Sprawozdanie - Apache Flink - NYC Yellow Taxi

full code available in github repository: https://github.com/JNeubau

Prepare the environment (Producent; skrypty inicjujące i zasilający)

```
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,FLINK \
--project ${PROJECT_ID} --max-age=3h \
--metadata "run-on-master=true" \
--initialization-actions gs://goog-dataproc-initialization-
actions-${REGION}/kafka/kafka.sh
```

1. Download package from https://github.com/JNeubau/BD-Flink-NYC-yellow-taxi.git. Unpack the files.
2. Make sure that source data is in your bucket in with the original name.
3. Execute`chmod +x *.sh`
4. Change vaiables in `environ.sh` file to match your data

- `INPUT_DATA_LOCATION` - your bucket
- `INPUT_DATA_LOC_FILE` - static input data
- `INPUT_DATA_DIR_TAXI` - folder with events

```
git clone https://github.com/JNeubau/BD-Flink-NYC-yellow-taxi.git
cd BD-Flink-NYC-yellow-taxi/
mv * ../
chmod +x *.sh
```

5. Run the `setup.sh` script to prepare environment
6. Using `vim flink.properties` create a file and pase the following data, changing `CASSANDRA_HOST`, `BOOTSTRAP_SERVERS`, `taxiData.directoryPath`, `zoneFile.path` according to your data. CASSANDRA_HOST is value of cluster name with '-m' (ex: `pdb-cluster-m`) and BOOTSTRAP_SERVERS is value of cluster name with '-w-0:9092' (ex: `pdb-cluster-w-0:9092`).

Example of the file:

```
taxiData.directoryPath = yellow_tripdata_result/
zoneFile.path = taxi_zone_lookup.csv

taxiEvents.maxElements = 10000
taxiEvents.elementDelayMillis = 100

FLINK_ANOMALY_TIME = 4
```

```
FLINK_ANOMALY_PEOPLE = 1000
DELAY_VERSION = C

BOOTSTRAP_SERVERS = pdb-cluster-w-0:9092
TAXI_INPUT_TOPIC = taxi-input-topic
LOC_INPUT_TOPIC = loc-input-topic
ANOMALY_OUTPUT_TOPIC = anomaly-topic
KAFKA_GROUP_ID = kafka-group-id
#FLINK_DELAY = flink-delay
#FLINK_CHECKPOINT_DIR = flink-checkpoint-dir

CASSANDRA_HOST = pdb-cluster-m
CASSANDRA_PORT = 9042
```

7. Execute these commands:

```
mkdir -p src/main/resources/
mv flink.properties src/main/resources/
```

8. Upload the jar file `TaxiEventsAnalysis.jar` and move it to to your working directory.

```
mv ~/TaxiEventsAnalysis.jar ./
```

9. In one of the terminals run the `sender-kafka_taxi.sh` script to start sending data via kafka producer.

- setup.sh

- reciver-kafka.sh

10. At this time, it should be possible to run the script using `./run.sh` unfortunately te JAR file does not work correctly and cannot be run on gcloud.

11. To check if sources work these commands can be used:

- for event data

```
source ./environ.sh (after running `sender-kafka_taxi.sh`)
kafka-console-consumer.sh --group my-consumer-group \
--bootstrap-server ${CLUSTER_NAME}-w-0:9092 \
--topic ${KAFKA_TOPIC_TAXI} --from-beginning
```

- for static data (after running `setup.sh`)

```
source ./environ.sh
kafka-console-consumer.sh --group my-consumer-group \
--bootstrap-server ${CLUSTER_NAME}-w-0:9092 \
--topic ${KAFKA_TOPIC_LOC} --from-beginning
```

