Kafka

1. **What is Apache Kafka?**

Apache Kafka is a publish-subscribe **open source** message broker application. This messaging application was coded in "Scala". Basically, this project was started by the Apache software. Kafka's design pattern is mainly based on the transactional logs design. For detailed understanding of Kafka, go through,
Kafka Tutorial.

2. **Enlist the several components in Kafka.**

The most important elements of Kafka are:

- Topic – Kafka Topic is the bunch or a collection of messages.
- Producer – In Kafka, Producers issue communications as well as publishes messages to a Kafka topic.
- Consumer – Kafka Consumers subscribes to a topic(s) and also reads and processes messages from the topic(s).
- Brokers –While it comes to manage storage of messages in the topic(s) we use Kafka Brokers. For detailed understanding of Kafka components, go through, Kafka – Architecture

3. **Explain the role of the offset.**

Ans. There is a sequential ID number given to the messages in the partitions what we call, an offset. So, to identify each message in the partition uniquely, we use these offsets.

4. **What is a Consumer Group?**

Ans. The concept of Consumer Groups is exclusive to Apache Kafka. Basically, every Kafka consumer group consists of one or more consumers that jointly consume a set of subscribed topics.

5. **What is the role of the ZooKeeper in Kafka?**

Ans. Apache Kafka is a distributed system is built to use Zookeeper. Although, Zookeeper's main role here is to build coordination between different nodes in a cluster. However, we also use Zookeeper to recover from previously committed offset if any node fails because it works as periodically commit offset.

6. **Is it possible to use Kafka without ZooKeeper?**

Ans. It is impossible to bypass Zookeeper and connect directly to the Kafka server, so the answer is no. If somehow, ZooKeeper is down, then it is impossible to service any client request.

7. **What do you know about Partition in Kafka?**

Ans. In every Kafka broker, there are few partitions available. And, here each partition in Kafka can be either a leader or a replica of a topic. https://stackoverflow.com/questions/49054946/kafka-topic-partitions

8. **Why is Kafka technology significant to use?**

Ans. There are some advantages of Kafka, which makes it significant to use:

- **High-throughput** : We do not need any large hardware in Kafka, because it is capable of handling high-velocity and high-volume data. Moreover, it can also support message throughput of thousands of messages per second.
- **Low Latency** : Kafka can easily handle these messages with the very low latency of the range of milliseconds, demanded by most of the new use cases.
- **Fault-Tolerant** : Kafka is resistant to node/machine failure within a cluster.
- **Durability** : As Kafka supports messages replication, so, messages are never lost. It is one of the reasons behind durability.
- **Scalability** : Kafka can be scaled-out, without incurring any downtime on the fly by adding additional nodes.
  Kafka – Pros & Cons

9. **What are main APIs of Kafka?**

Ans. Apache Kafka has 4 main APIs:

- Producer API
- Consumer API
- Streams API
- Connector API

10. **What are consumers or users?**

Ans. Mainly, Kafka Consumer subscribes to a topic(s), and also reads and processes messages from the topic(s). Moreover, with a consumer group name, Consumers label themselves. In other words, within each subscribing consumer group, each record published to a topic is delivered to one consumer instance. Make sure it is possible that Consumer instances can be in separate processes or on separate machines.

11. Explain the concept of Leader and Follower.

Ans. In every partition of Kafka, there is one server which acts as the Leader, and none or more servers plays the role as a Followers.

12. What ensures load balancing of the server in Kafka?

Ans. As the main role of the Leader is to perform the task of all read and write requests for the partition, whereas Followers passively replicate the leader. Hence, at the time of Leader failing, one of the Followers takeover the role of the Leader. Basically, this entire process ensures load balancing of the servers.

13. What roles do Replicas and the ISR play?

Ans. Basically, a list of nodes that replicate the log is Replicas. Especially, for a particular partition. However, they are irrespective of whether they play the role of the Leader. In addition, ISR refers to In-Sync Replicas. On defining ISR, it is a set of message replicas that are synced to the leaders.

14. Why are Replications critical in Kafka?

Ans. Because of Replication, we can be sure that published messages are not lost and can be consumed in the event of any machine error, program error or frequent software upgrades.

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

Ans. Simply, it implies that the Follower cannot fetch data as fast as data accumulated by the Leader.

16. What is the process for starting a Kafka server?

Ans. It is the very important step to initialize the ZooKeeper server because Kafka uses Zookeeper. So, the process for starting a Kafka server is: In order to **start the ZooKeeper server**: > bin/zookeeper-server-start.sh config/zookeeper.properties
Next, to **start the Kafka server**: > bin/kafka-server-start.sh config/server.properties

17. In the Producer, when does QueueFullException occur?

Ans. whenever the Kafka Producer attempts to send messages at a pace that the Broker cannot handle at that time QueueFullException typically occurs. However, to collaboratively handle the increased load, users will need to add enough brokers(servers, nodes), since the Producer doesn't block.
Kafka Broker

18. Explain the role of the Kafka Producer API.

Ans. An API which permits an application to publish a stream of records to one or more Kafka topics is what we call Producer API.

