DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorial – Kakfa Streaming in AWS

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to introduce the fundamentals of stream processing using Kafka. Three examples taken from: **Narkhede, N., Shapira, G., & Palino, T. (2017). Kafka: the definitive guide: real-time data and stream processing at scale. " O'Reilly Media, Inc."**, and are used to illustrate (i) how to develop the source code and (ii) how to test it.

First example is a simple word count example that is used to demonstrate the map/filter pattern and simple aggregates.

Second example contains a calculation of different statistics on stock market trades, which will allow us to demonstrate window aggregations.

Third example demonstrates the streaming joins.

The following contents is presented in this document.

Contents

A. Installing git in AWS EC2

In command line:

```
sudo yum install git
```

Check installation:

```
git --version
git version 2.47.1 (or later)
```

B. Installing mvn in AWS EC2

1. Create an EC2 Amazon linux image instance

2. Check that JDK 8 is available in the new created instance with the following commands:

```
[ec2-user@ip---- ~]$ java -version
openjdk version "17.0.14" 2025-01-21 LTS
OpenJDK Runtime Environment Corretto-17.0.14.7.1 (build 17.0.14+7-LTS)
OpenJDK 64-Bit Server VM Corretto-17.0.14.7.1 (build 17.0.14+7-LTS, mixed mode, sharing)
[ec2-user@ip-172-31-85-55 ~]$ javac -version
javac 17.0.14
```

Hint 1: if not available, use the following commands or similar:

```
[ec2-user@ip- ~]$ yum search java-17
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
================================================================================ N/S matched: java-
17 =================================================================================
java-17-amazon-corretto.x86_64 : Amazon Corretto development environment
java-17-amazon-corretto-debugsymbols.x86_64 : Amazon Corretto 17 zipped debug symbols
java-17-amazon-corretto-devel.x86_64 : Amazon Corretto 17 development tools
java-17-amazon-corretto-headless.x86_64 : Amazon Corretto headless development environment
java-17-amazon-corretto-javadoc.x86_64 : Amazon Corretto 17 API documentation
java-17-amazon-corretto-jmods.x86_64 : Amazon Corretto 17 jmods

[ec2-user@ip- ~]$ sudo yum -y install java-17-amazon-corretto-devel.x86_64
```

3. Execute the following commands in the EC2 instance:

```
[ec2-user@ip- - - -  ~]$ sudo wget https://repo.maven.apache.org/maven2/org/apache/maven/apache-
maven/3.9.9/apache-maven-3.9.9-bin.tar.gz

[ec2-user@ip- - - -  ~]$ tar xzvf apache-maven-3.9.9-bin.tar.gz

[ec2-user@ip- - - -  ~]$ cd apache-maven-3.9.9/bin

[ec2-user@ip- - - -  bin]$ export PATH=/home/ec2-user/apache-maven-3.9.9/bin:$PATH

[ec2-user@ip- - - -  ~]$ mvn -version
Maven home: /home/ec2-user/apache-maven-3.9.9
Java version: 17.0.14, vendor: Amazon.com Inc., runtime: /usr/lib/jvm/java-17-amazon-corretto.x86 64
Default locale: en_US, platform encoding: ANSI_X3.4-1968
OS name: "linux", version: "5.10.235-227.919.amzn2.x86_64", arch: "amd64", family: "unix"
```

Hint 2: To automate the maven path with instance login update the export command in the file:

```
.bash_profile
```

C. Testing the Word count example

C.1. Create working directory

```
mkdir WordCount
```

C.2. Change to working directory

```
cd WordCount
```

C.3. Clone the source code from git

```
[ec2-user@ip- WordCount]$ git clone https://github.com/gwenshap/kafka-streams-wordcount
Cloning into 'kafka-streams-wordcount'...
remote: Enumerating objects: 57, done.
remote: Total 57 (delta 0), reused 0 (delta 0), pack-reused 57
Unpacking objects: 100% (57/57), done.
```

C.4. Change directory

```
cd kafka-streams-wordcount/
```

C.5. Change pom.xml from:

```
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>0.10.3.0-SNAPSHOT</version>
</dependency>
```

