

Потоковая обработка данных

Оглавление

- [Введение](#)
- [1. Проверить работу потока](#)
- [2. Прочитать данные об акциях из Kafka](#)
- [3. Прочитать данные о подписчиках из Postgres](#)
- [4. Преобразование JSON в dataфейм](#)
- [5. Провести JOIN потоковых и статичных данных.](#)
- [6. Отправить результаты JOIN в Postgres для аналитики фидбэка](#)
- [7. Отправить данные, сериализованные в формат JSON, в Kafka для push-уведомлений.](#)
- [8. Персистентность dataфрейма.](#)

Введение ▲

Вы построили приложение для потоковой обработки данных. Пришла пора закрепить изученное в спринте! Вы продолжите работать с тем же продуктом — агрегатором доставки — и построите новое приложение. Задача будет похожая, но мы её немного усложним.

В учебном проекте, если условия рекламных кампаний и местоположение пользователя совпадали по заданным признакам, то ваш сервис отправлял сообщение в сервис `push`-уведомлений, а уже этот сервис `push`-уведомлений — уведомления пользователям.

Сейчас вам предстоит создать приложение, которое будет сужать круг пользователей ещё больше и доставлять уведомления об акциях с ограниченным сроком действия.

Описание задачи

Ваш агрегатор для доставки еды набирает популярность и вводит новую опцию — подписку. Она открывает для пользователей ряд возможностей, одна из них которых — добавлять рестораны в избранное. Только тогда пользователю будут поступать уведомления о специальных акциях с ограниченным сроком действия. Систему, которая поможет реализовать это обновление, вам и нужно будет создать.

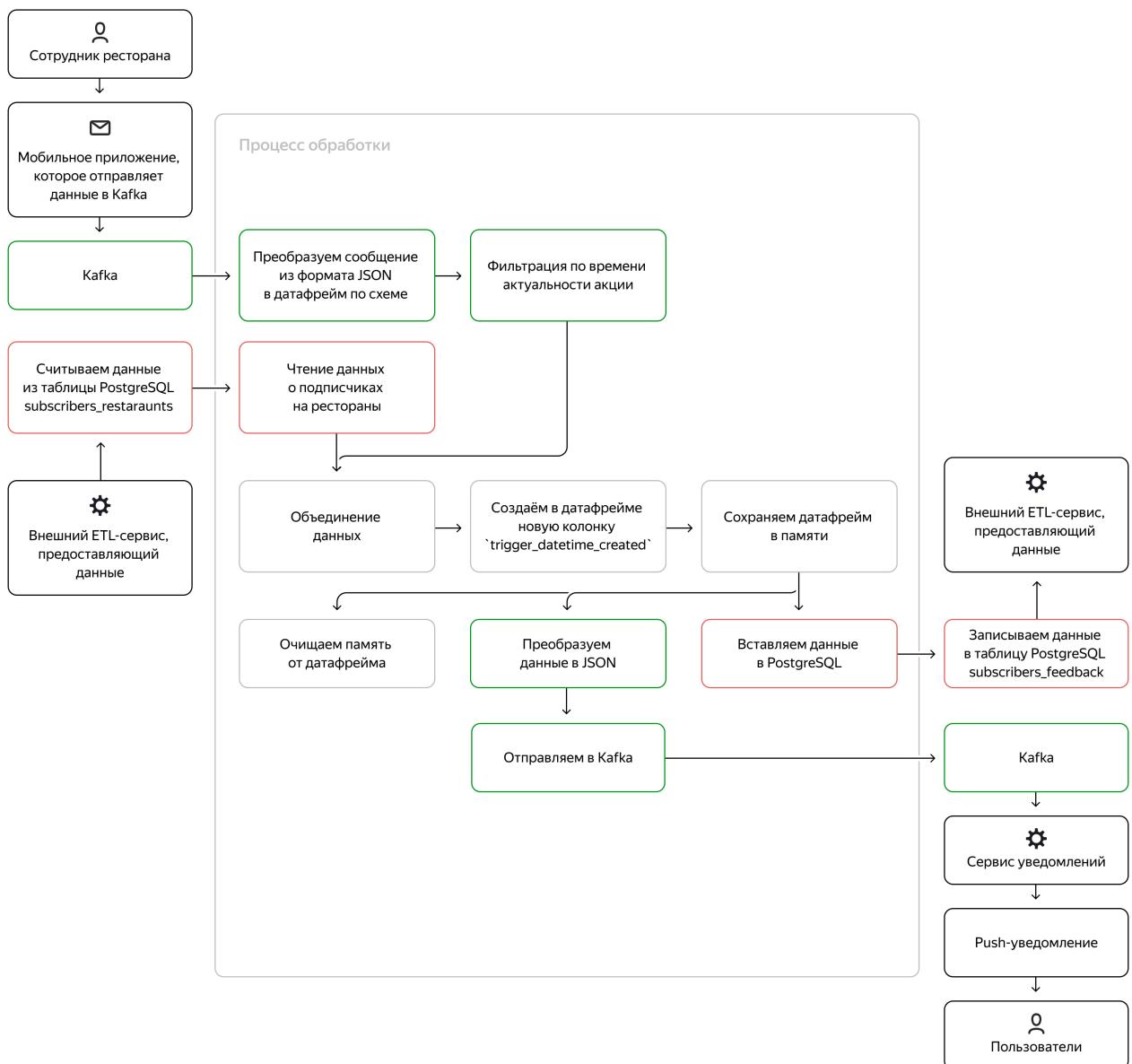
Благодаря обновлению рестораны смогут привлечь новую аудиторию, получить фидбэк на новые блюда и акции, продать профицит товара и увеличить продажи в непиковые часы. Акции делятся недолго, всего несколько часов, и часто бывают ситуативными, а значит, важно быстро доставить их кругу пользователей, у которых ресторан добавлен в избранное.

Система работает так:

1. Ресторан отправляет через мобильное приложение акцию с ограниченным предложением. Например, такое: «Вот и новое блюдо — его нет в обычном меню. Дарим на него скидку 70% до 14:00! Нам важен каждый комментарий о новинке».
2. Сервис проверяет, у кого из пользователей ресторан находится в избранном списке.
3. Сервис формирует заготовки для push-уведомлений этим пользователям о временных акциях. Уведомления будут отправляться только пока действует акция.

План проекта

Итак, ваша задача — реализовать сервис. Схематично его работа будет выглядеть так:



Сервис будет:

1. Читать данные из `Kafka` с помощью `Spark Structured Streaming` и `Python` в режиме реального времени.

В отличие от учебного проекта, у вас не будет входного потока данных в `Kafka` от генератора данных. Здесь вам нужно самостоятельно отправлять тестовое сообщение в `Kafka` с помощью `kcat`.

Обратите внимание: `restaurant_id` должен соответствовать входной таблице из `Postgres`, а поля `adv_campaign_datetime_start` и `adv_campaign_datetime_end` — актуальной информации, то есть текущему времени.

Посмотрите, как выглядит входное сообщение.

```
first_message:{"restaurant_id": "123e4567-e89b-12d3-a456-426614174000","adv_campaign_id": "123e4567-e89b-12d3-a456-426614174003","adv_campaign_content": "first campaign","adv_campaign_owner": "Ivanov Ivan Ivanovich","adv_campaign_owner_contact": "iiivanov@restaurant.ru","adv_campaign_datetime_start": 1659203516,"adv_campaign_datetime_end": 2659207116,"datetime_created": 1659131516}
```

В наглядном виде:

```
first_message:{  
    "restaurant_id": "123e4567-e89b-12d3-a456-426614174000",  
    "adv_campaign_id": "123e4567-e89b-12d3-a456-426614174003",  
    "adv_campaign_content": "first campaign",  
    "adv_campaign_owner": "Ivanov Ivan Ivanovich",  
    "adv_campaign_owner_contact": "iiivanov@restaurant.ru",  
    "adv_campaign_datetime_start": 1659203516,  
    "adv_campaign_datetime_end": 2659207116,  
    "datetime_created": 1659131516  
}
```

В этом сообщении:

- `first_message` — ключ. В рамках задачи он может быть произвольным, но для тестирования кода можно использовать номер сообщения.
- `:` — разделитель ключа и сообщения.

Далее идёт тело самого сообщения.

