Санкт-Петербургский государственный университет

Прикладная математика и информатика

Отчет по учебной практике 2 (научно-исследовательской работе) (семестр 3)

Потоковая обработка данных при помощи Kafka Streams

Выполнил:

Погребников Николай Вадимович, группа 21.Б05-мм

Научный руководитель:

Старший преподаватель, к.ф.-м.н., Ph.D.

Михаил Валерьевич Благов.

Кафедра прикладной кибернетики

Работа выполнена на отличном уровне и в срок. Оценка «А»

Санкт-Петербург

2022

ОТЗЫВ
научного руководителя на научно-исследовательскую работу (семестр 3) студента Погребникова Николая Вадимовича на тему «Потоковая обработка данных при помощи Kafka Streams».

Тема работы является актуальной. При выполнении работы студент проявил самостоятельность, разработал новое ПО. Работа выполнена на отличном уровне, отчет составлен грамотно и полностью отражает результаты работы студента.

Обучающимся освоены следующие компетенции: ОПК-3, ОПК4, ПКА-2, ПКП-1, ПКП2, ПКП-3, ПКП-4, УК-2, УКБ-1, УКБ-2.

Оценка ЗАЧТЕНО (‘A’).

Научный руководитель:

Старший преподаватель, Кафедра прикладной кибернетики,
к.ф. - м.н.

Благов Михаил Валерьевич

Оглавление

Введение

В современном мире всё чаще и чаще часто возникают задачи обеспечения связи и обмена информацией между приложениями или отдельными модулями в режиме реального времени. В большом количестве прикладных задач приложениям важно уметь общаться быстро и без перебоев. Kafka Streams является отказоустойчивой системой обработки сообщений, как раз эта потоковая платформа помогает многим программистам справляться с вышеперечисленными трудностями.

Kafka как инструмент потоковой обработки данных

Что такое потоковая передача событий?

“Потоковая передача событий - это цифровой эквивалент центральной нервной системы человеческого организма. Это технологическая основа для "постоянно работающего" мира, где бизнес становится все более и более программно-определяемым и автоматизированным.” Такое определение дают создатели Kafka [7].

С технической точки зрения, потоковая передача событий - это практика сбора данных в режиме реального времени из источников событий, таких как базы данных, датчики, мобильные устройства, облачные сервисы и программные приложения, в виде потоков событий; долговременное хранение этих потоков событий для последующего извлечения; управление, обработка событий в в режиме реального времени. Таким образом, потоковая передача событий обеспечивает непрерывный поток и интерпретацию данных, чтобы нужная информация была в нужном месте и в нужное время**.**

Потоковая передача событий применяется к широкому спектру вариантов использования во множестве отраслей и организаций. Многочисленные примеры включают в себя: обработку платежей и финансовых транзакций в режиме реального времени, например, на фондовых биржах, в банках и страховых компаниях; отслеживание и мониторинг легковых и грузовых автомобилей, автопарков и отправлений в режиме реального времени, например, в логистике и автомобильной промышленности; непрерывный сбор и анализ данных датчиков с устройств, например, на заводах.

Основные идеи Apache Kafka

Kafka объединяет три ключевые возможности, чтобы реализовать свои варианты использования потоковой передачи событий:

1. Публиковать (produce) и подписываться на (consume) потоки событий, включая непрерывный импорт/экспорт данных из других систем.
2. Хранить потоки событий надежно и настолько долго, насколько хочется.
3. Обрабатывать потоки событий по мере их возникновения.

И вся эта функциональность предоставляется распределенным, масштабируемым, гибким, отказоустойчивым и безопасным способом. Kafka может быть развернут на простом оборудовании, виртуальных

машинах и контейнерах, как локально, так и в облаке. Можно выбирать между самостоятельным управлением сервисом Kafka и использованием полностью управляемых сервисов.

Более подробную информацию об Apache Kafka можно узнать в книге “Apache Kafka. Потоковая обработка и анализ данных”[1] или в официальной документации[7].

