Sprawozdanie.md

Sprawozdanie

Zestaw danych: Netflix-Prize-Data

Technologia: Apache Spark Structured Streaming

Producent; skrypty inicjujące i zasilający

Uruchamiamy cluster dataproc i przechodzimy do konsoli węzła master

```
gcloud dataproc clusters create ${CLUSTER_NAME} \
--enable-component-gateway --bucket ${BUCKET_NAME} --region ${REGION} -
--master-machine-type n1-standard-4 --master-boot-disk-size 50 \
--num-workers 2 --worker-machine-type n1-standard-2 --worker-boot-disk-
--image-version 2.1-debian11 --optional-components DOCKER,ZOOKEEPER \
--project ${PROJECT_ID} --max-age=2h \
--metadata "run-on-master=true" \
--initialization-actions \
gs://goog-dataproc-initialization-actions-${REGION}/kafka/kafka.sh
```

Pobieramy repozytorium ze skryptami i uruchamiamy skrypt inicjujący środowisko. Alternatywnie możemy rozpakować archiwum z kodem źródłowym.

```
git clone https://github.com/MichalxPZ/Spark-Structured-Streaming---Net
mv Spark-Structured-Streaming---Netflix-Prize-Data/* .
rm -rf Spark-Structured-Streaming---Netflix-Prize-Data
```

Inicjalizujemy zmienne środowiskowe - możliwość edycji w pliku env.sh
Pobieramy dane, budujemy kody źródłowe oraz uruchamiamy kontener z postgresem.
Inicjalizujemy również schemat bazy danych i nadajemy odpowiednie uprawnienia.

```
source ./setup.sh
```

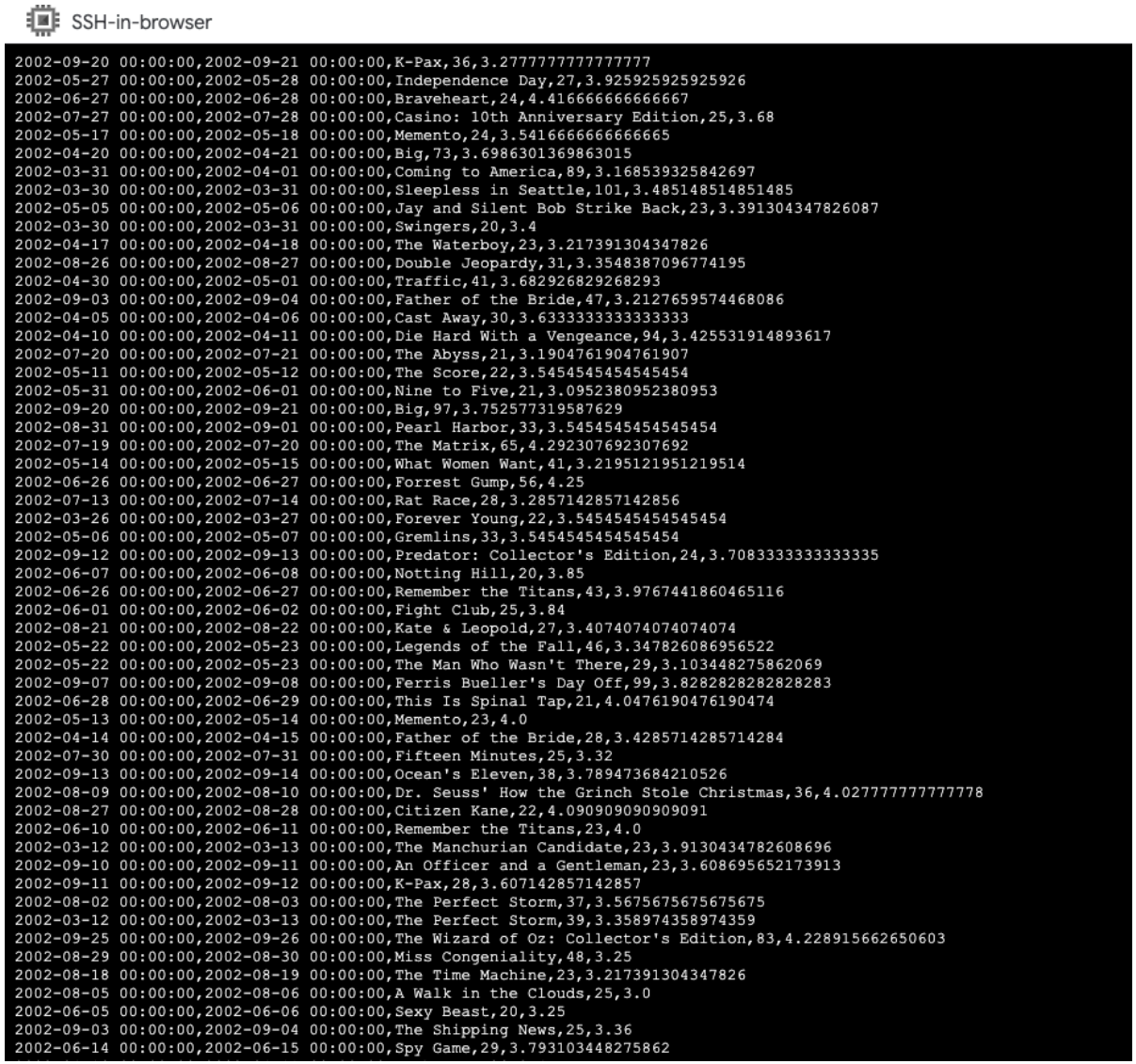
Polecenie uruchamiające producenta danych

```
source ./kafka_producer.sh
```