19. What is the main difference between Kafka and Flume?

Ans. The main difference between Kafka and Flume are: **Types of tool**

- Apache Kafka– As Kafka is a general-purpose tool for both multiple producers and consumers.
- Apache Flume– Whereas, Flume is considered as a special-purpose tool for specific applications.

Replication feature

- Apache Kafka– Kafka can replicate the events.
- Apache Flume- whereas, Flume does not replicate the events.

20. Is Apache Kafka is a distributed streaming platform? if yes, what you can do with it?

Ans. Undoubtedly, Kafka is a streaming platform. It can help: **To push records easily**. Also, can store a lot of records without giving any storage problems Moreover, it can process the records as they come in

21. What can you do with Kafka?

Ans. It can perform in several ways, such as:

- In order to transmit data between two systems, we can build a real-time stream of data pipelines with it.
- Also, we can build a real-time streaming platform with Kafka, that can actually react to the data.

22. What is the purpose of retention period in Kafka cluster?

Ans. However, retention period retains all the published records within the Kafka cluster. It doesn't check whether they have been consumed or not. Moreover, the records can be discarded by using a configuration setting for the retention period. And, it results as it can free up some space.

23. Explain the maximum size of a message that can be received by the Kafka?

Ans. The maximum size of a message that can be received by the Kafka is approx. 1000000 bytes.

24. What are the types of traditional method of message transfer?

Ans. Basically, there are two methods of the traditional message transfer method, such as:

- **Queuing**: It is a method in which a pool of consumers may read a message from the server and each message goes to one of them.
- **Publish-Subscribe**: Whereas in Publish-Subscribe, messages are broadcasted to all consumers.

25. What does ISR stand in Kafka environment?

Ans. ISR refers to In sync replicas. These are generally classified as a set of message replicas which are synced to be leaders.

26. What is Geo-Replication in Kafka?

Ans. For our cluster, Kafka MirrorMaker offers geo-replication. Basically, messages are replicated across multiple data centers or cloud regions, with MirrorMaker. So, it can be used in active/passive scenarios for backup and recovery; or also to place data closer to our users, or support data locality requirements.

27. Explain Multi-tenancy?

Ans. We can easily deploy Kafka as a multi-tenant solution. However, by configuring which topics can produce or consume data, Multi-tenancy is enabled.

28. What is Apache Kafka?

Apache Kafka is a distributed streaming platform that allows for publishing, subscribing to, storing, and processing streams of records in real-time. It's designed to handle high-throughput, fault-tolerant, and scalable data pipelines. Kafka is often used for building real-time data pipelines and streaming applications.

29. What are the key components of Kafka?

The key components of Kafka include:

1. Producer: Publishes messages to Kafka topics.
2. Consumer: Subscribes to topics and processes the published messages.
3. Broker: A Kafka server that stores and manages topics.
4. ZooKeeper: Manages and coordinates Kafka brokers.
5. Topic: A category or feed name to which records are published.
6. Partition: Topics are divided into partitions for scalability.

30. **What is a topic in Kafka?**

A topic in Kafka is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many

consumers that subscribe to the data written to it. Topics are split into partitions for improved scalability and parallel processing.

31. What is a partition in Kafka?

A partition is an ordered, immutable sequence of records that is continually appended to. Each partition is a structured commit log, and records in the partitions are each assigned a sequential id number called the offset. Partitions allow Kafka to scale horizontally and provide parallel processing capabilities.

32. What is the role of ZooKeeper in Kafka?

ZooKeeper is used for managing and coordinating Kafka brokers. It serves as a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. ZooKeeper keeps track of the status of Kafka cluster nodes, Kafka topics, and partitions.

33. What is a broker in Kafka?

A broker is a Kafka server that runs in a Kafka cluster. It receives messages from producers, assigns offsets to them, and commits the messages to storage on disk. It also services consumers, responding to fetch requests for partitions and responding with the messages that have been published.

34. How does Kafka ensure fault tolerance?

Kafka ensures fault tolerance through data replication. Each partition is replicated across a configurable number of servers for fault tolerance. One of the servers is designated as the leader, which handles all read and write requests for the partition, while the others are followers that passively replicate the leader.

35. What is the difference between a Kafka consumer and consumer group?

A Kafka consumer is an application that reads data from Kafka topics. A consumer group is a set of consumers that work together to consume data from one or more topics. The key difference is that each message is delivered to one consumer instance within each subscribing consumer group. This allows for parallel processing and load balancing of topic consumption.