- `"restaurant_id": "123e4567-e89b-12d3-a456-426614174000"`, — UUID ресторана;
- `"adv_campaign_id": "123e4567-e89b-12d3-a456-426614174003"`, — UUID рекламной кампании;
- `"adv_campaign_content": "first campaign"`, — текст кампании;
- `"adv_campaign_owner": "Ivanov Ivan Ivanovich"`, — сотрудник ресторана, который является владельцем кампании;
- `"adv_campaign_owner_contact": "iiivanov@restaurant.ru"`, — его контакт;
- `"adv_campaign_datetime_start": 1659203516`, — время начала рекламной кампании в формате `timestamp`
- `"adv_campaign_datetime_end": 2659207116`, — время её окончания в формате `timestamp`
- `"datetime_created": 1659131516` — время создания кампании в формате `timestamp`

Все поля, кроме `restaurant_id`, `adv_campaign_datetime_start` и `adv_campaign_datetime_end`, могут быть названы произвольно.

2. Получать список подписчиков из базы данных Postgres

Посмотрите на таблицу `subscribers_restaurants` — это входная таблица данных для проекта, которую вы будете использовать в дальнейшем.

```
-- DROP TABLE public.subscribers_restaurants;
```

```
CREATE TABLE public.subscribers_restaurants (
    id          serial4 NOT NULL,
    client_id   varchar NOT NULL,
    restaurant_id varchar NOT NULL,
    CONSTRAINT pk_id PRIMARY KEY (id)
);
```

Пример заполненных данных

	<code>id</code>	<code>client_id</code>	<code>restaurant_id</code>
1	223e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000	
2	323e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000	
3	423e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000	
4	523e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000	
5	623e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000	
6	723e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000	
7	823e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000	
8	923e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174001	
9	923e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174001	
10	023e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000	

3. Джойнить данные из Kafka с данными из БД.

4. Сохранять в памяти полученные данные, чтобы не собирать их заново после отправки в Postgres или Kafka

5. Отправлять выходное сообщение в Kafka

с информацией об акции, пользователе со списком избранного и ресторане, а ещё вставлять записи в `Postgres`, чтобы впоследствии получить фидбэк от пользователя. Сервис `push`-уведомлений будет читать сообщения из `Kafka` и формировать готовые уведомления. Получать и анализировать фидбэк в вашу задачу не входит — этим займутся аналитики.

Создайте в `Docker` таблицу — данные будут нужно положить в неё. Параметры подключения к `PostgreSQL` в `Docker`:

- адрес подключения — `localhost:5432`
- логин/пароль — `jovyan/jovyan`

Именно по такому примеру вы должны заполнить её для завершения задачи.

```
-- Выходная таблица
-- DROP TABLE public.subscribers_feedback;

CREATE TABLE public.subscribers_feedback (
    id                     serial4 NOT NULL,
    restaurant_id          text NOT NULL,
    adv_campaign_id         text NOT NULL,
    adv_campaign_content    text NOT NULL,
    adv_campaign_owner      text NOT NULL,
    adv_campaign_owner_contact text NOT NULL,
    adv_campaign_datetime_start int8 NOT NULL,
    adv_campaign_datetime_end int8 NOT NULL,
    datetime_created        int8 NOT NULL,
    client_id               text NOT NULL,
    trigger_datetime_created int4 NOT NULL,
    feedback                varchar NULL,
    CONSTRAINT id_pk PRIMARY KEY (id)
);
```

Пример заполненных данных

	id	restaurant_id	adv_campaign_id	adv_campaign_content	adv_campaign_owner	adv_campaign_owner_contact	adv_campaign_datetime_start	a
1	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174003		first campaign	Ivanov Ivan Ivanovich	iiivanov@restaurant.ru	1659203516	2
2	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174003		first campaign	Ivanov Ivan Ivanovich	iiivanov@restaurant.ru	1659203516	2
3	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174003		first campaign	Ivanov Ivan Ivanovich	iiivanov@restaurant.ru	1659203516	2
4	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174003		first campaign	Ivanov Ivan Ivanovich	iiivanov@restaurant.ru	1659203516	2
5	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174003		first campaign	Ivanov Ivan Ivanovich	iiivanov@restaurant.ru	1659203516	2
6	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174003		first campaign	Ivanov Ivan Ivanovich	iiivanov@restaurant.ru	1659203516	2
7	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174003		first campaign	Ivanov Ivan Ivanovich	iiivanov@restaurant.ru	1659203516	2
8	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174003		first campaign	Ivanov Ivan Ivanovich	iiivanov@restaurant.ru	1659203516	2

В результате вы заполните выходное сообщение. От входного оно отличается только полями.

```
{"restaurant_id": "123e4567-e89b-12d3-a456-426614174000", "adv_campaign_id": "123e4567-e89b-12d3-a456-426614174003", "adv_campaign_content": "first campaign", "adv_campaign_owner": "Ivanov Ivan Ivanovich", "adv_campaign_owner_contact": "iiivanov@restaurant.ru", "adv_campaign_datetime_start": 1659203516, "adv_campaign_datetime_end": 426614174000, "datetime_created": 1659131516, "trigger_datetime_created": 1659304828}
```

В наглядном виде:

```
{
  "restaurant_id": "123e4567-e89b-12d3-a456-426614174000",
  "adv_campaign_id": "123e4567-e89b-12d3-a456-426614174003",
  "adv_campaign_content": "first campaign", "adv_campaign_owner": "Ivanov Ivan Ivanovich",
  "adv_campaign_owner_contact": "iiivanov@restaurant.ru",
  "adv_campaign_datetime_start": 1659203516,
  "adv_campaign_datetime_end": 2659207116,
  "client_id": "023e4567-e89b-12d3-a456-426614174000",
  "datetime_created": 1659131516,
```

```
"trigger_datetime_created":1659304828
```

```
}
```

Добавляются два новых поля:

- "client_id": "023e4567-e89b-12d3-a456-426614174000" — UUID подписчика ресторана, который достаётся из таблицы Postgres .
- "trigger_datetime_created":1659304828 — время создания триггера, то есть выходного сообщения. Оно добавляется во время создания сообщения.

Воспользуйтесь кодом ниже, чтобы приступить к выполнению проекта.

Код

```
import os

from datetime import datetime
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, to_json, col, lit, struct
from pyspark.sql.types import StructType, StructField, StringType, LongType

# метод для записи данных в 2 target: в PostgreSQL для фидбеков и в Kafka для триггеров
def foreach_batch_function(df, epoch_id):
    # сохраняем df в памяти, чтобы не создавать df заново перед отправкой в Kafka
    ...
    # записываем df в PostgreSQL с полем feedback
    ...
    # создаем df для отправки в Kafka. СерIALIZАЦИЯ в json.
    ...
    # отправляем сообщения в результирующий топик Kafka без поля feedback
    ...
    # очищаем память от df
    ...

# необходимые библиотеки для интеграции Spark с Kafka и PostgreSQL
spark_jars_packages = ",".join(
    [
        "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0",
        "org.postgresql:postgresql:42.4.0",
    ]
)

# создаем spark сессию с необходимыми библиотеками в spark_jars_packages для интеграции с Kafka и PostgreSQL
spark = SparkSession.builder \
    .appName("RestaurantSubscribeStreamingService") \
    .config("spark.sql.session.timeZone", "UTC") \
    .config("spark.jars.packages", spark_jars_packages) \
    .getOrCreate()

# читаем из топика Kafka сообщения с акциями от ресторанов
restaurant_read_stream_df = spark.readStream \
    .format('kafka') \
    .option('kafka.bootstrap.servers', 'rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091') \
    .option('kafka.security.protocol', 'SASL_SSL') \
    .option('kafka.sasl.jaas.config', 'org.apache.kafka.common.security.scram.ScramLoginModule required\nusername="login" password="password";') \
    .option('kafka.sasl.mechanism', 'SCRAM-SHA-512') \
    .option('kafka.ssl.truststore.location', '/usr/lib/jvm/java-1.17.0-openjdk-amd64/lib/security/cacerts') \
    .option('kafka.ssl.truststore.password', 'changeit') \
    .option('subscribe', 'your_topic') \
    .load()

# определяем схему входного сообщения для json
incomming_message_schema = ...

# определяем текущее время в UTC в миллисекундах
current_timestamp_utc = int(round(datetime.utcnow().timestamp()))

# десериализуем из value сообщения json и фильтруем по времени старта и окончания акции
filtered_read_stream_df = ....

# вычитываем всех пользователей с подпиской на ресторана
subscribers_restaurant_df = spark.read \
    .format('jdbc') \
    .option('url', 'jdbc:postgresql://rc1a-fswjkpli01zafgjm.mdb.yandexcloud.net:6432/de')
```

```

        .option('driver', 'org.postgresql.Driver') \
        .option('dbtable', 'subscribers_restaurants') \
        .option('user', 'student') \
        .option('password', 'de-student') \
        .load()

# джойним данные из сообщения Kafka с пользователями подписки по restaurant_id (uuid). Добавляем время
# создания события.
result_df = ...
# запускаем стриминг
result_df.writeStream \
    .foreachBatch(foreach_batch_function) \
    .start() \
    .awaitTermination()

```

Инструкция по выполнению проекта

Подготовка

Шаг 1. Проверить работу потока.