Konsument: skrypt odczytujący wyniki przetwarzania

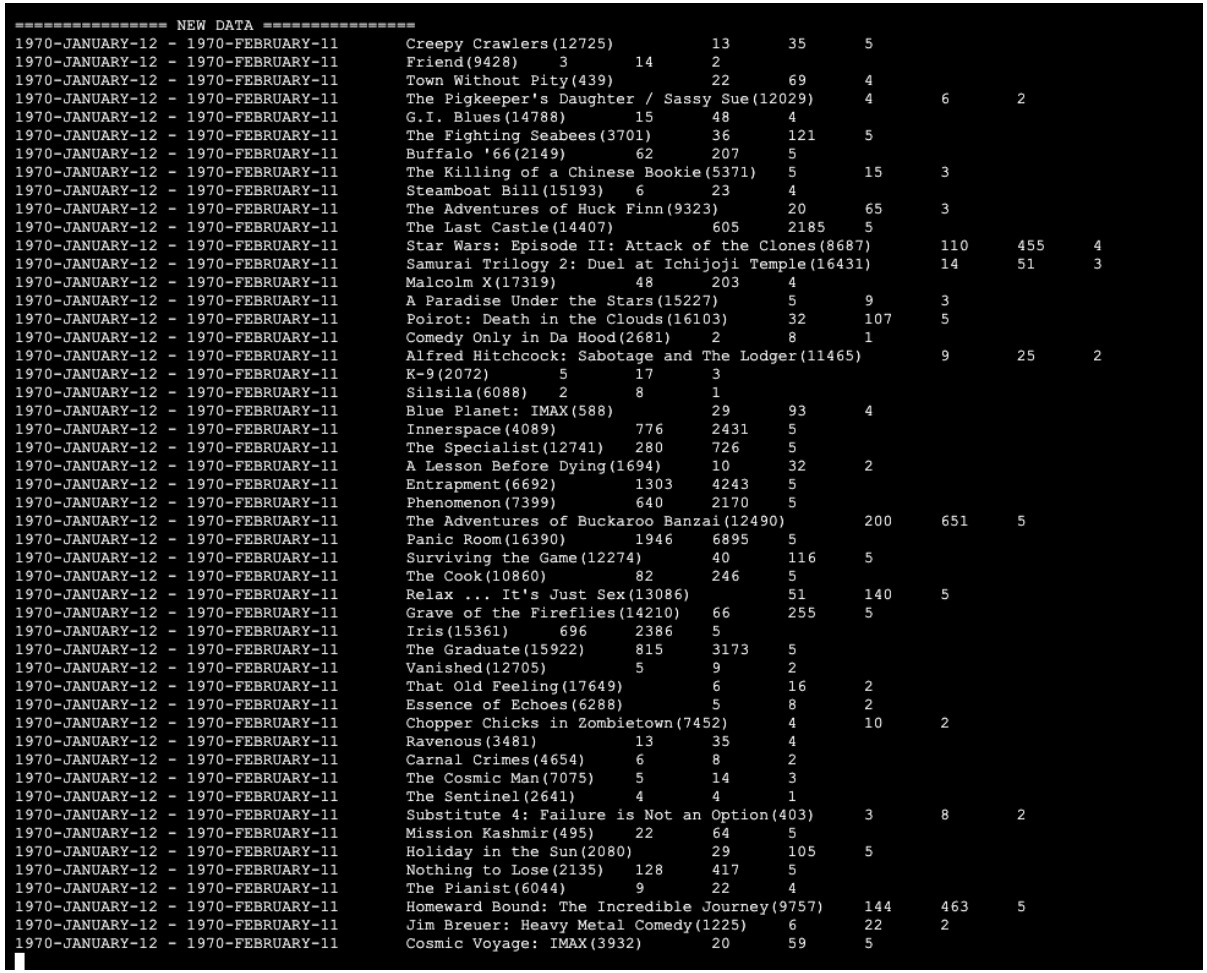
Aplikacja posiada dwóch konsumentów, którzy odczytują dane z bazy danych lub tematu Kafki.
Domyślnie konsument nasłuchujący na temat kafki zbiera dane dotyczące anomalii przetwarzania.

```
source ./kafka_consumer.sh
```



