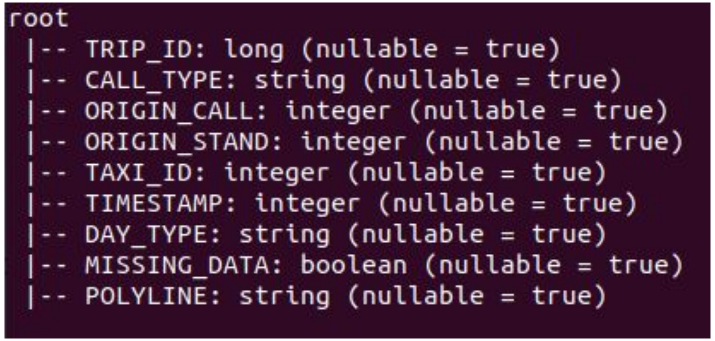
Big Data

Student:Teodora Stamenković 1460
Profesor: Dragan Stojanović

Dataset

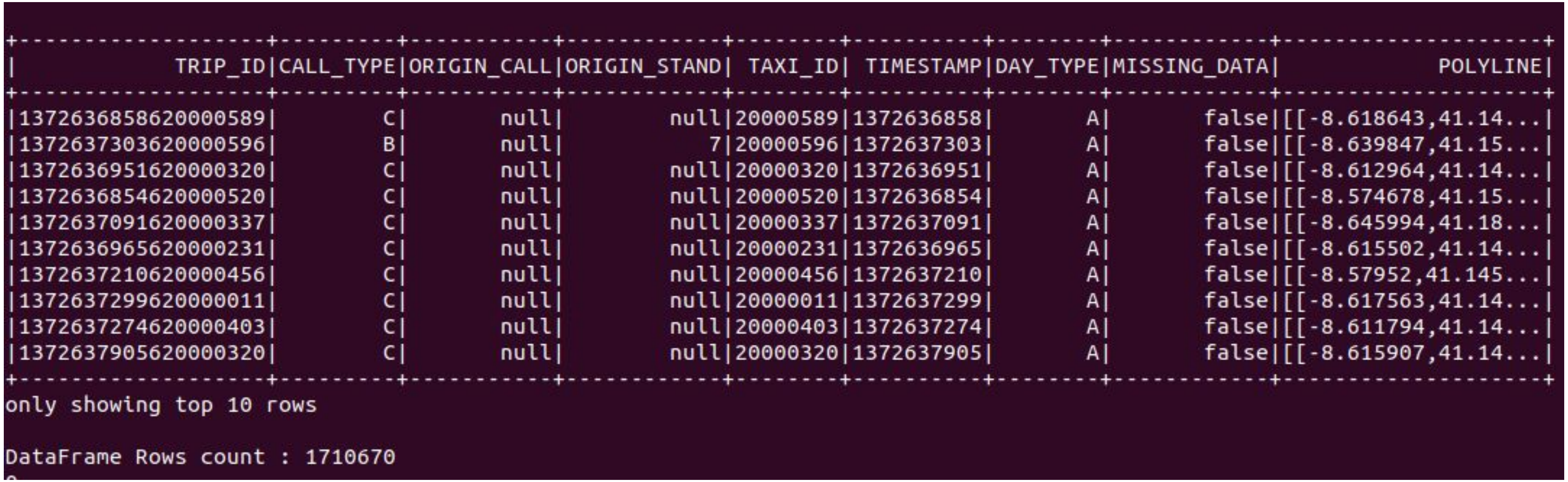
Putanje za 442 taksi vozila u Portu u periodu od 01.07.2013. do 30.06.2014.

- TRIP_ID
- CALL_TYPE:
    - A - poziv iz centrale
    - B - direktno kontaktiranje taksija na stajalištu
    - C - ostalo
- ORIGIN_CALL - identifikacija putnika ukoliko je CALL_TYPE='A'
- ORIGIN_STAND - identifikacija taksi stajališta ukoliko je CALL_TYPE='B'
- TAXI_ID - identifikacija taksi vozača
- TIMESTAMP - vreme početka
- DAY_TYPE
    - B - praznik
    - C - dan pre praznika
    - A - obican dan
- MISSING_DATA - da li nedostaju GPS podaci
- POLYLINE - lista parova (lat, lon) za svakih 15s putovanja



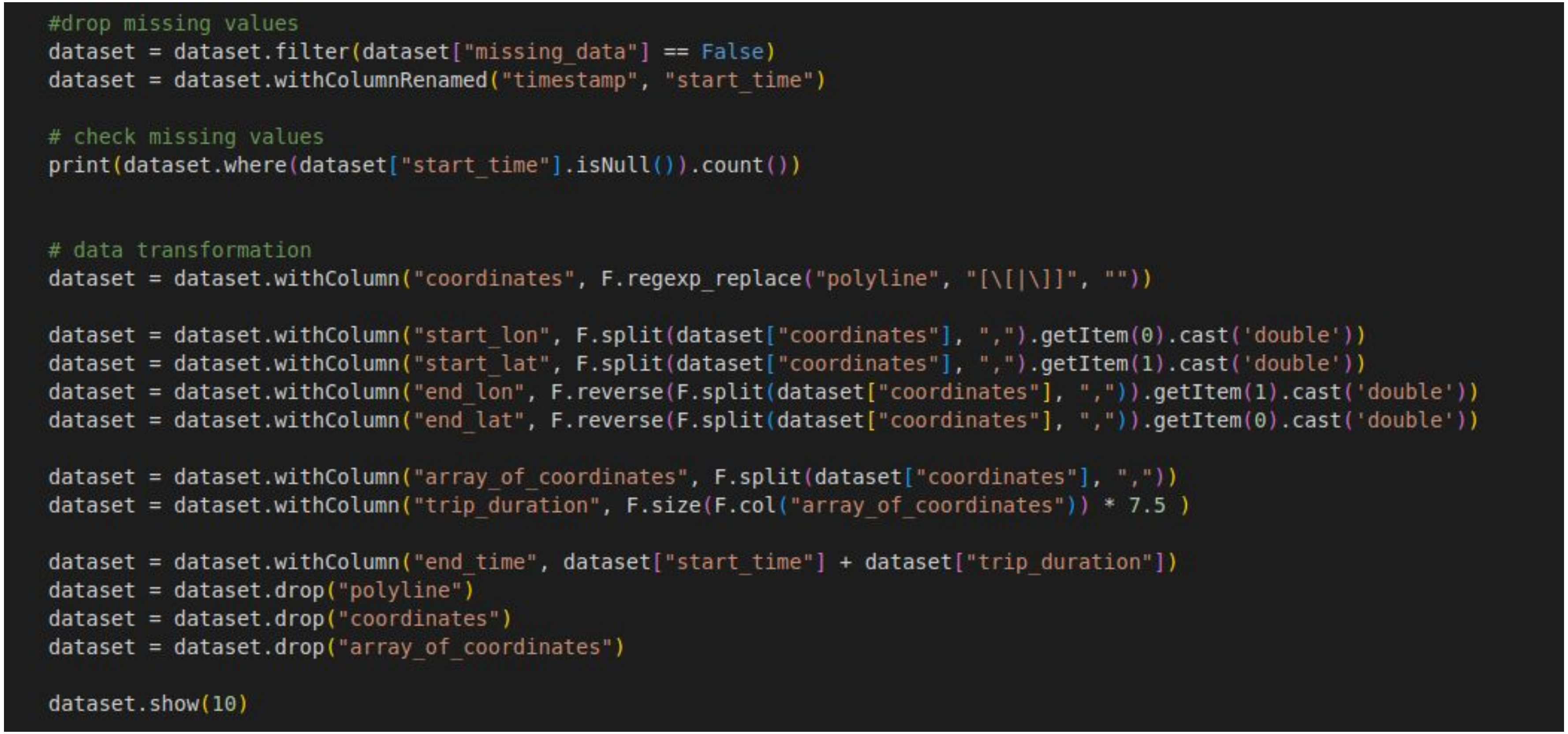
```
root
 |-- TRIP_ID: long (nullable = true)
 |-- CALL_TYPE: string (nullable = true)
 |-- ORIGIN_CALL: integer (nullable = true)
 |-- ORIGIN_STAND: integer (nullable = true)
 |-- TAXI_ID: integer (nullable = true)
 |-- TIMESTAMP: integer (nullable = true)
 |-- DAY_TYPE: string (nullable = true)
 |-- MISSING_DATA: boolean (nullable = true)
 |-- POLYLINE: string (nullable = true)
```

Prikaz podataka



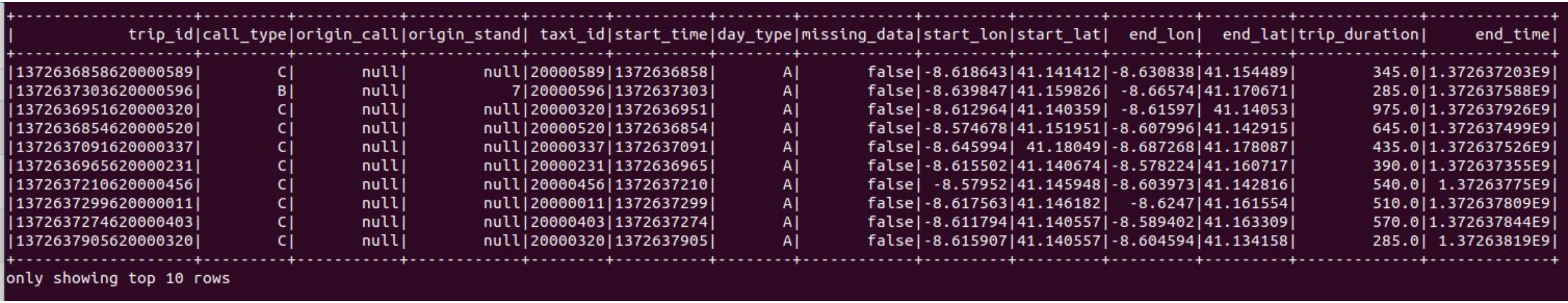
```
+-------------------+---------+-----------+------------+--------+----------+--------+------------+--------------------+
|            TRIP_ID|CALL_TYPE|ORIGIN_CALL|ORIGIN_STAND| TAXI_ID| TIMESTAMP|DAY_TYPE|MISSING_DATA|            POLYLINE|
+-------------------+---------+-----------+------------+--------+----------+--------+------------+--------------------+
|1372636858620000589|        C|       null|        null|20000589|1372636858|       A|       false|[[-8.618643,41.14...|
|1372637303620000596|        B|       null|           7|20000596|1372637303|       A|       false|[[-8.639847,41.15...|
|1372636951620000320|        C|       null|        null|20000320|1372636951|       A|       false|[[-8.612964,41.14...|
|1372636854620000520|        C|       null|        null|20000520|1372636854|       A|       false|[[-8.574678,41.15...|
|1372637091620000337|        C|       null|        null|20000337|1372637091|       A|       false|[[-8.645994,41.18...|
|1372636965620000231|        C|       null|        null|20000231|1372636965|       A|       false|[[-8.615502,41.14...|
|1372637210620000456|        C|       null|        null|20000456|1372637210|       A|       false|[[-8.57952,41.145...|
|1372637299620000011|        C|       null|        null|20000011|1372637299|       A|       false|[[-8.617563,41.14...|
|1372637274620000403|        C|       null|        null|20000403|1372637274|       A|       false|[[-8.611794,41.14...|
|1372637905620000320|        C|       null|        null|20000320|1372637905|       A|       false|[[-8.615907,41.14...|
+-------------------+---------+-----------+------------+--------+----------+--------+------------+--------------------+
only showing top 10 rows

DataFrame Rows count : 1710670
```

Transformacija podataka



```
#drop missing values
dataset = dataset.filter(dataset["missing_data"] == False)
dataset = dataset.withColumnRenamed("timestamp", "start_time")

# check missing values
print(dataset.where(dataset["start_time"].isNull()).count())


# data transformation
dataset = dataset.withColumn("coordinates", F.regexp_replace("polyline", "[\[|\]]", ""))

dataset = dataset.withColumn("start_lon", F.split(dataset["coordinates"], ",").getItem(0).cast('double'))
dataset = dataset.withColumn("start_lat", F.split(dataset["coordinates"], ",").getItem(1).cast('double'))
dataset = dataset.withColumn("end_lon", F.reverse(F.split(dataset["coordinates"], ",")).getItem(1).cast('double'))
dataset = dataset.withColumn("end_lat", F.reverse(F.split(dataset["coordinates"], ",")).getItem(0).cast('double'))

dataset = dataset.withColumn("array_of_coordinates", F.split(dataset["coordinates"], ","))
dataset = dataset.withColumn("trip_duration", F.size(F.col("array_of_coordinates")) * 7.5 )

dataset = dataset.withColumn("end_time", dataset["start_time"] + dataset["trip_duration"])
dataset = dataset.drop("polyline")
dataset = dataset.drop("coordinates")
dataset = dataset.drop("array_of_coordinates")

dataset.show(10)
```

Prikaz podataka nakon transformacija

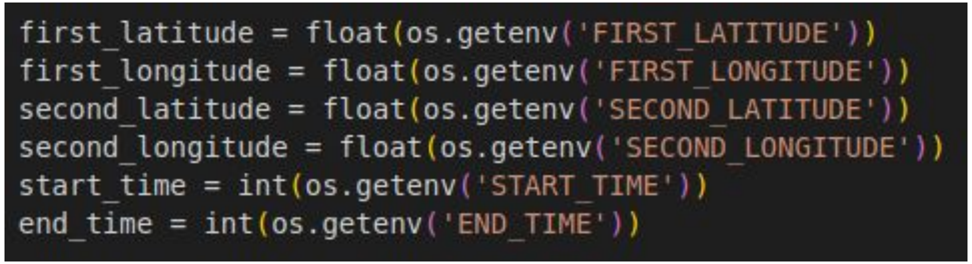


```
+-------------------+---------+-----------+------------+--------+----------+--------+------------+---------+---------+---------+---------+-------------+-------------+
|            trip_id|call_type|origin_call|origin_stand| taxi_id|start_time|day_type|missing_data|start_lon|start_lat|  end_lon|  end_lat|trip_duration|     end_time|
+-------------------+---------+-----------+------------+--------+----------+--------+------------+---------+---------+---------+---------+-------------+-------------+
|1372636858620000589|        C|       null|        null|20000589|1372636858|       A|       false|-8.618643|41.141412|-8.630838|41.154489|        345.0|1.372637203E9|
|1372637303620000596|        B|       null|           7|20000596|1372637303|       A|       false|-8.639847|41.159826| -8.66574|41.170671|        285.0|1.372637588E9|
|1372636951620000320|        C|       null|        null|20000320|1372636951|       A|       false|-8.612964|41.140359| -8.61597| 41.14053|        975.0|1.372637926E9|
|1372636854620000520|        C|       null|        null|20000520|1372636854|       A|       false|-8.574678|41.151951|-8.607996|41.142915|        645.0|1.372637499E9|
|1372637091620000337|        C|       null|        null|20000337|1372637091|       A|       false|-8.645994| 41.18049|-8.687268|41.178087|        435.0|1.372637526E9|
|1372636965620000231|        C|       null|        null|20000231|1372636965|       A|       false|-8.615502|41.140674|-8.578224|41.160717|        390.0|1.372637355E9|
|1372637210620000456|        C|       null|        null|20000456|1372637210|       A|       false| -8.57952|41.145948|-8.603973|41.142816|        540.0| 1.37263775E9|
|1372637299620000011|        C|       null|        null|20000011|1372637299|       A|       false|-8.617563|41.146182|  -8.6247|41.161554|        510.0|1.372637809E9|
|1372637274620000403|        C|       null|        null|20000403|1372637274|       A|       false|-8.611794|41.140557|-8.589402|41.163309|        570.0|1.372637844E9|
|1372637905620000320|        C|       null|        null|20000320|1372637905|       A|       false|-8.615907|41.140557|-8.604594|41.134158|        285.0| 1.37263819E9|
+-------------------+---------+-----------+------------+--------+----------+--------+------------+---------+---------+---------+---------+-------------+-------------+
only showing top 10 rows
```

