BIG DATA DAY LA

Los Angeles, California
August 5th 2017

Building Streaming Data Applications Using Apache Kafka

kafka

Slim Baltagi

ADVANCED ANALYTICS

Agenda

1. **A typical streaming data application**
2. Apache Kafka as a platform for building and running streaming data applications
3. Code and demo of an end-to-end Kafka-driven streaming data application

Batch data	Streaming data

1. A typical Streaming Data Application

A very simplified diagram!

Agenda

1. A typical streaming data application
2. **Apache Kafka as a platform for building and running streaming data applications**
3. Code and demo of an end-to-end Kafka-driven streaming data application

2. Apache Kafka as a platform for building and running streaming data applications

- **Apache Kafka** is an open source **streaming data platform** (a new category of software!)
  - to **import** event streams from other source data systems into Kafka and **export** event streams from **Kafka** to destination data systems
  - to **transport** and **store** event streams
  - to **process** event streams live as they occur.

2.1 Kafka Core: Event Streams Transport and Storage

2.1.1 What is Kafka Core?
2.1.2 Before Kafka Core?
2.1.3 Why Kafka Core?

2.1.1 What is Kafka Core?

- **Kafka** is a **software written in Scala and Java** and originally developed by Linkedin in **2010**.
- It was open sourced as an apache project in 2011 and became a Top Level Project in 2012.
- After 7 years, it is graduating to version **1.0** in **October 2017!!**
- **Kafka Core** is an **enterprise messaging system** to:
  - **publish** event streams
  - **subscribe to** event streams
  - **store** event streams
- Kafka Core is the the '**digital nervous system**' connecting all enterprise data and systems of many notable companies.
- **Diverse** and **rapidly growing** user base across **many industries** and **verticals**.

2.1.2 Before Kafka Core?

- **Before Kafka Core**, Linkedin had to build **many custom data pipelines**, for streaming and queueing data, that use **point to point communication** and need to be constantly **scaled individually**.

Total connections = N producers * M consumers

2.1.2 Before Kafka Core?

- **Traditional enterprise message systems** such as RabbitMQ, Apache ActiveMQ, IBM WebSphere MQ, TIBCO EMS **could not help** because of these **limitations:**
  - They can't accommodate **the web-scale requirements** of Linkedin
  - **Producers and consumers** are really **coupled** from a **performance perspective** because of the 'slow consumer problem'.
  - **Messages** are sent into a central message spool and stored only until they are **processed**, acknowledged and then they are **deleted**.
- **Linkedin had to create a new tool** as it could not leverage traditional enterprise message systems because of their limitations.

2.1.3 Why Kafka Core?

- **With Kafka Core**, Linkedin built a **central hub** to host all of its event streams, a **universal data pipeline** and **asynchronous services**.

Total connections = N producers + M consumers

2.1.3 Why Kafka Core?

- Apache Kafka is modeled as an **append only distributed log** which is suitable to model event streams.
- Apache Kafka comes with **out-the-box features** such as:
  - **High throughput**
  - **Low latency**
  - **Distributed** - Horizontal scaling
  - Support for **multiple consumers**
  - **Configurable persistence**
  - **Automatic recovery** from failure
  - **Polyglot ready** with its support for many languages
  - **Security**: support for encrypted data transfer

2.2 Kafka Connect: Event Import and Export

2.2.1 What is Kafka Connect?
2.2.2 Before Kafka Connect?
2.2.3 Why Kafka Connect?

2.2.1 What is Kafka Connect?

- **Kafka Connect** is a **framework, included** in **Apache Kafka** since Kafka 0.9 release on November 24th 2015, to rapidly stream events:
  - from external data systems into Kafka
  - out of Kafka to external data systems.
- Ready to use pre-built **Kafka connectors**
- **REST service** to define and manage Kafka connectors
- **Runtime** to run Kafka connectors in standalone or distributed mode
- **Java API** to build custom Kafka connectors

FTP
http://
CSV
HDFS
HIVE
KUDU
MQTT.ORG
splunk>
LOG
FLOGO
elastic
Solr
jmx
syncsort
10 01
Dbvisit
MySQL
Google BigQuery
CoAP
apache Ignite
AMQP
Advanced Message Queuing Protocol
ATTUNITY
hazelcast
ORACLE
GOLDEN GATE
Amazon SQS
slack
JDBC
PostgreSQL
JMS
PubNub
mongoDB
mixpanel
Jenkins
GitHub
IRC
Amazon DynamoDB
Bloomberg
striim
Couchbase
BLOCKCHAIN
APACHE
HBASE
INFOBRIGHT
OPENTSDB
kinetica
YAHOO!
FINANCE
EdgeIntelligence
SAP
HANA
cassandra
VOLTDB
AEROSPIKE
druid
VERTICA
RethinkDB
MarkLogic

