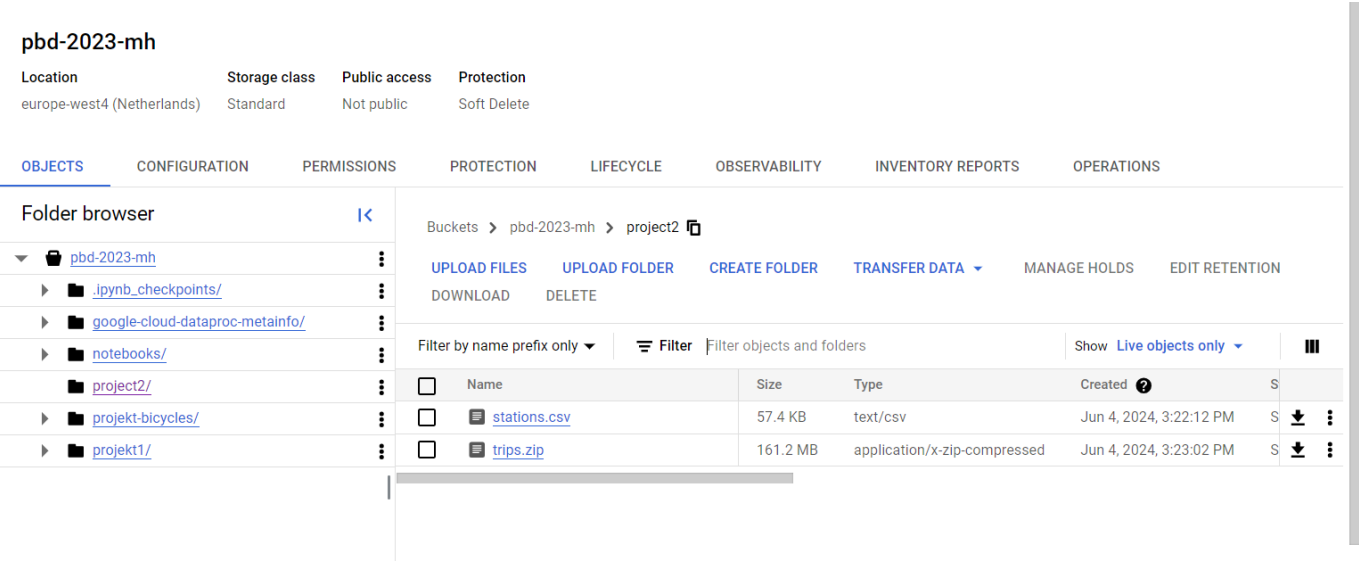


Instrukcja uruchomienia

Pobranie plików

- 1. Pobierz zbiór danych z zestawu 5 z ekursów.
- 2. Zmień nazwę pliku .csv na `stations.csv`, a pliku .zip na `trips.zip`.
- 3. Prześlij pliki do bucketa i umieść w katalogu `project2` tak, by ścieżki wyglądały następująco:
 - `gs://pbd-2023-mh/project2/stations.csv`
 - `gs://pbd-2023-mh/project2/trips.zip`