Konsument typu jdbc odczytuje dane zagregowane.

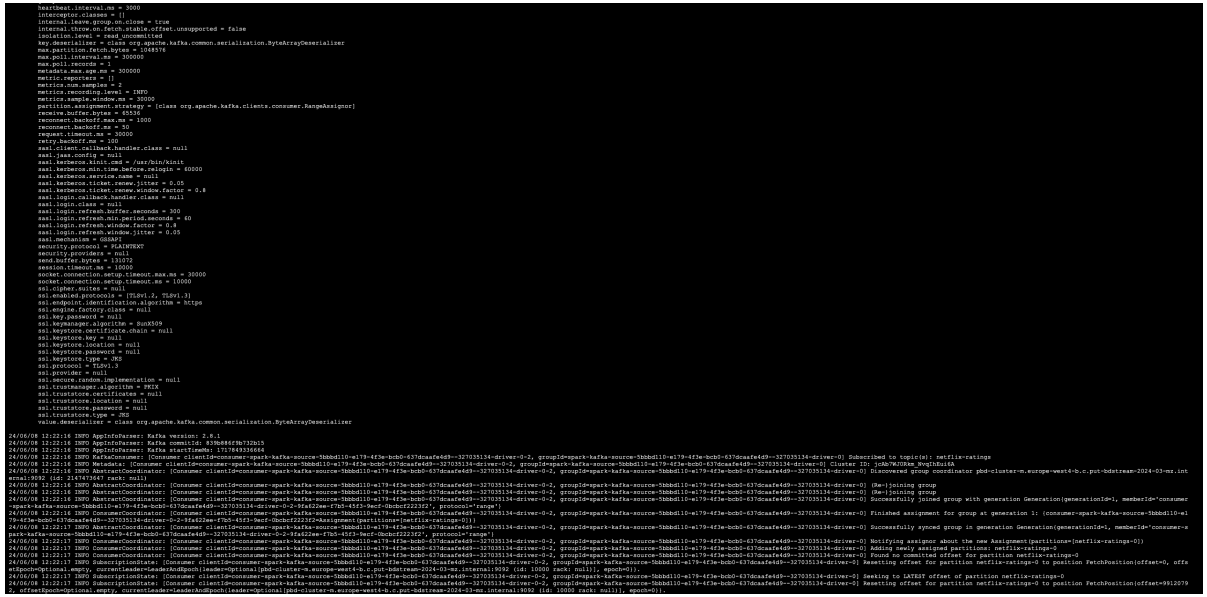
```
source ./jdbc_consumer.sh
```



Utrzymanie obrazu czasu rzeczywistego – transformacje

Aby uruchomić przetwarzanie strumieniowe, należy uruchomić skrypt

```
source ./processing_engine.sh
```



Ogólna idea przetwarzania danych

1. **Pobieranie danych z Kafki**

   Dane są pobierane z tematu Kafki (kafka_topic), które są w formacie CSV z czterema kolumnami: timestamp, movie_id, user_id i rating. Te dane są przetwarzane strumieniowo przy użyciu Spark Structured Streaming. Strumień danych jest odczytywany za pomocą SparkSession, a wartości są parsowane i przekształcane na odpowiednie typy danych.

2. **Agregowanie danych**

   Grupowanie i agregacja:

   Dane są grupowane według 30-dniowego okna czasowego i movie_id. Agregaty obejmują liczbę ocen (rating_count), sumę ocen (rating_sum) oraz liczbę unikalnych ocen (unique_rating_count). Zapis do bazy danych:

Wyniki agregacji są zapisywane do bazy danych PostgreSQL. W zależności od trybu przetwarzania (A dla trybu "append" i C dla trybu "complete").

3. **Wykrywanie anomalii**

   Watermark i grupowanie:

   Dane są przetwarzane z 1-dniowym watermarkiem, co pozwala na tolerowanie opóźnień w strumieniu danych. Dane są grupowane według okna czasowego (zdefiniowanego przez sliding_window_size_days) i movie_id.

   Wykrywanie anomalii:

   Wyliczane są średnie oceny (ratingMean) i liczba ocen (ratingCount) dla każdego okna czasowego i movie_id. Anomalie są wykrywane, jeśli liczba ocen i średnia ocena przekraczają odpowiednie progi (anomaly_rating_count_threshold i anomaly_rating_mean_threshold).

   Zapis anomalii do Kafki:

   Zidentyfikowane anomalie są wysyłane z powrotem do tematu Kafki (kafka_anomaly_topic).

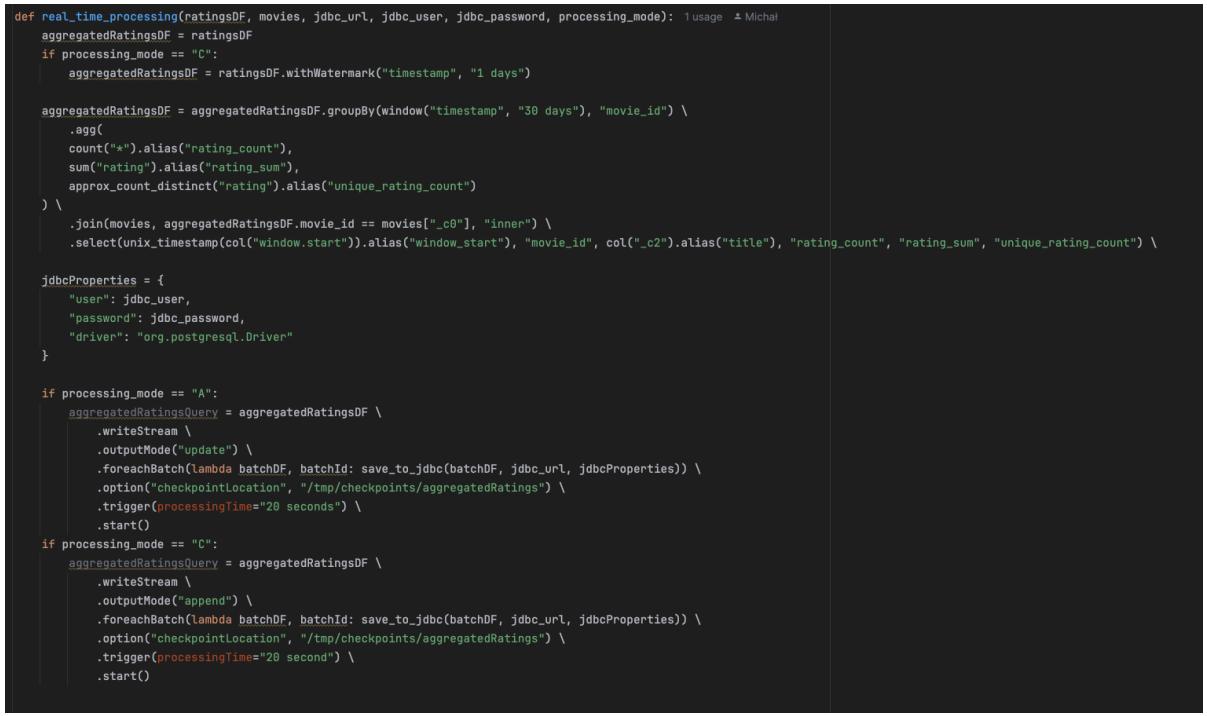
Utrzymanie obrazu czasu rzeczywistego – obsługa trybów