Убедитесь, что получается отправлять и читать данные из топика `Kafka`. Это нужно для отладки стримингового приложения.

1. Для этого с помощью `kcat` в одном окне терминала начните читать входной топик. В качестве имени входного топика используйте `ваш логин_in`. Для выходного топика — `ваш логин_out`. Если топиков на данный момент нет, они автоматически создадутся при отправке сообщения.
2. С помощью `kcat` отправьте сообщение в топик, из которого вы читаете сообщения. Напишите входное сообщение в ваш входной топик:
3. Проверьте, что в терминале, где был запущен консьюмер, отображается отправленное сообщение. Если это не так, то проверьте предыдущие шаги. Точно так же вы будете тестировать код следующих шагов.
4. Попробуйте написать код самостоятельно или воспользуйтесь готовым кодом.

Код

```

kafkacat -b rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091 \
-X security.protocol=SASL_SSL \
-X sasl.mechanisms=SCRAM-SHA-512 \
-X sasl.username="ваш_username" \
-X sasl.password="ваш_пароль" \
-X ssl.ca.location="путь до сертификата" \
-t base \
-K: \
-P
key:{"restaurant_id": "123e4567-e89b-12d3-a456-426614174000", "adv_campaign_id": "123e4567-e89b-12d3-a456-426614174003", "adv_campaign_content": "first campaign", "adv_campaign_owner": "Ivanov Ivan Ivanovich", "adv_campaign_owner_contact": "iiivanov@restaurant_id", "adv_campaign_datetime_start": 1659203516, "adv_campaign_datetime_end": 2659207116, "datetime_created": 1659131516}
#Весму ctrl+D, чтобы отправить сообщение

```

Реализация приложения

Шаг 2. Прочитать данные об акциях из `Kafka`

1. Напишите с помощью `PySpark` код стриминга для:
 - а. чтения сообщений об акциях из `Kafka`
 - б. вывода сообщений в консоль.
1. Протестируйте написанный код.

Шаг 3. Прочитать данные о подписчиках из `Postgres`

1. Дополните код чтением статичных данных из таблицы `Postgres` `subscribers_restaurants` и выводом содержимого на консоль.
2. Протестируйте написанный код.

Шаг 4. Преобразование `JSON` в датафрейм.

Сообщения из `Kafka` находятся в формате переменных «ключ-значение» (англ. "key-value"). В переменной `value` лежит содержимое `JSON`-файла, то есть текст в формате `JSON`. Вам же для работы нужен датафрейм, который в качестве колонок использует ключи `JSON`, поэтому придётся сначала десериализовать `value` в `JSON`, а затем преобразовать `JSON` в датафрейм.

1. Дополните код, чтобы он преобразовал `JSON` из `value` в датафрейм, где колонками будут ключи в формате `JSON`.
2. Отфильтруйте сообщения с рекламными кампаниями, для которых текущее время находится в промежутке между временем старта и временем окончания кампании.
3. Протестируйте написанный код.

Шаг 5. Провести `JOIN` потоковых и статичных данных.

1. Дополните код, чтобы сдженить данные из `Kafka` с данными из `Postgres` по полю `restaurant_id`.
2. Добавьте колонку с датой создания — текущей датой. В датафрейме не должно быть повторяющихся колонок.
3. Протестируйте написанный код.

Шаг 6. Отправить результаты `JOIN` в `Postgres` для аналитики фидбэка.

Вам нужно отправлять результаты обработки сразу в два стока — `Kafka` для `push`-уведомлений и `Postgres` для аналитики фидбэка. В обоих случаях вам потребуется метод `foreachBatch(...)`. С его помощью вы можете указать, какие функции нужно применить к каждому микробатчу выходных данных.

Начнём с `Postgres`. Дополните код функцией, которая будет отправлять данные в `Postgres` для получения фидбэка. Для этого:

1. Измените код в теле функции, чтобы записать `df` в `Postgres` с полем `feedback`. Данные нужно записывать в локальную БД, для которой вы создавали таблицу `subscribers_feedback`. Параметры подключения:
 - адрес подключения — `localhost:5432`
 - логин/пароль — `jovyan/jovyan`
1. Доработайте запуск стриминга методом `foreachBatch(...)`, который будет вызывать функцию.
2. Протестируйте написанный код.

Шаг 7. Отправить данные, сериализованные в формат `JSON`, в `Kafka` для `push`-уведомлений.

Дополните функцию из прошлого пункта, чтобы добавить ещё один сток — `Kafka`

1. Создайте датафрейм для отправки в `Kafka`.
2. Сериализуйте данные в сообщение формата «ключ-значение» и заполните только `value`. Для этого сериализуйте данные из датафрейма в `JSON` и положите `JSON` в колонку `value`.
3. Отправьте сообщения в результирующий топик `Kafka` без поля `feedback`.
4. В вызове `foreachBatch(...)` ничего менять не нужно.
5. Протестируйте написанный код.

Шаг 8. Персистентность датафрейма.

1. Сохраните датафрейм в памяти после объединения данных.
2. После отправки данных в два стока очистите память от датафрейма.
3. Протестируйте написанный код.

1. Проверить работу потока. ▲

Ставим один терминал в режим чтения и ждем

```
kafkacat -b rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091 \
-X security.protocol=SASL_SSL \
-X sasl.mechanisms=SCRAM-SHA-512 \
-X sasl.username="de-student" \
-X sasl.password="ltcneltyn" \
-X ssl.ca.location=/usr/local/share/ca-certificates/Yandex/YandexCA.crt \
-t mustdayker_in \
-C \
-o end
```

Запускаем на втором терминале передачу сообщения

```
kafkacat -b rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091 \
-X security.protocol=SASL_SSL \
-X sasl.mechanisms=SCRAM-SHA-512 \
```

```

-X sasl.username="de-student" \
-X sasl.password="ltcneltyn" \
-X ssl.ca.location=/usr/local/share/ca-certificates/Yandex/YandexCA.crt \
-t mustdayker_in \
-K: \
-P

key:{"restaurant_id": "123e4567-e89b-12d3-a456-426614174000", "adv_campaign_id": "123e4567-e89b-12d3-a456-426614174003", "adv_campaign_content": "first campaign", "adv_campaign_owner": "Ivanov Ivan Ivanovich", "adv_campaign_owner_contact": "iiivanov@restaurant_id", "adv_campaign_datetime_start": 1659203516, "adv_campaign_datetime_end": 2659207116, "datetime_created": 1659131516}

```

Еще можно передать так:

```

echo 'key:{"restaurant_id": "123e4567-e89b-12d3-a456-426614174000", "adv_campaign_id": "123e4567-e89b-12d3-a456-426614174003", "adv_campaign_content": "first campaign", "adv_campaign_owner": "Ivanov Ivan Ivanovich", "adv_campaign_owner_contact": "iiivanov@restaurant_id", "adv_campaign_datetime_start": 1659203516, "adv_campaign_datetime_end": 2659207116, "datetime_created": 1659131516}' | kafkacat -b rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091 \
-X security.protocol=SASL_SSL \
-X sasl.mechanisms=SCRAM-SHA-512 \
-X sasl.username="de-student" \
-X sasl.password="ltcneltyn" \
-X ssl.ca.location=/usr/local/share/ca-certificates/Yandex/YandexCA.crt \
-t mustdayker_in \
-P \
-K:

```

Данные для подключения к базе с маркетинговыми данными

- host: rc1a-fswjkpli01zafgjm.mdb.yandexcloud.net
- user: student
- password: de-student
- Порт (port): 6432 (используется по умолчанию для PostgreSQL)
- Использовать SSL - Use Secure Connection
- Название базы данных (database): de

```

SELECT
    table_schema,
    table_name,
    column_name,
    data_type,
    is_nullable
FROM information_schema.columns
where table_schema = 'public'
order by table_name