Projekat 1

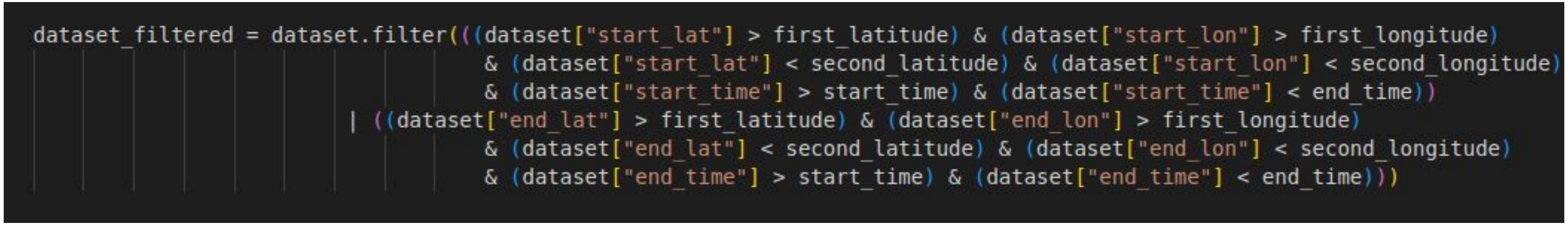
- Apache Spark aplikacija
- HDFS

Filtriranje podataka



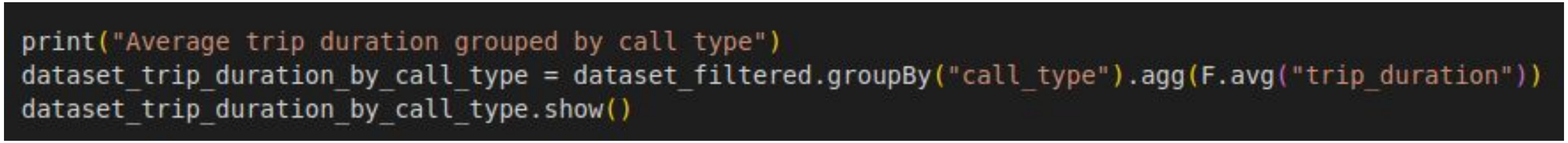
```
first_latitude = float(os.getenv('FIRST_LATITUDE'))
first_longitude = float(os.getenv('FIRST_LONGITUDE'))
second_latitude = float(os.getenv('SECOND_LATITUDE'))
second_longitude = float(os.getenv('SECOND_LONGITUDE'))
start_time = int(os.getenv('START_TIME'))
end_time = int(os.getenv('END_TIME'))
```

Podaci se filtriraju na osnovu ulaznih podataka (opseg koordinata i vremenski period)

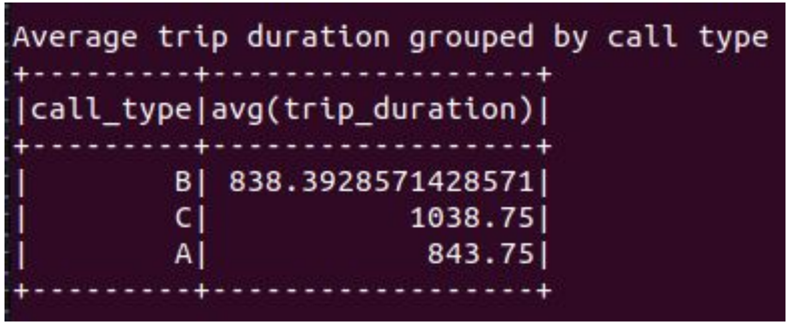


```
dataset_filtered = dataset.filter((((dataset["start_lat"] > first_latitude) & (dataset["start_lon"] > first_longitude)
                                    & (dataset["start_lat"] < second_latitude) & (dataset["start_lon"] < second_longitude)
                                    & (dataset["start_time"] > start_time) & (dataset["start_time"] < end_time))
                         | ((dataset["end_lat"] > first_latitude) & (dataset["end_lon"] > first_longitude)
                                    & (dataset["end_lat"] < second_latitude) & (dataset["end_lon"] < second_longitude)
                                    & (dataset["end_time"] > start_time) & (dataset["end_time"] < end_time)))
```

Analiza podataka

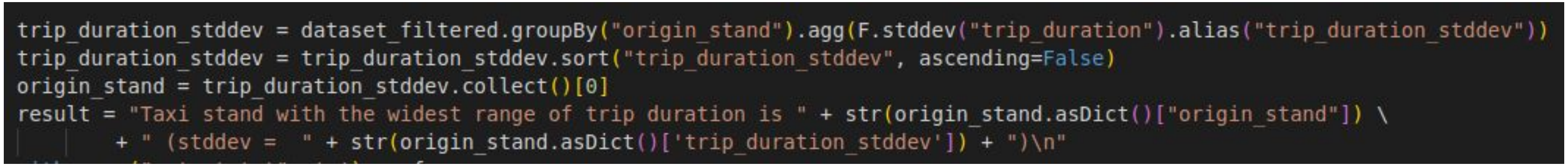


```
print("Average trip duration grouped by call type")
dataset_trip_duration_by_call_type = dataset_filtered.groupBy("call_type").agg(F.avg("trip_duration"))
dataset_trip_duration_by_call_type.show()
```



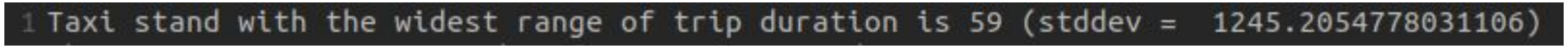
```
Average trip duration grouped by call type
+---------+------------------+
|call_type|avg(trip_duration)|
+---------+------------------+
|        B| 838.3928571428571|
|        C|           1038.75|
|        A|            843.75|
+---------+------------------+
```

Analiza podataka



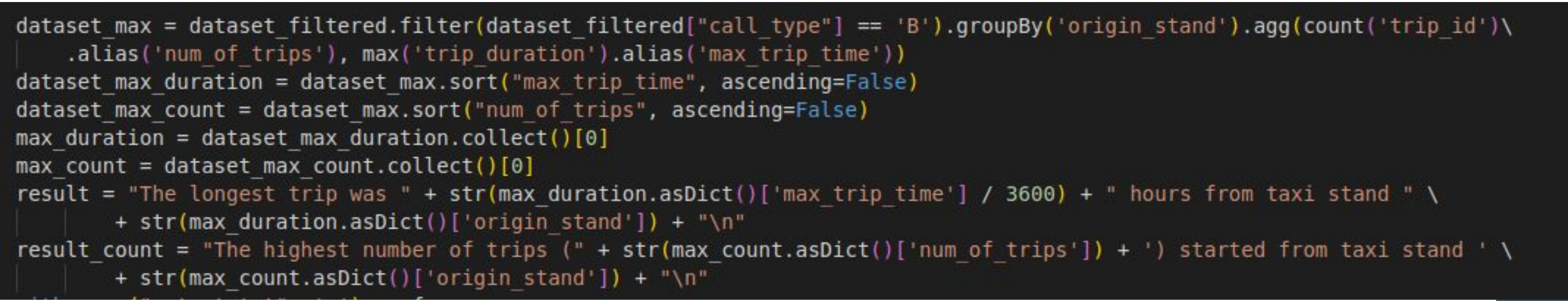
```
trip_duration_stddev = dataset_filtered.groupBy("origin_stand").agg(F.stddev("trip_duration").alias("trip_duration_stddev"))
trip_duration_stddev = trip_duration_stddev.sort("trip_duration_stddev", ascending=False)
origin_stand = trip_duration_stddev.collect()[0]
result = "Taxi stand with the widest range of trip duration is " + str(origin_stand.asDict()["origin_stand"]) \
        + " (stddev =  " + str(origin_stand.asDict()['trip_duration_stddev']) + ")\n"
```

Output:



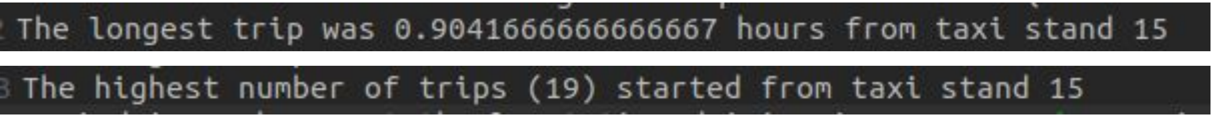
```
Taxi stand with the widest range of trip duration is 59 (stddev =  1245.2054778031106)
```

Analiza podataka



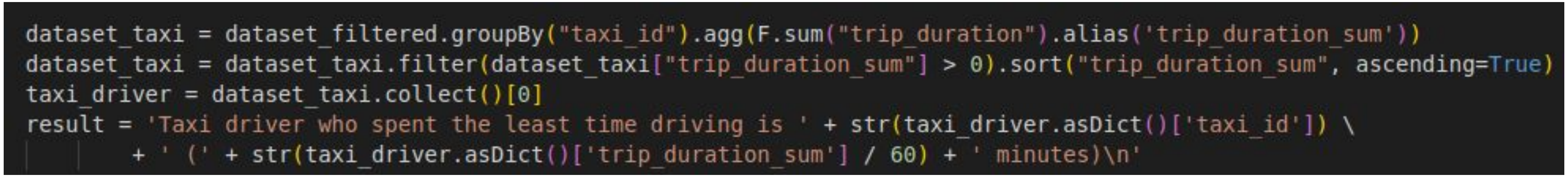
```python
dataset_max = dataset_filtered.filter(dataset_filtered["call_type"] == 'B').groupBy('origin_stand').agg(count('trip_id')\
    .alias('num_of_trips'), max('trip_duration').alias('max_trip_time'))
dataset_max_duration = dataset_max.sort("max_trip_time", ascending=False)
dataset_max_count = dataset_max.sort("num_of_trips", ascending=False)
max_duration = dataset_max_duration.collect()[0]
max_count = dataset_max_count.collect()[0]
result = "The longest trip was " + str(max_duration.asDict()['max_trip_time'] / 3600) + " hours from taxi stand " \
        + str(max_duration.asDict()['origin_stand']) + "\n"
result_count = "The highest number of trips (" + str(max_count.asDict()['num_of_trips']) + ') started from taxi stand ' \
        + str(max_count.asDict()['origin_stand']) + "\n"
```

Output:



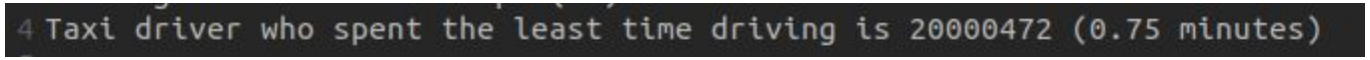
```
The longest trip was 0.9041666666666667 hours from taxi stand 15
The highest number of trips (19) started from taxi stand 15
```

Analiza podataka



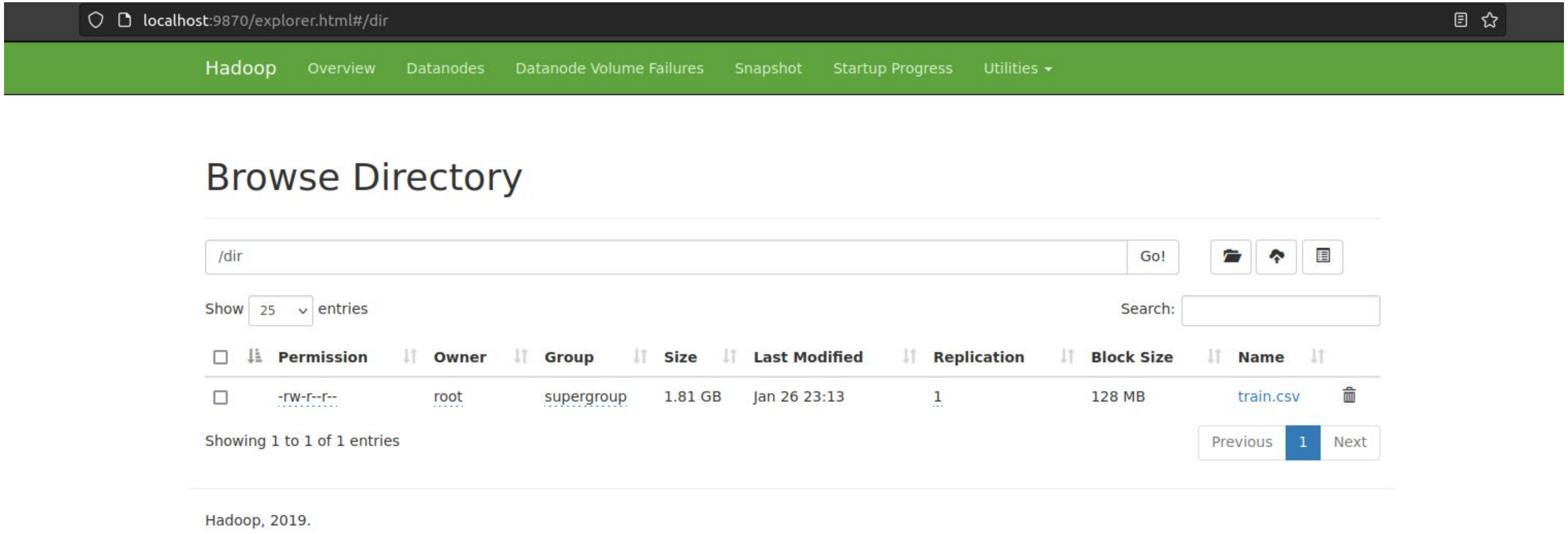
```python
dataset_taxi = dataset_filtered.groupBy("taxi_id").agg(F.sum("trip_duration").alias('trip_duration_sum'))
dataset_taxi = dataset_taxi.filter(dataset_taxi["trip_duration_sum"] > 0).sort("trip_duration_sum", ascending=True)
taxi_driver = dataset_taxi.collect()[0]
result = 'Taxi driver who spent the least time driving is ' + str(taxi_driver.asDict()['taxi_id']) \
        + ' (' + str(taxi_driver.asDict()['trip_duration_sum'] / 60) + ' minutes)\n'
```

Output:



```
Taxi driver who spent the least time driving is 20000472 (0.75 minutes)
```

Čuvanje podataka na HDFS



Browse Directory

/dir

Show 25 entries

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	1.81 GB	Jan 26 23:13	1	128 MB	train.csv

Showing 1 to 1 of 1 entries

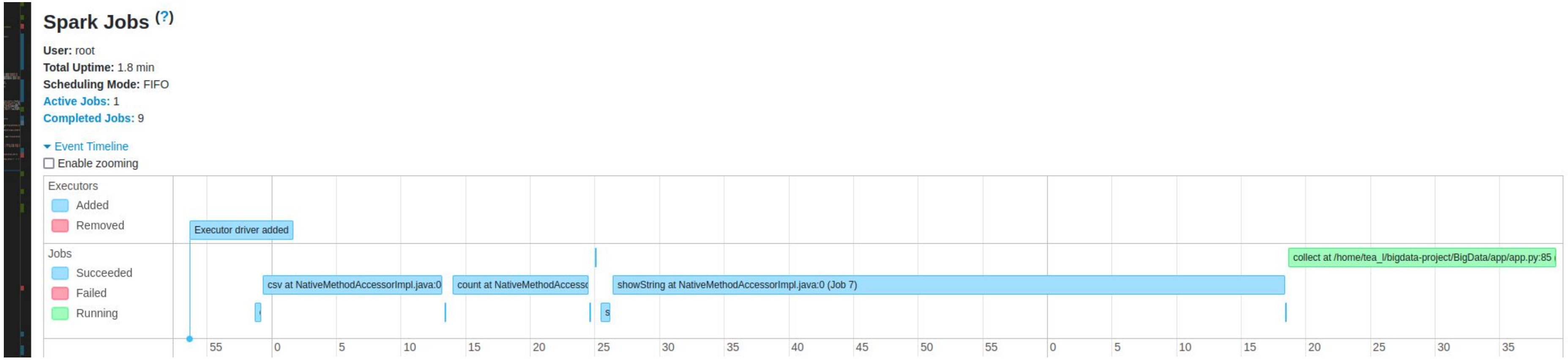
Previous 1 Next

Hadoop, 2019.

Lokalno izvršenje

```
input = "hdfs://localhost:9000/dir/train.csv"
spark = SparkSession.builder.appName(appName).master("local[2]").getOrCreate()
```

```
dataset = spark.read.option("inferSchema", True).option("header", True).csv(input)
```



Spark Jobs (?)