36. What is the purpose of the offset in Kafka?

The offset is a unique identifier of a record within a partition. It denotes the position of the consumer in the partition. Kafka maintains this offset per partition, per consumer group, allowing each consumer group to read from a different position in the partition. This enables Kafka to provide both queue and publish-subscribe messaging models.

37. How does Kafka handle message delivery semantics?

Kafka supports three message delivery semantics:

1. At most once: Messages may be lost but are never redelivered.
2. At least once: Messages are never lost but may be redelivered.
3. Exactly once: Each message is delivered once and only once. The choice depends on the specific use case and can be configured through producer and consumer settings.

38. What is the role of the Kafka producer API?

The Kafka producer API is used to publish streams of records to Kafka topics. It handles partitioning of messages, compression, and load balancing across multiple brokers. The producer is also responsible for retrying failed publish attempts and can be configured for different levels of delivery guarantees.

39. How does Kafka support scalability?

Kafka supports scalability through partitioning and distributed processing. Topics can be partitioned across multiple brokers, allowing for parallel processing. Consumers can be grouped to read from multiple partitions simultaneously. Brokers can be added to a cluster to increase capacity, and the cluster can be scaled without downtime.

40. What is log compaction in Kafka?

Log compaction is a mechanism to give finer-grained per-record retention, rather than the coarser-grained time-based retention. The idea is to selectively remove records where we have a more recent update with the same primary key. This way, the log is guaranteed to have at least the last state for each key.

41. How does Kafka handle message ordering?

Kafka guarantees order within a partition. Messages sent by a producer to a particular topic partition will be appended in the order they are sent. A consumer instance will read records in the order they are stored in the log. However, there's no guarantee of order across partitions.

42. What is the significance of the acks parameter in Kafka producers?

The acks parameter in Kafka producers controls the number of acknowledgments the producer requires the leader to have received before considering a request complete. It affects the durability of records and can be set to: 0: No acknowledgment 1: Leader acknowledgment only all: Full ISR (In-Sync Replica) acknowledgment

43. How does Kafka handle data retention?

Kafka handles data retention through configurable retention policies. These can be based on time (e.g., retain data for 7 days) or size (e.g., retain up to 1GB per partition).

After the retention limit is reached, old messages are discarded. Kafka also supports log compaction for topics where only the latest value for each key is needed.

44. What is the purpose of the Kafka Connect API?

Kafka Connect is a tool for scalably and reliably streaming data between Apache Kafka and other data systems. It makes it simple to quickly define connectors that move large collections of data into and out of Kafka. This can be used to connect Kafka with databases, key-value stores, search indexes, and file systems.

45. How does Kafka ensure high availability?

Kafka ensures high availability through:

1. Replication of partitions across multiple brokers
2. Automatic leader election when a broker fails
3. Ability to add brokers to a cluster without downtime
4. Configurable number of in-sync replicas for durability
5. ZooKeeper for distributed coordination and broker management

46. What is the difference between Kafka Streams and Apache Flink?

While both Kafka Streams and Apache Flink are stream processing frameworks, they have some key differences:

1. Kafka Streams is a client library for building applications and microservices, where the input and output data are stored in Kafka clusters. Flink is a distributed processing engine that can work with various data sources and sinks.
2. Kafka Streams is tightly integrated with Kafka, while Flink has a more general-purpose design.
3. Flink generally offers lower latency and higher throughput for complex operations, while Kafka Streams is simpler to deploy and operate.

47. How does Kafka handle message compression?

Kafka supports message compression to reduce the size of data transferred and stored. Compression can be configured at the producer level, and Kafka supports several compression types including gzip, snappy, lz4, and zstd. The broker can be configured to decompress messages to validate and convert them to the message format version on the broker.

48. What is the purpose of the Kafka Streams API?

The Kafka Streams API is a client library for building applications and microservices that process and analyse data stored in Kafka. It enables you to build stream processing applications with just standard Java and Kafka clients, without the need for a separate processing cluster. It supports stateful operations, windowing, joining streams and tables, and more.

49. How does Kafka handle message size limits?

Kafka has configurable message size limits. The default maximum message size is 1MB, but this can be increased by changing the 'message.max.bytes' configuration on the broker and the 'max.request.size' on the producer. However, very large messages can impact performance and memory usage, so it's generally recommended to keep messages relatively small.

50. What is the role of the group coordinator in Kafka?

The group coordinator in Kafka is responsible for managing consumer groups. It handles consumer group membership, assigns partitions to consumers within a group, and manages offset commits. When a consumer joins or leaves a group, the group coordinator triggers a rebalance to reassign partitions among the remaining consumers.