- example (static):

```
kafka-console-consumer.sh --group my-consumer-group \
--bootstrap-server ${CLUSTER_NAME}-w-0:9092 \
--topic ${KAFKA_TOPIC_LOC} --from-beginning
1,"EWR","Newark Airport","EWR"
2,"Queens","Jamaica Bay","Boro Zone"
3,"Bronx","Allerton/Pelham Gardens","Boro Zone"
4,"Manhattan","Alphabet City","Yellow Zone"
5,"Staten Island","Arden Heights","Boro Zone"
6,"Staten Island","Arrochar/Fort Wadsworth","Boro Zone"
7,"Queens","Astoria","Boro Zone"
8,"Queens","Astoria Park","Boro Zone"
9,"Queens","Auburndale","Boro Zone"
10,"Queens","Baisley Park","Boro Zone"
11,"Brooklyn","Bath Beach","Boro Zone"
12,"Manhattan","Battery Park","Yellow Zone"
13,"Manhattan","Battery Park City","Yellow Zone"
14,"Brooklyn","Bay Ridge","Boro Zone"
15,"Queens","Bay Terrace/Fort Totten","Boro Zone"
16,"Queens","Bayside","Boro Zone"
17,"Brooklyn","Bedford","Boro Zone"
18,"Bronx","Bedford Park","Boro Zone"
19,"Queens","Bellerose","Boro Zone"
20,"Bronx","Belmont","Boro Zone"
21,"Brooklyn","Bensonhurst East","Boro Zone"
22,"Brooklyn","Bensonhurst West","Boro Zone"
23,"Staten Island","Bloomfield/Emerson Hill","Boro Zone"
24,"Manhattan","Bloomingdale","Yellow Zone"
25,"Brooklyn","Boerum Hill","Boro Zone"
26,"Brooklyn","Borough Park","Boro Zone"
27,"Queens","Breezy Point/Fort Tilden/Riis Beach","Boro Zone"
28,"Queens","Briarwood/Jamaica Hills","Boro Zone"
29,"Brooklyn","Brighton Beach","Boro Zone"
30,"Queens","Broad Channel","Boro Zone"
31,"Bronx","Bronx Park","Boro Zone"
32,"Bronx","Bronxdale","Boro Zone"
33,"Brooklyn","Brooklyn Heights","Boro Zone"
34,"Brooklyn","Brooklyn Navy Yard","Boro Zone"
35,"Brooklyn","Brownsville","Boro Zone"
36,"Brooklyn","Bushwick North","Boro Zone"
37,"Brooklyn","Bushwick South","Boro Zone"
38,"Queens","Cambria Heights","Boro Zone"
39,"Brooklyn","Canarsie","Boro Zone"
40,"Brooklyn","Carroll Gardens","Boro Zone"
41,"Manhattan","Central Harlem","Boro Zone"
42,"Manhattan","Central Harlem North","Boro Zone"
43,"Manhattan","Central Park","Yellow Zone"
44,"Staten Island","Charleston/Tottenville","Boro Zone"
45,"Manhattan","Chinatown","Yellow Zone"
46,"Bronx","City Island","Boro Zone"
```

12. If the project worked, the output could be checked:

- for anomalies `./reciver-anomaly.sh` to read anomalies from kafka topic
- for normal output via `./show_output.sh` to view table with results. Without the working JAR, it will be empty. section-Read (Konsument: skrypt odczytujący wyniki przetwarzania)

Transformations (Utrzymanie obrazu czasu rzeczywistego – transformacje)

For current version, I could not verify the part reading from Kafka source and saving data in database and kafka. Those parts are placed in `connectors` package (without TaxiEventSource file). The classes

below will explain it's functions being based on reading local file and printing data in the InteliJ console. Sources code and explenation can be found at the bottom: Kafka sources

1. `Connectors.TaxiEventSource` is a file used for reading events data from localy placed file. `tools.EnrichWithLocData` is a file used to read the contents of the static file. It returns taxi data with added pieces of relevant information from static data file.