User: root
Total Uptime: 1.8 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 9

▾ Event Timeline
☐ Enable zooming

Executors
🟦 Added
🟥 Removed

Executor driver added

Jobs
🟦 Succeeded
🟥 Failed
🟩 Running

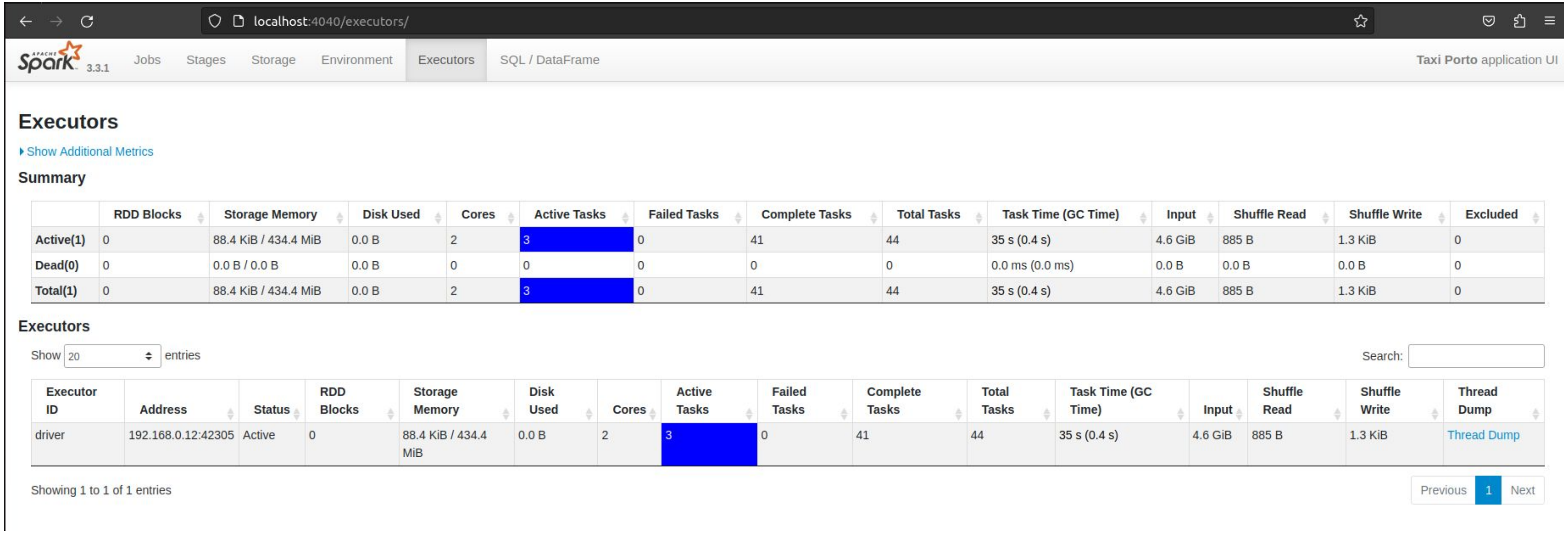
collect at /home/tea_l/bigdata-project/BigData/app/app.py:85

csv at NativeMethodAccessorImpl.java:0

count at NativeMethodAccess

showString at NativeMethodAccessorImpl.java:0 (Job 7)

Lokalno izvršenje - executors



Executors

▸ Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(1)	0	88.4 KiB / 434.4 MiB	0.0 B	2	3	0	41	44	35 s (0.4 s)	4.6 GiB	885 B	1.3 KiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	88.4 KiB / 434.4 MiB	0.0 B	2	3	0	41	44	35 s (0.4 s)	4.6 GiB	885 B	1.3 KiB	0

Executors

Show 20 entries

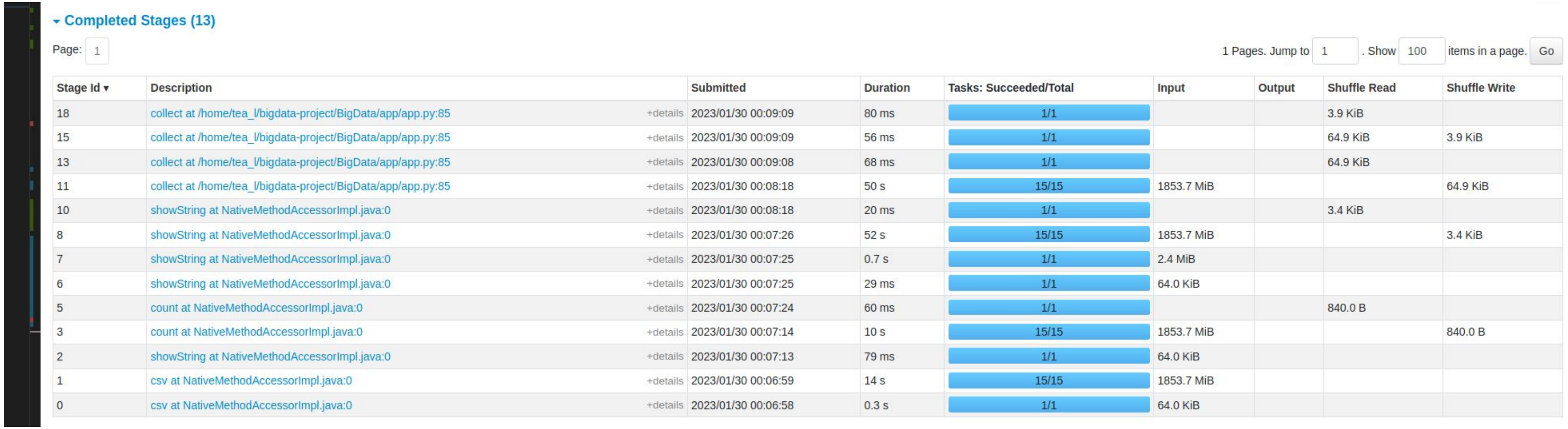
Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump
driver	192.168.0.12:42305	Active	0	88.4 KiB / 434.4 MiB	0.0 B	2	3	0	41	44	35 s (0.4 s)	4.6 GiB	885 B	1.3 KiB	Thread Dump

Showing 1 to 1 of 1 entries

Previous 1 Next

Lokalno izvršenje - stages



▾ Completed Stages (13)

Page: 1

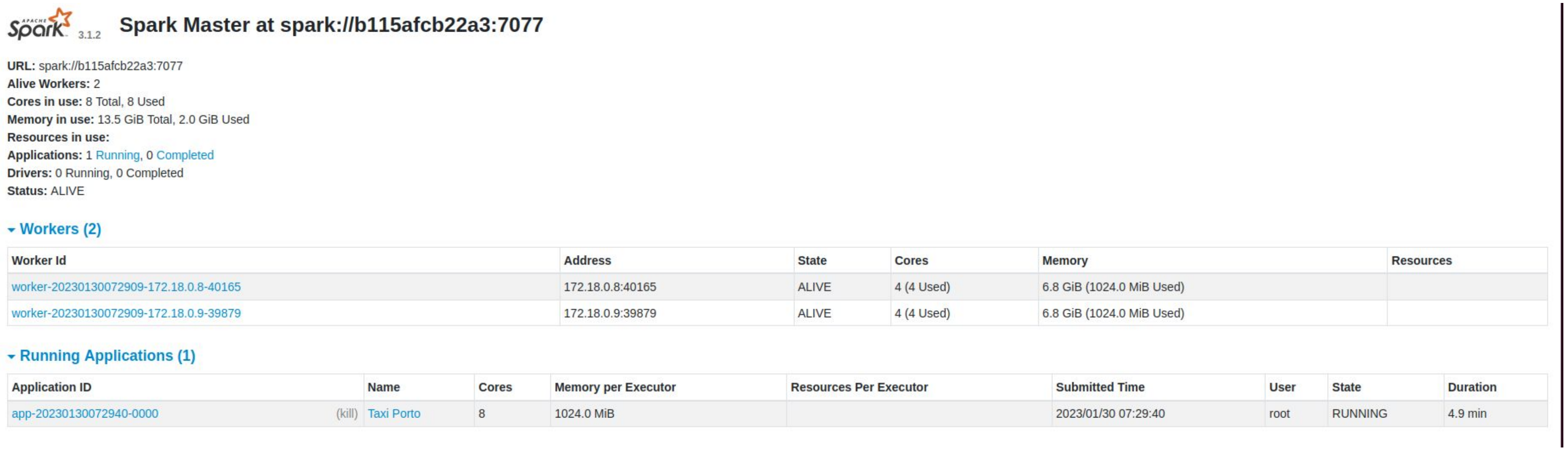
1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
18	collect at /home/tea_l/bigdata-project/BigData/app/app.py:85	+details	2023/01/30 00:09:09	80 ms	1/1			3.9 KiB	
15	collect at /home/tea_l/bigdata-project/BigData/app/app.py:85	+details	2023/01/30 00:09:09	56 ms	1/1			64.9 KiB	3.9 KiB
13	collect at /home/tea_l/bigdata-project/BigData/app/app.py:85	+details	2023/01/30 00:09:08	68 ms	1/1			64.9 KiB	
11	collect at /home/tea_l/bigdata-project/BigData/app/app.py:85	+details	2023/01/30 00:08:18	50 s	15/15	1853.7 MiB			64.9 KiB
10	showString at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:08:18	20 ms	1/1			3.4 KiB	
8	showString at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:07:26	52 s	15/15	1853.7 MiB			3.4 KiB
7	showString at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:07:25	0.7 s	1/1	2.4 MiB			
6	showString at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:07:25	29 ms	1/1	64.0 KiB			
5	count at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:07:24	60 ms	1/1			840.0 B	
3	count at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:07:14	10 s	15/15	1853.7 MiB			840.0 B
2	showString at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:07:13	79 ms	1/1	64.0 KiB			
1	csv at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:06:59	14 s	15/15	1853.7 MiB			
0	csv at NativeMethodAccessorImpl.java:0	+details	2023/01/30 00:06:58	0.3 s	1/1	64.0 KiB			

Izvršenje na klasteru

```
input = "hdfs://namenode:9000/dir/train.csv"
spark = SparkSession.builder.appName(appName).master("spark://spark-master:7077").getOrCreate()
```

```
dataset = spark.read.option("inferSchema", True).option("header", True).csv(input)
```



3.1.2 **Spark Master at spark://b115afcb22a3:7077**

URL: spark://b115afcb22a3:7077
Alive Workers: 2
Cores in use: 8 Total, 8 Used
Memory in use: 13.5 GiB Total, 2.0 GiB Used
Resources in use:
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

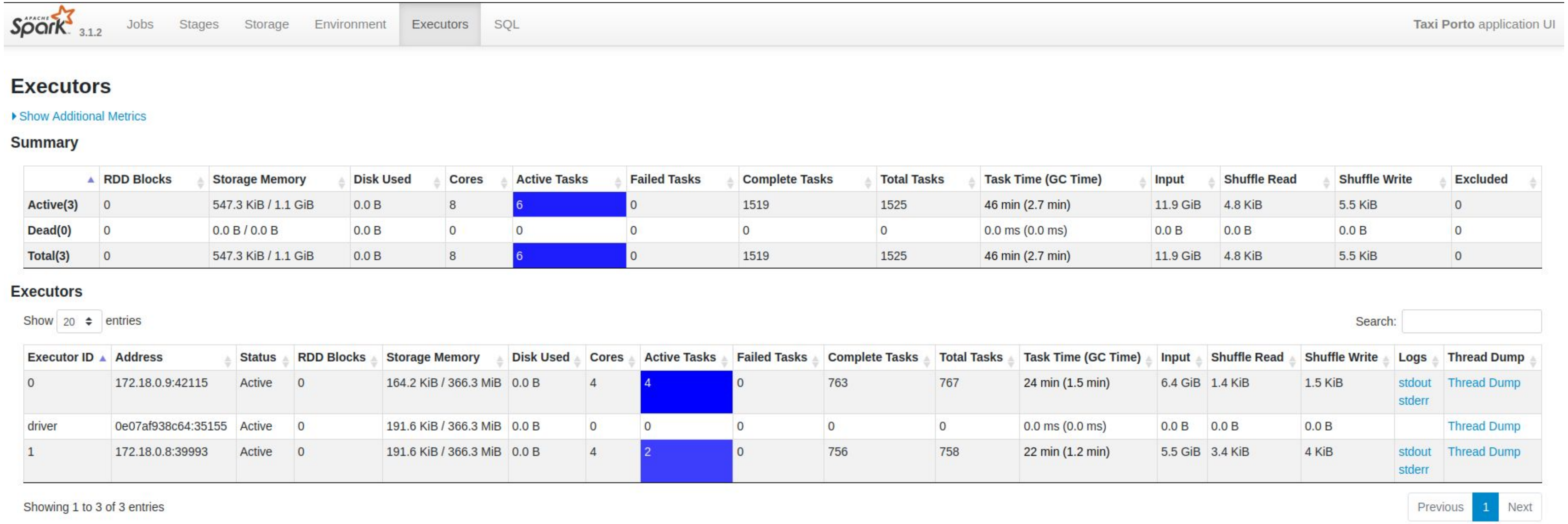
Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20230130072909-172.18.0.8-40165	172.18.0.8:40165	ALIVE	4 (4 Used)	6.8 GiB (1024.0 MiB Used)	
worker-20230130072909-172.18.0.9-39879	172.18.0.9:39879	ALIVE	4 (4 Used)	6.8 GiB (1024.0 MiB Used)	

Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20230130072940-0000 (kill)	Taxi Porto	8	1024.0 MiB		2023/01/30 07:29:40	root	RUNNING	4.9 min

Izvršenje na klasteru



Executors

Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(3)	0	547.3 KiB / 1.1 GiB	0.0 B	8	6	0	1519	1525	46 min (2.7 min)	11.9 GiB	4.8 KiB	5.5 KiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(3)	0	547.3 KiB / 1.1 GiB	0.0 B	8	6	0	1519	1525	46 min (2.7 min)	11.9 GiB	4.8 KiB	5.5 KiB	0

Executors

Show 20 entries

Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
0	172.18.0.9:42115	Active	0	164.2 KiB / 366.3 MiB	0.0 B	4	4	0	763	767	24 min (1.5 min)	6.4 GiB	1.4 KiB	1.5 KiB	stdout stderr	Thread Dump
driver	0e07af938c64:35155	Active	0	191.6 KiB / 366.3 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	172.18.0.8:39993	Active	0	191.6 KiB / 366.3 MiB	0.0 B	4	2	0	756	758	22 min (1.2 min)	5.5 GiB	3.4 KiB	4 KiB	stdout stderr	Thread Dump

