Sistemi za obradu i analizu velike količine podataka - Drugi projekat

Nikola Božinović 1030

Tehnologije

- Java 8
- Hadoop 2.7
- Spark 2.4.3
- Kafka 2.4.0 (Scala 2.11)
- Cassandra DB
- Docker

Streaming Producer

```
    KafkaProducer<String, String> producer = configureProducer(kafkaUrl);

    Configuration conf = new Configuration();
    conf.set(CommonConfigurationKeys.FS_DEFAULT_NAME_KEY, hdfsUrl);
    conf.set("fs.hdfs.impl", DistributedFileSystem.class.getName());
    conf.set("fs.file.impl", LocalFileSystem.class.getName());
    FSDataInputStream inputStream = null;
    FileSystem fs = null;
    try {
        fs = FileSystem.get(URI.create(hdfsUrl), conf);
        Path inFile = new Path(csvFilePath);
        inputStream = fs.open(inFile);
        if (!fs.exists(inFile)) {
            System.out.println("Input file not found");
            throw new IOException("Input file not found");
        }

        String line = inputStream.readLine();
        while(line != null) {
            EventData tmp = EventData.CreateEventData(line);

            if (tmp != null) {
                ProducerRecord<String, String> rec = new ProducerRecord<String, String>(TaxiTopic, line);
                producer.send(rec);
                System.out.println("[KAFKA TAXI DATA SENT}]: " + tmp.getVendorId());
                Thread.sleep(dataSendingSleep * 1000);
                System.out.println("Sleeping " + dataSendingSleep + "sec");
            }

            line = inputStream.readLine();
        }

    } finally {
        if (inputStream != null) {
            inputStream.close();
        }
        if (fs != null) {
            fs.close();
        }
    }

    producer.close();
    System.out.println("All done.");

    System.exit(1);
}
```

Streaming consumer

```
SparkConf conf = new SparkConf().setAppName("BigData-2").setMaster(sparkMasterUrl);
JavaStreamingContext streamingContext = new JavaStreamingContext(conf, new Duration(dataReceivingSleep * 1000));

Map<String, Object> kafkaParams = getKafkaParams(kafkaUrl);
Collection<String> topics = Collections.singletonList(TaxiTopic);

// Creates Kafka Direct Stream.

JavaInputDStream<ConsumerRecord<Object, String>> stream = KafkaUtils.createDirectStream(streamingContext,
        LocationStrategies.PreferConsistent(), ConsumerStrategies.Subscribe(topics, kafkaParams));


JavaDStream<String> receivedData = stream.map(ConsumerRecord::value);
JavaDStream<EventData> eventData = receivedData.map(EventData::CreateEventData);

// Prints received and serialized event data.

eventData.print();

Double minLongitude = -75.971770000000003;
Double maxLongitude = -71.995770000000003;
Double minLatitude = 38.706828000000003;
Double maxLatitude = 42.756998000000003;


// Filters trips with one passenger at a given location.

JavaDStream<EventData> filteredData = eventData.filter(ed -> ed != null &&
        ed.getPickupLongitute() >=  minLongitude && ed.getPickupLongitute() <= maxLongitude &&
        ed.getDropoffLongitude() >=  minLongitude && ed.getDropoffLongitude() <=  maxLongitude &&
        ed.getPickupLatitude() >=  minLatitude && ed.getPickupLatitude() <= maxLatitude &&
        ed.getDropoffLatitude() >=  minLatitude && ed.getDropoffLatitude() <= maxLatitude &&
        ed.getPassengerCount().equals("1"));
```

Cassandra connector

```java
public class CassandraConnector {

    private CqlSession session;
    private static CassandraConnector instance;


    public static CassandraConnector getInstance() {
        if (instance == null) {
            instance = new CassandraConnector();
        }
        return instance;
    }

    public void connect(String addr, Integer port) {
        CqlSessionBuilder builder = CqlSession.builder();
        builder.withAuthCredentials("cassandra", "cassandra")
            .withLocalDatacenter("datacenter1")
            .addContactPoint(new InetSocketAddress(addr, port));

        session = builder.build();
    }

    public CqlSession getSession() {
        return session;
    }


    public void close() {
        session.close();
    }

    private CassandraConnector() {

    }
}
```