51. How does Kafka handle data replication?

Kafka replicates data by maintaining multiple copies of each partition across different brokers. One broker is designated as the leader for a partition, handling all read and write requests, while others are followers that replicate the leader's data. If a leader fails, one of the followers becomes the new leader. The number of replicas is configurable per topic.

52. What is the purpose of the Idempotent Producer in Kafka?

The Idempotent Producer in Kafka ensures that messages are delivered exactly once to a partition, even in the case of retries. It achieves this by assigning a unique ID to each produce request and maintaining a sequence number for each producer-partition pair. This prevents duplicate messages due to network issues or producer retries.

53. How does Kafka handle consumer offsets?

Kafka maintains offsets for each consumer group per partition. These offsets represent the position of the consumer in the partition log. Consumers can commit these offsets either automatically (at a configurable interval) or manually. Kafka stores these offsets in a special Kafka topic called '__consumer_offsets', allowing consumers to resume from where they left off in case of restarts or failures.

54. What is the difference between a round-robin partitioner and a key-based partitioner in Kafka?

A round-robin partitioner distributes messages evenly across all partitions in a cyclic manner, regardless of any key. A key-based partitioner, on the other hand, uses a hash of the key to determine which partition a message should go to. This ensures that all messages with the same key always go to the same partition, which is crucial for maintaining order for key-based events.

55. How does Kafka handle message deletion?

Kafka doesn't delete messages individually. Instead, it uses a retention policy to manage message deletion. Messages are retained either for a configurable amount of time or until the topic reaches a certain size. Once the retention limit is reached, Kafka deletes messages in bulk by removing whole segments of the log file. For more fine-grained control, Kafka also supports log compaction.

56. What is the purpose of the Kafka Mirror Maker?

Kafka Mirror Maker is a tool used for replicating data between Kafka clusters, potentially across different data centers. It works by consuming from one Kafka cluster and producing to another. This is useful for maintaining a backup of your data, aggregating data from multiple datacenters into a central location, or for migrating data between clusters.

57. How does Kafka handle message versioning?

Kafka itself doesn't handle message versioning directly, but it provides mechanisms that allow users to implement versioning. One common approach is to include a version field in the message schema. For more complex versioning needs, many users leverage schema registries (like the Confluent Schema Registry) which can manage schema evolution and compatibility.

58. What is the role of the controller in a Kafka cluster?

The controller in a Kafka cluster is a broker that has additional responsibilities for managing the overall state of the cluster. It's responsible for electing partition leaders, managing the distribution of partitions across brokers, and handling administrative operations like adding or removing topics. If the controller fails, ZooKeeper helps elect a new controller from among the brokers.

59. How does Kafka ensure data consistency?

Kafka ensures data consistency through several mechanisms:

1. Replication: Each partition is replicated across multiple brokers.
2. In-Sync Replicas (ISR): Only replicas that are up-to-date with the leader can be part of the ISR.

3. Acknowledgments: Producers can be configured to wait for acknowledgments from the leader and ISRs.
4. Atomic writes: Writes to a partition are atomic and ordered.
5. Idempotent producers: Prevent duplicate messages in case of retries.

60. What is the purpose of the Kafka AdminClient API?

The Kafka AdminClient API provides administrative operations for managing and inspecting topics, brokers, configurations, and other Kafka objects. It can be used to create, delete, and describe topics, manage ACLs, get cluster information, and perform other administrative tasks programmatically.

61. How does Kafka handle message batching?

Kafka producers can batch messages to improve throughput. Instead of sending each message individually, the producer can group multiple messages destined for the same partition into a single request. This reduces network overhead and improves efficiency. The batch size and linger time (how long to wait for more messages before sending a batch) are configurable.

62. What is the difference between a Kafka consumer and a Kafka streams application?

A Kafka consumer is a client that reads data from Kafka topics and processes it in some way. It's typically used for simple consumption scenarios. A Kafka Streams application, on the other hand, is a more sophisticated client that can consume, process, and produce data back to Kafka. It provides a DSL for complex stream processing operations like filtering, transforming, aggregating, and joining streams.

63. How does Kafka handle message ordering within a partition?

Kafka guarantees that messages within a partition are ordered. Messages sent by a producer to a specific partition will be appended to the log in the order they are sent. Consumers read messages from a partition in the exact order they were written. This ordering guarantee is crucial for use cases that require event sequencing.

64. What is the purpose of the Kafka Transactions API?

The Kafka Transactions API allows for atomic updates to multiple topics and partitions. It enables exactly-once processing semantics for applications that read, process, and write data to Kafka. This is particularly useful for stream processing applications that need to ensure that each input event affects the output exactly once, even in the face of failures.

65. How does Kafka handle message key hashing?

When a key is provided with a message, Kafka uses a hash of the key to determine which partition the message should go to. By default, Kafka uses murmur2 algorithm for key hashing. This ensures that messages with the same key always go to the same partition, which is crucial for maintaining order for key-based events and for enabling local state in stream processing applications.

