Case Study Documentation:

Architecture (STREAM PROCESSING)

==========

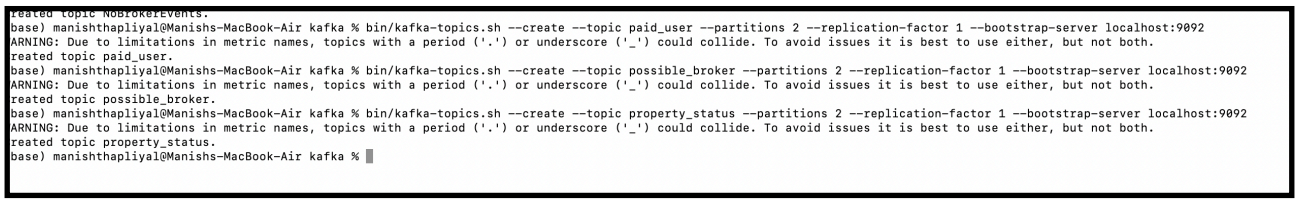
The source data is expected to be some streaming JSON log. Given that we are expecting 100000 records per minute, we can't scale the cluster to handle such a tremendous and streaming data using the live ETL. The best approach in such a scenario as per me is to use some Message-queuing framework so that we can treat the data in micro batches giving us the Near-Real-Time experience which is obviously one of the requirements from this case study.

To ensure the above scenario, I am using Kafka as the message-queuing framework as it can tackle streaming events very gracefully.
To ingest these data from the Kafka topics, spark-streaming is one of the best API as it can easily connect with Kafka and perform desired ETL on top of the data we will receive.

I am using spark 3.0.3 in my machine which is based on Scala 2.12. Hence, the JAR I am using to connect my spark with Kafka is also 2.12 dependent.
Below is the integration config I used in the code:

```
spark=SparkSession.builder\
    .config(conf=my_conf)\
    .config("spark.jars.packages","org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.3")\
    .getOrCreate()
```

The attached code file with this case study solution very well defines the logic used. To summarise, I've setup one standalone Kafka cluster on my local machine and created three topics namely Events, paid_user, possible_broker, property_status.
The input JSON stream will be of the form:



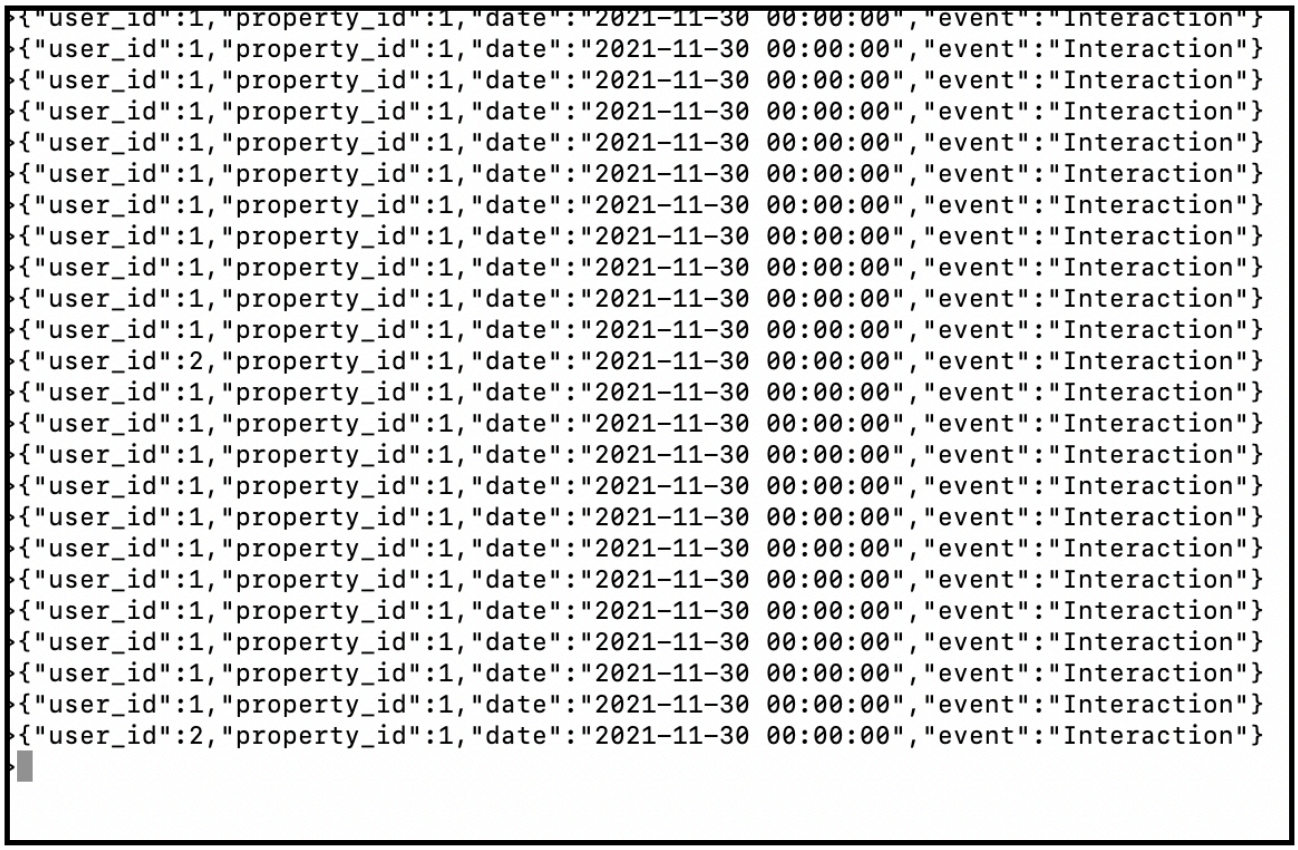
The ETL logic is set in such a way that the output events will be sinked to Kafka topic again based on the below logic:

1. If a user has done 9 or more interactions on any property, the user ID will be sent to the Kafka topic paid_user
2. If a property receives more than 5 interactions, we will send it to Kafka topic property_status
3. If a user does more than 10 interactions in a minute, the JSON record of that user will be pushed to the Kafka topic possible_broker

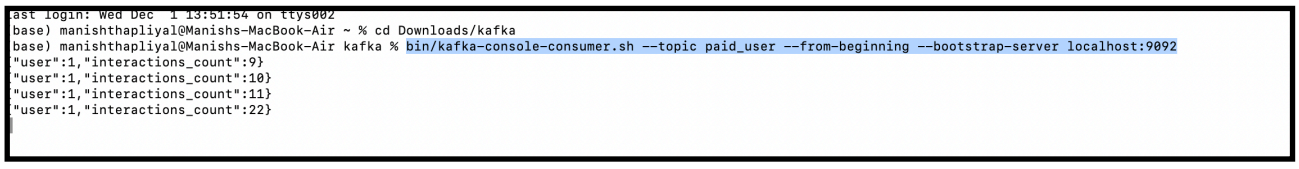
These messages in destination / sink Kafka topics can then be taken care by App Developers to take care of the further business logic incorporating the customer.

Below are the screenshot of destination Kafka topics that are serving the use case mentioned in the streaming architecture of given case study:

Producing the JSON strings in the input. If you see, the three conditions in the case study are getting met in the below input produced. Hence we are seeing the respective records in the corresponding Kafka topics:

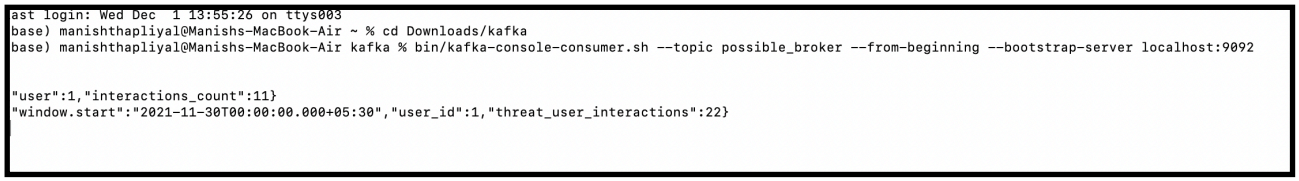


Topic 1 (paid_user)



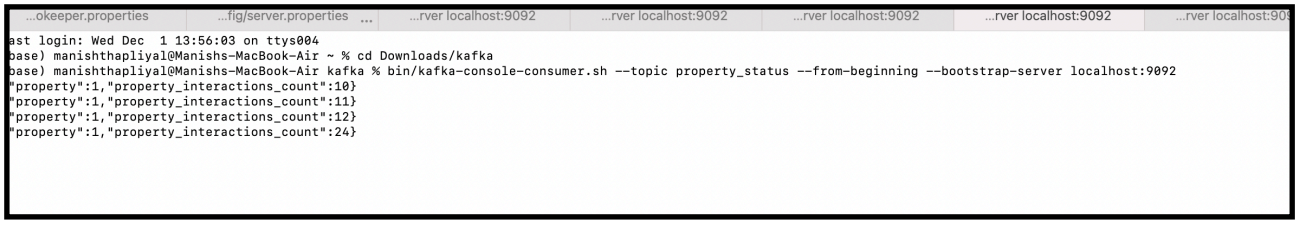
As can be seen, as soon as the user1 exceeds 9 transactions, the topic started displaying his/her details. Must be noted that I had sent 2 records for user 2 also but user 2 is not a part of **paid_user** topic. Hence the business rule is properly met.

Topic 2 (possible broker)



Here, user 1 entry was done 22 times within 1 minute of window and hence I used tumbling interval logic to display such users in this topic. Again user 2 must not be the part of this list which we can clearly see here.

Topic 3 (property status)



Here, the property 1 was viewed 24 times (22 times by user 1 and 2 times by user 2) and thus, such records are displayed in this topic. These records can then be taken up by other team to confirm if this property is still active.

So the above portion completes the streaming part (with all exhaustive hands-on as well).

One question here could be that the same user/property once exceeding threshold interactions will keep on getting displayed here. So what's the solution?
Answer: For this purpose, I have designed the HBase logic in the architecture diagram. HBase is very quick in inserts and updates as it is designed with NoSQL framework. It can handle millions of records per minute if not more.
The advantage with HBase is that the duplicate row keys are inserted as ***VERSIONS*** *instead of* ***NEW RECORDS*** *and hence we will have versioned history of each user/ property as a single entry in a database.*
So best scenario should be to insert the streaming output into both KAFKA topic and HBASE. Putting them to HBase will help us in quick retrieval of the records as well which can be used for analytics on archived data.
HBase focus on Consistency and Partition Tolerance part of CAP theorem (unlike Cassandra which focuses on Availability and Partition Tolerance). I believe, the strict

threshold of 9 interactions (for paid push user) and 5 interactions for broker threat is not compromizable. With this into assumption, I chose HBase over Cassandra

Architecture (BATCH PROCESSING)