Prepare Cassandra Keyspace

```
CassandraConnector conn = CassandraConnector.getInstance();
conn.connect(addr, port);
CqlSession session = conn.getSession();


String createKeyspaceCQL = String.format("CREATE KEYSPACE IF NOT EXISTS %s WITH replication "
        + "= {'class':'SimpleStrategy', 'replication_factor':1};", Keyspace);

//String dropTaxiRecordTable = String.format("DROP TABLE %s.%s;", Keyspace, TaxiRecordsTable);

String createTaxiRecordTable = String.format(
        "CREATE TABLE IF NOT EXISTS %s.%s ("
                    + " start_time timestamp PRIMARY KEY,"
                    + " end_time timestamp,"
                    + " fare_amount float,"
                    + " pickup_latitude double,"
                    + " pickup_longitude double,"
                    + " mta_tax float,"
                    + " rate_code text,"
                    + " surcharge float );",
        Keyspace, TaxiRecordsTable);

//String dropStatisticTable = String.format("DROP TABLE %s.%s;", Keyspace, StatisticsTable);

String createStatisticTable = String.format(
        "CREATE TABLE IF NOT EXISTS %s.%s ("
            + " min_pickup_time timestamp PRIMARY KEY,"
            + " max_pickup_time timestamp,"
            + " min_tip_amount float,"
            + " max_tip_amount float,"
            + " average_tip_amount float,"
            + " trip_count bigint,"
            + " top_payment_type_count int,"
            + " top_payment_type_name text,"
            + " top_rate_code_count int,"
            + " top_rate_code_name text );",
        Keyspace, StatisticsTable);

System.out.println("Preparing Cassandra Keyspace.");

session.execute(createKeyspaceCQL);
System.out.println("Keyspace created");

//session.execute(dropTaxiRecordTable);
//System.out.println(String.format("Table %s dropped.", TaxiRecordsTable));

session.execute(createTaxiRecordTable);
System.out.println(String.format("Table %s created.", StatisticsTable));

//session.execute(dropStatisticTable);
//System.out.println(String.format("Table %s dropped.", StatisticsTable));

session.execute(createStatisticTable);
System.out.println(String.format("Table %s created.", TaxiRecordsTable));
```

Obrada Streaming podataka

```
filteredData.foreachRDD(rdd -> {

    Long count = rdd.count();
    if (count <= 0) {
        System.out.println("INFO: There is no RDD over the filtered data, skipping.");
        return;
    }

    // Finds min, max and avg tip amount in current rdd batch.

    EventData minTipAmount = rdd.min(new TipAmountComparator());
    EventData maxTipAmount = rdd.max(new TipAmountComparator());

    EventData minPickupDate = rdd.min(new DateComparator());
    EventData maxPickupDate = rdd.max(new DateComparator());

    JavaRDD<Float> tipAmounts = rdd.map((tr) -> tr.getTipAmount());

    Float tipAmountSum = tipAmounts.reduce((tipAmount, accumulator) -> {
        return tipAmount + accumulator;
    });

    Float averageTipAmount = tipAmountSum / count;

    // Finds most used payment types in current rdd batch.

    JavaPairRDD<String, Integer> topPaymentTypes = rdd.mapToPair(r -> new Tuple2<>(r.getPaymentType(),1)).reduceByKey(Integer::sum);
    Integer mostPaymentTypeCount = topPaymentTypes.map(t -> t._2).max(Comparator.naturalOrder());

    List<String> topPaymentTypesNames = topPaymentTypes.filter(t -> t._2.equals(mostPaymentTypeCount)).map(t -> t._1).collect();
    String mostPaymentTypeNames = String.join(",", topPaymentTypesNames);

    // Finds most used rate codes in current rdd batch.

    JavaPairRDD<String, Integer> topRateCodes = rdd.mapToPair(r -> new Tuple2<>(r.getRateCode(),1)).reduceByKey(Integer::sum);
    Integer mostRateCodesCount = topRateCodes.map(t -> t._2).max(Comparator.naturalOrder());

    List<String> topRateCodeNames = topRateCodes.filter(t -> t._2.equals(mostRateCodesCount)).map(t -> t._1).collect();
    String mostRateCodeNames = String.join(",", topRateCodeNames);

    // Creates and stores statistic data in Cassandra Db.

    StatisticRecord statisticData = new StatisticRecord(minPickupDate.getPickupDateTime(), maxPickupDate.getPickupDateTime(),
            minTipAmount.getTipAmount(), maxTipAmount.getTipAmount(),averageTipAmount, count,
            mostPaymentTypeCount, mostPaymentTypeNames, mostRateCodesCount, mostRateCodeNames);

    //System.out.println("Statistic Data");
    //System.out.println(statisticData.toString());
    saveStatisticRecord(statisticData, cassandraUrl, cassandraPort);
    System.out.println("INFO: Statistics are stored in database.");
```