66. What is the role of the Kafka consumer coordinator?

The Kafka consumer coordinator is responsible for managing the state of the consumer group and coordinating the consumer group rebalance process. It assigns partitions to consumers in the group, ensures that each partition is consumed by only one consumer in the group, and manages the committed offsets for each partition.

67. How does Kafka handle message timestamps?

Kafka supports two types of timestamps:

1. CreateTime: The time the producer created the message.
2. LogAppendTime: The time the broker received the message. These timestamps can be used for log retention, log compaction, and time-based search in consumers. The timestamp type is configurable at the topic level.

68. What is the purpose of the Kafka Quota API?

The Kafka Quota API allows you to enforce quotas on produce and fetch requests to prevent a single client from consuming too many broker resources. Quotas can be defined on a per-client or per-user basis, and can limit the rate of data production or consumption. This helps in ensuring fair resource allocation and preventing denial of service scenarios.

69. How does Kafka handle message acknowledgments?

Kafka producers can be configured to require acknowledgments when sending messages. There are three settings:

1. acks=0: No acknowledgment (fire and forget)
2. acks=1: Leader acknowledgment only
3. acks=all: Full ISR (In-Sync Replica) acknowledgment The choice affects the trade-off between latency and durability. Higher levels of acknowledgment provide stronger durability guarantees but increase latency.

70. How does Kafka handle message acknowledgments?

Kafka producers can be configured to require acknowledgments when sending messages. There are three settings:

1. acks=0: No acknowledgment (fire and forget)
2.
    - The producer doesn't wait for any acknowledgment from the broker.
    - This option has the lowest latency but the weakest durability guarantees since the message may be lost if the broker goes down.
2. acks=1: Leader acknowledgment only
3.
    - The producer waits for the leader replica to acknowledge the message.
    - This provides better durability than acks=0, but there's still a risk of message loss if the leader fails immediately after acknowledging but before the followers have replicated the message.
3. acks=all: Full ISR (In-Sync Replica) acknowledgment
4.
    - The producer waits for the message to be acknowledged by all in-sync replicas.
    - This setting provides the strongest durability guarantee but has the highest latency.

The choice of acknowledgment level affects the trade-off between latency and durability. Higher levels of acknowledgment provide stronger durability guarantees but increase latency.

71. How does Kafka handle message serialization and deserialization?

Kafka itself treats message data as opaque byte arrays and doesn't perform any serialization or deserialization. However, Kafka producers and consumers can be configured with serializers and deserializers for keys and values. Common formats include String, Integer, and Avro. For complex objects, custom serializers and deserializers can be implemented.

72. What is the purpose of the Kafka Schema Registry?

The Kafka Schema Registry provides a serving layer for metadata. It provides a RESTful interface for storing and retrieving Avro schemas. It's used in conjunction with Kafka to ensure that producers and consumers use compatible schemas. This is particularly

useful in evolving data models over time while maintaining backward and forward compatibility.

73. How does Kafka handle topic deletion?

When a topic is deleted in Kafka, the following steps occur:

1. The topic is marked for deletion in ZooKeeper
2. Kafka stops serving data for that topic
3. The actual log segments on disk are asynchronously deleted This process ensures that topic deletion doesn't impact the performance of other operations. However, it's worth noting that in versions prior to Kafka 2.1, topic deletion could sometimes be incomplete if brokers were offline during the deletion process.

74. What is the difference between a Kafka consumer's poll() and subscribe() methods?

The subscribe() method is used to subscribe a consumer to one or more topics. It doesn't actually fetch any data. The poll() method, on the other hand, is used to fetch data from the subscribed topics. It returns records that have been published since the last fetch for the subscribed topics and partitions. poll() is typically called in a loop to continuously consume data.

75. How does Kafka handle message compression at the broker level?

Kafka brokers can be configured to handle message compression in several ways:

1. Pass-through: The broker stores the message in its original compressed format
2. Decompress on receipt: The broker decompresses the message on receipt and stores it uncompressed
3. Decompress and recompress: The broker decompresses the message and then recompresses it, potentially with a different algorithm The choice depends on factors like CPU usage, network bandwidth, and storage requirements.

76. What is the purpose of the Kafka consumer heartbeat thread?

The Kafka consumer heartbeat thread is responsible for sending periodic heartbeats to the Kafka broker (specifically, to the group coordinator). These heartbeats indicate that the consumer is alive and still part of the consumer group. If a consumer fails to send heartbeats for a configurable period, it's considered dead, and the group coordinator will trigger a rebalance to reassign its partitions to other consumers in the group.