В чём разница между Apache Kafka и Kafka Streams?

Apache Kafka - это серверное приложение, которое предоставляет способ обмена потоками событий между приложениями.
Kafka Streams - это API для написания клиентских приложений, которые преобразуют данные в Apache Kafka. Обычно это делается путём публикации преобразованных данных в топики. Сама обработка данных происходит в конкретном клиентском приложении, а не в брокере Kafka.

Основные понятия Kafka Streams

Событие фиксирует тот факт, что "что-то произошло" в мире или в конкретном бизнесе. В документации это также называется записью(record) или сообщением(message). Все данные в Kafka, представлены в форме событий. Концептуально событие имеет ключ, значение, временную метку и необязательные метаданные. Пример события:

- Ключ события: "Алиса"
- Значение события: "Осуществил платеж в размере 200 долларов США Бобу"
- Временная метка события: "25 июня 2020 года в 2:06 вечера".

Производители(producers) - это те клиентские приложения, которые публикуют (produce) события в Kafka, а потребители(consumers) - это те, которые подписываются на эти события. В Kafka производители и потребители полностью отделены и независимы друг от друга. События организованы и надежно хранятся в топиках(topic). Топик похож на папку в файловой системе, а события - это файлы в этой папке. Примером названия темы может быть "платежи". Топик в Kafka — неограниченная последовательность key-value пар. Ключи и значения — обычные массивы байтов, т.е. <byte[], byte[]>. Кроме того, топики разделены, имеют свои партиции(partition), откуда следует, что топик распределен по нескольким "корзинам", расположенным на разных брокерах Kafka.

При обсуждении данных, находящихся в таких системах, как Kafka, часто используется термин поток данных или стрим(stream). Стрим —

топик со схемой. Ключи и значения больше не массивы байтов, а имеют определённые типы, например <String, Integer>.

Таблица (table) — таблица в обычном смысле этого слова, однако с некоторыми оговорками. Если смотреть через призму потоковой обработки, можно сказать, что таблица также является агрегированным стримом.

Важным фактом является то, что в любой момент времени можно преобразовать таблицу в стрим и наоборот. Однако в отличие от систем для работы с базами данных, здесь во главе стоят стримы, а не таблицы. Это является ключевой особенностью Kafka Streams.

Основные функции Kafka Streams

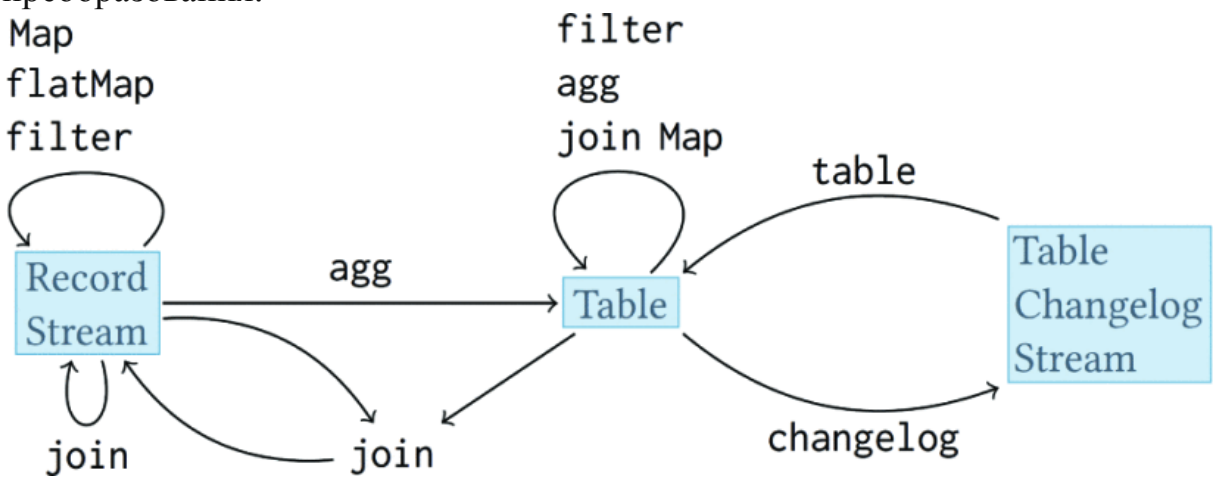
В Apache Kafka есть два потоковых API — низкоуровневый Processor API и высокоуровневый Streams DSL. Рассмотрим Kafka Streams DSL, который позволит задать приложение потоковой обработки путем описания последовательности преобразований событий потока. Преобразования могут быть простыми, например, фильтрами, или сложными, например, соединениями потоков. Создание приложения, задействующего API DSL, всегда начинается с формирования с помощью StreamBuilder топологии обработки — ориентированного ациклического графа (DAG) преобразований, применяемых ко всем событиям потоков. Затем на основе топологии создается исполняемый объект KafkaStreams. При запуске объекта KafkaStreams создается несколько потоков выполнения, каждый из которых использует топологию обработки к событиям из потоков. Обработка завершается по закрытии объекта KafkaStreams. Чтобы считывать топик, нужно применить функцию stream(String topic, Consumed <K, V> consumed) к экземпляру класса StreamBuilder, после чего создастся стрим, KStream, в котором в каждой записе ключом будет являться объект класса K, а значение - V. Кроме того, можно дополнительно применить одну из функций map(), flatMap(), flatMapValues() или mapValues(), чтобы изменить представление наших данных, например следующая строка преобразует значение записи из типа Long в String : mapValues(value->Long.toString(value));