2.2.2 Before Kafka Connect?

- Before Kafka Connect, to import data from other systems to Kafka or to export data from Kafka to other systems, you have 4 options:
  - **Option 1**: **Build your own Do It Yourself (DIY) solution:** custom code using the Kafka producer API or the Kafka consumer API.
  - **Option 2:** **Use one of the many existing tools** such as Linkedin Camus/Gobblin for Kafka to HDFS export, Flume, Sqoop, Logstash, Apache Nifi, StreamSets, ETL tool such as Talend, Pentaho, …
  - **Option 3:** **Use stream processors** to import data to Kafka or export it from Kafka! Example: Storm, Spark Streaming, Flink, Samza, …
  - **Option 4:** Use Confluent **REST Proxy API** (open source project maintained by Confluent) to read and write data to Kafka
- Each one of the 4 options above to import/export data to Kafka has its own **advantages** and **disadvantages.**

2.2.3 Why Kafka Connect?

- Using the Kafka Connect framework to stream data in and out of Kafka has the following advantages:
  - **alleviates the burden** of writing custom code or learning and integrating with a new tool to stream data in and out of Kafka for each data system!
  - **use pre-built Kafka connectors** to a **variety of data systems** just by **writing configuration files** and submitting them to Connect with minimal or no code necessary
  - **Out-of-the-box features** such as auto recovery, auto failover, automated load balancing, dynamic scaling, exactly-once delivery guarantees, …
  - **Out-of-the box integration** with the **Schema Registry** to capture schema information from sources if it is present
  - **enables to build custom Kafka connectors** leveraging the Kafka Connect framework

2.3 Kafka Streams: Event processing

2.3.1 What is Kafka Streams?
2.3.2 Before Kafka Streams?
2.3.3 Why Kafka Streams?

2.3.1 What is Kafka Streams?

- Kafka Streams is a **lightweight open source Java library, included** in Apache **Kafka** since 0.10 release in May 2016, for building stream processing applications on top of Apache Kafka.
- Kafka Streams is specifically designed to **consume** from & **produce data** to **Kafka topics**.
- A **high-level and declarative API** for common patterns like filter, map, aggregations, joins, stateful and stateless processing.
- A **low-level and imperative API** for building topologies of processors, streams and tables.

2.3.2 Before Kafka Streams?

- Before Kafka Streams, to process the data in Kafka you have 4 options:
  - **Option 1: Dot It Yourself (DIY)** – Write your own 'stream processor' using Kafka client libs, typically with a narrower focus.
  - **Option 2: Use a library** such as AkkaStreams-Kafka, also known as Reactive Kafka, RxJava, or Vert.x
  - **Option 3:** Use an **existing open source stream processing framework** such as Apache Storm, Spark Streaming, Apache Flink or Apache Samza for transforming and combining data streams which live in Kafka.
  - **Option 4**: Use an **existing commercial tool** for stream processing with adapter to Kafka such as IBM InfoSphere Streams, TIBCO StreamBase, …
- Each one of the 4 options above of processing data in Kafka has advantages and disadvantages.

2.3.3 Why Kafka Streams?

- Processing data in Kafka with Kafka Streams has the following advantages:
  - **No need to learn another framework or tool** for stream processing as Kafka Streams is already a library included in Kafka
  - **No need of external infrastructure** beyond Kafka. Kafka is already your cluster!
  - **Operational simplicity** obtained by getting rid of an additional stream processing cluster.
  - Kafka Streams **inherits operational characteristics** (low latency, elasticity, fault-tolerance, …) **from Kafka**.
  - **Low barrier to entry**: You can quickly write and run a small-scale proof-of-concept on a single machine

2.3.3 Why Kafka Streams?

- As a normal library, Kafka Streams is **easier to compose** with **other Java libraries** and **integrate** with your **existing applications and services**
- Kafka Streams **runs in your application code** and imposes no change in the Kafka cluster infrastructure, or within Kafka.
- Kafka Streams comes with **abstractions** and **features** for easier and efficient processing of event streams:
  - **KStream** and **KTable** as the two basic abstractions and there is a duality between them:
    - KStream = immutable log
    - KTable = mutable materialized view
  - **Interactive Queries**: Local queryable state is a fundamental primitive in Kafka Streams

2.3.3 Why Kafka Streams?

- **Exactly-One semantics** and **local transactions:**
- **Time** as a critical aspect in stream processing and how it is modeled and integrated: **Event** time, **Ingestion** time, **Processing** time.
- **Windowing** to control how to group records that have the same key for **stateful** operations such as **aggregations** or **joins** into so-called windows.

