**Report – Kafka challenge by Doodle**

**What did I do:**

Installed Kafka server and created 2 topics – one for input data and one for the output. The app processes streams preserving state over defined hopping windows of time. The window size is a minute and the time interval used to create a new window is a second. Each window accumulates uids and store them in a set (because it stores unique values) in a python dictionary. When window closes, the app calculates the number of unique users per window (the length of the set) and sink the number to the output topic along with timestamp and window interval values.

**Performance metrics:**

To see how many events are being processed every second by 1 instance of the consumer group, set IS\_PERF\_METRICS\_ON true.

My only instance of the consumer group processed 64 events/second in avg. Built-in faust monitor was used to collect the statistics.

**When to output the data:**

The data is being outputted every second. According to the requirements, the results should be displayed asap. The data is outputted when window closes.

**Possible errors in counting:**

1. When window closes there is a chance that not all data processed -> undercounting. To mitigate the issue a window should still exist some time after time is over, before closing.
2. Wrong counting when dealing with invalid data. Validation of the json should be implemented.

**For ingesting historical data,** kafka retention period was extended up to 2 years. (log.retention.hours). For historical analysis, moving such data into DWH can be an option.

**Scalability and HA**:

1. High availability must be provided before going on production. There is only 1 server used in my app. The replication factor should be defined according to the requirements and through testing process.
2. To achieve production performance all topics should use more partitions. There is only 1 partition per topic in my app, only for the prototype version. The number of required partitions should be defined through testing process and according to the business requirements. Increasing the number of partitions will require changes for the app code:
   1. The number of the consumer group instances should be increased respectively in order to improve reads performance.
   2. The number of the changelog topic partitions (used as a persistence storage for fault tolerance on the client) should be increased respectively.
   3. All consumers should collect a set of unique users per partition then send the set to the intermediate topic with 1 partition. Then another consumer will combine the number of unique users among all sets (by union method) coming from all partitions and calculate the number (by len method) of the unique users per window. Such aggregation can be based on batch\_id which can be calculated based on timestamp field.

**What else should be implemented before going on production:**

1. More metrics, including json ser/deser metrics should be collected and outputted to the log analytic tool to visualize statistics and knowledge. It can be, for example, DataDog or any other analytic tool used in the company. Based on the collected information, improvements can be suggested. Based on my personal experience json format usually shows the slowest time to read, so I would suggest comparing json with avro, orc, parquet formats and choose the best one based on carried out tests.
2. Validation of incoming jsons. At least basic type check should be implemented to improve quality of the data.
3. The strategy of disaster recovery should be tested and documented well. One of the options to consider can be to replay the changelog upon the failure, so the state of the table (python dictionary) can be recovered as it was before the failure.
4. At the moment the app processes events relatively to their incoming time. It can be reconsidered to the direction of being relative to ts field in the data.
5. Out of order events should be handled by faust library while window exists, however, more tests here are required.
6. Lots of python code refactoring required – config values should be moved to the config, model classes should be moved to separate modules, the logger should be moved to a separate module etc.