Smeštanje taxi rekorda u Cassandra DB

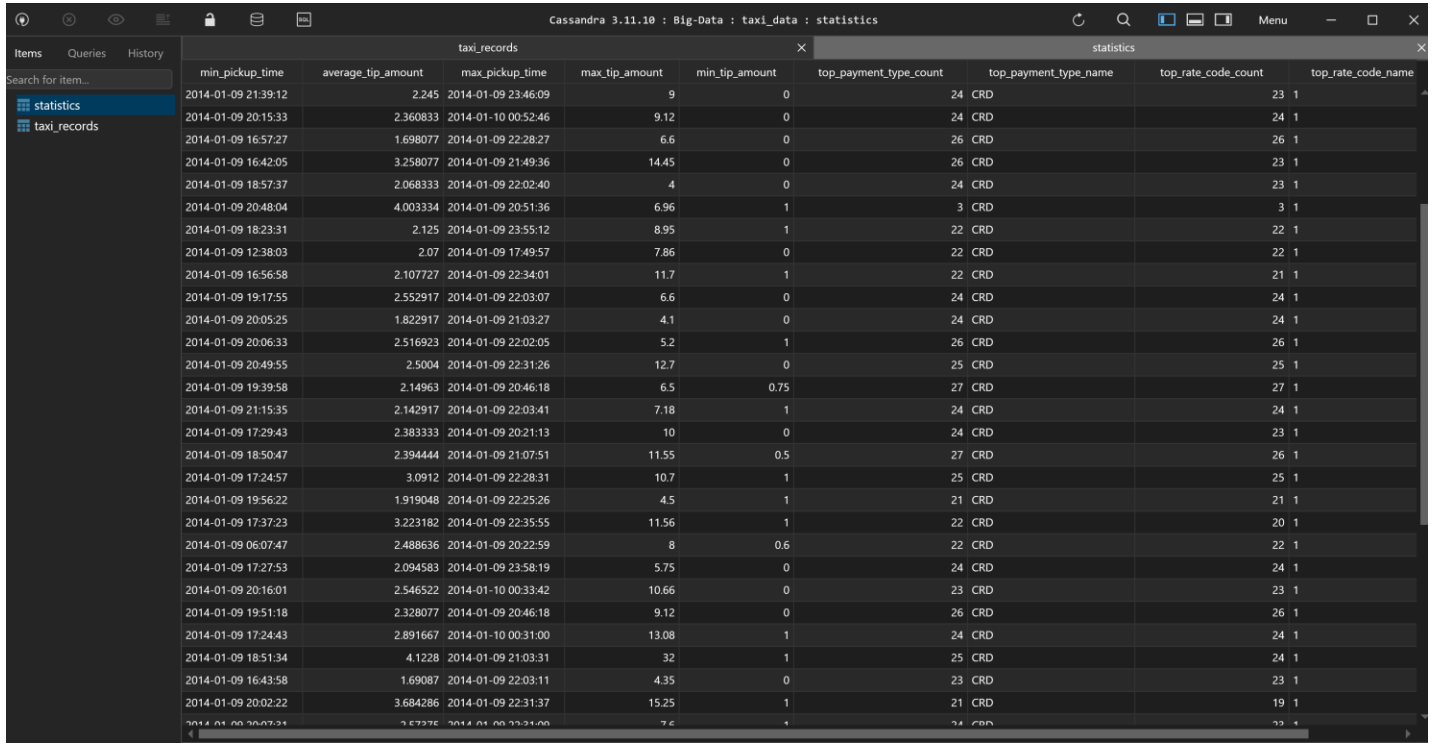
```
    // Stores some record attributes in Cassandra Db for testing purpose.

    rdd.foreach(c -> {

        TaxiRecord taxiRecord = new TaxiRecord(c.getPickupDateTime(), c.getDropoffDateTime(),
                c.getFareAmount(), c.getPickupLatitude(),
                c.getPickupLongitute(), c.getMtaTax(),
                c.getRateCode(), c.getSurcharge());
        saveTaxiRecord(taxiRecord, cassandraUrl, cassandraPort);
        System.out.println("INFO: Taxi records are stored in database.");
    });

    // Reading data from Cassandra Db.

    // readRecords(StatisticsTable, 10, cassandraUrl, cassandraPort);
    // readRecords(TaxiRecordsTable, 10, cassandraUrl, cassandraPort);
});

streamingContext.start();
streamingContext.awaitTermination();
```