W zależności od pamateru ustawionego w pliku env.sh, aplikacja działa w jednym z dwóch trybów: A lub C.

1. W pierwszym (delay=A) program ma dostarczać dane do obrazu czasu rzeczywistego z najmniejszym możliwym opóźnieniem, nawet jeśli

dostarczane wyniki nie są ostateczne i będzie trzeba je wielokrotnie aktualizować.

2. W drugim (delay=C) program ma dostarczać dane do obrazu czasu rzeczywistego najszybciej jak się da, ale tylko wyniki ostateczne, tak aby nie było potrzeby ich późniejszej aktualizacji.



```
def real_time_processing(ratingsDF, movies, jdbc_url, jdbc_user, jdbc_password, processing_mode):  1 usage  Michał
    aggregatedRatingsDF = ratingsDF
    if processing_mode == "C":
        aggregatedRatingsDF = ratingsDF.withWatermark("timestamp", "1 days")

    aggregatedRatingsDF = aggregatedRatingsDF.groupBy(window("timestamp", "30 days"), "movie_id") \
        .agg(
        count("*").alias("rating_count"),
        sum("rating").alias("rating_sum"),
        approx_count_distinct("rating").alias("unique_rating_count")
    ) \
        .join(movies, aggregatedRatingsDF.movie_id == movies["_c0"], "inner") \
        .select(unix_timestamp(col("window.start")).alias("window_start"), "movie_id", col("_c2").alias("title"), "rating_count", "rating_sum", "unique_rating_count") \

    jdbcProperties = {
        "user": jdbc_user,
        "password": jdbc_password,
        "driver": "org.postgresql.Driver"
    }

    if processing_mode == "A":
        aggregatedRatingsQuery = aggregatedRatingsDF \
            .writeStream \
            .outputMode("update") \
            .foreachBatch(lambda batchDF, batchId: save_to_jdbc(batchDF, jdbc_url, jdbcProperties)) \
            .option("checkpointLocation", "/tmp/checkpoints/aggregatedRatings") \
            .trigger(processingTime="20 seconds") \
            .start()
    if processing_mode == "C":
        aggregatedRatingsQuery = aggregatedRatingsDF \
            .writeStream \
            .outputMode("append") \
            .foreachBatch(lambda batchDF, batchId: save_to_jdbc(batchDF, jdbc_url, jdbcProperties)) \
            .option("checkpointLocation", "/tmp/checkpoints/aggregatedRatings") \
            .trigger(processingTime="20 second") \
            .start()
```

Tryb A - anomalie nie występują (należy zmienić wartość w pliku env.sh)

