Interview Questions - Advanced Demo

1. How will you calculate the amount of RAM required to hold the N no of partitions in a broker?

- The broker's buffer memory is defined by replica.fetch.max.bytes for each partition. Suppose you have 1000 partitions and replica.fetch.max.bytes is 1 MiB then 1 GiB of RAM is required. Remember (no of partitions) x (largest message size) shouldn't exceed the available memory.
- Be cautious when you're sending large messages because you don't get any exceptions.

2. What is the max size of the message a broker can receive from the producer?

- The maximum message size that broker accepts is defined by message.max.bytes. In case you are using old consumer this property should be lower than fetch.message.max.bytes otherwise the consumer will not be able to consume.

3. What is the largest size of the message that can be fetched by the customer?

- The consumer can fetch the largest size defined in fetch.message.max.bytes.
- The RAM size should be greater than the largest message present for each partition. As the message size becomes larger, you might want to use fewer partitions or try to increase the RAM.

4. What will you do when you exceed the resources available and you need to add more brokers into Kafka cluster?

- When you add a new broker to the cluster, new partitions are allocated to new brokers. But if you want to share the already present partition to the new broker you must assign them manually. For that, you should use bin/kafka-reassign-partitions.sh script. For doing this first you should create the topics you want to move, then using --generate options you should see the suggested distribution on your current list. Use --execute to commit from your list. And use --verify to verify the new distribution.
- you must consider redistribution when your system load is around 70%. Beyond this load, redistribution will have no resources to perform itself and the process will be slow.

5. What is the major problem when the message size is large in Kafka?

- If the message size is large, then the garbage collection pauses can be long because a large chunk of memory is used. To check the long paused look at the GC log and the server log. Long GC pauses can result in abandonment of the zookeeper session. In such cases, you may want to increase the zookeeper.session.timeout.ms.

6. What are the primary methods if we want to avoid large message size?

- We can compress messages from the producer. compression.codes and compressed.topics are producer configuration properties and it supports Gzip and Snappy compression. This is particularly useful when the message format is text.
- We can also use shared storage such as S3 to save large files and update the file location in the Kafka message.
- We can also split the messages into small chunks such as 1KB from the producer and use partition keys to ensure the correct order in a particular partition. At the other end, the consumer can reconstruct this message.
- If it is still required to send a large message, make sure that you modify the configurations as discussed before.

7. What is old consumer and new consumer?

- The old consumer was written in Scala whereas the new consumer is written in Java. To configure Kafka to handle large messages, both consumers have a different implementation.
- Old consumer: fetch.message.max.bytes is used to define the maximum amount of data the server should return for a fetch request. Remember if the batch size is larger than this, then the consumer will not be able to consume any incoming message from the partition.
- New consumer: Max.partition.fetch.bytes, Fetch.max.bytes.

8. What is a well-tuned Kafka Cluster?

- In a well tuned Kafka, the number of brokers is just enough to handle the topic throughput within the specified latency. For a real-time application, the acceptable latency is 1 millisecond or sub-millisecond.
- Tuning is basically configuring your producers, brokers and consumers in such a way so that the largest possible batch can be processed within a manageable amount of time, given the throughput.

9. What is a “future” in Kafka?

- The returned object from the send() method of a producer is called a future. It has methods to check the status of the message which is sent. Once the batch is received by the broker it responds about the transaction completion.

10. What will happen if you don't want to use futures?

- In case you don't want to use futures you can send one record at a time. Latency and throughput will be low.
- If you're using the send() method, the buffer is filled up first and once the buffer is full then the producer empties the buffer and sends it to Kafka broker.

11. What are the factors important for tuning in the producer?

- batch.size: is in terms of total bytes and not the number of messages that are sent. It defines the bytes that are collected and sent to the Kafka broker. You must keep it as high as possible limited by the available memory. The default is 16384. It might happen that this memory never gets full and because of the linger.ms. The best way to understand whether your throughput is good is by watching whether your producer is idle or not.
- linger.ms: tells the producer the time till which producer can buffer the data if it is using asynchronous mode. This property helps in improving the throughput but it increases the latency. If this property is not set, the producer doesn't wait. Setting this value decreases the no of messages to be sent but it results in higher latency.

12. Explain how consumers can create a problem in the Kafka cluster?

- As you already know that the maximum no of the consumer for a particular topic is the number of partitions present in the topic. Now you also need enough consumers to keep up to the producers producing. Hence you need to adjust the number of partition in each topic to provide each consumer one partition.

13. How checkpoint interval affects the throughput?

- replica.high.watermark.checkpoint.interval.ms: This defines the time in which the checkpoint should happen. If the watermark is set to checkpoint on every message, you will not lose a message but it will decrease the throughput whereas if you set it high then throughput will increase with a slight chance of losing some message.