2. In the main file `TaxiEventsAnalysis` the flow is as follows:

```
/* Join two types of data */
DataStream<TaxiLocEvent> taxiLocEventsDS = taxiEventsDS
    .map(new EnrichWithLocData(properties.getRequired("zoneFile.path")))
    .assignTimestampsAndWatermarks(
            WatermarkStrategy.
<TaxiLocEvent>forBoundedOutOfOrderness(Duration.ofDays(1))
                    .withTimestampAssigner(((taxiLocEvent, l) ->
taxiLocEvent.getTimestamp().toInstant().
                            getEpochSecond() * 1000))
    );
```

The `assignTimestampsAndWatermarks( WatermarkStrategy. <TaxiLocEvent>forBoundedOutOfOrderness(Duration.ofDays(1))...)` shows that data late for one day will be still considered in the given windows.

2.

```
/* Dokonywanie obliczeń dla
        - każdej dzielnicy
        - każdego kolejnego dnia
    Dla zmiennych:
        - ile było wyjazdów (startStop = 0),
        - ile było przyjazdów (startStop = 1)
        - liczba pasażerów obsłużona dla przyjazdów
        - liczba pasażerów obsłużona dla wyjazdów
*/
String delay = properties.getRequired("DELAY_VERSION");

DataStream<ResultData> taxiLocStatsDS = taxiLocEventsDS
    .keyBy(TaxiLocEvent::getBorough)
    .window(TumblingEventTimeWindows.of(Time.days(1)))
    .trigger((Objects.equals(delay, "A")) ? EveryEventTimeTrigger.create() :
EventTimeTrigger.create())
    .aggregate(new TaxiLocAggregator(), new GetFinalResultWindowFunction());
```

The `window(TumblingEventTimeWindows.of(Time.days(1)))` creates windows that last for one day and ensures there is max 1 window at given time. ResultData is the final form of data in the output.

Data is grouped by the *borough* variable. Windows are created aggregating data by one day. Trigger is chosen based on the delay option (*A* or *C*) with default value being *C* (completness). Tan data is aggregated using below AggregateFunction class and ProcessWindowFunction class.

```
public class TaxiLocAggregator implements AggregateFunction<TaxiLocEvent,
TaxiLocAccumulator, TaxiLocStats> {
    ...
    @Override
    public TaxiLocAccumulator add(TaxiLocEvent value, TaxiLocAccumulator
accumulator) {
        if (value.getStartStop() == 0) {
            accumulator.addDeparture(value.getPassengerCount());
        } else if (value.getStartStop() == 1) {
            accumulator.addArrival(value.getPassengerCount());
        }
        return accumulator;
    }
    ...
}
```

`GetFinalResultWindowFunction` is a function for processing the final ResultData and is ussed with aggregation function `TaxiLocAggregator`

```
public class GetFinalResultWindowFunction extends
ProcessWindowFunction<TaxiLocStats, ResultData, String, TimeWindow> {
    @Override
    public void process(String key, Context context, Iterable<TaxiLocStats> input,
Collector<ResultData> out) {
        int departures = 0;
        int arrivals = 0;
        int totalPassengersArr = 0;
        int totalPassengersDep = 0;

        for (TaxiLocStats stats : input) {
            departures += stats.getDepartures();
            arrivals += stats.getArrivals();
            totalPassengersArr += stats.getTotalPassengersArr();
            totalPassengersDep += stats.getTotalPassengersDep();
        }

        Instant windowStart = Instant.ofEpochMilli(context.window().getStart());
        Instant windowEnd = Instant.ofEpochMilli(context.window().getEnd());

        ResultData resultData = new ResultData(
                key,
                Date.from(windowStart),
                Date.from(windowEnd),
                departures,
                arrivals,
                totalPassengersArr,
```

```
                totalPassengersDep);

        out.collect(resultData);
    }
}
```

3. Anomalies are considered and filtered see in Anomalies (Wykrywanie anomalii) section

A delay (Utrzymanie obrazu czasu rzeczywistego – obsługa trybu A)

Delay can be changed in `flink.properties` file - `DELAY_VERSION` variable.