To a most recent library[1]:

```
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>3.2.0</version>
</dependency>
```

C.6. Study the following source code that is also located in:

```
/home/ec2-user/WordCount/kafka-streams-wordcount/src/main/java/com/shapira/examples/streams/wordcount/
WordCountExample.java
```

```
1   package com.shapira.examples.streams.wordcount;


2   import org.apache.kafka.clients.CommonClientConfigs;
3   import org.apache.kafka.clients.consumer.ConsumerConfig;
4   import org.apache.kafka.common.serialization.Serdes;
5   import org.apache.kafka.streams.KafkaStreams;
6   import org.apache.kafka.streams.KeyValue;
7   import org.apache.kafka.streams.StreamsConfig;
8   import org.apache.kafka.streams.kstream.KStream;
9   import org.apache.kafka.streams.kstream.KStreamBuilder;

10  import java.util.Arrays;
11  import java.util.Properties;
12  import java.util.regex.Pattern;

13  public class WordCountExample {

14  public static void main(String[] args) throws Exception{

15  Properties props = new Properties();
16  props.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount");
17  props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
18  props.put(StreamsConfig.KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
19  props.put(StreamsConfig.VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
```

[1] The most recent libraries should be consulted at https://mvnrepository.com/

```
20  // setting offset reset to earliest so that we can re-run the demo code with the same pre-loaded data
21  // Note: To re-run the demo, you need to use the offset reset tool:
22  // https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+Application+Reset+Tool
23  props.put(ConsumerConfig.AUTO OFFSET RESET CONFIG, "earliest");

24  // work-around for an issue around timing of creating internal topics
25  // Fixed in Kafka 0.10.2.0
26  // don't use in large production apps - this increases network load
27  // props.put(CommonClientConfigs.METADATA MAX AGE CONFIG, 500);

28  KStreamBuilder builder = new KStreamBuilder();

29  KStream<String, String> source = builder.stream("wordcount-input");


30  final Pattern pattern = Pattern.compile("\\W+");
31  KStream counts  = source.flatMapValues(value-> Arrays.asList(pattern.split(value.toLowerCase())))
32  .map((key, value) -> new KeyValue<Object, Object>(value, value))
33  .filter((key, value) -> (!value.equals("the")))
34  .groupByKey()
35  .count("CountStore").mapValues(value->Long.toString(value)).toStream();
36  counts.to("wordcount-output");

37  KafkaStreams streams = new KafkaStreams(builder, props);

38  // This is for reset to work. Don't use in production - it causes the app to re-load the state from Kafka
    on every start
39  streams.cleanUp();

40  streams.start();

41  // usually the stream application would be running forever,
42  // in this example we just let it run for some time and stop since the input data is finite.
43  Thread.sleep(5000L);

44  streams.close();

45  }
46  }
```

C.7. Change the location of your kafka server in the java source code:

```
/home/ec2-user/WordCount/kafka-streams-wordcount/src/main/java/com/shapira/examples/streams/wordcount/
WordCountExample.java
```

In the following source code line:

```
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "<YOUR_DNS_NAME>:9092");
```

C.8. Build the project with mvn package, this will generate an uber-jar with the streams app and all its dependencies.

```
mvn package
```

C.9. Create a wordcount-input topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server <YOUR DNS NAME>:9092 --create --topic wordcount-input
--partitions 1 --replication-factor 1
```

C.10. Produce some text to the topic. Don't forget to repeat words (so we can count higher than 1) and to use the word "the", so we can filter it.