77. How does Kafka handle message ordering across multiple partitions?

Kafka only guarantees message ordering within a single partition. Across multiple partitions, there is no guarantee of message ordering. If global ordering is required, it's typically achieved by using a single partition for the topic, but this limits scalability. For use cases requiring ordering and scalability, it's common to use a partition key that ensures related messages go to the same partition.

78. What is the role of the Kafka broker's log cleaner thread?

The log cleaner thread in Kafka is responsible for performing log compaction. Log compaction is a mechanism where Kafka removes redundant records from a log, keeping only the latest value for each key. This is useful for use cases where only the latest update for a given key is needed, such as maintaining a changelog or a database state. The log cleaner runs periodically to compact eligible topics.

79. How does Kafka handle consumer lag?

Consumer lag in Kafka refers to the difference between the offset of the last produced message and the offset of the last consumed message. Kafka provides tools and APIs to monitor consumer lag, such as the Kafka Consumer Groups command-line tool and the AdminClient API. High consumer lag can indicate performance issues or insufficient consumer capacity. Kafka doesn't automatically handle lag, but it provides the information needed for applications to make scaling or performance optimization decisions.

80. What is the purpose of the Kafka producer's Partitioner interface?

The Partitioner interface in Kafka's producer API determines which partition in the topic a message will be sent to. The default partitioner uses a hash of the key (if present) to choose the partition, ensuring that messages with the same key always go to the same partition. Custom partitioners can be implemented to control message distribution across partitions based on specific business logic or data characteristics.

81. How does Kafka handle message delivery timeouts?

Kafka producers can be configured with delivery timeouts. If a message cannot be successfully acknowledged within this timeout period, the producer will consider the send failed and may retry (depending on configuration). On the consumer side, there's a max.poll.interval.ms setting that controls how long a consumer can go without polling before it's considered failed and a rebalance is triggered.

82. What is the purpose of the Kafka Streams DSL?

The Kafka Streams DSL (Domain Specific Language) provides a high-level API for stream processing operations. It allows developers to express complex processing logic like filtering, transforming, aggregating, and joining streams of data. The DSL abstracts away

many of the low-level details of stream processing, making it easier to build and maintain stream processing applications.

83. How does Kafka handle message de-duplication?

Kafka itself doesn't provide built-in de-duplication of messages. However, it provides mechanisms that allow applications to implement de-duplication:

1. Idempotent producers prevent duplicate messages due to producer retries.
2. Exactly-once semantics in Kafka Streams ensure that each input record is processed once.
3. For custom applications, unique message IDs can be used to detect and handle duplicates at the consumer level.

84. What is the role of the Kafka consumer's position() method?

The position() method in a Kafka consumer returns the offset of the next record that will be fetched for a given partition. This is useful for tracking the progress of consumption and can be used in conjunction with the committed() method to determine how far behind the consumer is from its last committed position. This information can be valuable for monitoring and managing consumer performance.

85. How does Kafka handle message schema evolution?

Kafka itself is agnostic to message schemas, treating messages as byte arrays. However, schema evolution is typically handled using a schema registry (like Confluent Schema Registry) in conjunction with a serialization format that supports schema evolution (like Avro). The schema registry maintains versions of schemas and ensures compatibility between producer and consumer schemas. This allows for schema changes over time while maintaining backward and forward compatibility.

86. What is the purpose of the Kafka broker's controlled shutdown?

Controlled shutdown is a feature in Kafka that allows a broker to shut down gracefully. During a controlled shutdown:

1. The broker stops accepting new produce requests
2. It completes all ongoing produce and fetch requests
3. It transfers leadership of its partitions to other brokers in a controlled manner This process minimizes data loss and service disruption when a broker needs to be taken offline for maintenance or other reasons.

60. How does Kafka handle message validation?

Kafka itself doesn't perform message validation beyond ensuring that messages don't exceed the configured maximum size. Message validation is typically handled at the producer or consumer level. Producers can implement validation logic before sending messages, while consumers can validate messages after receiving them. For more complex validation scenarios, intermediate processing steps (like Kafka Streams applications) can be used to validate and potentially transform messages.

87. What is the role of the Kafka consumer's commitSync() and commitAsync() methods?

These methods are used to commit offsets in Kafka consumers:

- commitSync(): Synchronously commits the latest offset returned by poll(). It will retry until it succeeds or encounters a non-retriable error.
- commitAsync(): Asynchronously commits offsets. It doesn't retry on failures, making it faster but less reliable than commitSync(). The choice between these methods depends on the balance between performance and reliability required by the application.

88. How does Kafka handle message retention across multiple data centers?

Kafka can handle message retention across multiple data centers through a feature called MirrorMaker. MirrorMaker is a stand-alone tool for copying data between Kafka clusters. It consumes from one cluster and produces to another, allowing for replication of data across different data centers. This can be used for disaster recovery, geographic distribution of data, or aggregating data from multiple sources into a central location.