`EveryEventTimeTrigger` is an overwriten class of Trigger that enables the smallest possible delay. Use examples

```
public class EveryEventTimeTrigger extends Trigger<Object, TimeWindow> {
    ...
    @Override
    public TriggerResult onElement(
            Object element, long timestamp, TimeWindow window, TriggerContext ctx)
{
        return TriggerResult.FIRE;
    }

    @Override
    public TriggerResult onEventTime(long time, TimeWindow window, TriggerContext
ctx) {
        return TriggerResult.CONTINUE;
    }

    @Override
    public TriggerResult onProcessingTime(long time, TimeWindow window,
TriggerContext ctx) {
        return TriggerResult.CONTINUE;
    }
    ...
}
```

C delay (Utrzymanie obrazu czasu rzeczywistego – obsługa trybu C)

Delay can be changed in `flink.properties` file - `DELAY_VERSION` variable.

A standard `EventTimeTrigger` is used to achive completness trigger Use examples

Anomalies (Wykrywanie anomalii)

- call of `TaxiLocAggregator` (the same which was used in normal processing) and `GetAnomalyWindowFunction` (slightly changed class printing the data with different data model)

functions are used to aggregate data. `Filter` is used to choose data which are above chosen parameters.

The anomalies are controlled by parameters. They can be changed in `flink.properties` file. Specificly: `FLINK_ANOMALY_TIME` and `FLINK_ANOMALY_PEOPLE`.

```
DataStream<DeparturesAnomaly> anomalyOutput = taxiLocEventsDS
    .keyBy(TaxiLocEvent::getBorough)
    .window(SlidingEventTimeWindows.of(
        Time.hours(anomalyTime),
        Time.hours(1)))
    .aggregate(new TaxiLocAggregator(), new GetAnomalyWindowFunction())
    .filter(departuresAnomaly -> departuresAnomaly.getDifference() >=
anomalyPeople);
```

- Class processing data:

```
public class GetAnomalyWindowFunction extends ProcessWindowFunction<TaxiLocStats,
DeparturesAnomaly, String, TimeWindow> {
    @Override
    public void process(String key, Context context, Iterable<TaxiLocStats> input,
Collector<DeparturesAnomaly> out) {
        int departures = 0;
        int arrivals = 0;
        int totalPassengersArr = 0;
        int totalPassengersDep = 0;

        for (TaxiLocStats stats : input) {
            departures += stats.getDepartures();
            arrivals += stats.getArrivals();
            totalPassengersArr += stats.getTotalPassengersArr();
            totalPassengersDep += stats.getTotalPassengersDep();
        }

        Instant windowStart = Instant.ofEpochMilli(context.window().getStart());
        Instant windowEnd = Instant.ofEpochMilli(context.window().getEnd());

        DeparturesAnomaly departuresAnomaly = new DeparturesAnomaly(
                key,
                Date.from(windowStart),
                Date.from(windowEnd),
                totalPassengersArr,
                totalPassengersDep,
                totalPassengersDep - totalPassengersArr);

        out.collect(departuresAnomaly);
    }
}
```