```

id	table_schema	table_name	column_name	data_type	is_nullable
1	public	marketing_companies	description	text	YES
2	public	marketing_companies	name	text	NO
3	public	marketing_companies	start_time	time without time zone	NO
4	public	marketing_companies	radius	smallint	NO
5	public	marketing_companies	end_time	time without time zone	NO
6	public	marketing_companies	point_lon	double precision	NO
7	public	marketing_companies	point_lat	double precision	NO
8	public	marketing_companies	id	uuid	NO
80	public	subscribers_restaurants	id	integer	NO
81	public	subscribers_restaurants	restaurant_id	character varying	YES
82	public	subscribers_restaurants	client_id	character varying	YES
95	public	words	id	integer	NO
96	public	words	words	text	NO

Данные для подключения к локальной базе в Docker

- host: localhost
- user: jovyan
- password: jovyan
- Порт (port): 5432
- Использовать SSL - Standart Connection

- Название базы данных (database): de

2. Прочитать данные об акциях из Kafka ▲

_____ БЛОК ИМПОРТОВ _____

```
import os

from datetime import datetime
from time import sleep
from pyspark.sql import SparkSession
from pyspark.sql import functions as f
from pyspark.sql.functions import from_json, to_json, col, lit, struct
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType, IntegerType, LongType
```

_____ БЛОК ПЕРЕМЕННЫХ _____

```
# необходимые библиотеки для интеграции Spark с Kafka и PostgreSQL
spark_jars_packages = ",".join(
    [
        "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0",
        "org.postgresql:postgresql:42.4.0",
    ]
)
```

```
# truststore_location = "/etc/security/ssl"
# truststore_location = "/usr/lib/jvm/java-1.17.0-openjdk-amd64/lib/security/cacerts"
# truststore_pass = "de_sprint_8"
```

```
topic_in = 'mustdayker_in'
topic_out = 'mustdayker_out'

kafka_security_options = {
    'kafka.bootstrap.servers': 'rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091',
    'kafka.security.protocol': 'SASL_SSL',
    'kafka.sasl.mechanism': 'SCRAM-SHA-512',
    'kafka.sasl.jaas.config': 'org.apache.kafka.common.security.scram.ScramLoginModule required
username=\\"de-student\\" password=\\"ltcnelytyn\\";',
    'subscribe': topic_in,
#    'kafka.ssl.truststore.location': truststore_location,
#    'kafka.ssl.truststore.password': truststore_pass,
}
```

_____ СТАРТУЕМ СПАРК СЕССИЮ _____

```
# создаём spark сессию с необходимыми библиотеками в spark_jars_packages для интеграции с Kafka и PostgreSQL
spark = SparkSession.builder \
    .appName("RestaurantSubscribeStreamingService") \
    .config("spark.sql.session.timeZone", "UTC") \
    .config("spark.jars.packages", spark_jars_packages) \
    .getOrCreate()
```

_____ ЧИТАЕМ ПОТОК _____

```
# читаем из топика Kafka сообщения с акциями от ресторанов
restaurant_read_stream_df = spark.readStream \
    .format('kafka') \
    .options(**kafka_security_options) \
    .load()
```

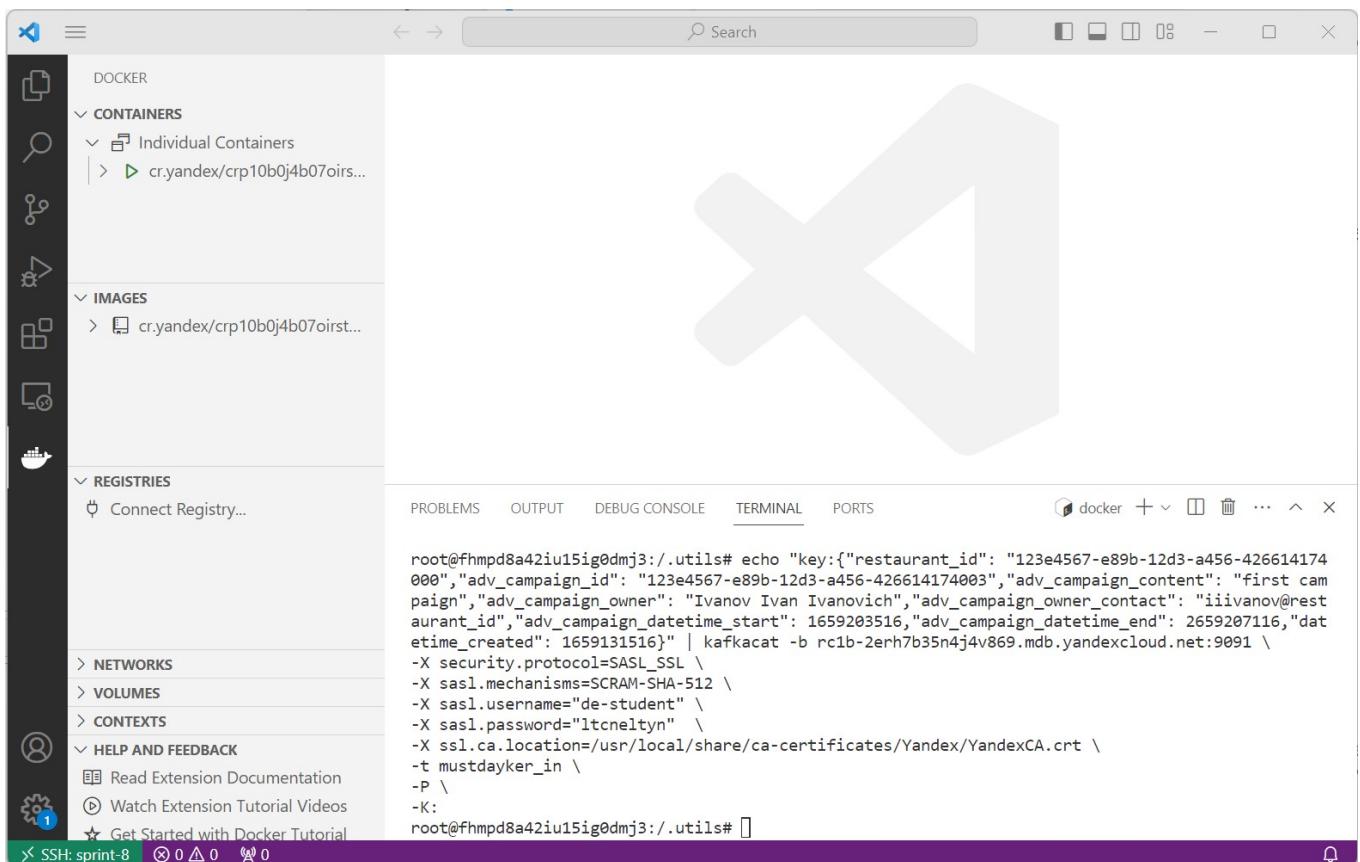
_____ ВЫВОДИМ ПОТОК В КОНСОЛЬ _____

```
query = (
    df
    .writeStream
    .outputMode("append")
    .format("console")
    .start()
```

```
)
```

```
query.awaitTermination()
```

Отправляем сообщение через kafkacat



Получаем:

```
-----
Batch: 1
-----
+-----+-----+-----+-----+-----+
| key|      value|      topic|partition|offset|      timestamp|timestampType|
+-----+-----+-----+-----+-----+
|[6B]||[D0 90 D0 91 D0 A...|mustdayker_in|       0| 37|2023-10-11 08:19:...|       0|
+-----+-----+-----+-----+-----+
```

3. Прочитать данные о подписчиках из Postgres ▲

```
# _____ БЛОК ИМПОРТОВ _____
```

```
import os

from datetime import datetime
from time import sleep
from pyspark.sql import SparkSession
from pyspark.sql import functions as f
from pyspark.sql.functions import from_json, to_json, col, lit, struct
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType, IntegerType, LongType
```

```
# _____ БЛОК ПЕРЕМЕННЫХ _____
```

```
# необходимые библиотеки для интеграции Spark с Kafka и PostgreSQL
spark_jars_packages = ",".join(
    [
        "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0",
        "org.postgresql:postgresql:42.4.0",
    ]
)
```

```

# truststore_location = "/etc/security/ssl"
# truststore_location = "/usr/lib/jvm/java-1.17.0-openjdk-amd64/lib/security/cacerts"
# truststore_pass = "de_sprint_8"

topic_in = 'mustdayker_in'
topic_out = 'mustdayker_out'

kafka_security_options = {
    'kafka.bootstrap.servers': 'rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091',
    'kafka.security.protocol': 'SASL_SSL',
    'kafka.sasl.mechanism': 'SCRAM-SHA-512',
    'kafka.sasl.jaas.config': 'org.apache.kafka.common.security.scram.ScramLoginModule required
username=\\"de-student\\" password=\\"ltcneltyn\\";',
    'subscribe': topic_in,
#     'kafka.ssl.truststore.location': truststore_location,
#     'kafka.ssl.truststore.password': truststore_pass,
}