Преобразование стрима в таблицу осуществляется с помощью этапа агрегации. Агрегация — это одна из разновидностей потоковой обработки, она означает, что для каждого ключа мы сжимаем множество значений в одно. Данные в Kafka представлены в виде пар ключ-значение. Одна из особенностей агрегаций в Kafka заключается в том, что все они вычисляются по ключу. Чтобы использовать агрегацию, нужно сгруппировать KStream через groupBy() или groupByKey(). Далее,

агрегации непрерывно обновляются как только новые данные поступают во входящие стримы. Вместе со свойством вычисления по ключу это требует наличия таблицы или, более точно, это требует изменяемую таблицу (mutable table) в качестве результата и, следовательно, типа возвращаемых агрегаций. Предыдущие значения (результаты агрегаций) для ключа постоянно перезаписываются новыми значениями. В Kafka Streams агрегации всегда возвращают таблицу.

Рассмотрим метод count(). Он является обычным сумматором, одной из разновидностей агрегации. count() может быть заменён на любую другую функцию, если использовать метод aggregate(), который имеет один из параметров - функцию. aggregate() можно использовать, например, для подсчёта среднего значения. Кроме того, эти два метода имеют параметр хранилища(state store). К этим хранилищам можно обращаться в режиме реального времени, а также гибко их настраивать.

Помимо этого, таблица в Kafka Streams имеет свой стрим вывода. Подобно записи данных об изменении в базах данных, каждое изменение в таблице в Kafka заносится во внутренний стрим изменений называемый changelog stream таблицы. Много вычислений в Kafka Streams фактически выполняются на changelog stream. Таким образом, можно преобразовать таблицу в стрим изменений через toStream(). Отличное пояснение того, как работает превращение таблицы в стрим и обратно есть в научной статье [6](Section 2). Следующая картинка наглядно демонстрирует данные преобразования.



Аналитика посещаемости мероприятия во времени, близкому к реальному

Постановка задачи

Научный руководитель поставил передо мной задачу подсчета метрик мероприятия по данным от специально созданного telegram бота, благодаря которому все пришедшие гости могут получить за участие в той или иной активности внутреннюю валюту, а затем обменять её на реальный товар. Требуется в реальном времени выводить статистику по процентному соотношению используемых команд в боте, средней стоимости купленного товара и вознаграждения, прохождению той или иной активности.

