scale your write rate or retention beyond the capability of a single machine. If you have 1000 partitions you could potentially use 1000 machines.

Each partition is totally ordered. If you want a total order over all writes you probably want to have just one partition.

Each partition is not consumed by more than one consumer thread/process in each consumer group. This allows to have each process consume in a single threaded fashion to guarantee ordering to the consumer within the partition (if we split up a partition of ordered messages and handed them out to multiple consumers even though the messages were stored in order they would be processed out of order at times).

Many partitions can be consumed by a single process, though. So you can have 1000 partitions all consumed by a single process.

Another way to say the above is that the partition count is a bound on the maximum consumer parallelism.

More partitions will mean more files and hence can lead to smaller writes if you don't have enough memory to properly buffer the writes and coalesce them into larger writes

Each partition corresponds to several znodes in zookeeper. Zookeeper keeps everything in memory so this can eventually get out of hand.

More partitions means longer leader fail-over time. Each partition can be handled quickly (milliseconds) but with thousands of partitions this can add up.

When we checkpoint the consumer position we store one offset per partition so the more partitions the more expensive the position checkpoint is.

It is possible to later expand the number of partitions BUT when we do so we do not attempt to reorganize the data in the topic. So if you are depending on key-based semantic partitioning in

your processing you will have to manually copy data from the old low partition topic to a new higher partition topic if you later need to expand.

43)How to replace a failed broker?

Answer)When a broker fails, Kafka doesn't automatically re-replicate the data on the failed broker to other brokers. This is because in the common case, one brings down a broker to apply code or config changes, and will bring up the broker quickly afterward. Re-replicating the data in this case will be wasteful. In the rarer case that a broker fails completely, one will need to bring up another broker with the same broker id on a new server. The new broker will automatically replicate the missing data.

44)Can I add new brokers dynamically to a cluster?

Answer)Yes, new brokers can be added online to a cluster. Those new brokers won't have any data initially until either some new topics are created or some replicas are moved to them using the partition reassignment tool.