```

_____ СТАРТУЕМ СПАРК СЕССИЮ _____

```

# создаём spark сессию с необходимыми библиотеками в spark_jars_packages для интеграции с Kafka и PostgreSQL
spark = SparkSession.builder \
    .appName("RestaurantSubscribeStreamingService") \
    .config("spark.sql.session.timeZone", "UTC") \
    .config("spark.jars.packages", spark_jars_packages) \
    .getOrCreate()

```

_____ ЧИТАЕМ ДАТАФРЕЙМ _____

```

# Вычитываем всех пользователей с подпиской на рестораны
subscribers_restaurant_df = spark.read \
    .format('jdbc') \
    .option('url', 'jdbc:postgresql://rc1a-fswjkpli01zafgjm.mdb.yandexcloud.net:6432/de') \
    .option('driver', 'org.postgresql.Driver') \
    .option('dbtable', 'subscribers_restaurants') \
    .option('user', 'student') \
    .option('password', 'de-student') \
    .load()

```

_____ ВЫВОДИМ ДАТАФРЕЙМ В КОНСОЛЬ _____

```
subscribers_restaurant_df.show()
```

Получаем:

id client_id restaurant_id
1 223e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000
2 323e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000
3 423e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000
4 523e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000
5 623e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000
6 723e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000
7 823e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000
8 923e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174001
9 923e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174001
10 023e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000

4. Преобразование JSON в датафрейм. ▲

_____ БЛОК ИМПОРТОВ _____

```

import os

from datetime import datetime
from time import sleep
from pyspark.sql import SparkSession

```

```

from pyspark.sql import functions as f
from pyspark.sql.functions import from_json, to_json, col, lit, struct
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType, IntegerType, LongType

# _____ БЛОК ПЕРЕМЕННЫХ _____

# необходимые библиотеки для интеграции Spark с Kafka и PostgreSQL
spark_jars_packages = ",".join(
    [
        "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0",
        "org.postgresql:postgresql:42.4.0",
    ]
)

# truststore_location = "/etc/security/ssl"
# truststore_location = "/usr/lib/jvm/java-1.17.0-openjdk-amd64/lib/security/cacerts"
# truststore_pass = "de_sprint_8"

topic_in = 'mustdayker_in'
topic_out = 'mustdayker_out'

kafka_security_options = {
    'kafka.bootstrap.servers': 'rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091',
    'kafka.security.protocol': 'SASL_SSL',
    'kafka.sasl.mechanism': 'SCRAM-SHA-512',
    'kafka.sasl.jaas.config': 'org.apache.kafka.common.security.scram.ScramLoginModule required\nusername=\"de-student\" password=\"ltcneltyn\";',
    'subscribe': topic_in,
}
# 'kafka.ssl.truststore.Location': truststore_location,
# 'kafka.ssl.truststore.password': truststore_pass,
}

# _____ СТАРТУЕМ СПАРК СЕССИЮ _____

# создаём spark сессию с необходимыми библиотеками в spark_jars_packages для интеграции с Kafka и PostgreSQL
spark = SparkSession.builder \
    .appName("RestaurantSubscribeStreamingService") \
    .config("spark.sql.session.timeZone", "UTC") \
    .config("spark.jars.packages", spark_jars_packages) \
    .getOrCreate()

# _____ ЧИТАЕМ ПОТОК _____

# читаем из топика Kafka сообщения с акциями от ресторанов
restaurant_read_stream_df = spark.readStream \
    .format('kafka') \
    .options(**kafka_security_options) \
    .load()

# _____ ПРЕОБРАЗУЕМ ПОТОК _____

# определяем схему входного сообщения для json
incomming_message_schema = StructType([
    StructField("restaurant_id", StringType(), True),
    StructField("adv_campaign_id", StringType(), True),
    StructField("adv_campaign_content", StringType(), True),
    StructField("adv_campaign_owner", StringType(), True),
    StructField("adv_campaign_owner_contact", StringType(), True),
    StructField("adv_campaign_datetime_start", LongType(), True),
    StructField("adv_campaign_datetime_end", LongType(), True),
    StructField("datetime_created", LongType(), True)
])

# десериализуем из value сообщения json и фильтруем по времени старта и окончания акции
filtered_read_stream_df = (restaurant_read_stream_df
    .withColumn('value', f.col('value').cast(StringType()))
    .withColumn('key', f.col('key').cast(StringType()))
    .withColumn('event', f.from_json(f.col('value'), incomming_message_schema))
    .selectExpr('event.*', '*').drop('event')
    .withColumn(

```

```

'adv_campaign_datetime_start',
f.from_unixtime(f.col('adv_campaign_datetime_start'), "yyyy-MM-dd"
'HH:mm:ss.SSS").cast(TimestampType()))
.withColumn(
    'adv_campaign_datetime_end',
f.from_unixtime(f.col('adv_campaign_datetime_end'), "yyyy-MM-dd"
'HH:mm:ss.SSS").cast(TimestampType())))
.withColumn(
    'datetime_created',
f.from_unixtime(f.col('datetime_created'), "yyyy-MM-dd' 'HH:mm:ss.SSS").cast(TimestampType())))
.filter(f.col("timestamp").between(
    f.col("adv_campaign_datetime_start"),
    f.col("adv_campaign_datetime_end")))
)
)

# filtered_read_stream_df = df \
#     .select(from_json(col("value").cast("string"), incomming_message_schema).alias("parsed_key_value")) \
#     .select(col("parsed_key_value.*"))

# _____ ВЫВОДИМ ПОТОК В КОНСОЛЬ _____

query = (
    filtered_read_stream_df
    .writeStream
    .outputMode("append")
    .format("console")
    .option("truncate", False)
    .start()
)

query.awaitTermination()

```

Результат:

```

-----
Batch: 1
-----

restaurant_id adv_campaign_id adv_campaign_content adv_campaign_owner adv_campaign_owner_contact adv_campaign_datetime_start adv_c
-----
123e4567-e89b-12d3-a456-426614174000 123e4567-e89b-12d3-a456-426614174000 first campaign Ivanov Ivan Ivanovich iiivanov@restaurant_id 2022-07-30 17:51:56 2054-

```

5. Провести JOIN ПОТОКОВЫХ И СТАТИЧНЫХ ДАННЫХ ▲

```

# _____ БЛОК ИМПОРТОВ _____

import os

from datetime import datetime
from time import sleep
from pyspark.sql import SparkSession
from pyspark.sql import functions as f
from pyspark.sql.functions import from_json, to_json, col, lit, struct
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType, IntegerType, LongType

# _____ БЛОК ПЕРЕМЕННЫХ _____

# необходимые библиотеки для интеграции Spark с Kafka и PostgreSQL
spark_jars_packages = ",".join([
    "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0",
    "com.github.sbt:sbt-dependency-graph:1.0.0"
])