89. What is the purpose of the Kafka producer's max.block.ms parameter?

The max.block.ms parameter in a Kafka producer controls how long the producer will block when calling send() and when explicitly requesting metadata via metadata(). If this time elapses before the producer can send the record, it will throw a TimeoutException. This parameter is useful for setting an upper bound on how long the application will wait in these scenarios, preventing indefinite blocking.

90. How does Kafka handle message consumption across consumer group rebalances?

When a consumer group rebalance occurs (due to consumers joining or leaving the group), Kafka ensures that:

1. All consumers stop consuming and commit their current offsets
2. The group coordinator reassigns partitions to the remaining consumers
3. Consumers start consuming from their newly assigned partitions, beginning from the last committed offset This process ensures that all messages are

consumed exactly once (assuming proper offset management) even as the set of consumers changes.

91. What is the role of the Kafka broker's log.segment.bytes configuration?

The log.segment.bytes configuration in Kafka brokers controls the maximum size of a single log segment file. When a log segment reaches this size, a new segment is created. This configuration affects:

1. How often segments are closed and become eligible for deletion
2. The granularity of log retention (Kafka can only delete entire segments)
3. The amount of data that needs to be moved during partition reassignments Smaller segments allow for more granular retention and faster reassignments but can lead to more file handles and slightly higher overhead.

92. How does Kafka handle message consumption patterns?

Kafka supports two main consumption patterns:

1. Queue: Each message is processed by one consumer within a consumer group. This is achieved by having multiple consumers in a group, each reading from exclusive partitions.
2. Publish-Subscribe: All messages are processed by all consumers. This is achieved by having each consumer in its own consumer group, allowing all consumers to read all messages. These patterns can be combined and customized to fit various use cases.

93. What is the purpose of the Kafka producer's linger.ms parameter?

The linger.ms parameter in a Kafka producer controls the amount of time to wait for additional messages before sending a batch of messages. Increasing this value leads to larger batches and higher throughput at the cost of increased latency. Setting this to 0 (the default) means messages are sent as soon as possible. This parameter allows for fine-tuning the trade-off between latency and throughput in message production.

94. How does Kafka handle message delivery guarantees?

Kafka provides different levels of delivery guarantees:

1. At most once: Messages may be lost but are never redelivered.
2. At least once: Messages are never lost but may be redelivered.
3. Exactly once: Each message is delivered once and only once. These guarantees are achieved through a combination of producer acknowledgments, consumer

offset management, and (for exactly once semantics) the transactions API. The choice depends on the specific requirements of the use case.

95. What is the role of the Kafka consumer's auto.offset.reset configuration?

The auto.offset.reset configuration in Kafka consumers determines what to do when there is no initial offset in Kafka or if the current offset no longer exists on the server. It can be set to:

- earliest: automatically reset the offset to the earliest offset
- latest: automatically reset the offset to the latest offset
- none: throw exception to the consumer if no previous offset is found This configuration is crucial for defining behavior when a consumer starts reading from a topic for the first time or when it has been offline for a long time.

96. How does Kafka handle message retrieval for consumers?

Kafka uses a pull model for message retrieval. Consumers request messages from brokers rather than brokers pushing messages to consumers. This allows consumers to control the rate at which they receive messages. Consumers make fetch requests to brokers, specifying the topics, partitions, and starting offset for each partition. The broker responds with messages up to a specified maximum byte limit. This model allows for better flow control and makes it easier to handle scenarios where consumers fall behind.

System Design using Kafka.

Draw a mix of high level/deployment diagram Input to system: A Kafka Topic in which you are getting messages containing Customer orders having customer ID and order amount. With every 100 rs spent, customer gets 1 reward point. Based on reward points accumulated, customer gets rank. Create two rest end points hosted on internet without authentication: Input: N Output: Top N customers based on rank Input: Customer ID Output: rank and reward points of this customer and +2 and -2 ranked customers. For simplicity, Assume 1000 requests / messages per second at any point. Also, assume millions of customers, If possible, choose open source and avoid public cloud as you will not be able to detail about inner working of let's say a database. Scalability and resilience are the main abilities that is the focus for now, so don't worry about reducing hardware cost etc What is the possible way to implement this If we have to update rank of million record on every order table? Is this possible or we can update rank at interval through a scheduler.