14. What is client id in Kafka?

- Client ID is an identifier for Kafka consumer which is optional and it is passed to a Kafka broker on every request. The purpose is to track the requests on the basis apart from IP address. This is majorly helpful when monitoring through logs.

15. What are Quotas in Kafka?

- Quotas are defined as bytes per second which is the upper limit on the amount of data a client ID can make a request of. If there are multiple instances sharing same client ID then the collective request size per second should be less than the quota defined.

16. How Quotas affect performance?

- Quotas protect against the issues that could arise due to the high consumption of data. High volumes of data are being consumed by producers and consumers which can cause network saturation, monopolise broker resources and may even deny service to other clients and brokers. The user experience may degrade when a small set of clients use high volumes of data, and hence quotas are very important for such multi-tenant clusters.
- quota.producer.default and quota.consumer.default defines the maximum rate of data that can be fetched or consumed from each broker per client ID. It can be different for each broker.

17. Can a client ID exceed the quota? What happens then?

- As soon as the client exceeds its quota, the broker attempts to slow down the client. There is no error being thrown by the broker instead it delays the response for the amount of time needed to bring a client under its quota. For this, it first calculates the amount of delay required to bring a client under its quota. This way the quota violation remains transparent to the clients. This also prevents clients from implementing special backoff and retry behaviour.

18. What is the default Quota in Kafka and how do you set the Quota?

- There is no default quota limit for a client. Each client ID gets an unlimited quota by default. You can set the default quota per producer and consumer Client ID using the following parameters.
- quota.producer.default
- quota.consumer.default
- Setting these values to 10485760, sets the default quota per producer and consumer client ID to 10 MB/s.
- Setting the Quota is similar to per-topic log configuration overrides. You need to write the client ID overrides to ZooKeeper. Client ID overrides need to be written under /config/clients in ZooKeeper. You can change the quotas anytime without restarting the cluster. The quota changes are effective immediately and all brokers read the overrides.

19. What does it mean when we say Spark Streaming application is not stable?

- When we say that the Spark Streaming application is not stable, we mean that the processing time of each micro-batch is equal to or less than the batch time. For example, if the streaming part of the application is receiving data in the 30-second window but it is taking between 4.5-6 minutes to process, then it is not a stable Spark Streaming Application. Similarly, if any batch job is running every one hour and taking more than one hour to complete then it is not a stable batch job.

20. What is Kafka Direct Approach in Spark Kafka Application?

- From Spark 1.3, to make end-to-end guarantees even better Kafka Direct Approach is introduced. Earlier receivers in Spark used to receive the data from Kafka cluster. This approach was load unbalanced. But now, Spark asks Kafka for the offsets for each topic and its partitions and defines the ranges of offset which each batch will process. When the tasks are going to execute by worker they use Kafka's simple consumer API to read the data from Kafka cluster in that much range. This approach gives parallelism, efficiency and exactly-once semantics.

21. What will be the partition per topic is each node is having 6 disks and there are 6 nodes present in the cluster?

- 36 partitions per topic = 6*6 (according to Kafka documentation it is advisable to use one partition of a topic per physical disk)

22. How is data serialisation important in reducing the memory usage of Spark?

- Often it is required by Spark to save the in-memory RDD in the disk using serialisation to decrease the memory usage.

23. What is Execution Memory and Storage Memory in Spark?

- Spark memory management involves two types of categories to handle: execution and storage. Execution memory is handles computing of shuffles, Joins and aggregation. Whereas the storage memory is used for caching and propagating internal data. These two memories use each other's space to optimise. Now memory.fraction defines the fraction of JVM heap that storage and execution memory combined use. memory.storageFraction defines the minimum threshold that should be available for storage in memory.fraction. The default value for memory.fraction is 0.6 whereas for memory.storageFraction it is 0.5.

24. How do you determine memory consumption in Spark?

- To know how much memory an RDD is occupying we should put the RDD into the cache and refer to the storage page in the Spark Web. For determining the memory of an object we can use SizeEstimator class estimate method.

25. What is Geohash? How are you making use of it in your project?

- Geohash library used in our code takes latitudes and longitudes as input and return a unique Geohash string. We have taken the length of the Geohash string to be 5 which has the km error of ±2.4. Hence a Geohash string covers the radius of 4.8 km. Latitudes and longitudes lying in the radius of 4.8 km will return the same Geohash string. We are calculating the surge price per Geohash string, i.e., 4.8 km. We, first, calculate the number of customers and drivers for a Geohash string and then calculate the surge in prices by dividing the number of customers by a number of drivers for that string.