Examples

```
1> ResultData{borough='Manhattan', from=Tue Aug 10 01:00:00 CET 1926, to=Wed Aug 11 01:00:00 CET 1926, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
1> ResultData{borough='Manhattan', from=Tue Dec 23 01:00:00 CET 2003, to=Wed Dec 24 01:00:00 CET 2003, departures=1, arrivals=1, totalPassengersArr=1, totalPassengersDep=1}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=61, arrivals=54, totalPassengersArr=112, totalPassengersDep=127}
1> ResultData{borough='Manhattan', from=Thu Jan 01 01:00:00 CET 2009, to=Fri Jan 02 01:00:00 CET 2009, departures=43, arrivals=57, totalPassengersArr=76, totalPassengersDep=52}
15> ResultData{borough='Brooklyn', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=1, arrivals=2, totalPassengersArr=2, totalPassengersDep=1}
15> ResultData{borough='Brooklyn', from=Thu Jan 01 01:00:00 CET 2009, to=Fri Jan 02 01:00:00 CET 2009, departures=2, arrivals=4, totalPassengersArr=9, totalPassengersDep=6}
12> ResultData{borough='Bronx', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=1, arrivals=1, totalPassengersArr=1, totalPassengersDep=1}
14> ResultData{borough='Unknown', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=2, arrivals=3, totalPassengersArr=3, totalPassengersDep=2}
14> ResultData{borough='Unknown', from=Thu Jan 01 01:00:00 CET 2009, to=Fri Jan 02 01:00:00 CET 2009, departures=0, arrivals=2, totalPassengersArr=4, totalPassengersDep=0}
13> ResultData{borough='EWR', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
3> ResultData{borough='Queens', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=21, arrivals=4, totalPassengersArr=4, totalPassengersDep=26}
3> ResultData{borough='Queens', from=Thu Jan 01 01:00:00 CET 2009, to=Fri Jan 02 01:00:00 CET 2009, departures=3, arrivals=6, totalPassengersArr=10, totalPassengersDep=7}
1> ResultData{borough='Manhattan', from=Sun Oct 28 02:00:00 CEST 2018, to=Mon Oct 29 01:00:00 CET 2018, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
3> ResultData{borough='Queens', from=Wed Oct 31 01:00:00 CET 2018, to=Thu Nov 01 01:00:00 CET 2018, departures=409, arrivals=344, totalPassengersArr=590, totalPassengersDep=677}
3> ResultData{borough='Queens', from=Thu Nov 01 01:00:00 CET 2018, to=Fri Nov 02 01:00:00 CET 2018, departures=4715, arrivals=4945, totalPassengersArr=7546, totalPassengersDep=7298}
1> ResultData{borough='Manhattan', from=Wed Oct 31 01:00:00 CET 2018, to=Thu Nov 01 01:00:00 CET 2018, departures=9035, arrivals=6698, totalPassengersArr=10894, totalPassengersDep=14671}
4> ResultData{borough='Staten Island', from=Wed Oct 31 01:00:00 CET 2018, to=Thu Nov 01 01:00:00 CET 2018, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
1> ResultData{borough='Manhattan', from=Thu Nov 01 01:00:00 CET 2018, to=Fri Nov 02 01:00:00 CET 2018, departures=83753, arrivals=78588, totalPassengersArr=120118, totalPassengersDep=128256}