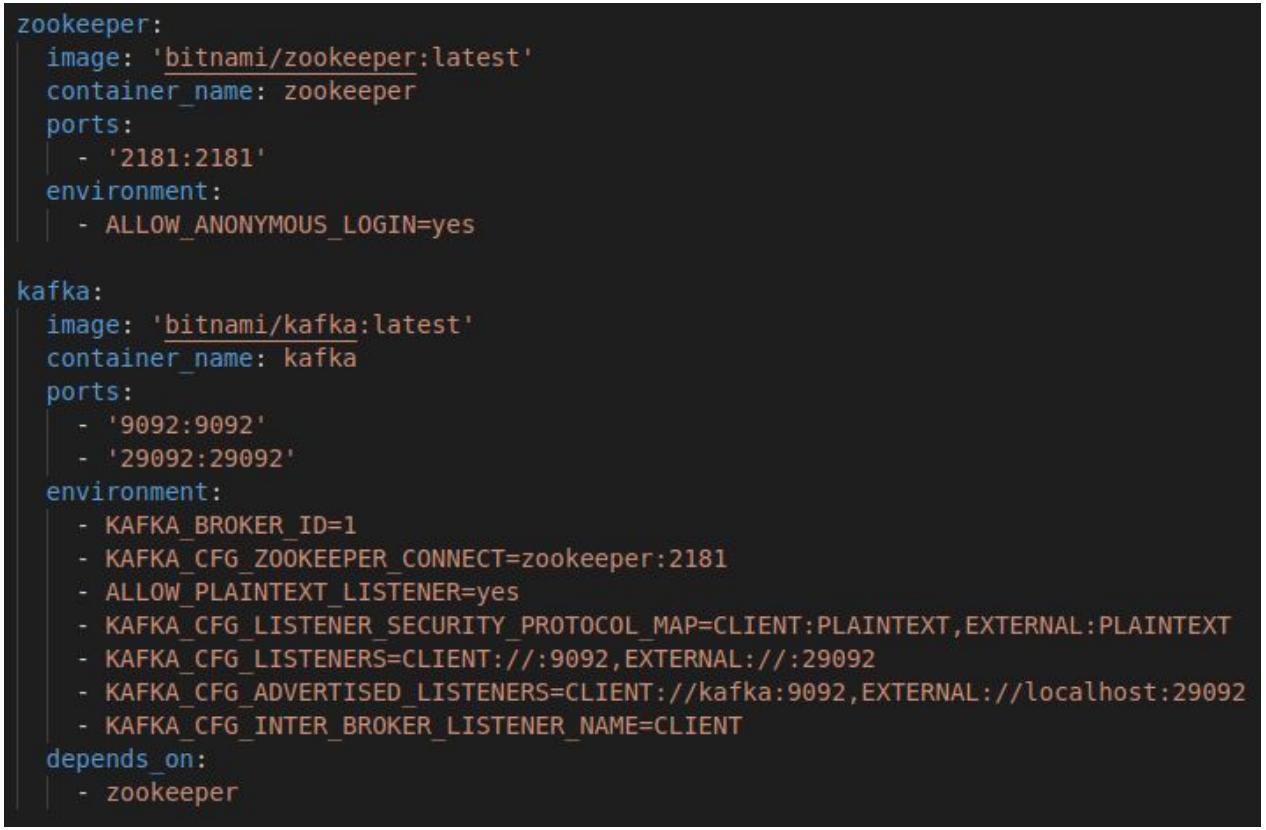
Showing 1 to 3 of 3 entries

Previous 1 Next

Projekat 2

- Kafka producer
- Apache Spark
- Apache Flink

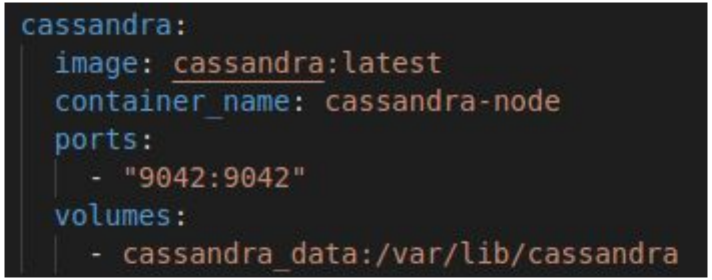
Kafka



```
zookeeper:
  image: 'bitnami/zookeeper:latest'
  container_name: zookeeper
  ports:
    - '2181:2181'
  environment:
    - ALLOW_ANONYMOUS_LOGIN=yes

kafka:
  image: 'bitnami/kafka:latest'
  container_name: kafka
  ports:
    - '9092:9092'
    - '29092:29092'
  environment:
    - KAFKA_BROKER_ID=1
    - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
    - ALLOW_PLAINTEXT_LISTENER=yes
    - KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=CLIENT:PLAINTEXT,EXTERNAL:PLAINTEXT
    - KAFKA_CFG_LISTENERS=CLIENT://:9092,EXTERNAL://:29092
    - KAFKA_CFG_ADVERTISED_LISTENERS=CLIENT://kafka:9092,EXTERNAL://localhost:29092
    - KAFKA_CFG_INTER_BROKER_LISTENER_NAME=CLIENT
  depends_on:
    - zookeeper
```

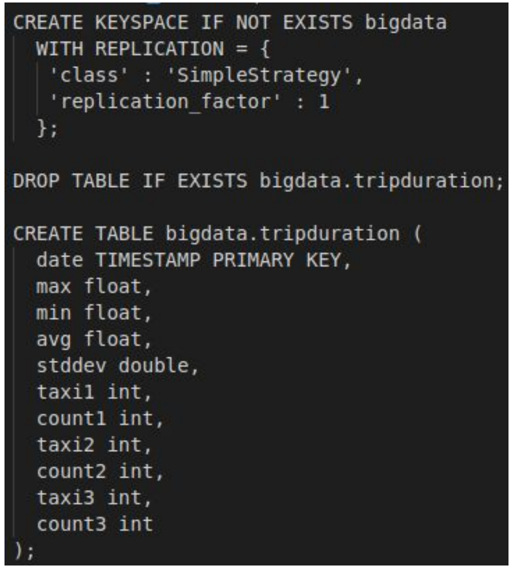
- Klijenti unutar Docker mreže:
  listener: CLIENT
  port: 9092
  host: kafka
- Klijenti van Docker mreže:
  listener: EXTERNAL
  port: 29092
  host: localhost

Cassandra

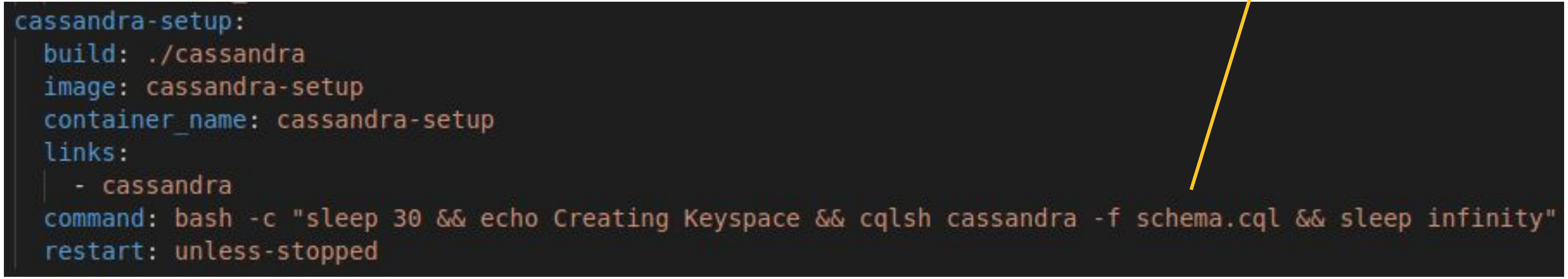


```yaml
cassandra:
  image: cassandra:latest
  container_name: cassandra-node
  ports:
    - "9042:9042"
  volumes:
    - cassandra_data:/var/lib/cassandra
```

schema.cql



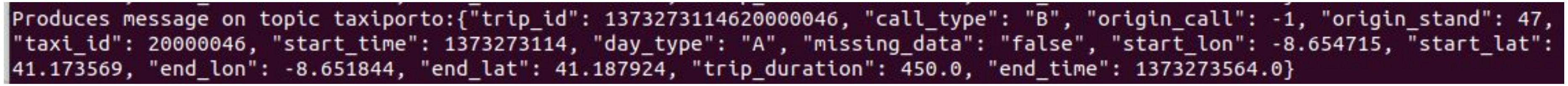
```sql
CREATE KEYSPACE IF NOT EXISTS bigdata
  WITH REPLICATION = {
    'class' : 'SimpleStrategy',
    'replication_factor' : 1
  };

DROP TABLE IF EXISTS bigdata.tripduration;

CREATE TABLE bigdata.tripduration (
  date TIMESTAMP PRIMARY KEY,
  max float,
  min float,
  avg float,
  stddev double,
  taxi1 int,
  count1 int,
  taxi2 int,
  count2 int,
  taxi3 int,
  count3 int
);
```



```yaml
cassandra-setup:
  build: ./cassandra
  image: cassandra-setup
  container_name: cassandra-setup
  links:
    - cassandra
  command: bash -c "sleep 30 && echo Creating Keyspace && cqlsh cassandra -f schema.cql && sleep infinity"
  restart: unless-stopped
```

Kafka producer

- Python aplikacija
- Lokalno izvršenje
- Čitanje iz *.csv* datoteke i slanje na Kafka topic

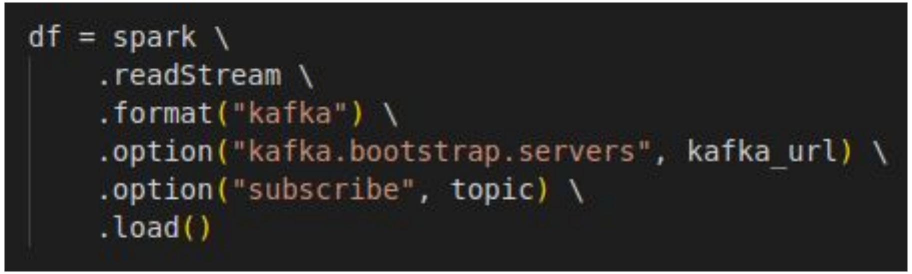


```
Produces message on topic taxiporto:{"trip_id": 1373273114620000046, "call_type": "B", "origin_call": -1, "origin_stand": 47,
"taxi_id": 20000046, "start_time": 1373273114, "day_type": "A", "missing_data": "false", "start_lon": -8.654715, "start_lat":
41.173569, "end_lon": -8.651844, "end_lat": 41.187924, "trip_duration": 450.0, "end_time": 1373273564.0}
```

Apache Spark

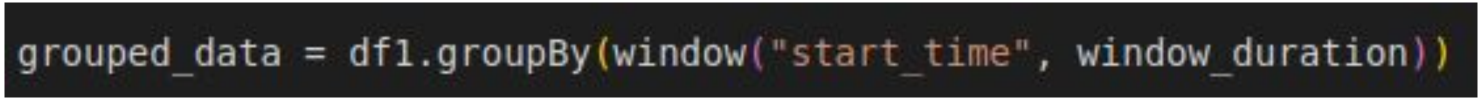
- Structured streaming - *micro-batch* obrada tokova podataka

Izvor podataka: **Kafka**



```
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", kafka_url) \
    .option("subscribe", topic) \
    .load()
```

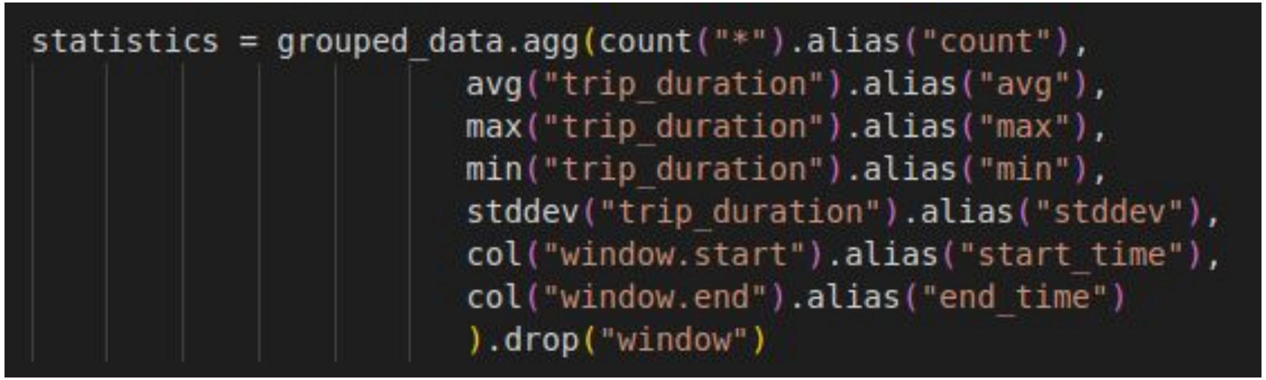
Podaci pročitani sa Kafka topic-a se obrađuju u određenom vremenskom prozoru (1h)



```
grouped_data = df1.groupBy(window("start_time", window_duration))
```

Apache Spark

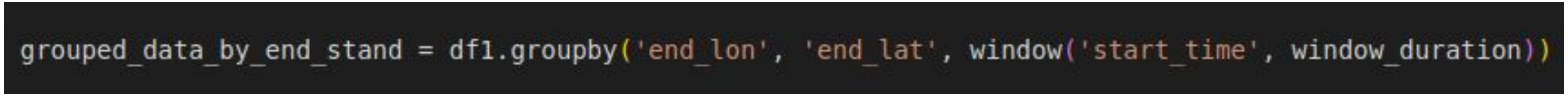
Broj taksi vožnji, maksimalna, minimalna, srednja vrednost i standardna devijacija dužine vožnje:



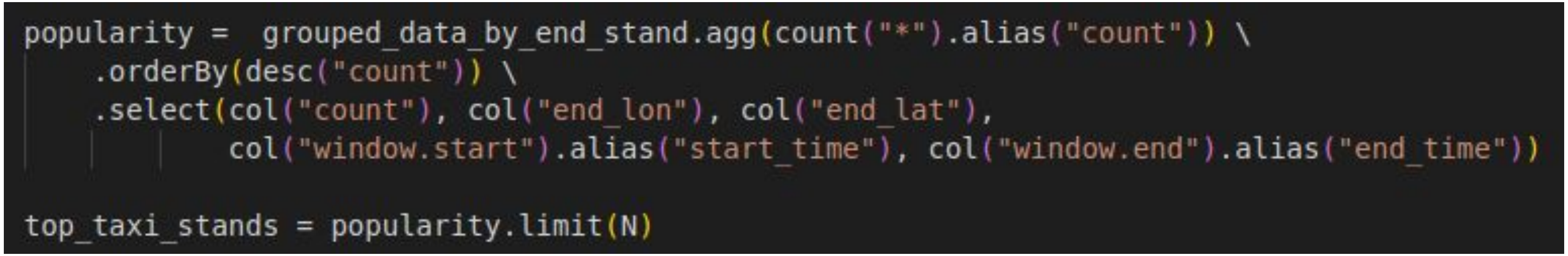
```
statistics = grouped_data.agg(count("*").alias("count"),
                              avg("trip_duration").alias("avg"),
                              max("trip_duration").alias("max"),
                              min("trip_duration").alias("min"),
                              stddev("trip_duration").alias("stddev"),
                              col("window.start").alias("start_time"),
                              col("window.end").alias("end_time")
                              ).drop("window")
```

Apache Spark

N najpopularnijih destinacija:



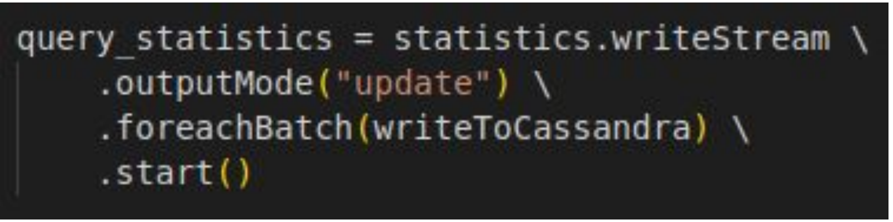
```
grouped_data_by_end_stand = df1.groupby('end_lon', 'end_lat', window('start_time', window_duration))
```



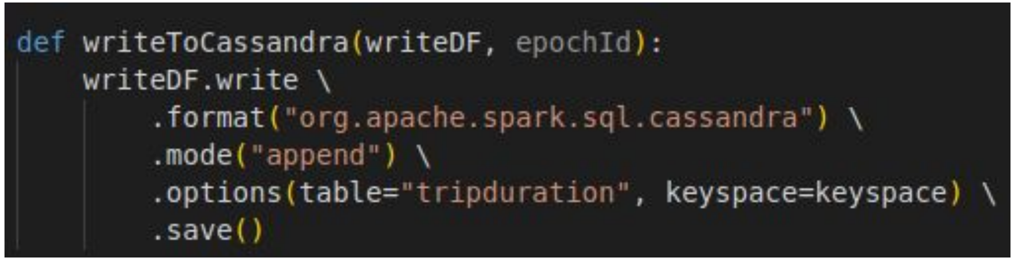
```
popularity =  grouped_data_by_end_stand.agg(count("*").alias("count")) \
    .orderBy(desc("count")) \
    .select(col("count"), col("end_lon"), col("end_lat"),
            col("window.start").alias("start_time"), col("window.end").alias("end_time"))

top_taxi_stands = popularity.limit(N)
```

Apache Spark

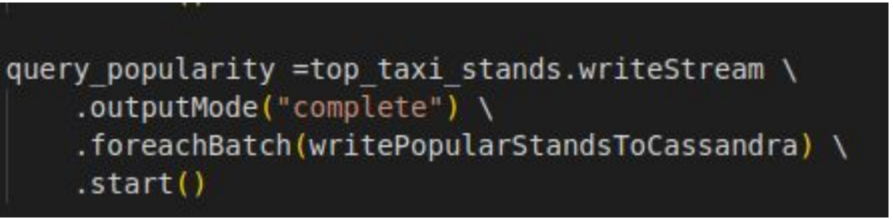
Rezultat analize se upisuje u Cassandra bazu