```

```

        "org.postgresql:postgresql:42.4.0",
    ]
}

# truststore_location = "/etc/security/ssl"
# truststore_location = "/usr/lib/jvm/java-1.17.0-openjdk-amd64/lib/security/cacerts"
# truststore_pass = "de_sprint_8"

topic_in = 'mustdayker_in'
topic_out = 'mustdayker_out'

kafka_security_options = {
    'kafka.bootstrap.servers': 'rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091',
    'kafka.security.protocol': 'SASL_SSL',
    'kafka.sasl.mechanism': 'SCRAM-SHA-512',
    'kafka.sasl.jaas.config': 'org.apache.kafka.common.security.scram.ScramLoginModule required\nusername=\"de-student\" password=\"ltcneltyn\";',
    'subscribe': topic_in,
#    'kafka.ssl.truststore.Location': truststore_location,
#    'kafka.ssl.truststore.password': truststore_pass,
}

# _____ СТАРТУЕМ СПАРК СЕССИЮ _____

# создаём spark сессию с необходимыми библиотеками в spark_jars_packages для интеграции с Kafka и PostgreSQL
spark = SparkSession.builder \
    .appName("RestaurantSubscribeStreamingService") \
    .config("spark.sql.session.timeZone", "UTC") \
    .config("spark.jars.packages", spark_jars_packages) \
    .getOrCreate()

# _____ ЧИТАЕМ ПОТОК _____

# читаем из топика Kafka сообщения с акциями от ресторанов
restaurant_read_stream_df = spark.readStream \
    .format('kafka') \
    .options(**kafka_security_options) \
    .load()

# _____ ПРЕОБРАЗУЕМ ПОТОК _____

# определяем схему входного сообщения для json
incomming_message_schema = StructType([
    StructField("restaurant_id", StringType(), True),
    StructField("adv_campaign_id", StringType(), True),
    StructField("adv_campaign_content", StringType(), True),
    StructField("adv_campaign_owner", StringType(), True),
    StructField("adv_campaign_owner_contact", StringType(), True),
    StructField("adv_campaign_datetime_start", LongType(), True),
    StructField("adv_campaign_datetime_end", LongType(), True),
    StructField("datetime_created", LongType(), True)
])

# десериализуем из value сообщения json и фильтруем по времени старта и окончания акции
filtered_read_stream_df = (restaurant_read_stream_df
    .withColumn('value', f.col('value').cast(StringType()))
    .withColumn('key', f.col('key').cast(StringType()))
    .withColumn('event', f.from_json(f.col('value'), incomming_message_schema))
    .selectExpr('event.*', '*').drop('event')
    .withColumn(
        'adv_campaign_datetime_start',
        f.from_unixtime(f.col('adv_campaign_datetime_start'), "yyyy-MM-dd'\n'HH:mm:ss.SSS").cast(TimestampType()))
    .withColumn(
        'adv_campaign_datetime_end',
        f.from_unixtime(f.col('adv_campaign_datetime_end'), "yyyy-MM-dd'\n'HH:mm:ss.SSS").cast(TimestampType()))
    .withColumn(
        'datetime_created',
        f.from_unixtime(f.col('datetime_created'), "yyyy-MM-dd' 'HH:mm:ss.SSS").cast(TimestampType()))
    .filter(f.col("timestamp").between(

```

```

        f.col("adv_campaign_datetime_start"),
        f.col("adv_campaign_datetime_end"))
    )
    .select(
        'restaurant_id',
        'adv_campaign_id',
        'adv_campaign_content',
        'adv_campaign_owner',
        'adv_campaign_owner_contact',
        'adv_campaign_datetime_start',
        'adv_campaign_datetime_end',
        'datetime_created'
    )
)
)

# filtered_read_stream_df = df \
#     .select(from_json(col("value").cast("string"), incomming_message_schema).alias("parsed_key_value")) \
#     .select(col("parsed_key_value.*"))


```

_____ ЧИТАЕМ СТАТИЧНЫЕ ДАННЫЕ ИЗ БД _____

```

# Вычитываем всех пользователей с подпиской на рестораны
subscribers_restaurant_df = spark.read \
    .format('jdbc') \
    .option('url', 'jdbc:postgresql://rc1a-fswjkpli01zafgjm.mdb.yandexcloud.net:6432/de') \
    .option('driver', 'org.postgresql.Driver') \
    .option('dbtable', 'subscribers_restaurants') \
    .option('user', 'student') \
    .option('password', 'de-student') \
    .load() \
    .dropDuplicates(['restaurant_id', 'client_id'])


```

_____ ДЖОЙНИМ ПОТОК С БД _____

```

# джойним данные из сообщения Kafka с пользователями подписки по restaurant_id (uuid). Добавляем время
создания события.
result_df = filtered_read_stream_df \
    .join(
        subscribers_restaurant_df.select('restaurant_id', 'client_id'),
        'restaurant_id',
        'inner'
    ) \
    .withColumn('trigger_datetime_created', f.lit(datetime.now())) \
    .select(
        'restaurant_id',
        'adv_campaign_id',
        'adv_campaign_content',
        'adv_campaign_owner',
        'adv_campaign_owner_contact',
        'adv_campaign_datetime_start',
        'adv_campaign_datetime_end',
        'client_id',
        'datetime_created',
        'trigger_datetime_created'
    ) \
    .dropDuplicates(['restaurant_id', 'client_id', 'adv_campaign_id'])


```

_____ ВЫВОДИМ ПОТОК В КОНСОЛЬ _____

```

# Выводим поток в консоль
query = (
    result_df
    .writeStream
    .outputMode("append")
    .format("console")
    .option("truncate", False)
    .start()
)
query.awaitTermination()


```

6. Отправить результаты JOIN в Postgres для аналитики фидбэка ▲

```
# _____ БЛОК ИМПОРТОВ _____  
  
import os  
  
from datetime import datetime  
from time import sleep  
from pyspark.sql import SparkSession  
from pyspark.sql import functions as f  
from pyspark.sql.functions import from_json, to_json, col, lit, struct  
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType, IntegerType, LongType  
import psycopg2  
  
# _____ БЛОК ПЕРЕМЕННЫХ _____  
  
# необходимые библиотеки для интеграции Spark с Kafka и PostgreSQL  
spark_jars_packages = ",".join(  
    [  
        "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0",  
        "org.postgresql:postgresql:42.4.0",  
    ]  
)  
  
# truststore_location = "/etc/security/ssl"  
# truststore_location = "/usr/lib/jvm/java-1.17.0-openjdk-amd64/lib/security/cacerts"  
# truststore_pass = "de_sprint_8"  
  
topic_in = 'mustdayker_in'  
topic_out = 'mustdayker_out'  
  
kafka_security_options = {  
    'kafka.bootstrap.servers': 'rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091',  
    'ssl.ca.location': '/etc/ssl/certs/ca-certificates.crt',  
    'ssl.truststore.location': '/etc/ssl/certs/cacerts.jks',  
    'ssl.truststore.password': 'de_sprint_8',  
    'ssl.key.password': 'de_sprint_8',  
    'ssl.keystore.location': '/etc/ssl/certs/cacerts.jks',  
    'ssl.keystore.password': 'de_sprint_8',  
    'ssl.enabled.protocols': 'TLSv1.3',  
    'ssl.enabled.cipher.suites': 'TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256',  
    'ssl.tls.insecure.ignore.valid.certificate.error': True,  
}
```

```

'kafka.security.protocol': 'SASL_SSL',
'kafka.sasl.mechanism': 'SCRAM-SHA-512',
'kafka.sasl.jaas.config': 'org.apache.kafka.common.security.scram.ScramLoginModule required
username=\\"de-student\\" password=\\"ltcnelytn\\";',
'subscribe': topic_in,
#   'kafka.ssl.truststore.location': truststore_location,
#   'kafka.ssl.truststore.password': truststore_pass,
}

# _____ СТАРТУЕМ СПАРК СЕССИЮ _____

# создаём spark сессию с необходимыми библиотеками в spark_jars_packages для интеграции с Kafka и PostgreSQL
spark = SparkSession.builder \
    .appName("RestaurantSubscribeStreamingService") \
    .config("spark.sql.session.timeZone", "UTC") \
    .config("spark.jars.packages", spark_jars_packages) \
    .getOrCreate()

# _____ СОЗДАЕМ ТАБЛИЦУ POSTGRES _____

URL = """
host=localhost
port=5432
dbname=de
user=jovyan
password=jovyan
target_session_attrs=read-write
"""

DDL = """
CREATE TABLE IF NOT EXISTS public.subscribers_feedback (
    id serial4 NOT NULL,
    restaurant_id text NOT NULL,
    adv_campaign_id text NOT NULL,
    adv_campaign_content text NOT NULL,
    adv_campaign_owner text NOT NULL,
    adv_campaign_owner_contact text NOT NULL,
    adv_campaign_datetime_start int8 NOT NULL,
    adv_campaign_datetime_end int8 NOT NULL,
    datetime_created int8 NOT NULL,
    client_id text NOT NULL,
    trigger_datetime_created int4 NOT NULL,
    feedback varchar NULL,
    CONSTRAINT id_pk PRIMARY KEY (id)
);
"""

CHECK_CREATION = """
SELECT EXISTS (
    SELECT 1
    FROM information_schema.tables
    WHERE table_schema = 'public'
    AND table_name = 'subscribers_feedback'
);
"""

# создаем таблицу в postgres внутри docker контейнера
conn = psycopg2.connect(URL)
try:
    with conn:
        with conn.cursor() as cur:
            cur.execute(DDL)
finally:
    conn.close()

# проверяем создание таблицы
conn = psycopg2.connect(URL)
try:
    with conn:
        with conn.cursor() as cur:
            cur.execute(CHECK_CREATION)
            result = cur.fetchone()
finally:
    conn.close()
print(result)

# _____ ЧИТАЕМ ПОТОК _____

