Задание для разработчиков (Java)

> Если ваша сильная сторона - администрирование, можете не пытаться выполнить это задание, а сразу переходить на следующий шаг.

Цель: проверить полученные знания на практике, создав ReadProcessWrite приложение для генерации real-time статистики продаж.

Задача:

Давайте представим, что у нас есть успешный интернет-магазин. На прошлой неделе наши инженеры установили кластер Apache Kafka, в топик которого отправляются сообщения о всех проведённых оплатах на нашем сайте. Формат сообщения следующий:

```
{
   "uuid":"3ee4184a-f48d-4dc3-91d3-08bbc477f2ce",
   "productName":"Fantastic Aluminum Car",
   "chargedAmount":100,
   "creditCardNumber":"6767-6333-6265-7320",
   "creditCardType":"SOLO",
   "countryCode":"RO",
   "isSuccessful":false
}
```

Наша задача — создать аналитическое приложение, которое бы считывало успешные платежи из топика *(см. флаг “isSuccessful”)* и агрегировало их суммы в поминутные срезы *(i.o.w. поминутные окна, Tumbling Window)*. Подсчитанные агрегации должны отправляться в отдельный топик. Пример такой минутной агрегации:

```
{
   "windowStartTime": "30-03-2021T05:57:37+00:00",
   "windowEndTime": "30-03-2021T05:58:37+00:00",
   "windowStats": {
```

```
        "byCountry”: {
            "NL": 3429,
            "BE": 215
        },

        "byCreditCardType": {
            "VISA": 3413,
            "SOLO": 13
        }
    }
}
```

Постарайтесь подойти к задаче итеративно, двигаясь от простого решения к более сложному — кажущаяся простота скрывает под собой массу вопросов связанных с обработкой времени. Для решения подобного класса задач созданы целые фреймворки, например Kafka Streams или Apache Flink!

- В качестве генератора случайных оплат вы можете воспользоваться классом io.slurm.kafka.TestProducer из https://gitlab.slurm.io/edu/kafka/-/tree/master/test-clients
- Вы также можете воспользоваться Docker для запуска кластера Apache Kafka локально: https://gitlab.slurm.io/edu/kafka/-/tree/master/kafka-docker
- В качестве примера ReadWriteApp приложения посмотрите на https://gitlab.slurm.io/edu/kafka/-/blob/master/test-clients/src/main/java/io/slurm/kafka/ReadProcessWriteExactlyOnceApp.java
- Подумайте, каким образом вы будете высчитывать границы минутного окна? Можем ли мы использовать для этого локальное время нашего приложения (Processing Time), какие плюсы и минусы есть у этого подхода по сравнению с использованием времени отправки/генерации самих сообщений (Event Time [0])? **Учтите, что корректная имплементация Event Time Windowing крайне сложна и выходит за рамки нашего курса!**
- Для упрощения задачи, предположим, что у в исходном топике оплат не бывает out-of-order или late сообщений. Мы можем гарантировать [1] это выставив конфигурационную опцию топика message.timestamp.type в LogAppendTime
- Каким образом мы можем гарантировать корректность репорта: например, отсутствие повторных обработок одних и тех же сообщений или потерь данных? [2]
- При желании, попробуйте так же имплементировать репорт используя библиотеку Kafka Streams — https://kafka.apache.org/27/documentation/streams/tutorial

[0] — помните, что каждое сообщение в Apache Kafka имеет встроенный атрибут timestamp, который либо выставляется продюсером при отправке (CreateTime), либо добавляется самим брокером при сохранении сообщения на диск (LogAppendTime).

[1] — на самом деле LogAppendTime не дает 100% защиту от out-of-order message timestamps в Apache Kafka. В зависимости от временной разницы (clock drift) между вашими брокерами очередность timestamp у сообщений может быть сломана при фейловере лидера партиции. Всегда пользуйтесь NTP для синхронизации таймеров на ваших брокерах!

[2] — подумайте, какой встроенный функционал клиентов Apache Kafka поможет нам этого добиться?

Ответы на это задание можно найти в репозитории курса:
https://gitlab.slurm.io/edu/kafka/-/tree/master/final-practice-clients