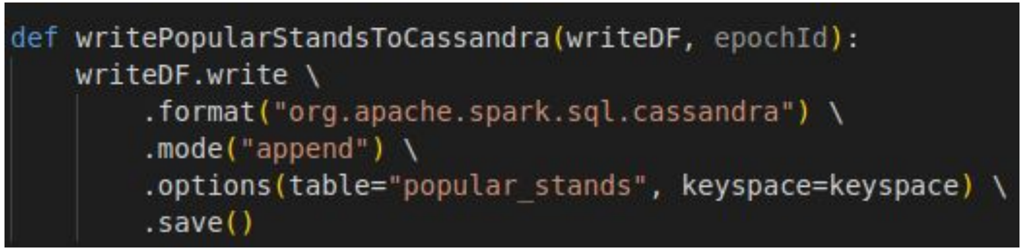
```
query_statistics = statistics.writeStream \
    .outputMode("update") \
    .foreachBatch(writeToCassandra) \
    .start()
```



```
def writeToCassandra(writeDF, epochId):
    writeDF.write \
        .format("org.apache.spark.sql.cassandra") \
        .mode("append") \
        .options(table="tripduration", keyspace=keyspace) \
        .save()
```



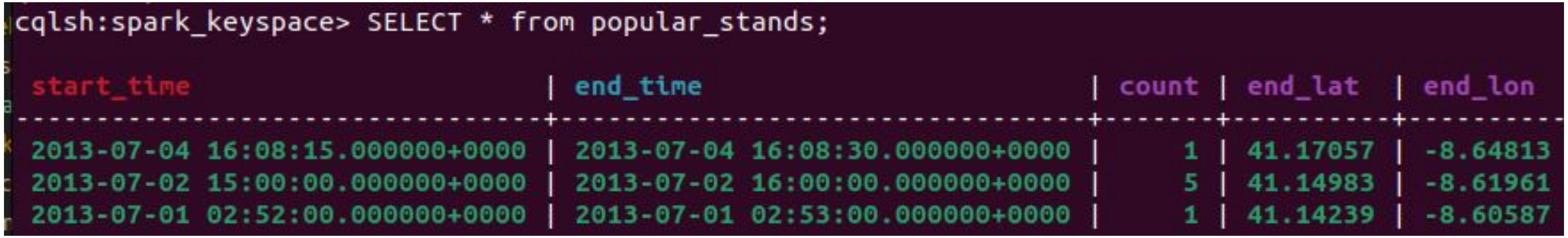
```
query_popularity =top_taxi_stands.writeStream \
    .outputMode("complete") \
    .foreachBatch(writePopularStandsToCassandra) \
    .start()
```



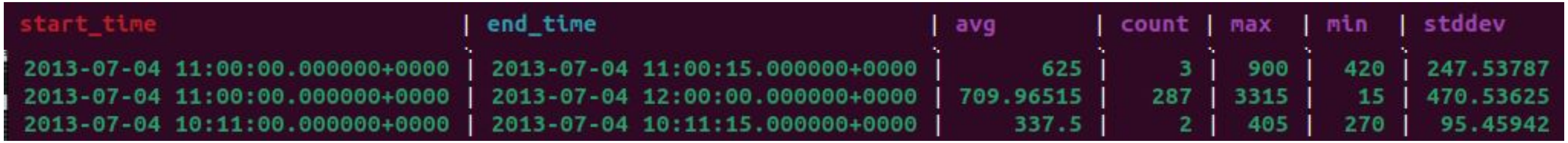
```
def writePopularStandsToCassandra(writeDF, epochId):
    writeDF.write \
        .format("org.apache.spark.sql.cassandra") \
        .mode("append") \
        .options(table="popular_stands", keyspace=keyspace) \
        .save()
```

Apache Spark

Rezultat analize se upisuje u Cassandra bazu



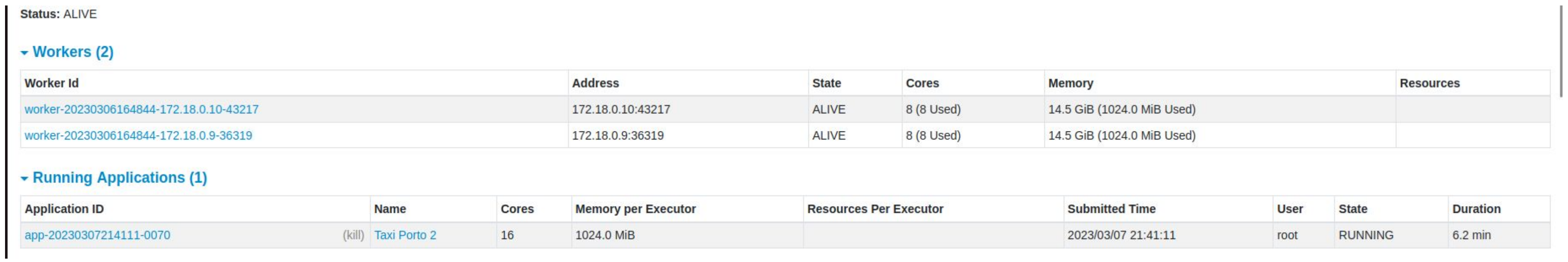
```
cqlsh:spark_keyspace> SELECT * from popular_stands;

 start_time                      | end_time                        | count | end_lat  | end_lon
---------------------------------+---------------------------------+-------+----------+---------
 2013-07-04 16:08:15.000000+0000 | 2013-07-04 16:08:30.000000+0000 |     1 | 41.17057 | -8.64813
 2013-07-02 15:00:00.000000+0000 | 2013-07-02 16:00:00.000000+0000 |     5 | 41.14983 | -8.61961
 2013-07-01 02:52:00.000000+0000 | 2013-07-01 02:53:00.000000+0000 |     1 | 41.14239 | -8.60587
```



```
 start_time                      | end_time                        | avg       | count | max  | min | stddev
 2013-07-04 11:00:00.000000+0000 | 2013-07-04 11:00:15.000000+0000 |       625 |     3 |  900 | 420 | 247.53787
 2013-07-04 11:00:00.000000+0000 | 2013-07-04 12:00:00.000000+0000 | 709.96515 |   287 | 3315 |  15 | 470.53625
 2013-07-04 10:11:00.000000+0000 | 2013-07-04 10:11:15.000000+0000 |     337.5 |     2 |  405 | 270 |  95.45942
```

Apache Spark

Spark Master



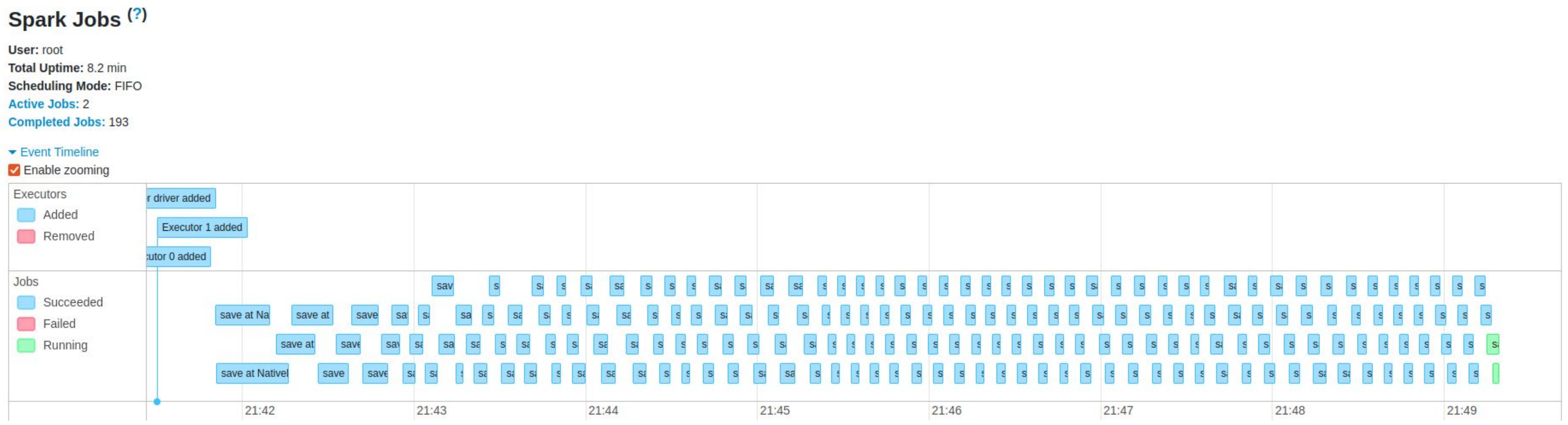
Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20230306164844-172.18.0.10-43217	172.18.0.10:43217	ALIVE	8 (8 Used)	14.5 GiB (1024.0 MiB Used)	
worker-20230306164844-172.18.0.9-36319	172.18.0.9:36319	ALIVE	8 (8 Used)	14.5 GiB (1024.0 MiB Used)	

Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20230307214111-0070 (kill)	Taxi Porto 2	16	1024.0 MiB		2023/03/07 21:41:11	root	RUNNING	6.2 min



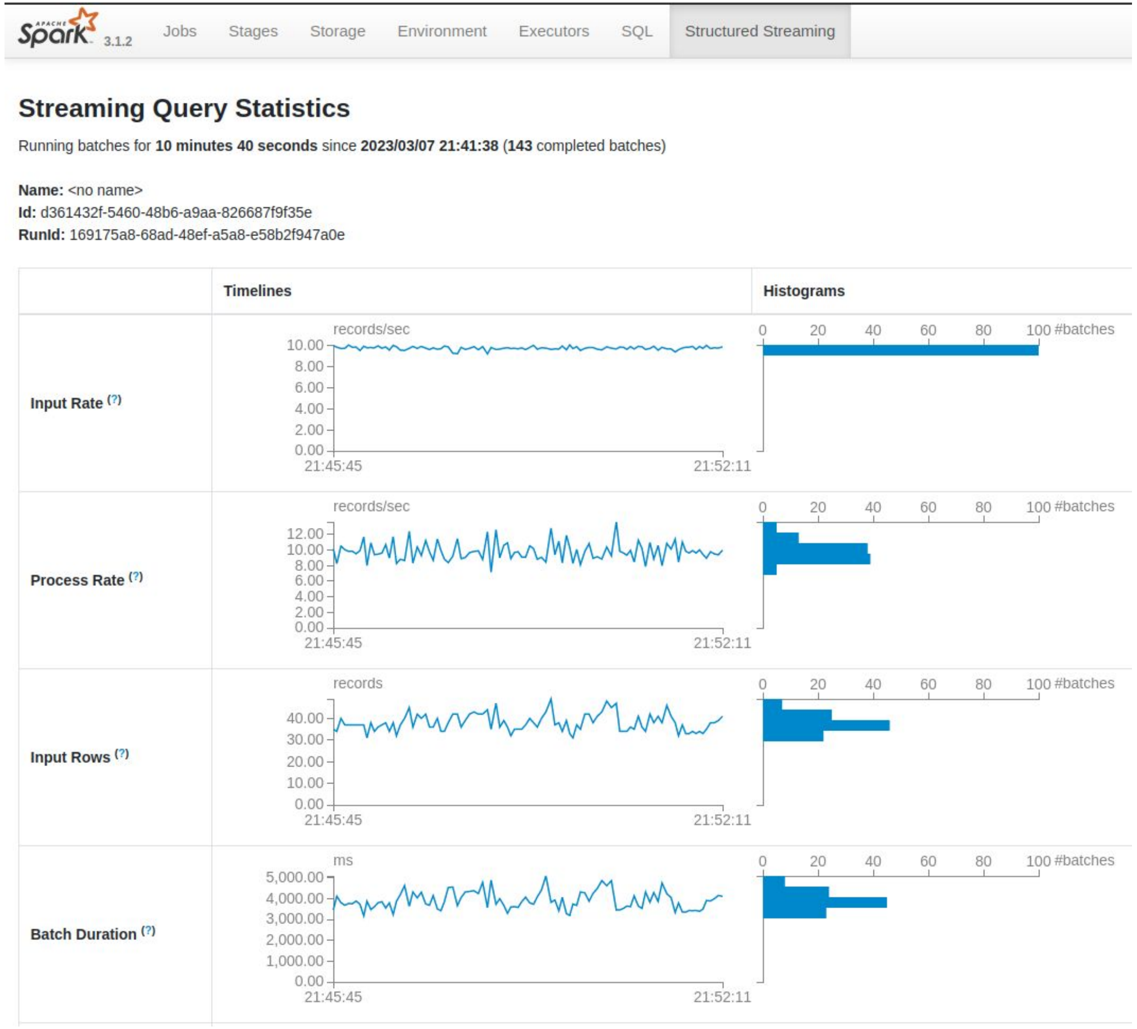
Spark Jobs (?)

User: root
Total Uptime: 8.2 min
Scheduling Mode: FIFO
Active Jobs: 2
Completed Jobs: 193

Event Timeline
☑ Enable zooming

App UI

Apache Spark



Streaming Query Statistics

Running batches for **10 minutes 40 seconds** since **2023/03/07 21:41:38** (**143** completed batches)

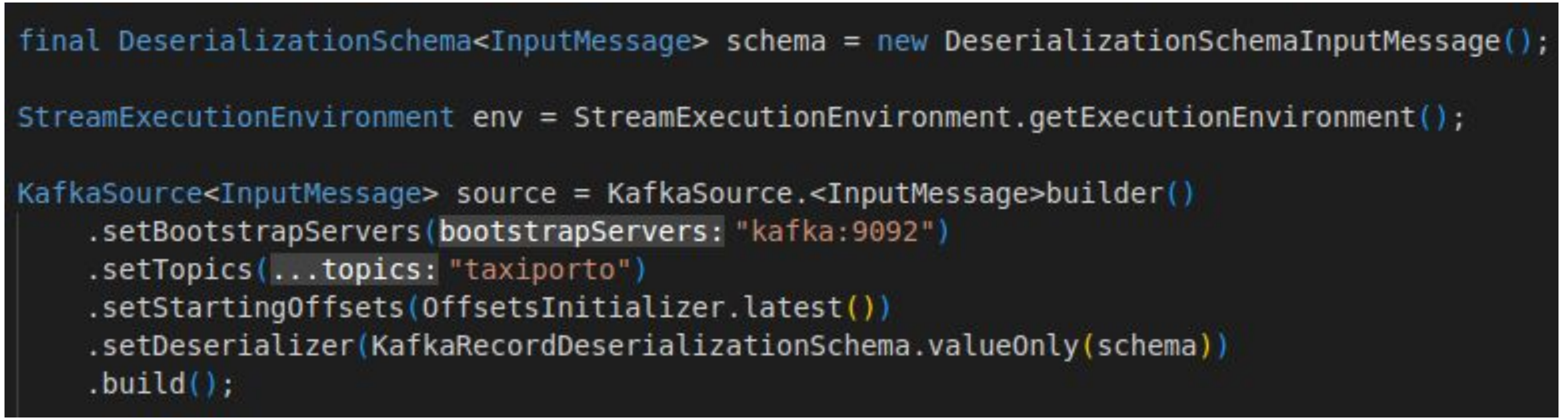
Name: <no name>
Id: d361432f-5460-48b6-a9aa-826687f9f35e
RunId: 169175a8-68ad-48ef-a5a8-e58b2f947a0e

	Timelines	Histograms
Input Rate (?)	records/sec; 0.00–10.00; 21:45:45 – 21:52:11	0 20 40 60 80 100 #batches
Process Rate (?)	records/sec; 0.00–12.00; 21:45:45 – 21:52:11	0 20 40 60 80 100 #batches
Input Rows (?)	records; 0.00–40.00; 21:45:45 – 21:52:11	0 20 40 60 80 100 #batches
Batch Duration (?)	ms; 0.00–5,000.00; 21:45:45 – 21:52:11	0 20 40 60 80 100 #batches