```

```

# читаем из топика Kafka сообщения с акциями от ресторанов
restaurant_read_stream_df = spark.readStream \
    .format('kafka') \
    .options(**kafka_security_options) \
    .load()

# _____ ПРЕОБРАЗУЕМ ПОТОК _____

# определяем схему входного сообщения для json
incomming_message_schema = StructType([
    StructField("restaurant_id", StringType(), True),
    StructField("adv_campaign_id", StringType(), True),
    StructField("adv_campaign_content", StringType(), True),
    StructField("adv_campaign_owner", StringType(), True),
    StructField("adv_campaign_owner_contact", StringType(), True),
    StructField("adv_campaign_datetime_start", LongType(), True),
    StructField("adv_campaign_datetime_end", LongType(), True),
    StructField("datetime_created", LongType(), True)
])

# десериализуем из value сообщения json и фильтруем по времени старта и окончания акции
filtered_read_stream_df = (restaurant_read_stream_df
    .withColumn('value', f.col('value').cast(StringType()))
    .withColumn('key', f.col('key').cast(StringType()))
    .withColumn('event', f.from_json(f.col('value'), incomming_message_schema))
    .selectExpr('event.*', '*').drop('event')
    .withColumn('timestamp', f.unix_timestamp('timestamp'))
    .filter(f.col("timestamp").between(
        f.col("adv_campaign_datetime_start"),
        f.col("adv_campaign_datetime_end")))
    .select(
        'restaurant_id',
        'adv_campaign_id',
        'adv_campaign_content',
        'adv_campaign_owner',
        'adv_campaign_owner_contact',
        'adv_campaign_datetime_start',
        'adv_campaign_datetime_end',
        'datetime_created'
    )
)

# _____ ЧИТАЕМ СТАТИЧНЫЕ ДАННЫЕ ИЗ БД _____

# вычитываем всех пользователей с подпиской на рестораны
subscribers_restaurant_df = spark.read \
    .format('jdbc') \
    .option('url', 'jdbc:postgresql://rc1a-fswjkpli01zafgjm.mdb.yandexcloud.net:6432/de') \
    .option('driver', 'org.postgresql.Driver') \
    .option('dbtable', 'subscribers_restaurants') \
    .option('user', 'student') \
    .option('password', 'de-student') \
    .load() \
    .dropDuplicates(['restaurant_id', 'client_id'])

# _____ ДЖОЙНИМ ПОТОК С БД _____

# джойним данные из сообщения Kafka с пользователями подписки по restaurant_id (uuid). Добавляем время
# создания события.
result_df = filtered_read_stream_df \
    .join(
        subscribers_restaurant_df.select('restaurant_id', 'client_id'),
        'restaurant_id',
        'inner'
    ) \
    .withColumn("trigger_datetime_created", lit(int(round(datetime.utcnow().timestamp())))) \
    .select(
        'restaurant_id',
        'adv_campaign_id',
        'adv_campaign_content',
        'adv_campaign_owner',
        'adv_campaign_owner_contact',
        'adv_campaign_datetime_start',
        'adv_campaign_datetime_end',
        'client_id',
        'datetime_created',
        'trigger_datetime_created'
    )

```

```
'datetime_created',
'trigger_datetime_created'
)\ \
.dropDuplicates(['restaurant_id', 'client_id', 'adv_campaign_id'])
```

ФУНКЦИЯ FOREBATCH

```
# метод для записи данных в 2 target: в PostgreSQL для фидбеков и в Kafka для триггеров
def foreach batch function(df, epoch id):
```

```
# сохраняем df в памяти, чтобы не создавать df заново перед отправкой в Kafka  
df.persist()
```

```
# записываем df в PostgreSQL с полем feedback
df.write \
    .mode("append") \
    .format("jdbc") \
    .option("url", "jdbc:postgresql://localhost:5432/de") \
    .option('driver', 'org.postgresql.Driver') \
    .option("dbtable", "subscribers_feedback") \
    .option("user", "jovyan") \
    .option("password", "jovyan") \
    .save()
```

создаём df для отправки в Kafka. Сериализация в json

11

```
# очищаем память от df  
df.unpersist()
```

ОТПРАВКА ДАННЫХ В СТОК

```
result_df.writeStream \
    .foreachBatch(foreach_batch_function) \
    .start() \
    .awaitTermination()
```

Результат:

7. Отправить данные, сериализованные в формат JSON, в Kafka для push-уведомлений ▲

_____ БЛОК ИМПОРТОВ

```
import os
```

```
from datetime import datetime
from time import sleep
from pyspark.sql import SparkSession
from pyspark.sql import functions as f
from pyspark.sql.functions import from_json, to_json, col, lit, struct
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType, IntegerType, LongType
import psycopg2
```

_____ БЛОК ПЕРЕМЕННЫХ

```
# необходимые библиотеки для интеграции Spark с Kafka и PostgreSQL  
spark_jars_packages = ",".join(
```

```
[  
    "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0",  
    "org.postgresql:postgresql:42.4.0",  
]  
}
```

```
topic_in = 'mustdayker_in' # входной топик  
topic_out = 'mustdayker_out' # выходной топик
```

```
kafka_security_options = {
    'kafka.bootstrap.servers': 'rc1b-2erh7b35n4j4v869.mdb.yandexcloud.net:9091',
    'kafka.security.protocol': 'SASL_SSL',
    'kafka.sasl.mechanism': 'SCRAM-SHA-512',
    'kafka.sasl.jaas.config': 'org.apache.kafka.common.security.scram.ScramLoginModule required
username=\"de-student\" password=\"ltcneltyn\";'
}
```

СТАРТУЕМ СПАРК СЕССИЮ

```
# создаём spark сессию с необходимыми библиотеками в spark_jars_packages для интеграции с Kafka и PostgreSQL
spark = SparkSession.builder.\
```

```
R = SparkSession.builder() \
    .appName("RestaurantSubscribeStreamingService") \
    .config("spark.sql.session.timeZone", "UTC") \
    .config("spark.jars.packages", spark_jars_packages) \
```

```

    .getOrCreate()

# _____ СОЗДАЕМ ТАБЛИЦУ POSTGRES _____

URL = """
host=localhost
port=5432
dbname=de
user=jovyan
password=jovyan
target_session_attrs=read-write
"""

DDL = """
CREATE TABLE IF NOT EXISTS public.subscribers_feedback (
    id serial4 NOT NULL,
    restaurant_id text NOT NULL,
    adv_campaign_id text NOT NULL,
    adv_campaign_content text NOT NULL,
    adv_campaign_owner text NOT NULL,
    adv_campaign_owner_contact text NOT NULL,
    adv_campaign_datetime_start int8 NOT NULL,
    adv_campaign_datetime_end int8 NOT NULL,
    datetime_created int8 NOT NULL,
    client_id text NOT NULL,
    trigger_datetime_created int4 NOT NULL,
    feedback varchar NULL,
    CONSTRAINT id_pk PRIMARY KEY (id)
);
"""

CHECK_CREATION = """
SELECT EXISTS (
    SELECT 1
    FROM   information_schema.tables
    WHERE  table_schema = 'public'
    AND    table_name = 'subscribers_feedback'
);
"""

# создаем таблицу в postgres внутри docker контейнера
conn = psycopg2.connect(URL)
try:
    with conn:
        with conn.cursor() as cur:
            cur.execute(DDL)
finally:
    conn.close()

# проверяем создание таблицы
conn = psycopg2.connect(URL)
try:
    with conn:
        with conn.cursor() as cur:
            cur.execute(CHECK_CREATION)
            result = cur.fetchone()
finally:
    conn.close()
print(result)

# _____ ЧИТАЕМ ПОТОК _____

# читаем из топика Kafka сообщения с акциями от ресторанов
restaurant_read_stream_df = spark.readStream \
    .format('kafka') \
    .options(**kafka_security_options) \
    .option("subscribe", topic_in)\ \
    .load()

# _____ ПРЕОБРАЗУЕМ ПОТОК _____

# определяем схему входного сообщения для json
incomming_message_schema = StructType([