26. What is the objective of this project and explain the various scenarios which were tested?

- The objective of this project is to calculate the surge in prices in real time for a cab provider. The live data is coming from the applications to the system which we need to process to calculate the surge prices in real time which can then be updated on the application and/or used to gather insights from the data. So, we first had to ingest the data, then process it and finally write it on some storage system. The scenarios which were tested in the project are -
- The surge in prices at different locations will be different. The code should be able to handle this basic property and output surge in prices geographically. Our code handled this efficiently using Geohash.
- The other scenario was how to ingest the data. We used Kafka to effectively ingest the data. While ingesting data using Kafka, we had to decide the number of topics. We made only one topic, but if the throughput becomes too large then we would have to use multiple topics. Throughput is the number of events coming per second.
- The third scenario which we tested was how to process the data. We used Spark for this purpose. We made use of JavaDStream to store the incoming stream of data. Then the issue was to decide the window size which we chose to be 60 seconds.
- Then the issue was how to calculate surge for which we used the following formula - (number of customers)/(number of drivers). The surge was calculated for each Geohash string hence making sure the surge in prices is different for different locations.

- Then the fifth scenario was how to store the processed data so that it can be visualised and analysed to extract insights from the data. For this, we used ElasticSearch to store the data and Kibana to visualise it.

27. What was the size of your batch window in Spark? Why?

- The batch window size was taken to be 60 seconds or 1 minute. The window size was taken to be 60 seconds to prevent queueing of data. If the batch window is too small then it may be the case that till spark processes the data other batch comes in. The data will vary more and hence some batch window might get much more data as compared to other windows and hence processing it might take some time due to which queueing of the incoming batches might occur. 60 seconds is a decent batch window and chances of queueing of incoming batches gets reduced significantly. Further, since the window is more, the data will be distributed in a much more uniform manner across the batches.

28. How will you check whether the batch interval is set correctly?

- As we have already discussed that the Spark Streaming application is stable if end-to-end delay in processing the stream is not increasing. Note that sometimes, for some batches, there may be increase in the delay but that should not be considered for instability of Spark Streaming as long as that increase in delay is temporary. The total delay can be searched in log4j logs or it can be figured out from StreamingListener interface. Start with setting small batch interval and low rate of data. Now increase the data rate or decrease the delay according to your need to find out more optimised state.

29. Can you explain the approach used in Spark Streaming?

- Following steps are taken to calculate the surge in Spark:
- Driver and Customer DStreams are separated using the filter on user type.
- To determine the last location of each user, first, we need to group the users based on their ID and then find the last location from the group of locations for a particular ID.
- Once the last location is determined, Geohash from that location is calculated.
- Then users are grouped using the Geohash and the count of each user type is calculated on each Geohash.
- Finally, the surge is calculated using formula surge = demand (customer count at each location) / supply (driver count at each location)

30. How was surge calculated?

- The surge was calculated by finding the number of drivers and number of customers for a Geohash string and then dividing the number of customers by the number of drivers for that Geohash string. The number of customers is the demand for cabs and the number of drivers is the supply. So to calculate surge we use the formula - demand/supply so as demand increases surge increases and as supply increases surge decreases.

31. Was scalability taken into account?

- No. As the project was for academic purposes, the scalability was not taken into account.

32. What is the Spark library for elasticsearch that you used?

- The spark library used for elasticsearch is elasticsearch.spark.streaming.EsSparkStreaming.

33. How did you manage to push the data into ELK?

- We are writing the data to ES using saveToES() function from EsSparkStreaming library.

34. How did you visualise the data saved on elasticsearch?

- We visualise the data on Kibana. The Coordinate Map was used to visualise the output data which was taken from Elasticsearch using index pattern.

35. What was the refresh frequency and spark batch interval? Batch interval and refresh frequency did not sync? Why?

- Automatic refresh was not set in our project since that data was not streaming and all the computation was done within the first 60-second batch. Generally, if your batch interval is 60 seconds and the data is streaming you should keep the refresh frequency to be equal to 60 seconds. There is no point in keeping the refresh frequency to be less than the batch interval because the new results come on every batch interval. Keeping refresh frequency more than the batch interval will lead to seeing the same data on Kibana even though we have the new data already computed. Remember till now we have not set the sliding interval and if the sliding interval is set then the refresh frequency should be equal to the sliding interval.

36. Was your project on Eclipse or on the cloud?

- The project was run on Eclipse but the Kafka, Zookeeper, Elasticsearch and Kibana were present on EC2.

37. Why was Spark used? Any alternative for the same?

- It is easier to implement lambda architecture on Spark because batch and streaming API are common in Spark Streaming and Spark. (Remember we haven't used lambda architecture but often in the industry it is required)
- If the latency is an issue then Storm can be a better candidate.