Apache Flink

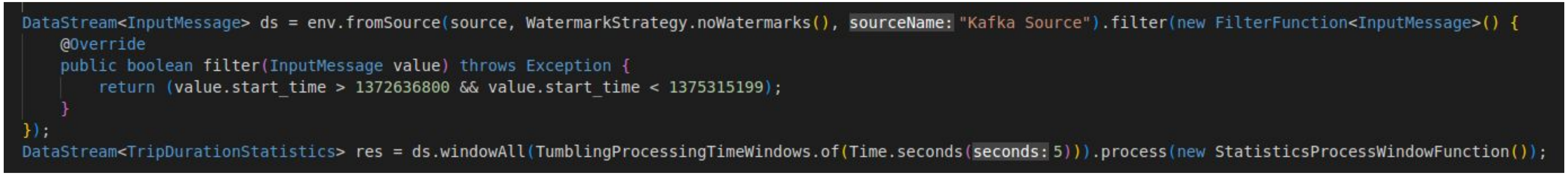
- Java Maven aplikacija

Izvor podataka: Kafka

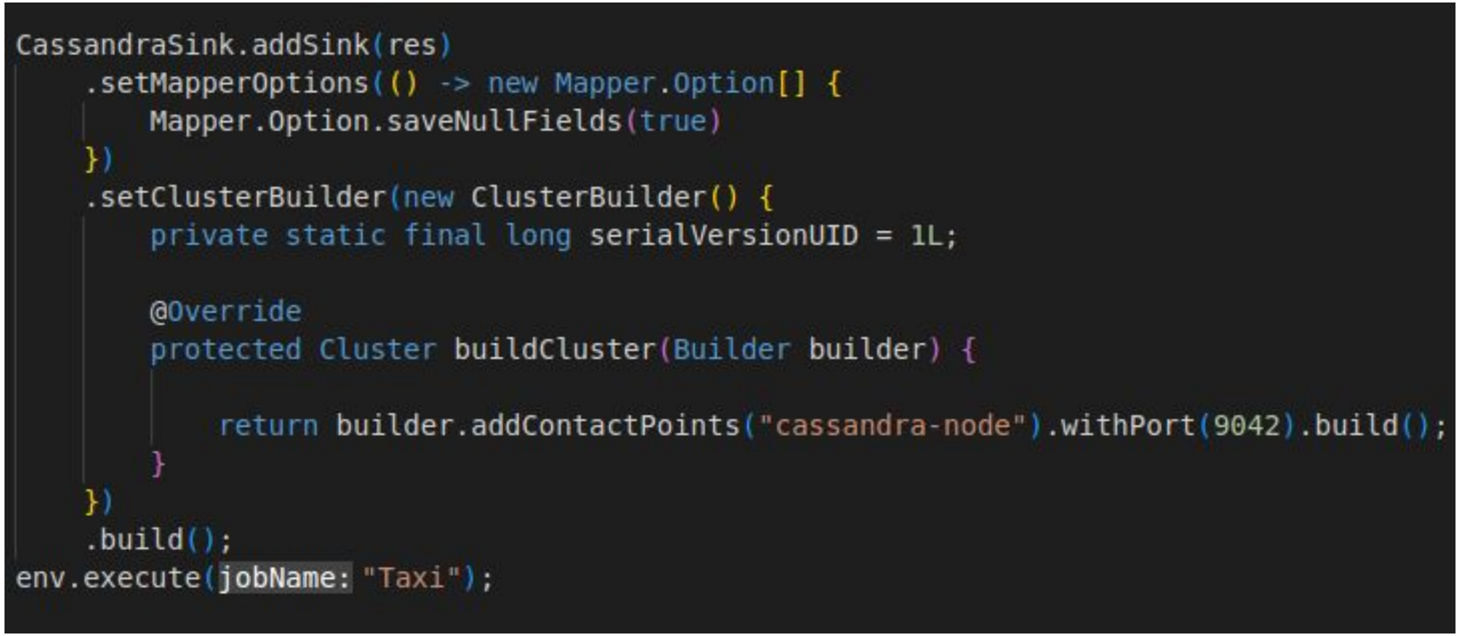


```java
final DeserializationSchema<InputMessage> schema = new DeserializationSchemaInputMessage();

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

KafkaSource<InputMessage> source = KafkaSource.<InputMessage>builder()
    .setBootstrapServers(bootstrapServers: "kafka:9092")
    .setTopics(...topics: "taxiporto")
    .setStartingOffsets(OffsetsInitializer.latest())
    .setDeserializer(KafkaRecordDeserializationSchema.valueOnly(schema))
    .build();
```

Obrada podataka

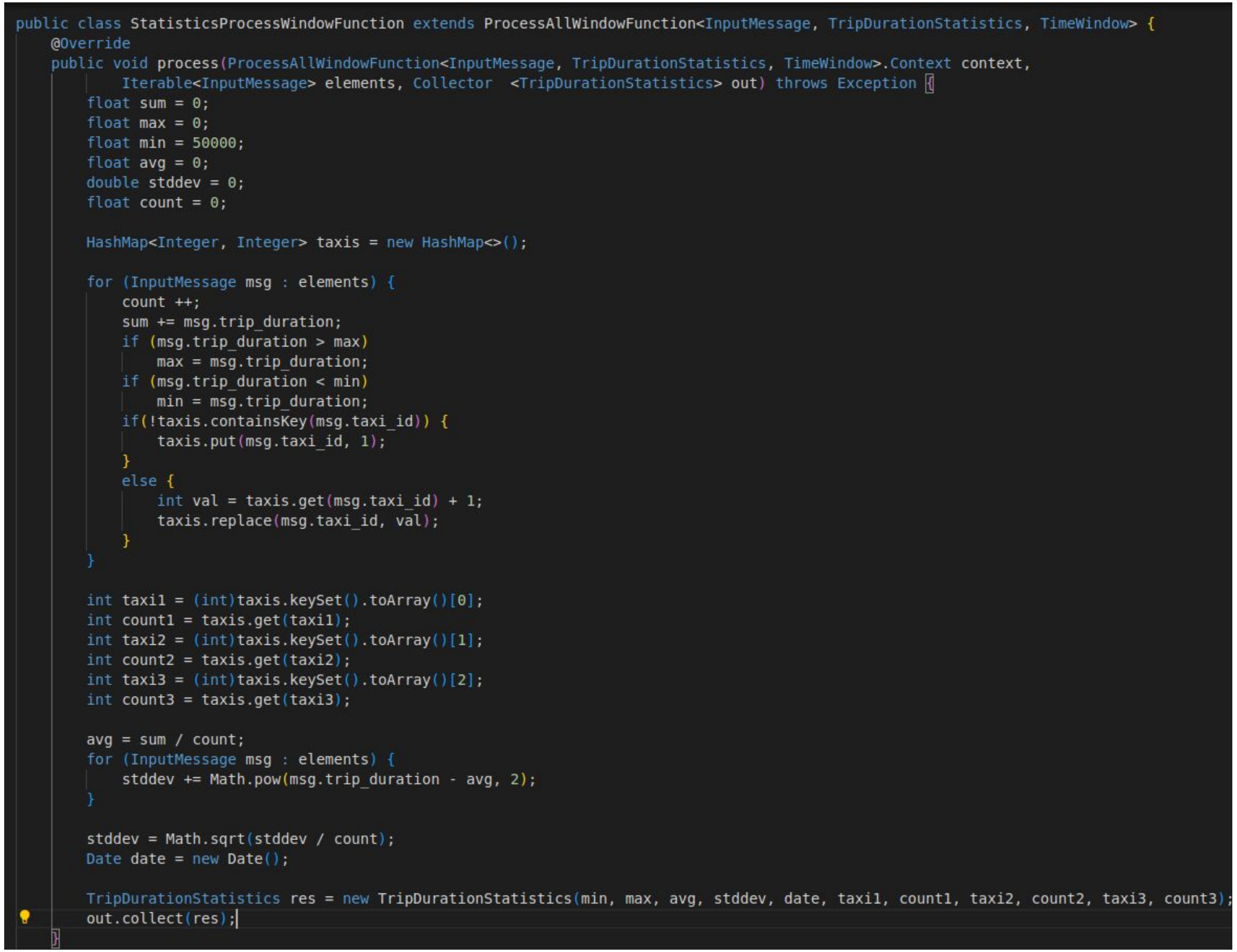


```java
DataStream<InputMessage> ds = env.fromSource(source, WatermarkStrategy.noWatermarks(), sourceName: "Kafka Source").filter(new FilterFunction<InputMessage>() {
    @Override
    public boolean filter(InputMessage value) throws Exception {
        return (value.start_time > 1372636800 && value.start_time < 1375315199);
    }
});
DataStream<TripDurationStatistics> res = ds.windowAll(TumblingProcessingTimeWindows.of(Time.seconds(seconds: 5))).process(new StatisticsProcessWindowFunction());
```



```java
CassandraSink.addSink(res)
     .setMapperOptions(() -> new Mapper.Option[] {
         Mapper.Option.saveNullFields(true)
     })
     .setClusterBuilder(new ClusterBuilder() {
         private static final long serialVersionUID = 1L;

         @Override
         protected Cluster buildCluster(Builder builder) {

             return builder.addContactPoints("cassandra-node").withPort(9042).build();
         }
     })
     .build();
env.execute(jobName: "Taxi");
```

Kreiranje Cassandra sink-a

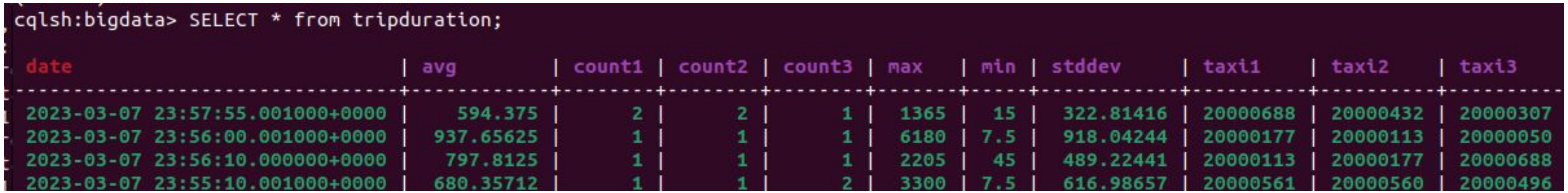


```java
public class StatisticsProcessWindowFunction extends ProcessAllWindowFunction<InputMessage, TripDurationStatistics, TimeWindow> {
    @Override
    public void process(ProcessAllWindowFunction<InputMessage, TripDurationStatistics, TimeWindow>.Context context,
            Iterable<InputMessage> elements, Collector  <TripDurationStatistics> out) throws Exception {
        float sum = 0;
        float max = 0;
        float min = 50000;
        float avg = 0;
        double stddev = 0;
        float count = 0;

        HashMap<Integer, Integer> taxis = new HashMap<>();

        for (InputMessage msg : elements) {
            count ++;
            sum += msg.trip_duration;
            if (msg.trip_duration > max)
                max = msg.trip_duration;
            if (msg.trip_duration < min)
                min = msg.trip_duration;
            if(!taxis.containsKey(msg.taxi_id)) {
                taxis.put(msg.taxi_id, 1);
            }
            else {
                int val = taxis.get(msg.taxi_id) + 1;
                taxis.replace(msg.taxi_id, val);
            }
        }

        int taxi1 = (int)taxis.keySet().toArray()[0];
        int count1 = taxis.get(taxi1);
        int taxi2 = (int)taxis.keySet().toArray()[1];
        int count2 = taxis.get(taxi2);
        int taxi3 = (int)taxis.keySet().toArray()[2];
        int count3 = taxis.get(taxi3);

        avg = sum / count;
        for (InputMessage msg : elements) {
            stddev += Math.pow(msg.trip_duration - avg, 2);
        }

        stddev = Math.sqrt(stddev / count);
        Date date = new Date();

        TripDurationStatistics res = new TripDurationStatistics(min, max, avg, stddev, date, taxi1, count1, taxi2, count2, taxi3, count3);
        out.collect(res);
    }
```

Klasa i funkcija za analizu toka podataka

Apache Flink

Cassandra - prikaz podataka

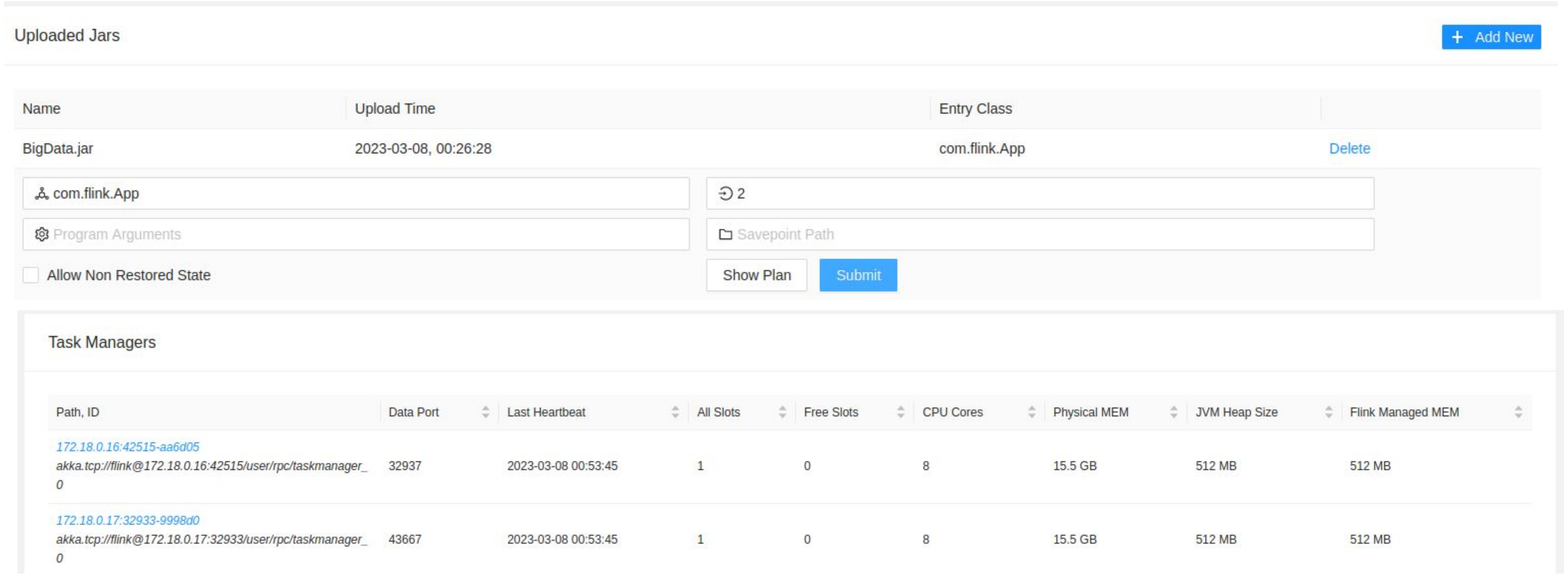


```
cqlsh:bigdata> SELECT * from tripduration;

 date                            | avg       | count1 | count2 | count3 | max  | min | stddev    | taxi1    | taxi2    | taxi3
---------------------------------+-----------+--------+--------+--------+------+-----+-----------+----------+----------+----------
 2023-03-07 23:57:55.001000+0000 |   594.375 |      2 |      2 |      1 | 1365 |  15 | 322.81416 | 20000688 | 20000432 | 20000307
 2023-03-07 23:56:00.001000+0000 | 937.65625 |      1 |      1 |      1 | 6180 | 7.5 | 918.04244 | 20000177 | 20000113 | 20000050
 2023-03-07 23:56:10.000000+0000 |  797.8125 |      1 |      1 |      1 | 2205 |  45 | 489.22441 | 20000113 | 20000177 | 20000688
 2023-03-07 23:55:10.001000+0000 | 680.35712 |      1 |      1 |      2 | 3300 | 7.5 | 616.98657 | 20000561 | 20000560 | 20000496
```

Apache Flink

1 Job Manager
2 Task Manager-a

Job submit



Uploaded Jars

Name	Upload Time	Entry Class	
BigData.jar	2023-03-08, 00:26:28	com.flink.App	Delete

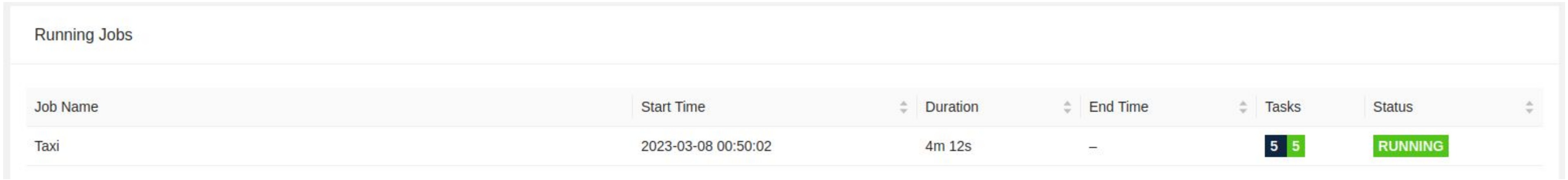
com.flink.App	2
Program Arguments	Savepoint Path