Smeštanje podataka u Cassandra Db

```java
public static void saveStatisticRecord(StatisticRecord statisticRecord, String addr, Integer port) {

    CassandraConnector conn = CassandraConnector.getInstance();
    conn.connect(addr, port);
    CqlSession session = conn.getSession();


    RegularInsert insertInto = QueryBuilder.insertInto(Keyspace, StatisticsTable)
            .value("min_pickup_time", QueryBuilder.bindMarker())
            .value("max_pickup_time", QueryBuilder.bindMarker())
            .value("min_tip_amount", QueryBuilder.bindMarker())
            .value("max_tip_amount", QueryBuilder.bindMarker())
            .value("average_tip_amount", QueryBuilder.bindMarker())
            .value("trip_count", QueryBuilder.bindMarker())
            .value("top_payment_type_count", QueryBuilder.bindMarker())
            .value("top_payment_type_name", QueryBuilder.bindMarker())
            .value("top_rate_code_count", QueryBuilder.bindMarker())
            .value("top_rate_code_name", QueryBuilder.bindMarker());

    SimpleStatement insertStatement = insertInto.build();
    PreparedStatement preparedStatement = session.prepare(insertStatement);

    BoundStatement boundStatement = preparedStatement.bind()
            .setInstant(0, statisticRecord.getMinPickupTime().toInstant())
            .setInstant(1, statisticRecord.getMaxPickupTime().toInstant())
            .setFloat(2, statisticRecord.getMinTipAmount())
            .setFloat(3, statisticRecord.getMaxTipAmount())
            .setFloat(4, statisticRecord.getAverageTipAmount())
            .setLong(5, statisticRecord.getTripCount())
            .setInt(6, statisticRecord.getTopPaymentTypeCount())
            .setString(7, statisticRecord.getTopPaymentTypeName())
            .setInt(8, statisticRecord.getTopRateCodeCount())
            .setString(9, statisticRecord.getTopRateCodeName());

    session.execute(boundStatement);
    System.out.println(String.format("Data stored in %s table.", TaxiRecordsTable));

    session.close();
    conn.close();
}
```