Uruchomienie klastra

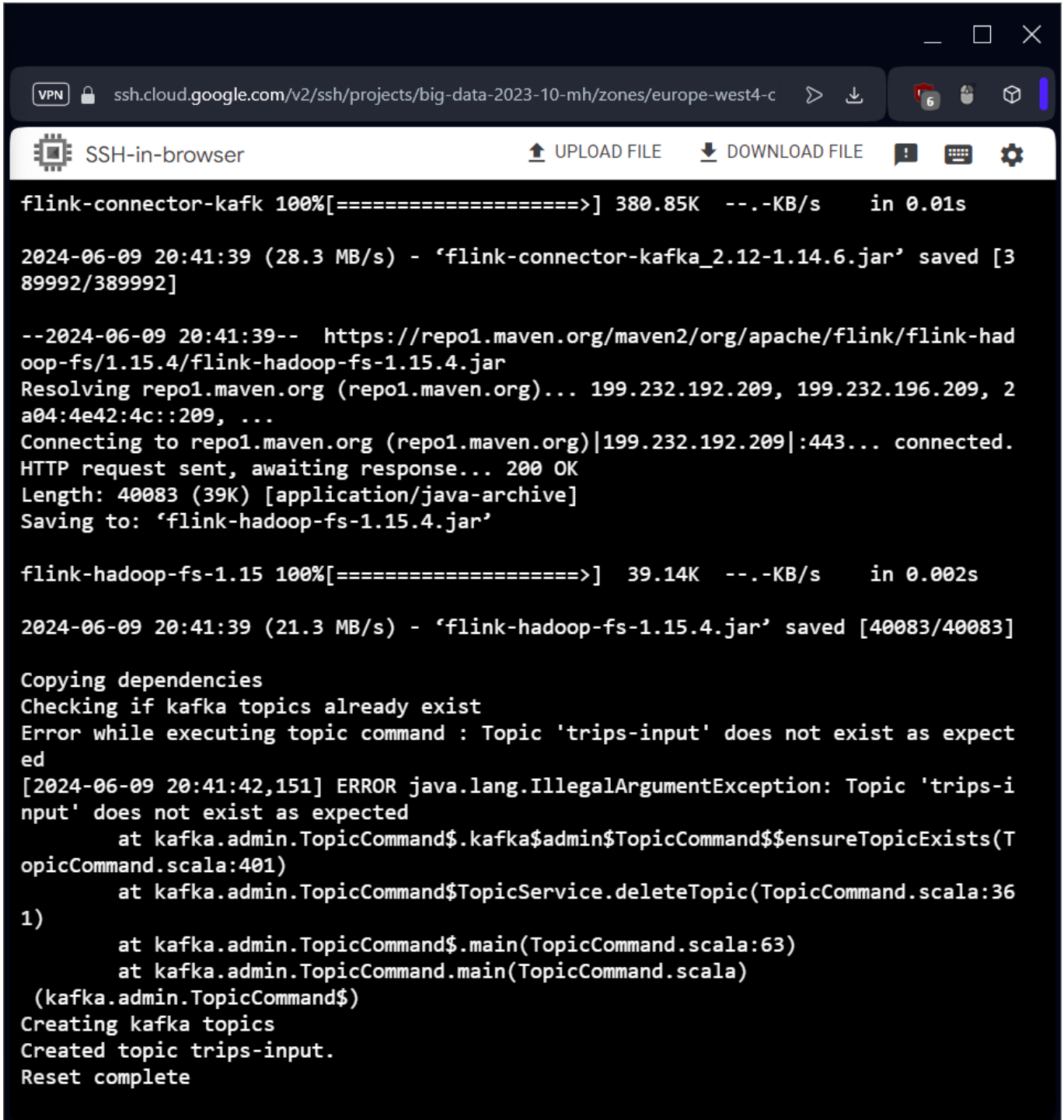
- 1. Uruchom klaster następującym poleceniem. Możesz zmienić czas działania klastra na inny niż 10h.

```
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 ${PROJECT_ID} --max-age=10h --metadata "run-on-
master=true" --initialization-actions gs://goog-dataproc-initialization-
actions-${REGION}/kafka/kafka.sh
```

Pobranie projektu

- 1. Pobierz pliki z ekursów lub sklonuj projekt z adresu XYZ bezpośrednio na klaster.
- 2. Przejdź do katalogu z projektem.
- 3. Nadaj uprawnienia dla plików wykonywalnych poleceniem `chmod +x *.sh`.
- 4. Edytuj plik ze zmiennymi np. poleceniem `nano vars.sh`. Podaj swoją nazwę bucketa.
- 5. Załaduj zmienne środowiskowe poleceniem `source vars.sh`.
 - jeśli jakieś polecenie nie wykonuje się prawidłowo, sprawdź czy zmienne zostały poprawnie załadowane. Spróbuj zrobić to ponownie.