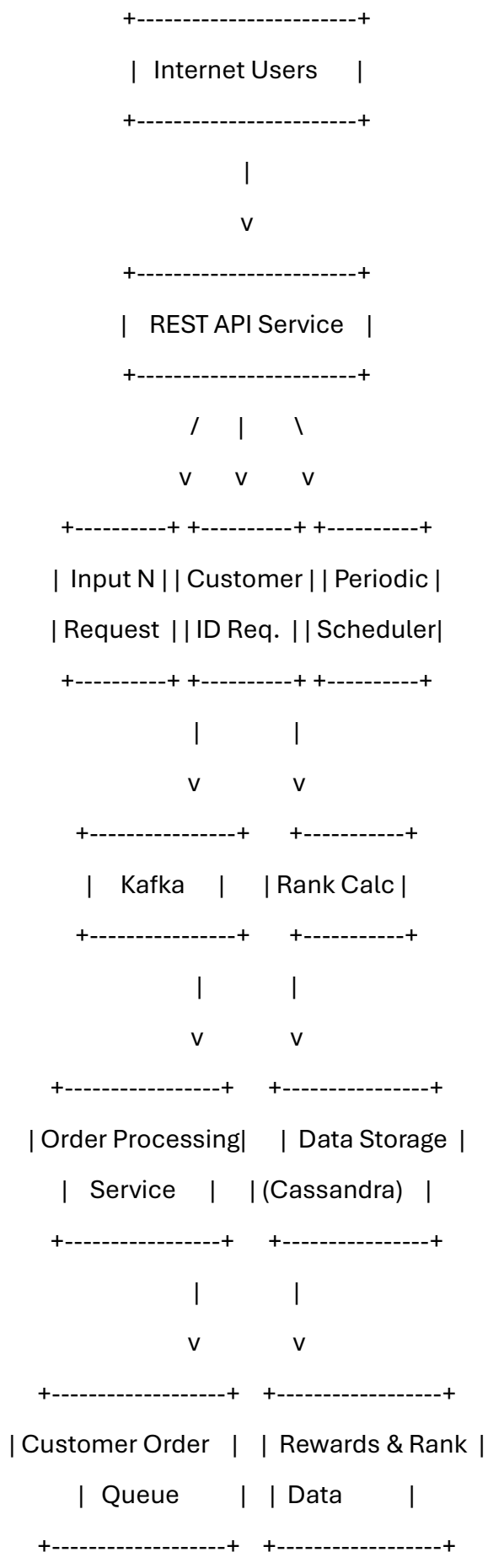
To create a scalable and resilient system for managing customer orders, reward points, and ranking, we can use the following architecture:

High-Level Architecture

1. **Kafka**: Acts as the message broker, receiving and distributing customer order messages.
2. **Order Processing Service**: Listens to the Kafka topic, processes incoming orders, calculates reward points, and updates customer ranks.
3. **Data Storage**: A scalable and resilient database system, like Apache Cassandra, to store customer information, reward points, and ranks.
4. **Rank Calculation Service**: Periodically calculates customer ranks based on accumulated reward points.
5. **REST API Service**: Provides endpoints to fetch top N customers and individual customer ranks.

Deployment Diagram

Here is a simplified version of the deployment diagram:


Internet Users
REST API Service
Input N Request
Customer ID Req.
Periodic Scheduler
Kafka
Rank Calc
Order Processing Service
Data Storage (Cassandra)
Customer Order Queue
Rewards & Rank Data

Detailed Workflow

1. **Order Processing Service**:
    - Listens to the Kafka topic for incoming order messages.
    - Parses messages to extract customer ID and order amount.
    - Updates the customer's reward points in the database.
    - Sends a message to the Rank Calculation Service if a threshold of reward points is crossed.
2. **Rank Calculation Service**:
    - Periodically (e.g., every 5 minutes) retrieves updated reward points for all customers.
    - Calculates ranks for customers based on accumulated reward points.
    - Updates ranks in the database.
3. **REST API Service**:
    - Provides endpoints to fetch top N customers based on rank and details of a specific customer and their neighbours in rank.
    - Interfaces directly with the database to fetch the required data.

REST Endpoints

- **Get Top N Customers**:
    - **Input**: N (number of top customers to fetch)
    - **Output**: List of top N customers based on rank
- **Get Customer Rank and Reward Points**:
    - **Input**: Customer ID
    - **Output**: Rank and reward points of the specified customer, along with +2 and -2 ranked customers

Scalability and Resilience

- **Kafka**: Can handle a high throughput of messages (1000 requests per second).
- **Cassandra**: Provides horizontal scaling, high availability, and distributed data storage.
- **Order Processing Service**: Can scale horizontally to process messages concurrently.
- **Rank Calculation**: Can be done periodically using a scheduler to avoid frequent updates for millions of records.

Additional Considerations

- **Eventual Consistency**: Due to the periodic rank calculation, there might be slight delays in rank updates, but it ensures scalability.
- **Fault Tolerance**: Use replication and failover mechanisms in Kafka and Cassandra to ensure high availability and fault tolerance.

I hope this helps! Follow Prashanth Acharya for more Study Materials.