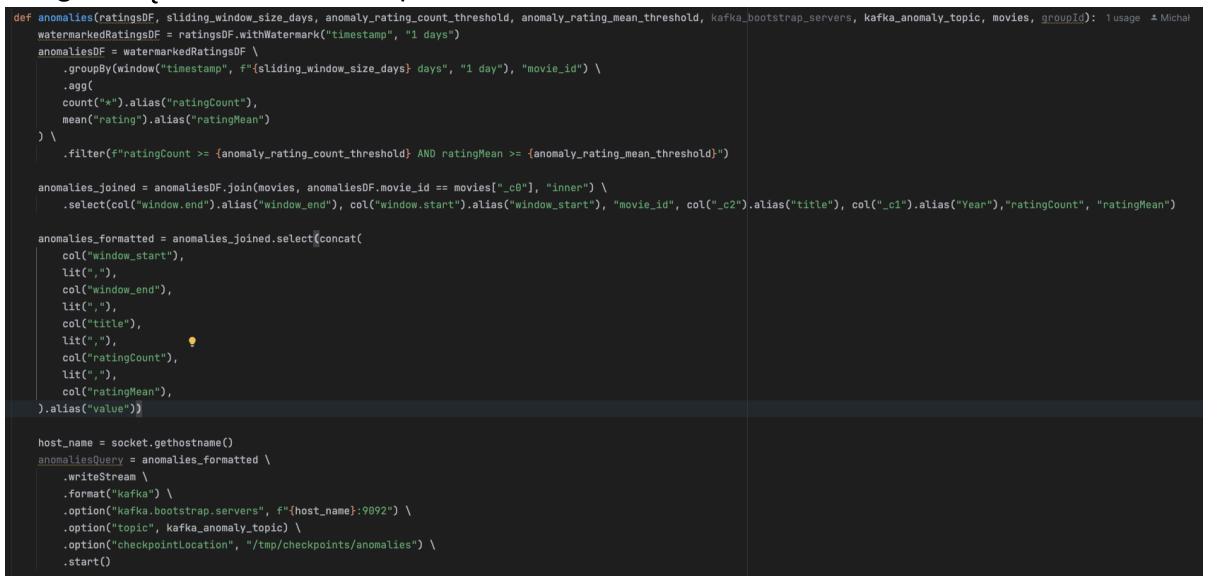
```
export ANOMALY_PERIOD_LENGTH=30
export ANOMALY_RATING_COUNT=70
export ANOMALY_RATING_MEAN=4
```

Tryb C - anomalie występują stosunkowo często (należy zmienić wartość w pliku env.sh)

```
export ANOMALY_PERIOD_LENGTH=1
export ANOMALY_RATING_COUNT=2
export ANOMALY_RATING_MEAN=2
```

Wykrywanie anomalii

Aplikacja wykrywa anomalie w danych, jeśli liczba ocen i średnia ocena przekraczają odpowiednie progi.
Progi te są zdefiniowane w pliku env.sh.



```
def anomalies(ratingsDF, sliding_window_size_days, anomaly_rating_count_threshold, anomaly_rating_mean_threshold, kafka_bootstrap_servers, kafka_anomaly_topic, movies, groupId): 1 usage  Michał
    watermarkedRatingsDF = ratingsDF.withWatermark("timestamp", "1 days")
    anomaliesDF = watermarkedRatingsDF \
        .groupBy(window("timestamp", f"{sliding_window_size_days} days", "1 day"), "movie_id") \
        .agg(
        count("*").alias("ratingCount"),
        mean("rating").alias("ratingMean")
    ) \
        .filter(f"ratingCount >= {anomaly_rating_count_threshold} AND ratingMean >= {anomaly_rating_mean_threshold}")

    anomalies_joined = anomaliesDF.join(movies, anomaliesDF.movie_id == movies["_c0"], "inner") \
        .select(col("window.end").alias("window_end"), col("window.start").alias("window_start"), "movie_id", col("_c2").alias("title"), col("_c1").alias("Year"),"ratingCount", "ratingMean")

    anomalies_formatted = anomalies_joined.select(concat(
        col("window_start"),
        lit(","),
        col("window_end"),
        lit(","),
        col("title"),
        lit(","),
        col("ratingCount"),
        lit(","),
        col("ratingMean"),
    ).alias("value"))

    host_name = socket.gethostname()
    anomaliesQuery = anomalies_formatted \
        .writeStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", f"{host_name}:9092") \
        .option("topic", kafka_anomaly_topic) \
        .option("checkpointLocation", "/tmp/checkpoints/anomalies") \
        .start()
```