6. Uruchom skrypt `reset.sh` w celu zresetowania środowiska oraz pobrania i zainstalowania wszystkich zależności. Skrypt:
 - kopiuje z Bucketa plik zip z przejazdami i rozpakowuje go
 - Pobiera dependencje z mavena i kopiuje je do katalogu flink/lub
 - Usuwa temat kafki dla strumienia wejściowego
 - Tworzy temat Kafki dla strumienia wejściowego
7. Upewnij się, że wszystkie pliki zostały pobrane i zainstalowane poprawnie.
- 8.



```
flink-connector-kafk 100%[=====] 380.85K --.-KB/s in 0.01s

2024-06-09 20:41:39 (28.3 MB/s) - 'flink-connector-kafka_2.12-1.14.6.jar' saved [389992/389992]

--2024-06-09 20:41:39-- https://repo1.maven.org/maven2/org/apache/flink/flink-hadoop-fs/1.15.4/flink-hadoop-fs-1.15.4.jar
Resolving repo1.maven.org (repo1.maven.org)... 199.232.192.209, 199.232.196.209, 2a04:4e42:4c::209, ...
Connecting to repo1.maven.org (repo1.maven.org)|199.232.192.209|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 40083 (39K) [application/java-archive]
Saving to: 'flink-hadoop-fs-1.15.4.jar'

flink-hadoop-fs-1.15 100%[=====] 39.14K --.-KB/s in 0.002s

2024-06-09 20:41:39 (21.3 MB/s) - 'flink-hadoop-fs-1.15.4.jar' saved [40083/40083]

Copying dependencies
Checking if kafka topics already exist
Error while executing topic command : Topic 'trips-input' does not exist as expected
[2024-06-09 20:41:42,151] ERROR java.lang.IllegalArgumentException: Topic 'trips-input' does not exist as expected
    at kafka.admin.TopicCommand$.kafka$admin$TopicCommand$$ensureTopicExists(TopicCommand.scala:401)
    at kafka.admin.TopicCommand$TopicService.deleteTopic(TopicCommand.scala:361)
    at kafka.admin.TopicCommand$.main(TopicCommand.scala:63)
    at kafka.admin.TopicCommand.main(TopicCommand.scala)
(kafka.admin.TopicCommand$)
Creating kafka topics
Created topic trips-input.
Reset complete
```

Uruchomienie bazy danych

1. Zjrzyj do pliku `run_docker.sh`.
2. Uruchom skrypt.

3. Skrypt powinien stworzyć kontener z bazą danych.
4. Sprawdź czy kontener działa poleceniem `docker ps`.
5. Wejdź do kontenera i sprawdź czy widzisz bazę danych oraz tabelę `aggsink`.

```
mariusz_hybiak_bigdata@pbd-cluster-m:~/bigdata$ ./run_docker.sh
d941b183a3b7ebb973d37acafeaebb5ecd13ff54c8584eea90e3fb76e319f2d3
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1396 (HY000) at line 1: Operation CREATE USER failed for 'streamuser'@'%'
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql: [Warning] Using a password on the command line interface can be insecure.
Tabela sink istnieje.
mariusz_hybiak_bigdata@pbd-cluster-m:~/bigdata$
```

Uruchomienie zasilania danymi strumienia

1. Zajrzyj do pliku `produce.sh`.
2. Uruchom skrypt (korzysta on z pliku `KafkaProducer.jar`).
 - kod źródłowy znajduje się w katalogu `KafkaProducer`.
3. Skrypt powinien wysyłać dane do tematu Kafka.

```
mariusz_hybiak_bigdata@pbd-cluster-m:~/bigdata$ ./produce.sh
Producing data to Kafka
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info
.
```