==========

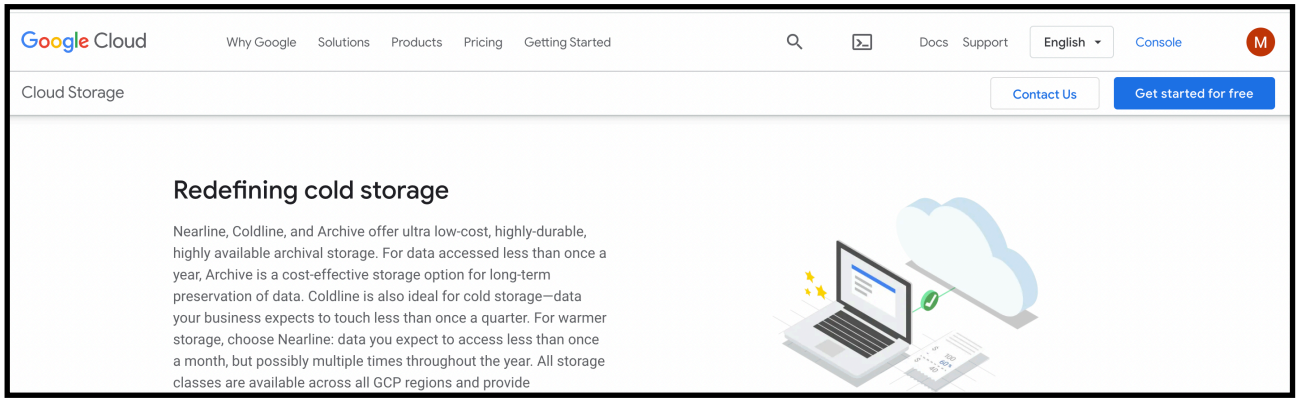
Solution 1

The batch part will be straight forward.

Instead of pushing the processed JSON into Kafka topics (as key value record), we can use spark streaming sink as csv/parquet file itself.
These files can be dumped to various storages such as HDFS/GCS/S3/ADLS etc.
As we know that stream processing will be series of micro-batches and hence we can store the files processed in a GCS buckets with the objects inside the buckets as csv files generated by micro-batches (logic could be developed to segregate the Objects on date basis.
This way one object will contain all files which were processed that day. Sample GCS bucket locations will look like:

gs://caseStudy/data_2021_11_25/
gs://caseStudy/data_2021_11_26/
gs://caseStudy/data_2021_11_27/
gs://caseStudy/data_2021_11_28/ and so on…

If we put them to GCS, we can use Archived Storage of Google Cloud.



The archived storages are best suited when we want to access data once in 365 days. Such storages as shown in the screenshot are very cheap and provided as SLA of about 99.9% on single region and 99.95% on dual-region. We can put the data in dual region archived storage to make it fault tolerant.

The GCS files can then be easily imported to GCP BQ which is Analytics and Warehouse solution hosted on Google cloud.
We can create attractive reporting tables on BQ for ML and analytics.
The final tables can be connected to visualisation tools like Power BI using live gateway connection between Power BI service and BQ.

Ideally Looker and Tableau have better edge for streaming data visualisation compared to Tableau. The reason I mentioned Power BI is my understanding and hands-on with it as a visualisation tool.

The GCP BQ API can easily interact with GCS API and can take the data in JSON format itself to create the table on top of it.
Below syntax can be used for the same:

bq load —source format=JSON —replace project-name:schema_name:table_name gs://gs://caseStudy/*

Once the data exists in BQ, we can connect it to power BI for visualisation purpose. All kind of ML can also be applied if we connect GCS data to Apache spark. We can think of transferring the processed data to BQ afterwords with this scenario.

Here is the documentation by Microsoft to connect Power BI with BQ using Native connector
https://docs.microsoft.com/en-us/power-bi/connect-data/desktop-connect-bigquery

Solution 2

The same logic mentioned in solution 1 can be implemented using solution 2 if we want to use Hadoop architecture everywhere.
1. Instead of GCS, we can use HDFS as sink (for storage purpose)
2. Instead of Google BigQuery, we can use Hive as open-source DataWarehouse solution for OLAP.

The reason I am giving HDFS also as an alternate to GCS is that Cloud storage might not display us all filesystem information and may have slightly higher latency. Also, it is not open source.

Big Query on the other hand runs on separate engine and is way faster than Hive. This definitely comes with some extra cost and Hence Hive can be treated as an alternative in such cases