Data Engineer Challenge

The goal of the challenge is to have a tool that is able to stream data from kafka and count unique things within this data. The simplest use case is that we want to calculate unique users per minute, day, week, month, year. [1] For a very first version, business wants us to provide just the unique users per minute. [1]

- The data consists of (Log)-Frames of JSON data that are streamed into/from apache kafka.
- Each frame has a timestamp property which is unixtime, the name of the property is `ts`.
- Each frame has a user id property calles `uid`.
- You can assume that 99.9% of the frames arrive with a maximum latency of 5 seconds.
- you want to display the results as soon as possible
- the results should be forwarded to a new kafka topic (again as json.) choose a suitable structure. [1]
- for an advanced solution you should assume that you can *not* guarantee that events are always strictly ordered. [2]

Requirements:

- provide a Readme that shows how to build and run the code on Linux or OS X
- write a report: what did you do? what was the reasons you did it like that?
- measure at least one performance metric (e.g. frames per second)
- document your approach on how you decide **when** to output the data [1]
- document the estimated error in counting [3]
- it should be possible to ingest historical data. e.g. from the last 2 years. [1]

sample data

For a quick start you can use the sample data provided at http://tx.tamedia.ch.s3.amazonaws.com/challenge/data/stream.jsonl.gz:

it was generated using

```
./data-generator -c 1000000 -o stream.jsonl -r 1000 -n 100000
```

```
cat stream.jsonl | jq .uid -r | gsort --parallel=4 | guniq | wc -l
```

yields `99993` unique ids.

you can just input this file to you kafka topic once by something like

```
gzcat stream.gz | kafka-console-producer --broker-list localhost:9092 --topic
mytopic
```

You can also generate your own data by using the data-generator tool. it allows you to introduce random-ness if you have a more advanced solution and want to test it. it requires you to install a Dlang compiler and D's package manager. then execute`dub build` to build the binary. binaries for os x and linux are in the bin directory.

suggested steps:

basic solution

1. install kafka
2. create a topic
3. use the kafka producer from kafka itself to send our test data to your topic
4. create a small app that reads this data from kafka and prints it to stdout
5. find a suitable data structure for counting [1] and implement a simple counting mechanism, output the results to stdout

   ##### advanced solution

6. benchmark
7. Output to a new Kafka Topic instead of stdout [1]
8. try to measure performance and optimize
9. write about how you could scale [1]
10. only now think about the edge cases, options and other things [3]

Evaluation Criteria & Expectations

- ability to break down *business* requirements into simple prototype code
- clean project setup and documentation
- research and use of suitable data structure for a specific use case. explain which and why. [3]
- ability to write performant code to handle streaming data. measure and document *how* fast your solution is.
- Understanding how to benchmark and analyze performance bottle necks. what tools did you use? [3]
- awareness of the mechanisms and costs of serialization. Explain (and ideally prove!) why json is an ideal format here or why not and then suggest a better solution.
- scalability: *explain* how you *would* scale your approach [2]

Bonus Questions / Challenges:

- how do you scale it to improve troughput. [1]
- you may want count things for different time frames but only do json parsing once. [1]
- explain how you would cope with failure if the app crashes mid day / mid year. [1]
- when creating e.g. per minute statistics, how do you handle frames that arrive late or frames with a random timestamp (e.g. hit by a bitflip), describe a strategy? [2]
- make it accept data also from std-in (instead of kafka) for rapid prototyping. (this might be helpful to have for testing anyways)
- measure also the performance impact / overhead of the json parser

Hints:

- tap into other peoples know-how and code.
- expected time is ~8 hours. If you are above that think about which parts to leave out and just document HOW you would do them.
- you should not use any big data framework, just your favourite fast programming language that has a Kafka driver. The most simplest version to archieve step 5 can be done in about 10 lines of code (plus the boilerplate to read from kafka).
- If you found the cheatcode for level 5 (well done!) you should try to archieve level 10 ;)
- check that your last commit compiles.
- be smart and impress us. It does not matter if you impress us with nice clean code or with a very clever hack to archieve the business goal in short time.