```
/usr/local/kafka/bin/kafka-console-producer.sh --broker-list <YOUR_DNS_NAME>:9092 --topic wordcount-input
```

C.11. Run the app:

```
java -cp target/uber-kafka-streams-wordcount-1.0-SNAPSHOT.jar
com.shapira.examples.streams.wordcount.WordCountExample
```

C.12. View the results obtained:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --topic wordcount-output --from-beginning --bootstrap-server
<YOUR_DNS_NAME>:9092 --property print.key=true


You will receive something similar with:
um      1
este    2
outro   1
testte  1teste  1
tets    1
test    4
        1
a       1
```

C.13. If you want to reset state and re-run the application (maybe with some changes?) on existing input topic, you can stop the kafka producer and consumer and then:

Reset internal topics (used for shuffle the topic and state-stores):

```
/usr/local/kafka/bin/kafka-streams-application-reset.sh --application-id wordcount --bootstrap-servers
<YOUR_DNS_NAME>:9092 --input-topics wordcount-input
```

(optional) Delete the output topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server <YOUR DNS NAME>:9092 --delete --topic wordcount-
output
```

D. Testing the Stock Market Statistics example

D.1. Create working directory

```
mkdir StockMarketStatistics
```

D.2. Change to working directory

```
cd StockMarketStatistics
```

D.3. Clone the source code from git

```
[ec2-user@ip- StockMarketStatistics]$ git clone https://github.com/gwenshap/kafka-streams-stockstats
Cloning into 'kafka-streams-stockstats'...
remote: Enumerating objects: 173, done.
remote: Total 173 (delta 0), reused 0 (delta 0), pack-reused 173
Receiving objects: 100% (173/173), 27.77 KiB | 3.08 MiB/s, done.
Resolving deltas: 100% (45/45), done.
```

D.4. Change pom.xml from:

```
    <properties>
        <kafka.version>2.8.0</kafka.version>
        <confluent.version>6.1.0</confluent.version>
        <avro.version>1.8.2</avro.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
```

To a most recent library[2]:

```
    <properties>
        <kafka.version>3.2.0</kafka.version>
        <confluent.version>6.1.0</confluent.version>
        <avro.version>1.8.2</avro.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
```

D.5. Study the following source code that is also located in:

```
/home/ec2-user/StockMarketStatistics/kafka-streams-stockstats/src/main/java/com/Shapira/examples/streams/
stockstats/StockStatsExample.java
```

```
1   package com.shapira.examples.streams.stockstats;

2   import com.shapira.examples.streams.stockstats.serde.JsonDeserializer;
3   import com.shapira.examples.streams.stockstats.serde.JsonSerializer;
4   import com.shapira.examples.streams.stockstats.serde.WrapperSerde;
5   import com.shapira.examples.streams.stockstats.model.Trade;
6   import com.shapira.examples.streams.stockstats.model.TradeStats;
7   import org.apache.kafka.clients.admin.AdminClient;
8   import org.apache.kafka.clients.admin.DescribeClusterResult;
9   import org.apache.kafka.clients.consumer.ConsumerConfig;
10  import org.apache.kafka.common.serialization.Serdes;
11  import org.apache.kafka.common.utils.Bytes;
12  import org.apache.kafka.streams.KafkaStreams;
13  import org.apache.kafka.streams.StreamsConfig;
14  import org.apache.kafka.streams.Topology;
15  import org.apache.kafka.streams.kstream.KStream;
16  import org.apache.kafka.streams.StreamsBuilder;
17  import org.apache.kafka.streams.kstream.Materialized;
18  import org.apache.kafka.streams.kstream.Produced;
19  import org.apache.kafka.streams.kstream.TimeWindows;
20  import org.apache.kafka.streams.kstream.Windowed;
21  import org.apache.kafka.streams.kstream.WindowedSerdes;
22  import org.apache.kafka.streams.state.WindowStore;
```

[2] The most recent libraries should be consulted at https://mvnrepository.com/

```
23 import java.util.Properties;

24 /**
   Input is a stream of trades
   Output is two streams: One with minimum and avg "ASK" price for every 10 seconds window
   Another with the top-3 stocks with lowest minimum ask every minute
25 */
26 public class StockStatsExample {

27 public static void main(String[] args) throws Exception {

28 Properties props;
29 if (args.length==1)
30 props = LoadConfigs.loadConfig(args[0]);
31 else
32 props = LoadConfigs.loadConfig();

33 props.put(StreamsConfig.APPLICATION_ID_CONFIG, "stockstat-2");
34 props.put(StreamsConfig.DEFAULT KEY SERDE CLASS CONFIG, Serdes.String().getClass().getName());
35 props.put(StreamsConfig.DEFAULT VALUE SERDE CLASS CONFIG, TradeSerde.class.getName());

36 // setting offset reset to earliest so that we can re-run the demo code with the same pre-loaded data
37 // Note: To re-run the demo, you need to use the offset reset tool:
38 // https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+Application+Reset+Tool
39 props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

40 // creating an AdminClient and checking the number of brokers in the cluster, so I'll know how many
   replicas we want...