Решение проблемы

Логика работы в терминах Kafka Streams: бот является производителем сообщений, записи отправляются на вычисление метрик в отдельное приложение, а затем отсылаются потребителю в обработанном виде.

Для начала рассмотрим производителя - бота. Мною были созданы три независимых друг от друга топика с названиями “command-counter-input”, “avg-input”, “trials-input” для подсчета процентного соотношения команд к общему числу, средних значений и выполнения активностей, соответственно. Этот производитель сообщений создавался в программе MyProducers.cs [3] на языке C#(бот был написан на данном языке). Всякое событие, которое может изменить статистику, отправлялось в нужный топик, например:

```
MyProducers.Produce("avg-input", "prize", trial.Reward.ToString());
```

Здесь запись с ключом “prize” и значением строкового типа, эквивалентным количеству валюты за выполнение определённого задания, отправлялось в топик “avg-input”.

Все эти топики обрабатываются потоками Kafka в файле MyKafkaStreams.java [1]. Для определения процентного соотношения команд к общему числу пришлось создать отдельный класс с статическим полем sum и не статическим count. sum будет считать общее количество команд, а count - количество использований конкретной команды. Была создана отдельная Kafka таблица для подсчета команд и хранилище с названием “cm-stream-store”. Сама функция aggregate выглядит следующим образом:

```
aggregate(
    ()->new JsonCountAndSumCommands.CountAndSumCommands(0L),
    (key, value, aggregate) -> {
      aggregate.setCount(aggregate.getCount() + 1);
      JsonCountAndSumCommands.CountAndSumCommands.setSum(
      JsonCountAndSumCommands.CountAndSumCommands.getSum() + 1);
      return aggregate;
    },
     Materialized.<String,
     JsonCountAndSumCommands.CountAndSumCommands>as(cmStore)
          .withKeySerde(Serdes.String())
          .withValueSerde(new JsonCountAndSumCommands.JSONSerde<>())
          .withLoggingDisabled());
```

Здесь увеличиваются на единицу счётчики общего числа команд и конкретной команды, а затем сохраняются в хранилище, принимающее за ключ строку и за значение вышеупомянутый класс. Кроме того, строка withLoggingDisabled() означает, что при закрытии программы данные из хранилища удалятся(потребитель итак сохраняет их в отдельный файл). Функция aggregate() вернёт KTable с названием cmTable.

Затем была создана ещё одна Kafka таблица, основанная на cmTable, которая вычисляет процентное соотношение с помощью метода mapValues() и отправляет в выходной топик:

```
final KTable<String, Double> commandsPercent = cmTable
    .mapValues(value -> (((double) value.getCount()) / (double)
      JsonCountAndSumCommands.CountAndSumCommands.getSum())*100,
     .Materialized.<String, Double>as(percentStore)
      .withKeySerde(Serdes.String())
      .withValueSerde(Serdes.Double())
      .withLoggingDisabled()
    );
commandsPercent.toStream().mapValues((v)->v.toString())
      .to("command-percent-output", Produced.with(Serdes.String(),
        Serdes.String()));
```

Аналогично происходит с обработкой средних значений, только немного меняются функции агрегации и mapValues:

```
.aggregate(
    () -> new JsonAvg.CountAndSum(0, 0),
    (key, value, aggregate) -> {
       aggregate.setCount(aggregate.getCount() + 1);
       aggregate.setSum(aggregate.getSum() + value);
```

```
        return aggregate;
    }, …);

…

.mapValues(value ->  ((double)value.getSum()) / (double) value.getCount(), …);
```

А для подсчёта количества выполненных испытаний вместо метода aggregate используется обычный count. Всё это отправляется в соответствующие выходные топики.