☐ Allow Non Restored State

Task Managers

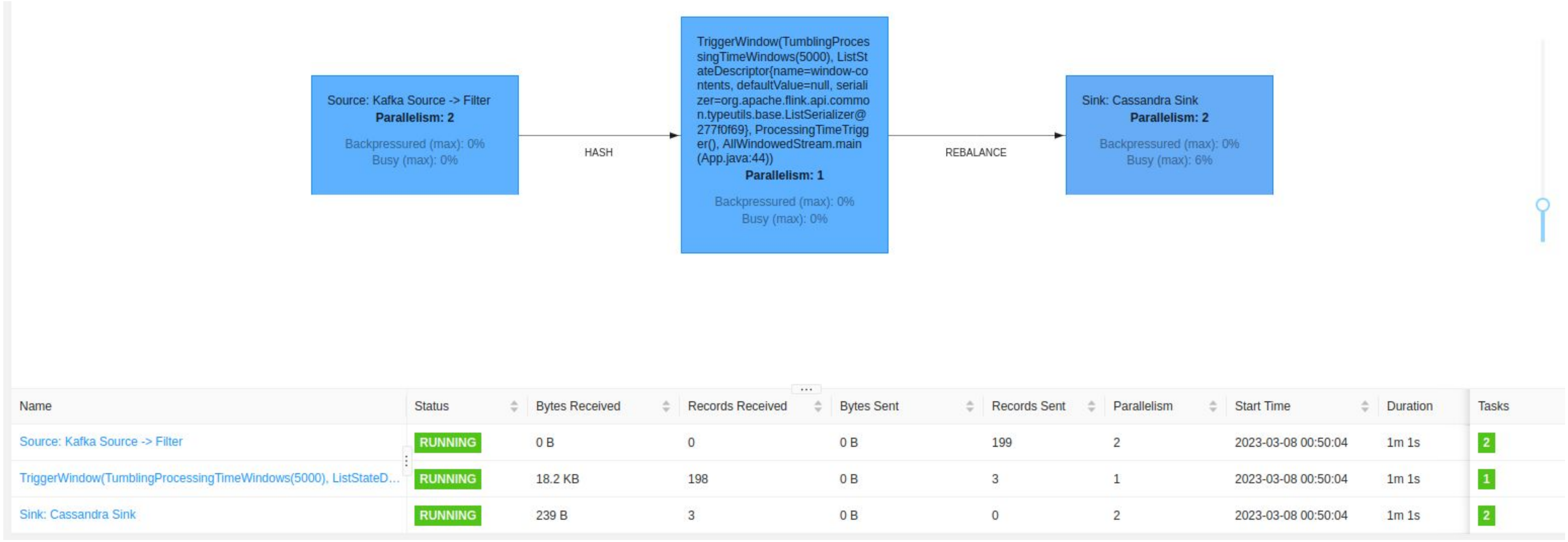
Path, ID	Data Port	Last Heartbeat	All Slots	Free Slots	CPU Cores	Physical MEM	JVM Heap Size	Flink Managed MEM
172.18.0.16:42515-aa6d05 *akka.tcp://flink@172.18.0.16:42515/user/rpc/taskmanager_0*	32937	2023-03-08 00:53:45	1	0	8	15.5 GB	512 MB	512 MB
172.18.0.17:32933-9998d0 *akka.tcp://flink@172.18.0.17:32933/user/rpc/taskmanager_0*	43667	2023-03-08 00:53:45	1	0	8	15.5 GB	512 MB	512 MB

Apache Flink



Running Jobs

Job Name	Start Time	Duration	End Time	Tasks	Status
Taxi	2023-03-08 00:50:02	4m 12s	–	5 5	RUNNING


Source: Kafka Source -> Filter
Parallelism: 2
Backpressured (max): 0%
Busy (max): 0%
HASH
TriggerWindow(TumblingProcessingTimeWindows(5000), ListStateDescriptor{name=window-contents, defaultValue=null, serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@277f0f69}, ProcessingTimeTrigger(), AllWindowedStream.main(App.java:44))
Parallelism: 1
Backpressured (max): 0%
Busy (max): 0%
REBALANCE
Sink: Cassandra Sink
Parallelism: 2
Backpressured (max): 0%
Busy (max): 6%

Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parallelism	Start Time	Duration	Tasks
Source: Kafka Source -> Filter	RUNNING	0 B	0	0 B	199	2	2023-03-08 00:50:04	1m 1s	2
TriggerWindow(TumblingProcessingTimeWindows(5000), ListStateD...	RUNNING	18.2 KB	198	0 B	3	1	2023-03-08 00:50:04	1m 1s	1
Sink: Cassandra Sink	RUNNING	239 B	3	0 B	0	2	2023-03-08 00:50:04	1m 1s	2

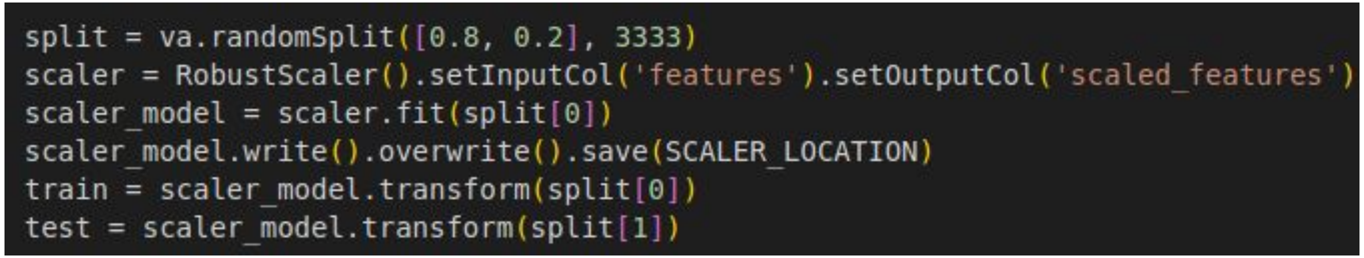
Projekat 3

- Spark aplikacija za treniranje modela mašinskog učenja
- Spark Streaming aplikacija
- InfluxDB
- Grafana

Spark ML

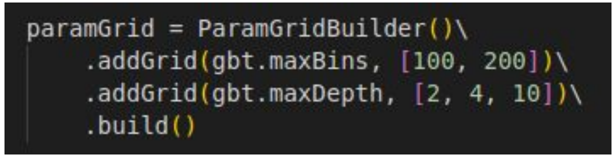
- Predikcija trajanja taksi vožnje
- *Gradient-Boosted Tree* (`pyspark.ml.regression.GBTRegressor`)

- Skaliranje podataka korišćenjem robust scaler-a



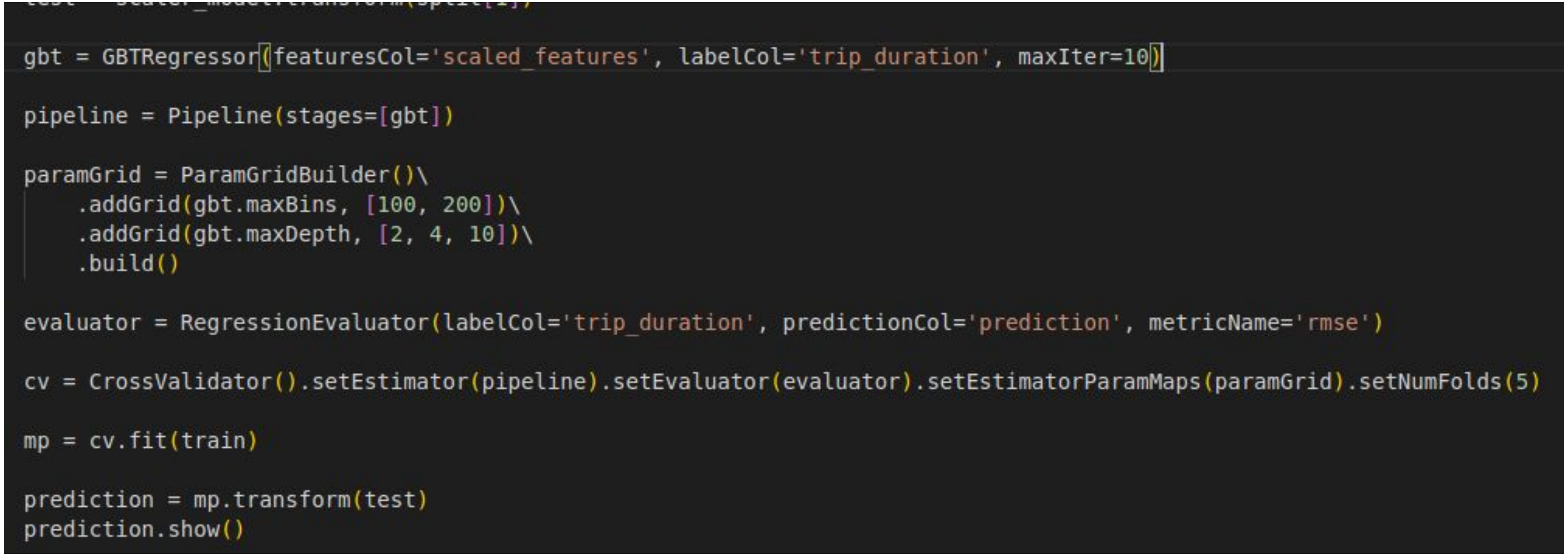
```
split = va.randomSplit([0.8, 0.2], 3333)
scaler = RobustScaler().setInputCol('features').setOutputCol('scaled_features')
scaler_model = scaler.fit(split[0])
scaler_model.write().overwrite().save(SCALER_LOCATION)
train = scaler_model.transform(split[0])
test = scaler_model.transform(split[1])
```

- Optimizacija hiperparametara (`pyspark.ml.tuning.ParamGridBuilder`)



```
paramGrid = ParamGridBuilder()\
    .addGrid(gbt.maxBins, [100, 200])\
    .addGrid(gbt.maxDepth, [2, 4, 10])\
    .build()
```

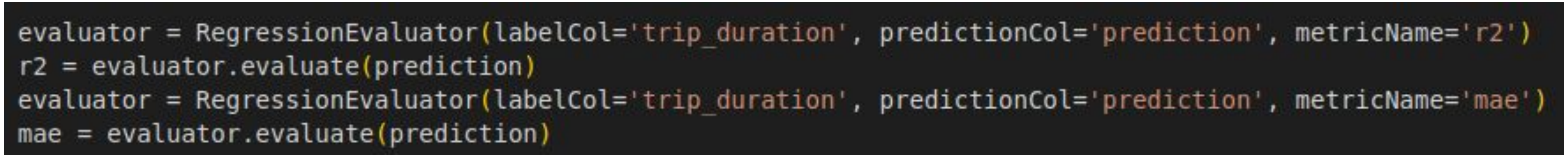
Spark ML



```
gbt = GBTRegressor(featuresCol='scaled_features', labelCol='trip_duration', maxIter=10)

pipeline = Pipeline(stages=[gbt])

paramGrid = ParamGridBuilder()\
    .addGrid(gbt.maxBins, [100, 200])\
    .addGrid(gbt.maxDepth, [2, 4, 10])\
    .build()

evaluator = RegressionEvaluator(labelCol='trip_duration', predictionCol='prediction', metricName='rmse')

cv = CrossValidator().setEstimator(pipeline).setEvaluator(evaluator).setEstimatorParamMaps(paramGrid).setNumFolds(5)

mp = cv.fit(train)

prediction = mp.transform(test)
prediction.show()
```

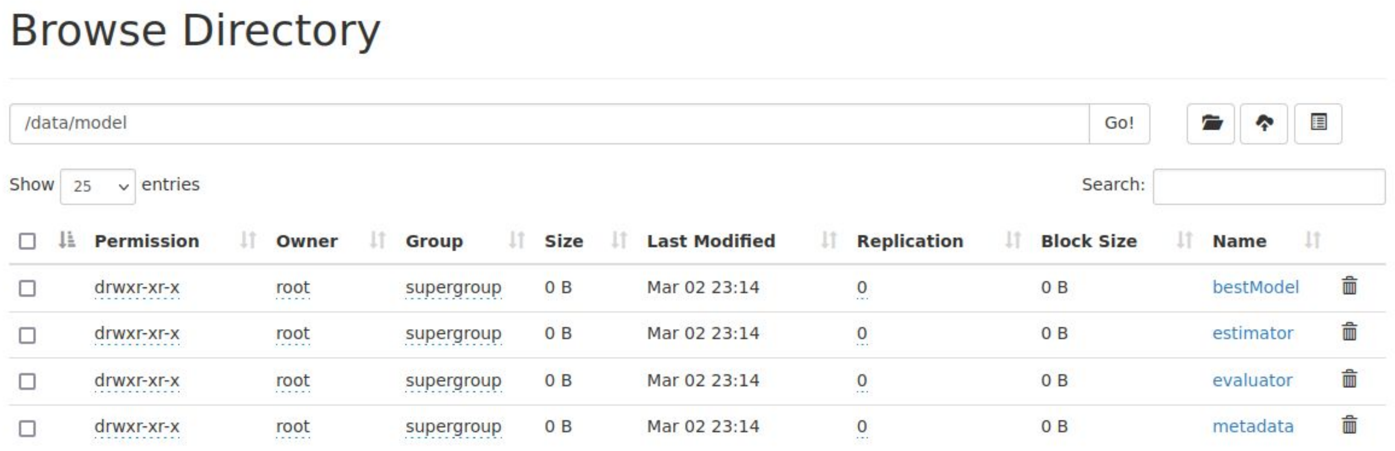
Spark ML

- Evaluacija



```
evaluator = RegressionEvaluator(labelCol='trip_duration', predictionCol='prediction', metricName='r2')
r2 = evaluator.evaluate(prediction)
evaluator = RegressionEvaluator(labelCol='trip_duration', predictionCol='prediction', metricName='mae')
mae = evaluator.evaluate(prediction)
```

- Model se čuva na HDFS



Browse Directory

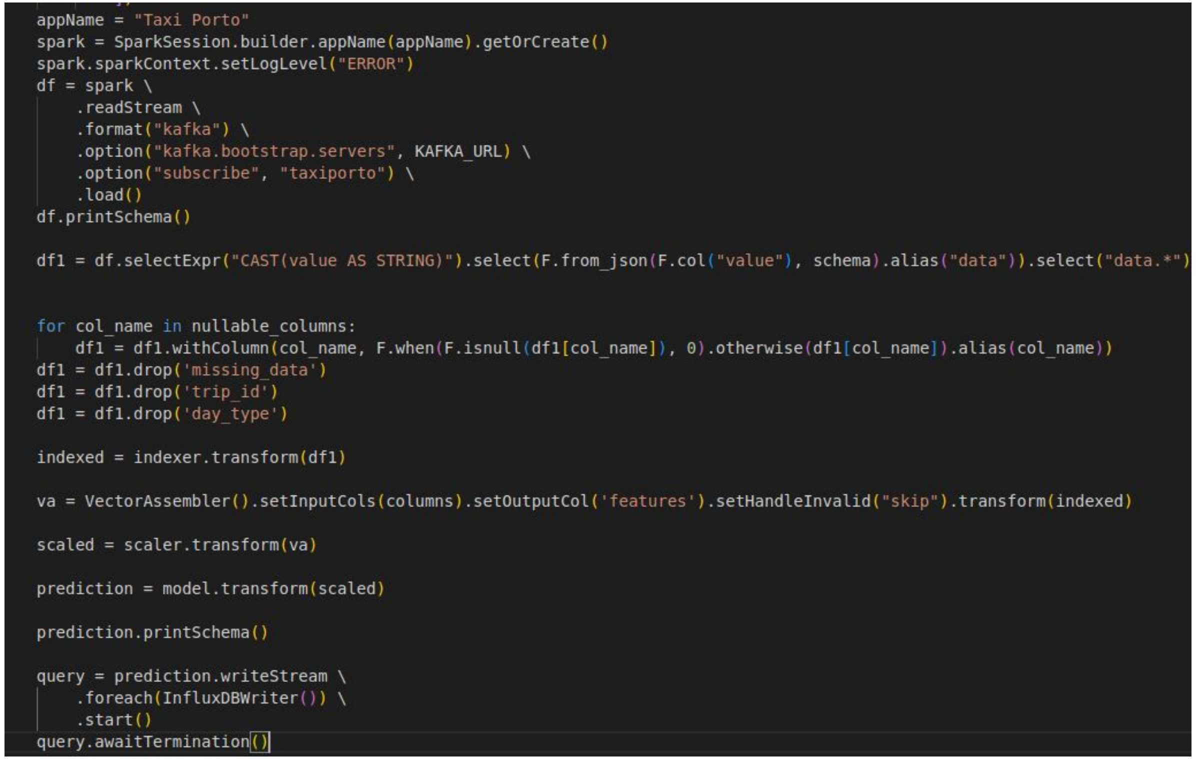
/data/model Go!