41 AdminClient ac = AdminClient.create(props);
42 DescribeClusterResult dcr = ac.describeCluster();
43 int clusterSize = dcr.nodes().get().size();

44 if (clusterSize<3)
45 props.put("replication.factor",clusterSize);
46 else
47 props.put("replication.factor",3);

48 StreamsBuilder builder = new StreamsBuilder();

49 KStream<String, Trade> source = builder.stream(Constants.STOCK_TOPIC);

50 KStream<Windowed<String>, TradeStats> stats = source
51 .groupByKey()
52 .windowedBy(TimeWindows.of(5000).advanceBy(1000))
53 .<TradeStats>aggregate(() -> new TradeStats(),(k, v, tradestats) -> tradestats.add(v),
54 Materialized.<String, TradeStats, WindowStore<Bytes, byte[]>>as("trade-aggregates")
55 .withValueSerde(new TradeStatsSerde()))
56 .toStream()
57 .mapValues((trade) -> trade.computeAvgPrice());

58 stats.to("stockstats-output", Produced.keySerde(WindowedSerdes.timeWindowedSerdeFrom(String.class)));

59 Topology topology = builder.build();

60 KafkaStreams streams = new KafkaStreams(topology, props);

61 System.out.println(topology.describe());

62 streams.cleanUp();

63 streams.start();

64 // Add shutdown hook to respond to SIGTERM and gracefully close Kafka Streams
65 Runtime.getRuntime().addShutdownHook(new Thread(streams::close));

66 }

67 static public final class TradeSerde extends WrapperSerde<Trade> {
68 public TradeSerde() {
69 super(new JsonSerializer<Trade>(), new JsonDeserializer<Trade>(Trade.class));
70 }
71 }

72 static public final class TradeStatsSerde extends WrapperSerde<TradeStats> {
73 public TradeStatsSerde() {
74 super(new JsonSerializer<TradeStats>(), new JsonDeserializer<TradeStats>(TradeStats.class));
75 }
76 }

77 }
```

D.6. Change directory

```
cd kafka-streams-stockstats
```

D.7. Build the project with mvn package, this will generate an uber-jar with the streams app and all its dependencies.

```
mvn package
```

D.8. Change to working directory

```
cd kafka-streams-stockstats
```

D.9. Create a stocks input topic and output topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic stocks --partitions 1
--replication-factor 1

/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic stockstats-output --
partitions 1 --replication-factor 1
```

D.10. Create the configuration file, next to you JAVA project, with the command:

```
cat > configfile << EOF
```

and then write the following content to the file directly in the command line:

```
bootstrap.servers = PLAINTEXT://<YOUR DNS NAME>:9092
EOF
```

D.11. Generate some trades so we can analyze them. Start running the trades producer and stop it with ctrl-c when you think there's enough data:

```
java -cp target/uber-kafka-streams-stockstats-1.1-SNAPSHOT.jar -DLOGLEVEL=INFO
com.shapira.examples.streams.stockstats.StockGenProducer configfile
```

D.12. Run the streams app:

```
java -cp target/uber-kafka-streams-stockstats-1.1-SNAPSHOT.jar -DLOGLEVEL=INFO
com.shapira.examples.streams.stockstats.StockStatsExample configfile
```

D.13. Check the results:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --topic stockstats-output --from-beginning --bootstrap-server
<YOUR_DNS_NAME>:9092 --property print.key=true
```

D.14. If you want to reset state and re-run the application (maybe with some changes?) on existing input topic, you can stop the kafka producer and consumer and then:

Reset internal topics (used for shuffle the topic and state-stores):