```

```
StructField("restaurant_id", StringType(), True),
StructField("adv_campaign_id", StringType(), True),
StructField("adv_campaign_content", StringType(), True),
StructField("adv_campaign_owner", StringType(), True),
StructField("adv_campaign_owner_contact", StringType(), True),
StructField("adv_campaign_datetime_start", LongType(), True),
StructField("adv_campaign_datetime_end", LongType(), True),
StructField("datetime_created", LongType(), True)
])
```

```
# десериализуем из value сообщения json и фильтруем по времени старта и окончания акции
filtered_read_stream_df = (restaurant_read_stream_df
    .withColumn('value', f.col('value').cast(StringType()))
    .withColumn('key', f.col('key').cast(StringType()))
    .withColumn('event', f.from_json(f.col('value'), incoming_message_schema))
    .selectExpr('event.*', '*').drop('event')
    .withColumn('timestamp', f.unix_timestamp('timestamp'))
    .filter(f.col("timestamp").between(
        f.col("adv_campaign_datetime_start"),
        f.col("adv_campaign_datetime_end"))))
    .select(
        'restaurant_id',
        'adv_campaign_id',
        'adv_campaign_content',
        'adv_campaign_owner',
        'adv_campaign_owner_contact',
        'adv_campaign_datetime_start',
        'adv_campaign_datetime_end',
        'datetime_created'
    )
)
```

```
# _____ ЧИТАЕМ СТАТИЧНЫЕ ДАННЫЕ ИЗ БД _____
```

```
# вычитываем всех пользователей с подпиской на рестораны
subscribers_restaurant_df = spark.read \
    .format('jdbc') \
    .option('url', 'jdbc:postgresql://rc1a-fswjkpli01zafgjm.mdb.yandexcloud.net:6432/de') \
    .option('driver', 'org.postgresql.Driver') \
    .option('dbtable', 'subscribers_restaurants') \
    .option('user', 'student') \
    .option('password', 'de-student') \
    .load() \
    .dropDuplicates(['restaurant_id', 'client_id'])
```

```
# _____ ДЖОЙНИМ ПОТОК С БД _____
```

```
# джойним данные из сообщения Kafka с пользователями подписки по restaurant_id (uuid). Добавляем время
создания события.
result_df = filtered_read_stream_df \
    .join(
    subscribers_restaurant_df.select('restaurant_id', 'client_id'),
    'restaurant_id',
    'inner'
) \
    .withColumn("trigger_datetime_created", lit(int(round(datetime.utcnow().timestamp())))) \
    .select(
        'restaurant_id',
        'adv_campaign_id',
        'adv_campaign_content',
        'adv_campaign_owner',
        'adv_campaign_owner_contact',
        'adv_campaign_datetime_start',
        'adv_campaign_datetime_end',
        'client_id',
        'datetime_created',
        'trigger_datetime_created'
) \
    .dropDuplicates(['restaurant_id', 'client_id', 'adv_campaign_id'])
```

```
# _____ ФУНКЦИЯ FOREBATCH _____
```

```
# метод для записи данных в 2 target: в PostgreSQL для фидбэков и в Kafka для триггеров
```

```

def foreach_batch_function(df, epoch_id):
    # сохраняем df в памяти, чтобы не создавать df заново перед отправкой в Kafka
    df.persist()

    # записываем df в PostgreSQL с полем feedback
    df.write \
        .mode("append") \
        .format("jdbc") \
        .option("url", "jdbc:postgresql://localhost:5432/de") \
        .option('driver', 'org.postgresql.Driver') \
        .option("dbtable", "subscribers_feedback") \
        .option("user", "jovyan") \
        .option("password", "jovyan") \
        .save()

    # создаём df для отправки в Kafka. Сериализация в json.
    kafka_df = result_df.select(f.to_json(f.struct(
        'restaurant_id',
        'adv_campaign_id',
        'adv_campaign_content',
        'adv_campaign_owner',
        'adv_campaign_owner_contact',
        'adv_campaign_datetime_start',
        'adv_campaign_datetime_end',
        'client_id',
        'datetime_created',
        'trigger_datetime_created'
    )).alias('value')))

    # отправляем сообщения в результирующий топик Kafka без поля feedback
    kafka_df\
        .writeStream\
        .outputMode("append")\
        .format("kafka")\
        .options(**kafka_security_options)\.
        .option("topic", topic_out)\.
        .trigger(processingTime="15 seconds")\
        .option("checkpointLocation", "/root/spark_checkpoint")\
        .option("truncate", False)\.
        .start()

    # очищаем память от df
    df.unpersist()

```

_____ ОТПРАВКА ДАННЫХ В СТОК _____

```

result_df.writeStream \
    .foreachBatch(foreach_batch_function) \
    .start() \
    .awaitTermination()

```

_____ Вспомогательный блок _____

```

# # Выбодим поток в консоль
# query = (
#     kafka_df
#     .writeStream
#     .outputMode("append")
#     .format("console")
#     .option("truncate", False)
#     .start()
# )

# query.awaitTermination()

```

```

# kafka_df = result_df.select(f.to_json(f.struct(
#     'restaurant_id',
#     'adv_campaign_id',
#     'adv_campaign_content',
#     'adv_campaign_owner',
#     'adv_campaign_owner_contact',
#     'adv_campaign_datetime_start',
#     'adv_campaign_datetime_end',
#     'client_id',
#     'datetime_created',
#     'trigger_datetime_created'
# )).alias('value'))
#
# query = (kafka_df
#         .writeStream
#         .outputMode("append")
#         .format("kafka")
#         .options(**kafka_security_options)
#         .option("topic", topic_out)
#         .trigger(processingTime="15 seconds")
#         .option("checkpointLocation", "/root/spark_checkpoint")
#         .option("truncate", False)
#         .start())
#
# query.awaitTermination()

```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS docker + × ☰ ... ^ ×

t":1659203516,"adv_campaign_datetime_end":2659207116,"client_id":"723e4567-e89b-12d3-a456-426614174000","datetime_created":165913 1516,"trigger_datetime_created":1697181012}
% Reached end of topic mustdayker_out [0] at offset 77
{"restaurant_id":"123e4567-e89b-12d3-a456-426614174000","adv_campaign_id":"ФИНАЛЬНАЯ ПРОВЕРКА","adv_campaign_content":"first camp
aign","adv_campaign_owner":"ФИНАЛЬНЫЙ ВЛАДЕЛЕЦ","adv_campaign_owner_contact":"iiivanov@restaurant_id","adv_campaign_datetime_star
t":1659203516,"adv_campaign_datetime_end":2659207116,"client_id":"223e4567-e89b-12d3-a456-426614174000","datetime_created":165913 1516,"trigger_datetime_created":1697181012}
{"restaurant_id":"123e4567-e89b-12d3-a456-426614174000","adv_campaign_id":"ФИНАЛЬНАЯ ПРОВЕРКА","adv_campaign_content":"first camp
aign","adv_campaign_owner":"ФИНАЛЬНЫЙ ВЛАДЕЛЕЦ","adv_campaign_owner_contact":"iiivanov@restaurant_id","adv_campaign_datetime_star
t":1659203516,"adv_campaign_datetime_end":2659207116,"client_id":"623e4567-e89b-12d3-a456-426614174000","datetime_created":165913 1516,"trigger_datetime_created":1697181012}
% Reached end of topic mustdayker_out [0] at offset 79
{"restaurant_id":"123e4567-e89b-12d3-a456-426614174000","adv_campaign_id":"ФИНАЛЬНАЯ ПРОВЕРКА","adv_campaign_content":"first camp
aign","adv_campaign_owner":"ФИНАЛЬНЫЙ ВЛАДЕЛЕЦ","adv_campaign_owner_contact":"iiivanov@restaurant_id","adv_campaign_datetime_star
t":1659203516,"adv_campaign_datetime_end":2659207116,"client_id":"323e4567-e89b-12d3-a456-426614174000","datetime_created":165913 1516,"trigger_datetime_created":1697181012}
% Reached end of topic mustdayker_out [0] at offset 80
root@fhm1tgbjqt1j0h594iv:/util$ []
```

8. Персистентность датафрейма ▲

```
df.persist()  
df.unpersist()
```