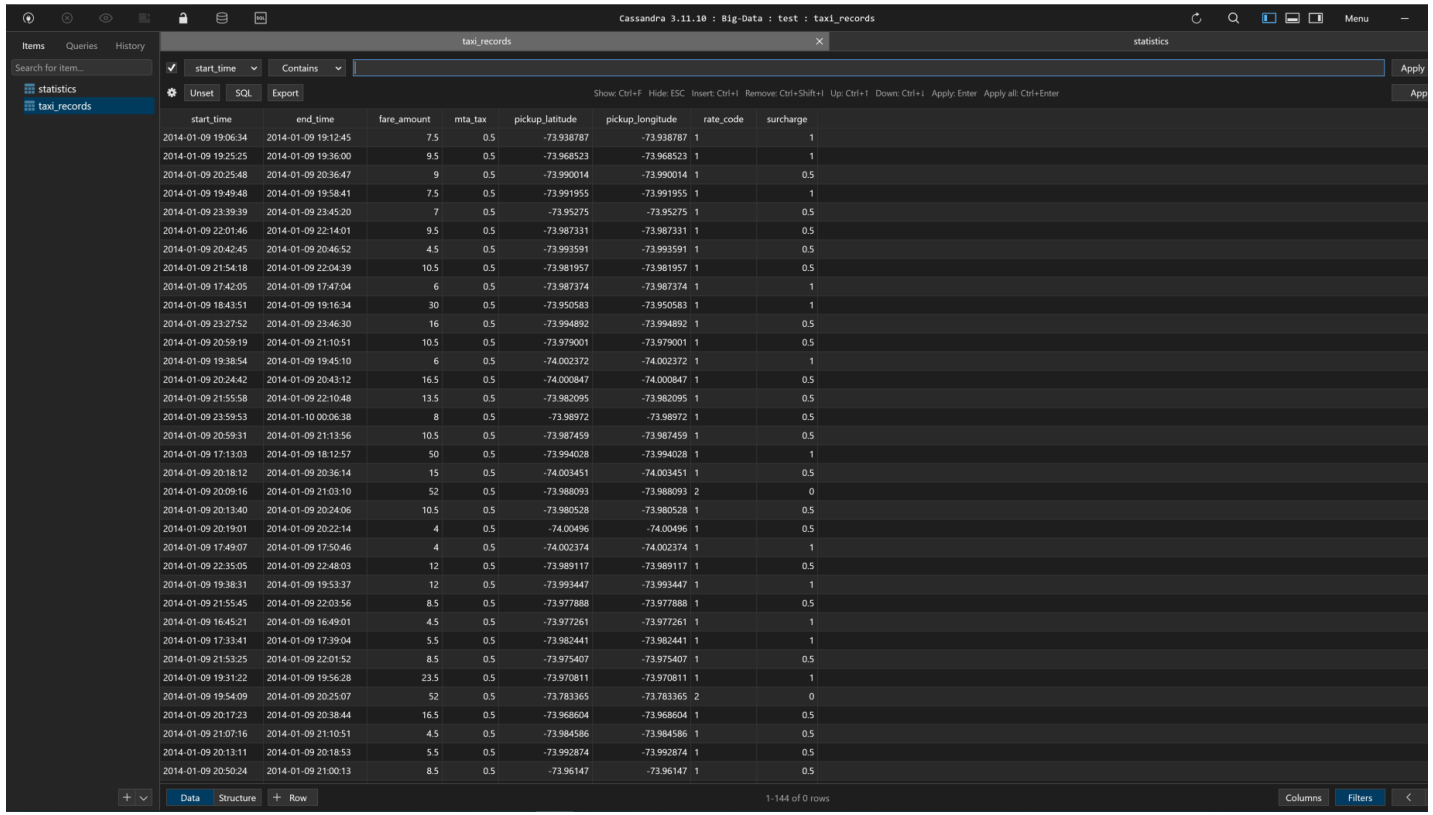
```java
public static void saveTaxiRecord(TaxiRecord taxiRecord, String addr, Integer port) {

    CassandraConnector conn = CassandraConnector.getInstance();
    conn.connect(addr, port);
    CqlSession session = conn.getSession();


    RegularInsert insertInto = QueryBuilder
        .insertInto(Keyspace, TaxiRecordsTable)
        .value("start_time", QueryBuilder.bindMarker())
        .value("end_time", QueryBuilder.bindMarker())
        .value("fare_amount", QueryBuilder.bindMarker())
        .value("pickup_latitude", QueryBuilder.bindMarker())
        .value("pickup_longitude", QueryBuilder.bindMarker())
        .value("mta_tax", QueryBuilder.bindMarker())
        .value("rate_code", QueryBuilder.bindMarker())
        .value("surcharge", QueryBuilder.bindMarker());

    SimpleStatement insertStatement = insertInto.build();
    PreparedStatement preapredStatement = session.prepare(insertStatement);

    BoundStatement boundStatement = preapredStatement.bind()
        .setInstant(0, taxiRecord.getStartTime().toInstant())
        .setInstant(1, taxiRecord.getEndTime().toInstant())
        .setFloat(2, taxiRecord.getFareAmount())
        .setDouble(3, taxiRecord.getPickupLatitude())
        .setDouble(4, taxiRecord.getPickupLongutude())
        .setFloat(5, taxiRecord.getMtaTax())
        .setString(6, taxiRecord.getRateCode())
        .setFloat(7, taxiRecord.getSurcharge());

    session.execute(boundStatement);
    conn.close();
}
```