13> ResultData{borough='EWR', from=Wed Oct 31 01:00:00 CET 2018, to=Thu Nov 01 01:00:00 CET 2018, departures=1, arrivals=0, totalPassengersArr=0, totalPassengersDep=1}
13> ResultData{borough='EWR', from=Thu Nov 01 01:00:00 CET 2018, to=Fri Nov 02 01:00:00 CET 2018, departures=3, arrivals=141, totalPassengersArr=250, totalPassengersDep=14}
14> ResultData{borough='Unknown', from=Wed Oct 31 01:00:00 CET 2018, to=Thu Nov 01 01:00:00 CET 2018, departures=180, arrivals=134, totalPassengersArr=171, totalPassengersDep=237}
14> ResultData{borough='Unknown', from=Thu Nov 01 01:00:00 CET 2018, to=Fri Nov 02 01:00:00 CET 2018, departures=1599, arrivals=1582, totalPassengersArr=2126, totalPassengersDep=2143}
12> ResultData{borough='Bronx', from=Wed Oct 31 01:00:00 CET 2018, to=Thu Nov 01 01:00:00 CET 2018, departures=9, arrivals=68, totalPassengersArr=112, totalPassengersDep=16}
15> ResultData{borough='Brooklyn', from=Wed Oct 31 01:00:00 CET 2018, to=Thu Nov 01 01:00:00 CET 2018, departures=221, arrivals=663, totalPassengersArr=1071, totalPassengersDep=346}
12> ResultData{borough='Bronx', from=Thu Nov 01 01:00:00 CET 2018, to=Fri Nov 02 01:00:00 CET 2018, departures=255, arrivals=817, totalPassengersArr=1273, totalPassengersDep=374}
1> DeparturesAnomaly{borough='Manhattan', from=Wed Oct 31 22:00:00 CET 2018, to=Thu Nov 01 02:00:00 CET 2018, totalPassengersDeparted=25051, totalPassengersArrived=20588, difference=4463}
15> ResultData{borough='Brooklyn', from=Thu Nov 01 01:00:00 CET 2018, to=Fri Nov 02 01:00:00 CET 2018, departures=1716, arrivals=3816, totalPassengersArr=5973, totalPassengersDep=2549}
1> DeparturesAnomaly{borough='Manhattan', from=Wed Oct 31 23:00:00 CET 2018, to=Thu Nov 01 03:00:00 CET 2018, totalPassengersDeparted=32975, totalPassengersArrived=27658, difference=5317}
4> ResultData{borough='Staten Island', from=Thu Nov 01 01:00:00 CET 2018, to=Fri Nov 02 01:00:00 CET 2018, departures=9, arrivals=26, totalPassengersArr=42, totalPassengersDep=12}
1> DeparturesAnomaly{borough='Manhattan', from=Thu Nov 01 00:00:00 CET 2018, to=Thu Nov 01 04:00:00 CET 2018, totalPassengersDeparted=38380, totalPassengersArrived=32469, difference=5911}
1> DeparturesAnomaly{borough='Manhattan', from=Thu Nov 01 03:00:00 CET 2018, to=Thu Nov 01 07:00:00 CET 2018, totalPassengersDeparted=25186, totalPassengersArrived=20576, difference=4610}
1> DeparturesAnomaly{borough='Manhattan', from=Thu Nov 01 04:00:00 CET 2018, to=Thu Nov 01 08:00:00 CET 2018, totalPassengersDeparted=38973, totalPassengersArrived=32816, difference=6157}
1> DeparturesAnomaly{borough='Manhattan', from=Thu Nov 01 05:00:00 CET 2018, to=Thu Nov 01 09:00:00 CET 2018, totalPassengersDeparted=56624, totalPassengersArrived=50803, difference=5821}
1> DeparturesAnomaly{borough='Manhattan', from=Thu Nov 01 06:00:00 CET 2018, to=Thu Nov 01 10:00:00 CET 2018, totalPassengersDeparted=72543, totalPassengersArrived=68108, difference=4435}
```