Program przetwarzający strumienie danych; skrypt uruchamiający

Aby uruchomić przetwarzanie strumieniowe, należy uruchomić skrypt

```
source ./processing_engine.sh
```

```
source ./env.sh

$SPARK_HOME/bin/spark-submit \
  --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0,org.postgresql:postgresql:42.6.0 \
  processing/spark-structured.py \
  "$INPUT_FILE_PATH" \
  "$KAFKA_BOOTSTRAP_SERVERS" \
  "$KAFKA_DATA_TOPIC_NAME" \
  "$KAFKA_GROUP_ID" \
  "$JDBC_URL" \
  "$JDBC_USERNAME" \
  "$JDBC_PASSWORD" \
  "$ANOMALY_PERIOD_LENGTH" \
  "$ANOMALY_RATING_COUNT" \
  "$ANOMALY_RATING_MEAN" \
  "$KAFKA_ANOMALY_TOPIC_NAME" \
  "$PROCESSING_TYPE"
```

Miejsce utrzymywania obrazów czasu rzeczywistego –

skrypt tworzący

Miejsce utrzymywania obrazów czasu rzeczywistego jest tworzone w skrypcie setup.sh

Jest to kontener z bazą PostgreSQL, który jest uruchamiany w klastrze dataproc.

Schematy bazy danych oraz tabele są tworzone w skrypcie setup.sql

```
DROP DATABASE IF EXISTS :"db_name";
CREATE DATABASE :"db_name" WITH ENCODING 'UTF8';
DROP USER IF EXISTS :"user";
CREATE USER :"user" WITH PASSWORD :'password';
ALTER DATABASE :"db_name" OWNER TO :"user";
ALTER SCHEMA public owner to :"user";
GRANT ALL PRIVILEGES ON DATABASE :"db_name" TO :"user";
GRANT USAGE ON SCHEMA public TO :"user";
GRANT CREATE ON SCHEMA public TO :"user";
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO :"user";

\c :"db_name";
DROP TABLE IF EXISTS movie_ratings;
CREATE TABLE IF NOT EXISTS movie_ratings (
     window_start BIGINT NOT NULL,
     movie_id VARCHAR(32) NOT NULL,
     title VARCHAR(128) NOT NULL,
     rating_count INTEGER NOT NULL,
     rating_sum INTEGER NOT NULL,
     unique_rating_count INTEGER NOT NULL,
     PRIMARY KEY (window_start, movie_id)
);
GRANT ALL PRIVILEGES ON TABLE movie_ratings TO :"user";
ALTER TABLE movie_ratings OWNER TO :"user";
```

Miejsce utrzymywania obrazów czasu rzeczywistego – cechy

PostgreSQL jest idealny do przechowywania zagregowanych danych w procesach Big Data dzięki skalowalności, która umożliwia obsługę dużych wolumenów danych, oraz zaawansowanym funkcjom analitycznym i agregacyjnym. Dzięki wsparciu dla równoległego przetwarzania zapytań i optymalizacji wydajności, PostgreSQL efektywnie zarządza skomplikowanymi analizami na dużych zbiorach danych. Dodatkowo, łatwa integracja z narzędziami Big Data i elastyczność w obsłudze różnych typów danych czynią go wszechstronnym rozwiązaniem.

Czyszczenie środowiska

Aby wyczyścić środowisko, należy uruchomić skrypt

```
source ./cleanup.sh
```