Теперь рассмотрим потребителя. Он находится в файле MyConsumer.java[2]. Ниже показан кусок кода, в котором потребитель создаётся и подписывается на события из выходных топиков:

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
List<String> lst = new ArrayList<>();
lst.add("command-percent-output");
lst.add("counters-output");
lst.add("average-output");
consumer.subscribe(lst);
```

Данный потребитель сначала считывает записи с помощью строки “ConsumerRecords<String, String> records = consumer.poll(100);” , а затем запоминает их в словари. Если написать в командную строку “1”, он выведет все записи на экран, а если “0”, то прекратит считывание новых записей и сохранит существующие в отдельные файлы(для каждого топика свой файл), а следом завершит свою работу.

Что удалось выяснить с помощью Kafka Streams?

Одну из активностей мероприятия под названием “задать вопрос” не выполнил ни один из участников. Всего бота запускали 79 человек, и самым популярным испытанием стало “сделать пасхальное фото” - 30 конкурсантов его выполнили. Средняя стоимость закупки - 483 внутренней валюты, а среднее вознаграждение от одного задания - 85. Самыми популярными командами, если не брать в расчёт команды от организаторов, стали “Назад” - 12.7%, что означает вернуться назад в меню, “Прислать котика” - 8.7% (отправляет случайное фото кота), “Текущие миссии” - 3.5%(выводит список непройденных активностей), процент ошибочных команд - 1. Полная информация о статистике мероприятия находится в приложениях [4], [5], [6].

Заключение

Благодаря Kafka можно получить гибкую, масштабируемую и отказоустойчивую распределенную потоковую обработку. Я разобрался в использовании такого инструмента как Kafka Streams, применил его на практике и получил удовлетворительный результат. Поставленная задача была решена.

Приложения

1. Код обработки команд бота
   https://github.com/xSICHx/KafkaStreams/tree/master/src/main/java/Stream
2. Код потребителя
   https://github.com/xSICHx/KafkaStreams/blob/master/src/main/java/Consumer/MyConsumer.java
3. Код производителя
   https://github.com/xSICHx/DVFUTelegramBotC-/tree/kafka/TelegramBotDVFU
4. Процентное соотношение команд, используемых в боте, к общему числу
   https://github.com/xSICHx/KafkaStreams/blob/master/commandPercentage.txt
5. Средние значения закупки и выигрыша
   https://github.com/xSICHx/KafkaStreams/blob/master/avg.txt
6. Счётчики количества выполненных испытаний и человек
   https://github.com/xSICHx/KafkaStreams/blob/master/counters.txt

Список литературы

1. Нархид Ния, Шапира Гвен, Палино Тодд. Apache Kafka. Потоковая обработка и анализ данных. — СПб.: Питер, 2019. — 320 с.
2. Koutanov Emil. Effective Kafka A Hands-On Guide to Building Robust and Scalable Event-Driven Applications with Code Examples in Java. — Victoria.: Leanpub, 2021. — 394p.
3. Nishant Garg. Apache Kafka Set up Apache Kafka clusters and develop custom message producers and consumers using practical, hands-on examples. — Birmingham.: Packt Publishing Ltd, 2013. — 88p.
4. H. Wu, Z. Shang, K. Wolter. Performance Prediction for the Apache Kafka Messaging System. — IEEE 21st International Conference on High Performance Computing and Communications.: 2019. — Режим доступа : https://ieeexplore.ieee.org/abstract/document/8855525 — Загл. с экрана.
5. A. B. A. Alaasam, G. Radchenko, A. Tchernykh. Stateful Stream Processing for Digital Twins: Microservice-Based Kafka Stream DSL. — 2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON).: 2019. — Режим доступа : https://ieeexplore.ieee.org/abstract/document/8958367 — Загл. с экрана.
6. S. Langhi, R. Tommasini, E. D. Valle. Extending Kafka Streams for Complex Event Recognition. — 2020 IEEE International Conference on Big Data (Big Data).: 2020. — Режим доступа : https://ieeexplore.ieee.org/abstract/document/9378217 — Загл. с экрана.
7. Apache Kafka documentation. — Режим доступа: https://kafka.apache.org/documentation. — Загл. с экрана.