```
/usr/local/kafka/bin/kafka-streams-application-reset.sh --application-id wordcount --bootstrap-servers
<YOUR_DNS_NAME>:9092 --input-topics stocks
```

(optional) Delete the output topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server <YOUR_DNS_NAME>:9092 --delete --topic stockstats-
output
```

E. Testing the click Stream Enrichment example

E.1. Proceed accordingly with the previous examples:

```
[ec2-user@ip- ~]$ mkdir StreamEnrichment

[ec2-user@ip- ~]$ cd StreamEnrichment/

[ec2-user@ip- StreamEnrichment]$ git clone https://github.com/gwenshap/kafka-clickstream-enrich
Cloning into 'kafka-clickstream-enrich'...
remote: Enumerating objects: 65, done.
remote: Total 65 (delta 0), reused 0 (delta 0), pack-reused 65
Unpacking objects: 100% (65/65), done.

[ec2-user@ip- StreamEnrichment]$ cd kafka-clickstream-enrich
```

E.2. Study the following source code that is also located in:

```
/home/ec2-user/StreamEnrichment/kafka-clickstream-enrich/src/main/java/com/shapira/examples/streams/
clickstreamenrich/ClickstreamEnrichment.java
```

```
1   package com.shapira.examples.streams.clickstreamenrich;
2   import com.shapira.examples.streams.clickstreamenrich.model.PageView;
3   import com.shapira.examples.streams.clickstreamenrich.model.Search;
4   import com.shapira.examples.streams.clickstreamenrich.model.UserActivity;
5   import com.shapira.examples.streams.clickstreamenrich.model.UserProfile;
6   import com.shapira.examples.streams.clickstreamenrich.serde.JsonDeserializer;
7   import com.shapira.examples.streams.clickstreamenrich.serde.JsonSerializer;
8   import com.shapira.examples.streams.clickstreamenrich.serde.WrapperSerde;
9   import org.apache.kafka.clients.consumer.ConsumerConfig;
10  import org.apache.kafka.common.serialization.Serdes;
11  import org.apache.kafka.streams.KafkaStreams;
12  import org.apache.kafka.streams.StreamsConfig;
13  import org.apache.kafka.streams.kstream.JoinWindows;
14  import org.apache.kafka.streams.kstream.KStream;
15  import org.apache.kafka.streams.kstream.KStreamBuilder;
16  import org.apache.kafka.streams.kstream.KTable;

17  import java.util.Properties;

18  public class ClickstreamEnrichment {

19  public static void main(String[] args) throws Exception {

1   Properties props = new Properties();
2   props.put(StreamsConfig.APPLICATION_ID_CONFIG, "clicks");
3   props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, Constants.BROKER);
4   // Since each step in the stream will involve different objects, we can't use default Serde

5   // setting offset reset to earliest so that we can re-run the demo code with the same pre-loaded data
6   // Note: To re-run the demo, you need to use the offset reset tool:
7   // https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+Application+Reset+Tool
8   props.put(ConsumerConfig.AUTO OFFSET RESET CONFIG, "earliest");

9   // work-around for an issue around timing of creating internal topics
10  // this was resolved in 0.10.2.0 and above
11  // don't use in large production apps - this increases network load
12  // props.put(CommonClientConfigs.METADATA_MAX_AGE_CONFIG, 500);

13  KStreamBuilder builder = new KStreamBuilder();

14  KStream<Integer, PageView> views = builder.stream(Serdes.Integer(), new PageViewSerde(),
    Constants.PAGE_VIEW_TOPIC);
15  KTable<Integer, UserProfile> profiles = builder.table(Serdes.Integer(), new ProfileSerde(),
    Constants.USER PROFILE TOPIC, "profile-store");
16  KStream<Integer, Search> searches = builder.stream(Serdes.Integer(), new SearchSerde(),
    Constants.SEARCH_TOPIC);

17  KStream<Integer, UserActivity> viewsWithProfile = views.leftJoin(profiles,
18  (page, profile) -> {
19  if (profile != null)
```

```
20  return new UserActivity(profile.getUserID(), profile.getUserName(), profile.getZipcode(),
    profile.getInterests(), "", page.getPage());
21  else
22  return new UserActivity(-1, "", "", null, "", page.getPage());
23  });

24  KStream<Integer, UserActivity> userActivityKStream = viewsWithProfile.leftJoin(searches,
25  (userActivity, search) -> {
26  if (search != null)
27  userActivity.updateSearch(search.getSearchTerms());
28  else
29  userActivity.updateSearch("");
30  return userActivity;
31  },
32  JoinWindows.of(1000), Serdes.Integer(), new UserActivitySerde(), new SearchSerde());

33  userActivityKStream.to(Serdes.Integer(), new UserActivitySerde(), Constants.USER_ACTIVITY_TOPIC);

34  KafkaStreams streams = new KafkaStreams(builder, props);

35  streams.cleanUp();

36  streams.start();

37  // usually the stream application would be running forever,
38  // in this example we just let it run for some time and stop since the input data is finite.
39  Thread.sleep(60000L);

40  streams.close();


41  }

42  static public final class PageViewSerde extends WrapperSerde<PageView> {
43  public PageViewSerde() {
44      super(new JsonSerializer<PageView>(), new JsonDeserializer<PageView>(PageView.class));
45  }
46  }

47  static public final class ProfileSerde extends WrapperSerde<UserProfile> {
48  public ProfileSerde() {
49      super(new JsonSerializer<UserProfile>(), new JsonDeserializer<UserProfile>(UserProfile.class));
50  }
51  }

52  static public final class SearchSerde extends WrapperSerde<Search> {
53  public SearchSerde() {
54      super(new JsonSerializer<Search>(), new JsonDeserializer<Search>(Search.class));
55  }
56  }

57  static public final class UserActivitySerde extends WrapperSerde<UserActivity> {
58  public UserActivitySerde() {
59       super(new JsonSerializer<UserActivity>(), new JsonDeserializer<UserActivity>(UserActivity.class));
60  }
61  }
62  }
```