Agenda

1. A typical streaming data application
2. Apache Kafka as a platform for building and running streaming data applications
3. **Code and demo of an end-to-end Kafka-driven streaming data application**

3. Code and Demo of an end-to-end Streaming Data Application using Kafka

3.1 Scenario of this demo
3.2 Architecture of this demo
3.3 Setup of this demo
3.4 Results of this demo
3.5 Stopping the demo!

3.1. Scenario of this demo

- This demo consists of:
  - reading live stream of data (tweets) from Twitter using **Kafka Connect** connector for Twitter
  - storing them in Kafka broker leveraging **Kafka Core** as publish-subscribe message system.
  - performing some basic stream processing on tweets in Avro format from a Kafka topic using **Kafka Streams** library to do the following:
    - **Raw word count** - every occurrence of individual words is counted and written to the **topic** ***wordcount*** (a predefined list of stopwords will be ignored)
    - **5-Minute word count** - words are counted per 5 minute window and every word that has more than 3 occurrences is written to the **topic** ***wordcount5m***
    - **Buzzwords** - a list of special interest words can be defined and those will be tracked in the **topic** ***buzzwords***

3.1. Scenario of this demo

- This demo is **adapted** from one that was given by Sönke Liebau on July 27th 2016 from OpenCore, Germany. See blog entry titled: 'Processing Twitter Data with Kafka Streams" http://www.opencore.com/blog/2016/7/kafka-streams-demo/ and related code at GitHub https://github.com/opencore/kafkastreamsdemo
- What is specific to this demo :
  - Use of a **Docker container** instead of the confluent platform they are providing with their Virtual Machine defined in Vagrant.
  - Use of **Kafka Connect UI** from Landoop for easy and fast configuration of Twitter connector and also other Landoop's Fast Data Web UIs.

3.2. Architecture of this demo

3.3. Setup of this demo

Step 1: Setup your Kafka Development Environment

Step 2: Get twitter credentials to connect to live data

Step 3: Get twitter live data into Kafka broker

Step 4: Write and test the application code in Java

Step 5: Run the application

Step 1: Setup your Kafka Development Environment

- The easiest way to get up and running quickly is to use a **Docker container** with all components needed.
- **First,** install Docker on your desktop or on the cloud https://www.docker.com/products/overview and start it

Step 1: Setup your Kafka Development Environment

- **Second**, install **Fast-data-dev**, a **Docker image for Kafka** developers which is packaging:
  - **Kafka broker**
  - **Zookeeper**
  - **Open source version of the Confluent Platform** with its Schema registry, REST Proxy and bundled connectors
  - Certified DataMountaineer **Connectors** (ElasticSearch, Cassandra, Redis, ..)
  - Landoop's Fast Data **Web UIs** : schema-registry, kafka-topics, kafka-connect.
  - Please note that **Fast Data Web UIs are licensed under BSL**. You should contact Landoop if you plan to use them on production clusters with more than 4 nodes.

by executing the command below, while Docker is running and you are connected to the internet:

```
docker run --rm -it --net=host landoop/fast-data-dev
```

- If you are on Mac OS X, you have to expose the ports instead:

```
docker run --rm -it \
        -p 2181:2181 -p 3030:3030 -p 8081:8081 \
        -p 8082:8082 -p 8083:8083 -p 9092:9092 \
        -e ADV_HOST=127.0.0.1 \
        landoop/fast-data-dev
```

- This will download the fast-data-dev Docker image from the Dock Hub. https://hub.docker.com/r/landoop/fast-data-dev/
- Future runs will use your local copy.
- More details about Fast-data-dev docker image https://github.com/Landoop/fast-data-dev

Step 1: Setup your Kafka Development Environment

- **Points of interest:**
  - the **-p** flag is used to publish a network port. Inside the container, ZooKeeper listens at 2181 and Kafka at 9092. If we don't publish them with -p, they are not available outside the container, so we can't really use them.
  - the **–e** flag sets up environment variables.
  - the last part specifies the image we want to run: **landoop/fast-data-dev**
  - Docker will realize it doesn't have the landoop/fast-data-dev image locally, so it will first download it.
- That's it.
  - Your **Kafka Broker** is at localhost:9092,
  - your **Kafka REST Proxy** at localhost:8082,
  - your **Schema Registry** at localhost:8081,
  - your **Connect Distributed** at localhost:8083,
  - your **ZooKeeper** at localhost:2181

Step 1: Setup your Kafka Development Environment

- At http://localhost:3030, you will find **Landoop's Web UIs** for:
  - Kafka Topics
  - Schema Registry
  - as well as a integration test report for connectors & infrastructure using Coyote. https://github.com/Landoop/coyote
- If you want to stop all services and remove everything, simply hit **Control+C**.