Show 25 entries Search:

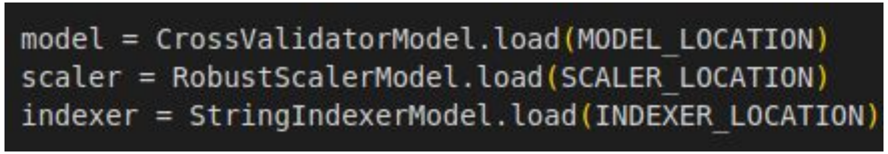
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	drwxr-xr-x	root	supergroup	0 B	Mar 02 23:14	0	0 B	bestModel
☐	drwxr-xr-x	root	supergroup	0 B	Mar 02 23:14	0	0 B	estimator
☐	drwxr-xr-x	root	supergroup	0 B	Mar 02 23:14	0	0 B	evaluator
☐	drwxr-xr-x	root	supergroup	0 B	Mar 02 23:14	0	0 B	metadata

Spark Streaming aplikacija

- Izvor podataka: Kafka topic
- Model se učitava sa HDFS-a

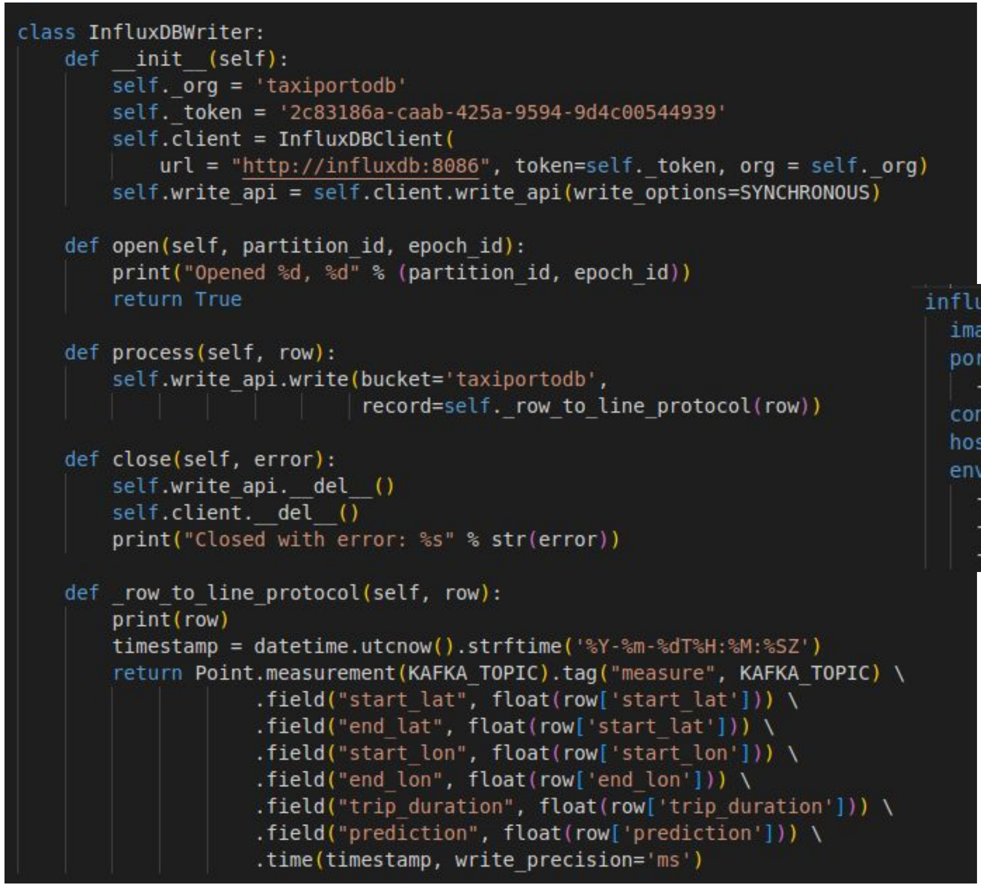


```python
appName = "Taxi Porto"
spark = SparkSession.builder.appName(appName).getOrCreate()
spark.sparkContext.setLogLevel("ERROR")
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_URL) \
    .option("subscribe", "taxiporto") \
    .load()
df.printSchema()

df1 = df.selectExpr("CAST(value AS STRING)").select(F.from_json(F.col("value"), schema).alias("data")).select("data.*")


for col_name in nullable_columns:
    df1 = df1.withColumn(col_name, F.when(F.isnull(df1[col_name]), 0).otherwise(df1[col_name]).alias(col_name))
df1 = df1.drop('missing_data')
df1 = df1.drop('trip_id')
df1 = df1.drop('day_type')

indexed = indexer.transform(df1)

va = VectorAssembler().setInputCols(columns).setOutputCol('features').setHandleInvalid("skip").transform(indexed)

scaled = scaler.transform(va)

prediction = model.transform(scaled)

prediction.printSchema()

query = prediction.writeStream \
    .foreach(InfluxDBWriter()) \
    .start()
query.awaitTermination()
```

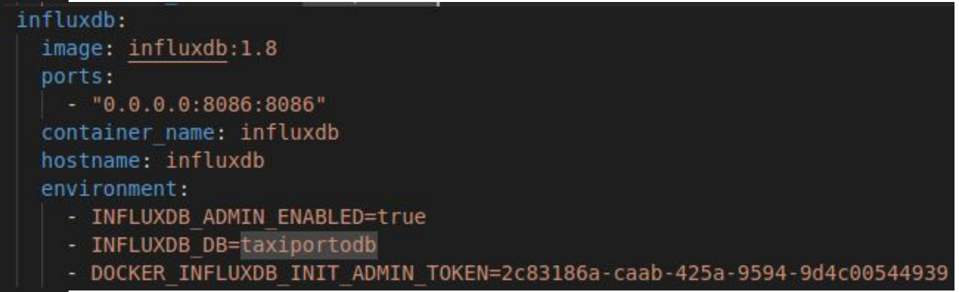


```python
model = CrossValidatorModel.load(MODEL_LOCATION)
scaler = RobustScalerModel.load(SCALER_LOCATION)
indexer = StringIndexerModel.load(INDEXER_LOCATION)
```

InfluxDB

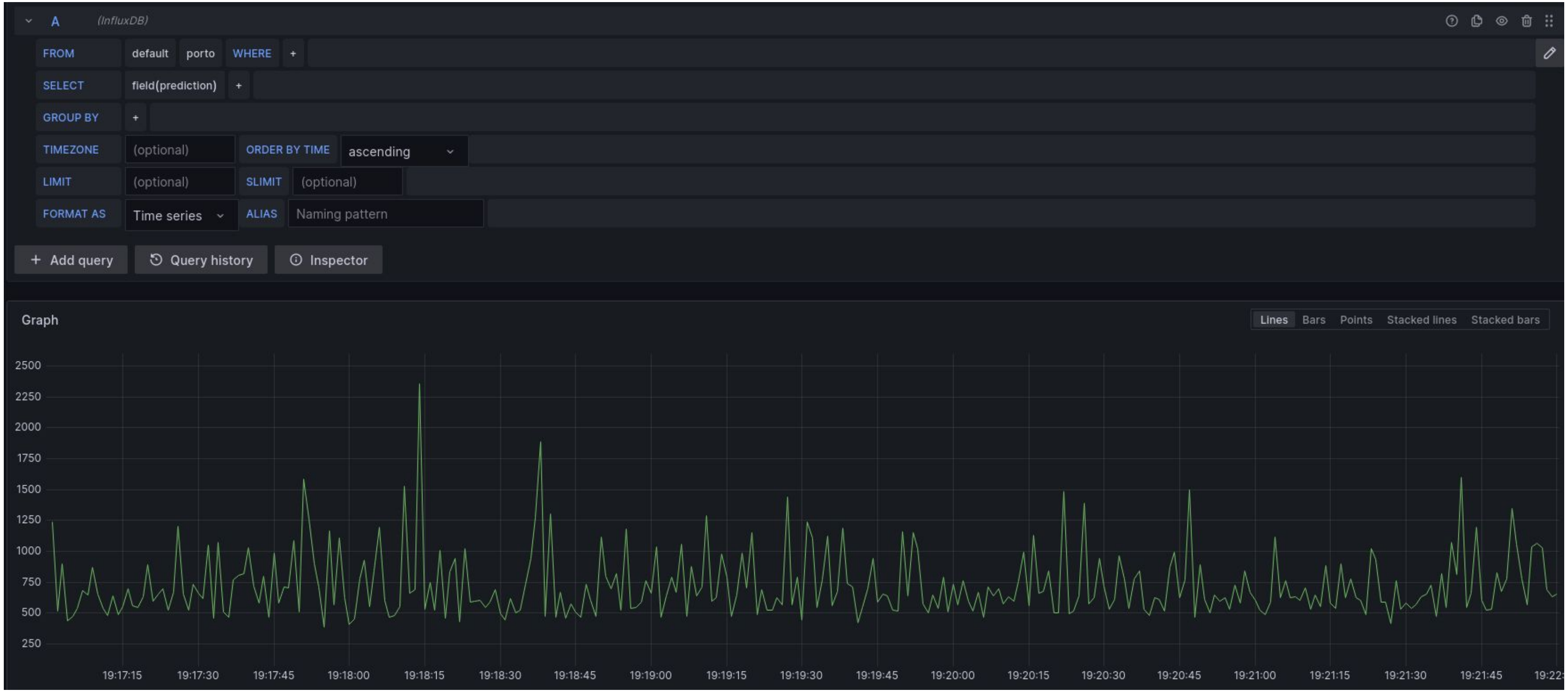


```python
class InfluxDBWriter:
    def __init__(self):
        self._org = 'taxiportodb'
        self._token = '2c83186a-caab-425a-9594-9d4c00544939'
        self.client = InfluxDBClient(
            url = "http://influxdb:8086", token=self._token, org = self._org)
        self.write_api = self.client.write_api(write_options=SYNCHRONOUS)

    def open(self, partition_id, epoch_id):
        print("Opened %d, %d" % (partition_id, epoch_id))
        return True

    def process(self, row):
        self.write_api.write(bucket='taxiportodb',
                             record=self._row_to_line_protocol(row))

    def close(self, error):
        self.write_api.__del__()
        self.client.__del__()
        print("Closed with error: %s" % str(error))

    def _row_to_line_protocol(self, row):
        print(row)
        timestamp = datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')
        return Point.measurement(KAFKA_TOPIC).tag("measure", KAFKA_TOPIC) \
                  .field("start_lat", float(row['start_lat'])) \
                  .field("end_lat", float(row['start_lat'])) \
                  .field("start_lon", float(row['start_lon'])) \
                  .field("end_lon", float(row['end_lon'])) \
                  .field("trip_duration", float(row['trip_duration'])) \
                  .field("prediction", float(row['prediction'])) \
                  .time(timestamp, write_precision='ms')
```



```yaml
influxdb:
  image: influxdb:1.8
  ports:
    - "0.0.0.0:8086:8086"
  container_name: influxdb
  hostname: influxdb
  environment:
    - INFLUXDB_ADMIN_ENABLED=true
    - INFLUXDB_DB=taxiportodb
    - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=2c83186a-caab-425a-9594-9d4c00544939
```

Vizuelizacija u Grafani



A (InfluxDB)

FROM default porto WHERE +

SELECT field(prediction) +

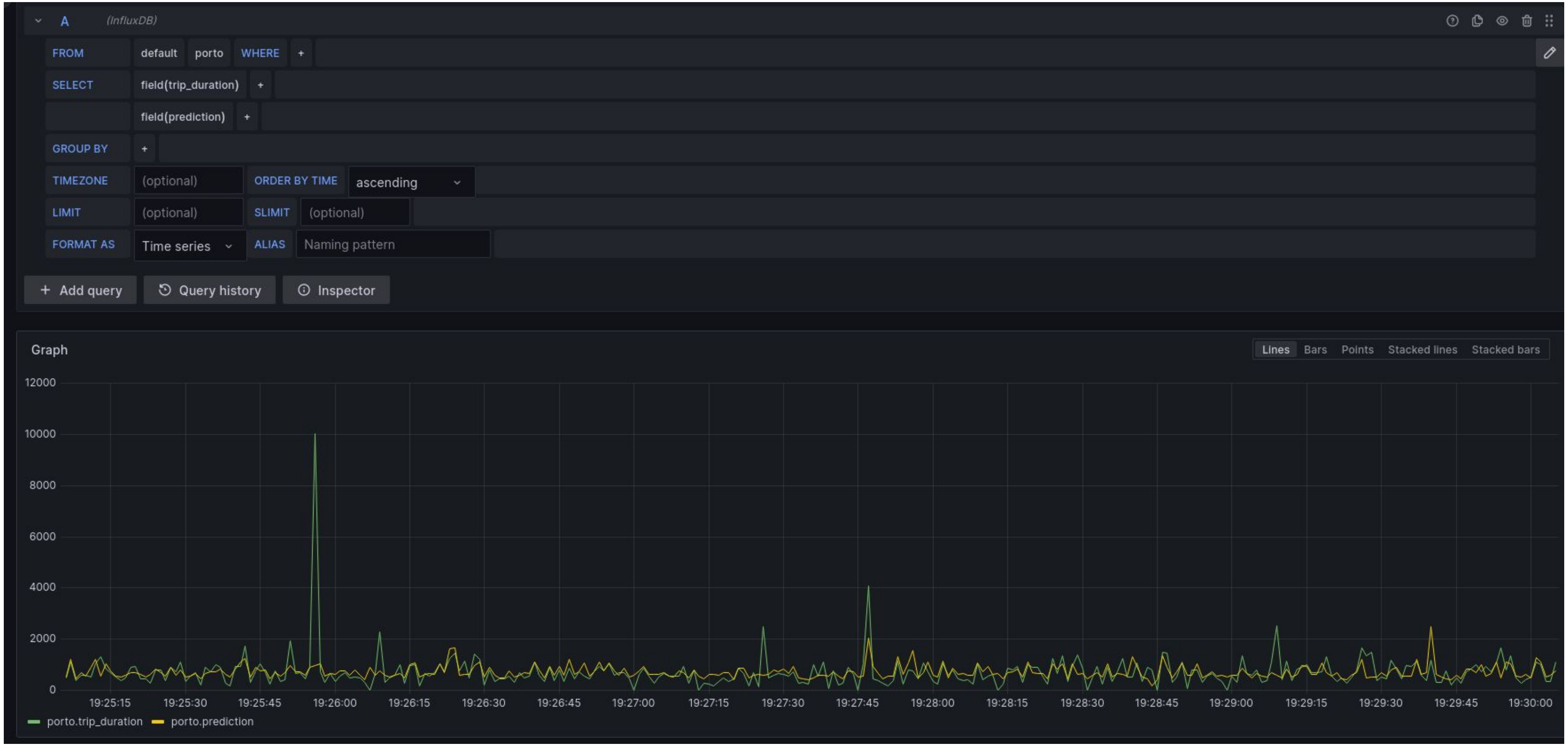
GROUP BY +

TIMEZONE (optional) ORDER BY TIME ascending

LIMIT (optional) SLIMIT (optional)

FORMAT AS Time series ALIAS Naming pattern

Vizuelizacija u Grafani


A (InfluxDB)

FROM default porto WHERE +

SELECT field(trip_duration) +

field(prediction) +

GROUP BY +

TIMEZONE (optional) ORDER BY TIME ascending

LIMIT (optional) SLIMIT (optional)

FORMAT AS Time series ALIAS Naming pattern