The above piece of output is generated for ***C type*** and ***Anomalie***, both printed to console output (in InteliJ) for ***200 000*** elements in the input data. A ***4-hour*** time window during which the difference between people, who departed the borough and people who arravied to it, is over ***4000*** people is considered an anomaly.

Lines contianing `ResultData` are the output line, while lines contianing `DeparuresAnomalies` are lines with anomalies.

The bigger the anomalies time window, the more probable that the anomalies will be detected.

Similarly, the smaller the difference in number of people, the more anomalies will be detected.

```
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=1, arrivals=0, totalPassengersArr=0, totalPassengersDep=1}
1> ResultData{borough='Manhattan', from=Tue Aug 10 01:00:00 CET 1926, to=Wed Aug 11 01:00:00 CET 1926, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
1> ResultData{borough='Manhattan', from=Tue Dec 23 01:00:00 CET 2003, to=Wed Dec 24 01:00:00 CET 2003, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
1> ResultData{borough='Manhattan', from=Tue Dec 23 01:00:00 CET 2003, to=Wed Dec 24 01:00:00 CET 2003, departures=1, arrivals=1, totalPassengersArr=1, totalPassengersDep=1}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=1, arrivals=1, totalPassengersArr=1, totalPassengersDep=1}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=2, arrivals=1, totalPassengersArr=1, totalPassengersDep=2}
3> ResultData{borough='Queens', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=3, arrivals=1, totalPassengersArr=1, totalPassengersDep=3}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=3, arrivals=2, totalPassengersArr=2, totalPassengersDep=3}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=4, arrivals=2, totalPassengersArr=2, totalPassengersDep=5}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=5, arrivals=2, totalPassengersArr=2, totalPassengersDep=6}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=5, arrivals=3, totalPassengersArr=4, totalPassengersDep=6}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=5, arrivals=4, totalPassengersArr=5, totalPassengersDep=6}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=6, arrivals=4, totalPassengersArr=5, totalPassengersDep=8}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=6, arrivals=5, totalPassengersArr=7, totalPassengersDep=8}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=7, arrivals=5, totalPassengersArr=7, totalPassengersDep=9}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=8, arrivals=5, totalPassengersArr=7, totalPassengersDep=10}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=8, arrivals=6, totalPassengersArr=8, totalPassengersDep=10}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=8, arrivals=7, totalPassengersArr=9, totalPassengersDep=10}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=9, arrivals=7, totalPassengersArr=9, totalPassengersDep=11}
13> ResultData{borough='EWR', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=10, arrivals=7, totalPassengersArr=9, totalPassengersDep=17}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=10, arrivals=8, totalPassengersArr=15, totalPassengersDep=17}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=11, arrivals=8, totalPassengersArr=15, totalPassengersDep=22}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=11, arrivals=9, totalPassengersArr=20, totalPassengersDep=22}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=12, arrivals=9, totalPassengersArr=20, totalPassengersDep=24}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=12, arrivals=10, totalPassengersArr=22, totalPassengersDep=24}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=13, arrivals=10, totalPassengersArr=22, totalPassengersDep=29}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=13, arrivals=11, totalPassengersArr=27, totalPassengersDep=29}
3> ResultData{borough='Queens', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=1, arrivals=1, totalPassengersArr=1, totalPassengersDep=1}
14> ResultData{borough='Unknown', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=0, arrivals=1, totalPassengersArr=1, totalPassengersDep=0}
1> ResultData{borough='Manhattan', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=14, arrivals=11, totalPassengersArr=27, totalPassengersDep=34}
3> ResultData{borough='Queens', from=Wed Dec 31 01:00:00 CET 2008, to=Thu Jan 01 01:00:00 CET 2009, departures=2, arrivals=1, totalPassengersArr=1, totalPassengersDep=2}
```

The above piece is the output generated for ***A type***, printed to the console (in InteliJ).

The diffference between the two outputs can be seen easily.

Run the script (Program przetwarzający strumienie danych; skrypt uruchamiający)

`./run.sh`

Output

Create (Miejsce utrzymywania obrazów czasu rzeczywistego – skrypt tworzący)

Last part of initialization script takes care of setting up the Cassandra database. It is achived by utilizing the `Docker Compose` mechanism.

```
# docker-compose.yml
services:
  cassandra:
    image: cassandra:latest
    container_name: cassandra
    ports:
      - "9042:9042"
    environment:
      - CASSANDRA_USER=admin
      - CASSANDRA_PASSWORD=admin
    healthcheck:
      test: [ "CMD", "cqlsh", "-u cassandra", "-p cassandra" ,"-e describe
keyspaces" ]
      interval: 15s
      timeout: 10s
      retries: 10
```

```
# a part of setup.sh
docker exec -it cassandra cqlsh -e "CREATE KEYSPACE IF NOT EXISTS taxi_data WITH
replication = {'class': 'SimpleStrategy', 'replication_factor' : 1};
    USE taxi_data;
    CREATE TABLE IF NOT EXISTS taxi_events_sink
    (
        borough                TEXT,
        from_val               TEXT,
        to_val                 TEXT,
        departures             BIGINT,
        arrivals               BIGINT,
        totalPassengersArr     BIGINT,
        totalPassengersDep     BIGINT,
        PRIMARY KEY ((borough), from_val, totalPassengersArr, totalPassengersDep)
    );
    TRUNCATE taxi_data.taxi_events_sink;"
```