E.3. Change pom.xml from:

```
<dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka-clients</artifactId>
       <version>0.10.3.0-SNAPSHOT</version>
</dependency>
<dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka-streams</artifactId>
       <version>0.10.3.0-SNAPSHOT</version>
</dependency>
```

To a most recent library:

```
<dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka-clients</artifactId>
```

```
        <version>3.2.0</version>
</dependency>
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>3.2.0</version>
</dependency>
```

E.4. Change the location of your kafka server in the java source code:

```
/home/ec2-user/StreamEnrichment/kafka-clickstream-
enrich/src/main/java/com/shapira/examples/streams/clickstreamenrich/Constants.java
```

In the line 8:

```
public static final String BROKER = "<YOUR_DNS_NAME>:9092";
```

E.5. Build the project with mvn package:

```
mvn package
```

E.6. Generate some clicks, searches, and profiles. Run the generator. It should take about 5 seconds to run. Don't worry about complete lack of output:

```
java -cp target/uber-kafka-clickstream-enrich-1.0-SNAPSHOT.jar
com.shapira.examples.streams.clickstreamenrich.GenerateData
```

E.7. Run the streams app:

```
java -cp target/uber-kafka-clickstream-enrich-1.0-SNAPSHOT.jar
com.shapira.examples.streams.clickstreamenrich.ClickstreamEnrichment
```

E.8. Check the results:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --topic clicks.user.activity --from-beginning --bootstrap-
server <YOUR_DNS_NAME>:9092 --property print.key=true
```

E.9. If you want to reset state and re-run the application (maybe with some changes?) on existing input topic, you can stop the kafka producer and consumer and then:

Reset internal topics (used for shuffle the topic and state-stores):

```
/usr/local/kafka/bin/kafka-streams-application-reset.sh --application-id clicks --bootstrap-servers
<YOUR_DNS_NAME>:9092 --input-topics clicks.user.profile,clicks.pages.views,clicks.search
```

(optional) Delete the output topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server <YOUR_DNS_NAME>:9092 --delete --topic
clicks.user.activity
```

F. Other tools and references

- If you prefer, install emacs in AWS EC2 using the following command:

```
sudo yum install emacs
```

- Where to download apache maven https://maven.apache.org/download.cgi
- How to install apache maven https://maven.apache.org/install.html