# BigData Projekt 2

## Klaster

```
gcloud dataproc clusters create ${CLUSTER_NAME} \
--enable-component-gateway --region ${REGION} --subnet default \
--master-machine-type n1-standard-4 --master-boot-disk-size 50 \
--num-workers 2 --worker-machine-type n1-standard-2 --worker-boot-disk-size 50 \
--image-version 2.1-debian11 --optional-components ZOOKEEPER,DOCKER \
--project ${PROJECT_ID} --max-age=3h \
--metadata "run-on-master=true" \
--initialization-actions \
gs://goog-dataproc-initialization-actions-${REGION}/kafka/kafka.sh
```

## Jak korzystać ze skryptów w praktyce

1. Utwórz 4 terminale
2. Na jednym z nich uruchom

```
> unzip projekt2.zip
> chmod +x *.sh
> ./init.sh
```

3. Na osobnych terminalach uruchom

```
terminal1> ./start_processing.sh
```

zaczeka ok minuty aż silnik będzie gotowy (wtedy kiedy wypisują się jsony z offsetami: 0)

```
terminal2> ./kafka_producer.sh
```

Odczytywanie można uruchomić kiedykolwiek:

```
terminal3> ./kafka_console_consumer.sh
terminal4> ./read_output.sh
```

4. Aby wykonać punkt 3. od nowa uruchom

```
> ./reset.sh
```

Parametry dla silnika przetwarzającego znajdują się w params.sh Można je zmodyfikować i ponownie wykonać punkt 3.

`read_output.sh` tylko odczytuje bazę danych poprzez select *

Aby wykonywać własne zapytania:

```
> source ./env.sh
> psql -h localhost -p 8432 -U $PG_USER -v db_name="$DB_NAME" -v
db_table="$DB_TABLE"
> \c :db_name
> select count(*) from :db_table;
```

# Producent; skrypty inicjujące i zasilający

## Skrypt inicjujący

`init.sh`

- pobiera dane
- instaluje bazę danych
- tworzy tematy kafki
- tworzy jary

## Skryp resetujący środowisko

- do tworzenia/resetowania tematów Kafkia: `kafka_topics_reset.sh` (pojawiają się błedy jak tematy nie istnieją, ale nie są one istotne)
- do resetowania środowiska: `reset.sh`, zawiera dodadkowo resetowanie bazy danych i checkpointów

## Skrypt zasilający tematy Kafki

`kafka_producer.sh` - uruchamia program javy `StocksProcessing.jar`. Program listuje pliki w podanym katalogu, sortuje je po nazwie i wypisuje zawartość wszystkich wierszy do tematu kafki. Błędy SLF4J są nieistotne.

```
List<Path> filePaths = new ArrayList<>();
try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(directoryPath)) {
    for (Path path : directoryStream) {
        if (Files.isRegularFile(path)) {
            filePaths.add(path);
        }
    }
}

Collections.sort(filePaths);
```

```
for (Path path : filePaths) {
    Files.lines(path)
            .skip(1)
            .forEach(line -> producer.send(new ProducerRecord<>(kafkaTopic,
line)));
}
```

## Utrzymanie obrazu czasu rzeczywistego – transformacje

`processing.py`

```
lines -> parsed_df -> joined_df -> aggregated_df -> aggregated_query
        company_df /
```

## Utrzymanie obrazu czasu rzeczywistego – obsługa trybu A

W `params.sh` ustawić `PROCESSING_MODE="update"`

```
.outputMode(working_mode)
```

## Utrzymanie obrazu czasu rzeczywistego – obsługa trybu C

W `params.sh` ustawić `PROCESSING_MODE="complete"`

```
.outputMode(working_mode)
```

## Wykrywanie anomalii

`processing.py`

```
lines -> parsed_df -> joined_df -> anomalies_df -> anomalies_query
        company_df /
```

## Program przetwarzający strumienie danych; skrypt uruchamiający

Całe przetwarzanie znajduje się w pliku `processing.py`. Komentarze w kodzie wskazują w którym miejscu co się dzieje.

Skrypt uruchamiający: `start_processing.sh`

Skrypt przyjmuje argumenty: `<symbols_meta> <bootstrap_servers> <checkpoints_location> <kafka_data_topic> <kafka_anomalies_topic> <db_user> <db_password> <db_name> <db_table> <working_mode> <anomalies_window_size> <anomalies_threshold>`

Wariantem uruchomienia można sterować z poziomu pliku `params.sh`

Brak anomali: `ANOMALIES_WINDOW_SIZE="1" ANOMALIES_THRESHOLD="1000000"`

Częste anomalie: `ANOMALIES_WINDOW_SIZE="7" ANOMALIES_THRESHOLD="40"`

## Miejsce utrzymywania obrazów czasu rzeczywistego – skrypt tworzący

Relacyjna baza danych postgres. Skrypt instalujący + tworzący bazę: `prepare_db.sh`

Skrypt tworzący/resetujący bazę: `db_reset.sh` i `prepare_database.sql`

## Miejsce utrzymywania obrazów czasu rzeczywistego – cechy

- dobra integracja z systemami przetwarzania strumieni danych
- danych nie jest aż tak dużo: `liczba miesięcy * liczba spółek` (kilka milionów), aby była duża konieczność wykorzystywania bardziej skalowalnych baz danych.

## Konsument: skrypt odczytujący wyniki przetwarzania

Odczyt obrazu czasu rzeczywistego: `read_output.sh`

Odczyt anomalii: `kafka_console_consumer.sh` wykorzystuje `StocksConsoleConsumer.jar`

## Co jest źle

- Funkcja window przyjmuje na wejście offset od 01.01.1970, a w zbiorze są dane od 1962. Timestampy przed 1970 są w jakiś sposób konwertowane na po 1970, ale te wyniki oczywiście będą złe.