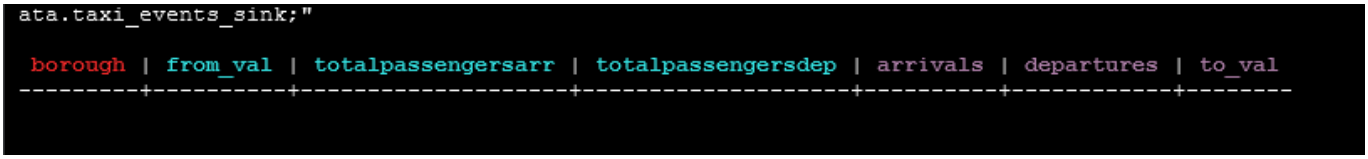
Characteristics (Miejsce utrzymywania obrazów czasu rzeczywistego – cechy)

Why Cassandra?

- available conncetor for Flink, making transfering of data simple
- scalability - NoSQL database, that can be easily expanded to match app requirements
- efficiency with writing data - perfect for processing data in real time and frequest data changes

Read (Konsument: skrypt odczytujący wyniki przetwarzania)

`show_output.sh` file contains command to show contents of Cassandra database as shown below:



Before running the Flink JAR file, the database contents should be empty as shown abouve.

Kafka sources

Kafka Sources are available with `getCassandraAggSink` in the `connectors.Connectors` file.

```
public static KafkaSource<TaxiEvent> getTaxiSource(ParameterTool properties) {
        return KafkaSource.<TaxiEvent>builder()
                .setBootstrapServers(properties.getRequired("BOOTSTRAP_SERVERS"))
                .setTopics(properties.getRequired("TAXI_INPUT_TOPIC"))
                .setGroupId(properties.getRequired("KAFKA_GROUP_ID"))

.setStartingOffsets(OffsetsInitializer.committedOffsets(OffsetResetStrategy.EARLIE
ST))
                .setDeserializer(new TaxiDeserializator())
                .build();
    }
```

TaxiDeserializarot class needed for Kafka input source to deserialize obtained data and convert it into `TaxiEvents` class considered in next steps of processing.

```
public class TaxiDeserializator implements
KafkaRecordDeserializationSchema<TaxiEvent> {

    @Override
    public void deserialize(ConsumerRecord<byte[], byte[]> consumerRecord,
Collector<TaxiEvent> collector) throws IOException {
        try {
            TaxiEvent taxiEvent = TaxiEvent.fromString( new
String(consumerRecord.value()));
            collector.collect(taxiEvent);
        } catch (ParseException e) {
            // Print malformed line to stderr
            System.err.println("Malformed line: " +
Arrays.toString(consumerRecord.value()));
        }
```

```
    }

    @Override
    public TypeInformation<TaxiEvent> getProducedType() {
        return TypeInformation.of(TaxiEvent.class);
    }
}
```

Kafka Output source for anomalies data.

```
public static KafkaSink<String> getAnomalySink(ParameterTool properties) {
        return KafkaSink.<String>builder()
                .setBootstrapServers(properties.getRequired("BOOTSTRAP_SERVERS"))
                .setRecordSerializer(KafkaRecordSerializationSchema.builder()
                        .setTopic(properties.get("ANOMALY_OUTPUT_TOPIC"))
                        .setValueSerializationSchema(new SimpleStringSchema())
                        .build()
                )
                .setDeliverGuarantee(DeliveryGuarantee.AT_LEAST_ONCE)
                .build();
    }
```