Statistic Table (TablePlus DB Management alat)



min_pickup_time	average_tip_amount	max_pickup_time	max_tip_amount	min_tip_amount	top_payment_type_count	top_payment_type_name	top_rate_code_count	top_rate_code_name
2014-01-09 21:39:12	2.245	2014-01-09 23:46:09	9	0	24	CRD	23	1
2014-01-09 20:15:33	2.360833	2014-01-10 00:52:46	9.12	0	24	CRD	24	1
2014-01-09 16:57:27	1.698077	2014-01-09 22:28:27	6.6	0	26	CRD	26	1
2014-01-09 16:42:05	3.258077	2014-01-09 21:49:36	14.45	0	26	CRD	23	1
2014-01-09 18:57:37	2.068333	2014-01-09 22:02:40	4	0	24	CRD	23	1
2014-01-09 20:48:04	4.003334	2014-01-09 20:51:36	6.96	1	3	CRD	3	1
2014-01-09 18:23:31	2.125	2014-01-09 23:55:12	8.95	1	22	CRD	22	1
2014-01-09 12:38:03	2.07	2014-01-09 17:49:57	7.86	0	22	CRD	22	1
2014-01-09 16:56:58	2.107727	2014-01-09 22:34:01	11.7	1	22	CRD	21	1
2014-01-09 19:17:55	2.552917	2014-01-09 22:03:07	6.6	0	24	CRD	24	1
2014-01-09 20:05:25	1.822917	2014-01-09 21:03:27	4.1	0	24	CRD	24	1
2014-01-09 20:06:33	2.516923	2014-01-09 22:02:05	5.2	1	26	CRD	26	1
2014-01-09 20:49:55	2.5004	2014-01-09 22:31:26	12.7	0	25	CRD	25	1
2014-01-09 19:39:58	2.14963	2014-01-09 20:46:18	6.5	0.75	27	CRD	27	1
2014-01-09 21:15:35	2.142917	2014-01-09 22:03:41	7.18	1	24	CRD	24	1
2014-01-09 17:29:43	2.383333	2014-01-09 20:21:13	10	0	24	CRD	23	1
2014-01-09 18:50:47	2.394444	2014-01-09 21:07:51	11.55	0.5	27	CRD	26	1
2014-01-09 17:24:57	3.0912	2014-01-09 22:28:31	10.7	1	25	CRD	25	1
2014-01-09 19:56:22	1.919048	2014-01-09 22:25:26	4.5	1	21	CRD	21	1
2014-01-09 17:37:23	3.223182	2014-01-09 22:35:55	11.56	1	22	CRD	20	1
2014-01-09 06:07:47	2.488636	2014-01-09 20:22:59	8	0.6	22	CRD	22	1
2014-01-09 17:27:53	2.094583	2014-01-09 23:58:19	5.75	0	24	CRD	24	1
2014-01-09 20:16:01	2.546522	2014-01-10 00:33:42	10.66	0	23	CRD	23	1
2014-01-09 19:51:18	2.328077	2014-01-09 20:46:18	9.12	0	26	CRD	26	1
2014-01-09 17:24:43	2.891667	2014-01-10 00:31:00	13.08	1	24	CRD	24	1
2014-01-09 18:51:34	4.1228	2014-01-09 21:03:31	32	1	25	CRD	24	1
2014-01-09 16:43:58	1.69087	2014-01-09 22:03:11	4.35	0	23	CRD	23	1
2014-01-09 20:02:22	3.684286	2014-01-09 22:31:37	15.25	1	21	CRD	19	1

Taxi Record Table (TablePlus DB Management alat)



Pomoćne funkcije

```
public static void readRecords(String tableName, int limit, String addr, Integer port) {

    CassandraConnector conn = CassandraConnector.getInstance();
    conn.connect(addr, port);
    CqlSession session = conn.getSession();


    String query = String.format("SELECT * FROM %s.%s limit %d" ,Keyspace, tableName, limit);
    ResultSet result = session.execute(query);

    System.out.println(String.format("First %d records from table %s", limit, tableName));
    System.out.println(result.all());

    session.close();
    conn.close();
}
```

```
public static Map<String, Object> getKafkaParams(String brokers) {

    Map<String, Object> kafkaParams = new HashMap<String, Object>();
    kafkaParams.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers);
    kafkaParams.put(ConsumerConfig.CLIENT_ID_CONFIG, "streaming-consumer");
    kafkaParams.put(ConsumerConfig.GROUP_ID_CONFIG, "streaming-consumer");
    kafkaParams.put(ConsumerConfig.AUTO_OFFSET_RESET_DOC, "earliest");
    kafkaParams.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
    kafkaParams.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
    kafkaParams.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "latest");
    kafkaParams.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, false);

    return kafkaParams;
}
```