Uruchomienie aplikacji Flink

1. Wykorzystaj dostarczony `jar` lub skompiluj go samodzielnie.
2. Uruchom kompilację poleceniem `compile_flink.sh`. Plik `BicycleDataAnalysis.jar` powinien zostać przekopiowany do katalogu w którym jesteś.

```

SSH-in-browser
[↑] UPLOAD FILE [↓] DOWNLOAD FILE [!] [⌨] [⚙]

Downloaded from central: https://repo.maven.apache.org/maven2/org/iq80/snappy/snappy/0.4/snappy-0.4.jar (58 kB at 840 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interpolation/1.26/plexus-interpolation-1.26.jar (85 kB at 1.1 MB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/4.4.0/plexus-archiver-4.4.0.jar (211 kB at 2.4 MB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/tukaani/xz/1.9/xz-1.9.jar (116 kB at 1.2 MB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.4.2/plexus-utils-3.4.2.jar (267 kB at 2.5 MB/s)
[INFO] Building jar: /home/mariusz_hybiak_bigdata/bigdata/BicycleDataAnalysis/target/BicycleDataAnalysis.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 35.999 s
[INFO] Finished at: 2024-06-09T21:02:01Z
[INFO] -----
Copying jar to home directory
Done
mariusz_hybiak_bigdata@pbd-cluster-m:~/bigdata$ ls
BicycleDataAnalysis      flink-connector-jdbc-1.15.4.jar
BicycleDataAnalysis.jar  flink-connector-kafka-1.15.4.jar
KafkaProducer            flink-connector-kafka_2.12-1.14.6.jar
KafkaProducer.jar        flink-hadoop-fs-1.15.4.jar
README.md                mysql-connector-j-8.0.33.jar
bicycle-input             produce.sh
bicycle-input.zip         reset.sh
compile_flink.sh         run_docker.sh
consume_agg.sh            run_flink.sh
consume_anomaly.sh        run_flink_checkpoint.sh
create_docker.sh          tutorials
flink-connector-cassandra_2.12-1.15.4.jar  vars.sh
mariusz_hybiak_bigdata@pbd-cluster-m:~/bigdata$

```

3. Uruchom aplikację poleceniem `run_flink.sh`.
4. Zwróć uwagę na parametry aplikacji.
5. Dane po przetwarzaniu powinny zostać zapisane w bazie danych.

Konsumpcja wyników

1. Użyj skryptu `consume_agg.sh` do konsumpcji danych z bazy danych.

Informacje o przetwarzaniu

```

DataStream<Trip> tripsDS = env.fromSource(
    Connectors.getTripsSource(properties),
    WatermarkStrategy
        .<Trip>forBoundedOutOfOrderness(Duration.ofHours(1))
        .withTimestampAssigner((event, timestamp) ->
            event.getEventTime().toEpochSecond(ZoneOffset.UTC) * 1000),

```

```
        "Trips Source"  
    );
```

Odpowiada za wczytanie danych z tematu Kafki.

```
String path = properties.get(Parameters.STATION_INPUT_FILE);  
DataStream<Station> stations = env.readTextFile(path)  
    .map(a -> a.split(","))  
    .filter(a -> !a[0].equals("ID"))  
    .map(a -> new Station(  
        Integer.parseInt(a[0]),  
        a[1],  
        Integer.parseInt(a[2]),  
        Integer.parseInt(a[3]),  
        a[4],  
        a[5],  
        a[6],  
        a[7] + " " + a[8]  
    ));
```

Odpowiada za wczytanie danych z pliku CSV.

```
KeyedStream<Trip, Integer> keyedTrips = tripsDS.keyBy(Trip::getStationId);  
KeyedStream<Station, Integer> keyedStations =  
stations.keyBy(Station::getId);  
  
DataStream<TripStation> tripStationDS = keyedTrips  
    .join(keyedStations)  
    .where(Trip::getStationId)  
    .equalTo(Station::getId)  
    .window(TumblingProcessingTimeWindows.of(Time.seconds(10)))  
    .apply(new TripStationJoinFunction());  
  
DataStream<StationAggregate> aggOutput = tripStationDS  
    .map(StationAggregate::fromTripStation)  
    .keyBy(StationAggregate::getId)  
    .window(TumblingProcessingTimeWindows.of(Time.days(1)))  
    .reduce((a, b) -> new StationAggregate(  
        a.getId(),  
        a.getStart() + b.getStart(),  
        a.getStop() + b.getStop()  
    ));
```

Odpowiada za połączenie danych i ich przetwarzanie. W ramach stacji są zliczane przyjazdy i odjazdy rowerów. Wyniki są agregowane w oknach czasowych.

```
aggOutput.addSink(Connectors.getAggSink(properties));
```

Odpowiada za zapisanie wyników do bazy danych.

Jako ujście został wykorzystany temat Kafka:

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

Odpowiada za zapisanie wyników anomalii do tematu Kafka.

## Dodatkowe informacje

- W pliku `consume_anomaly.sh` znajduje się skrypt do konsumpcji danych dotyczących anomalii (wykrywania anomalii nie zaimplementowano).
- Udało mi się przetestować i potwierdzić, że program działa poprawnie do etapu połączenia dwóch strumieni i zapisania tego do bazy danych.