Step 1: Setup your kafka Development Environment

- Explore Integration test results at http://localhost:3030/coyote-tests/

localhost:3030/coyote-tests/

Landoop Kafka Dev... | Other Bookmarks

Coyote | Confluent Tests, 2017 Mar 02, Thu, 16:14 UTC

100.00% Passed

Total tests: 39

- Brokers — Passed 3 out of 3 | 37.47s
- REST Proxy — Passed 16 out of 16 | 8.54s
- Schema Registry — Passed 7 out of 7 | 0.17s
- Connect — Passed 13 out of 13 | 50.42s

Total duration: **96.60s**

Results

Brokers

Action	Time (sec)	Command	Exit code
Create Topic (basic kafka)	3.37	kafka-topics --zookeeper 127.0.0.1:2181 --topic coyote_basic_1488471144874 --partition 1 --replication 1 --create	0
List Topics (basic kafka)	2.95	kafka-topics --zookeeper 127.0.0.1:2181 --list	0
Performance Test (basic kafka)	31.14	kafka-producer-perf-test --topic coyote_basic_1488471144874 --throughput 100000 --record-size 1000 --num-records 500000 --producer-props bootstrap.servers="127.0.0.1:9092"	0

Passed 3 out of 3 | 37.47 seconds

REST Proxy

Action	Time (sec)	Command	Exit code
List Topics (rest proxy)	0.09	curl -vs --stderr - "http://127.0.0.1:8082/topics"	0
Topic Information (rest proxy)	1.26	curl -vs --stderr - "http://127.0.0.1:8082/topics/coyote_basic_1488471144874"	0
Topic Partitions (rest proxy)	0.11	curl -vs --stderr - "http://127.0.0.1:8082/topics/coyote_basic_1488471144874/partitions"	0
Delete Topic (basic kafka)	2.23	kafka-topics --zookeeper 127.0.0.1:2181 --topic coyote_basic_1488471144874 --delete	0
Produce Avro Message (rest proxy, schema registry)	0.93	curl -vs --stderr - -XPOST -H "Content-Type: application/vnd.kafka.avro.v1+json" --data '{"value_schema": "{\"type\": \"record\", \"name\": \"User\", \"fields\": [{\"name\": \"name\", \"type\": \"string\"}]}", "records": [{"value": {"name": "testUser"}}]}'	0

Step 2: Get twitter credentials to connect to live data

- Now that our **single-node Kafka cluster is fully up and running**, we can proceed to preparing the input data:
  - First you need to **register an application with Twitter.**
  - Second, once the application is created **copy the Consumer key and Consumer Secret.**
  - Third, **generate the Access Token Access and Secret Token** required to give your twitter account access to the new application
- Full instructions are here: **https://apps.twitter.com/app/new**

Step 3: Get twitter live data into Kafka broker

- First, **create a new Kafka Connect for Twitter**

Step 3: Get twitter live data into Kafka broker

- Second, **configure this Kafka Connect for Twitter** to write to the **topic *twitter*** by entering your own track.terms and also the values of twitter.token, twitter.secret, twitter.comsumerkey and twitter.consumer.secret

Step 3: Get twitter live data into Kafka broker

- **Kafka Connect for Twitter** is now configured to write data to the **topic *twitter***.

Step 3: Get twitter live data into Kafka broker

- Data is now being written to the **topic *twitter***.

Step 4: Write and test the application code in Java

- Instead of writing our own code for this demo, we will be leveraging an existing code from GitHub by Sonke Liebau: https://github.com/opencore/kafkastreamsdemo

Code | Issues 0 | Pull requests 0 | Projects 0 | Insights

Branch: master | Find file | Copy path

kafkastreamsdemo / src / main / java / com / opencore / sapwebinarseries / KafkaStreamsDemo.java

Sönke Liebau Fixed bug that caused hashtags to not be correctly recognized as buzz... d3dc4a7 on Aug 1, 2016

0 contributors

168 lines (145 sloc) | 8.1 KB | Raw | Blame | History

```
/**
 * Copyright 2016 OpenCore GmbH & Co. KG
 * <p>
 * Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
 * in compliance with the License. You may obtain a copy of the License at
 * <p>
 * http://www.apache.org/licenses/LICENSE-2.0
 * <p>
 * Unless required by applicable law or agreed to in writing, software distributed under the License
 * is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
 * or implied. See the License for the specific language governing permissions and limitations under
 * the License.
 */

package com.opencore.sapwebinarseries;

import com.opencore.sapwebinarseries.utils.GenericAvroSerde;
import io.confluent.kafka.serializers.AbstractKafkaAvroSerDeConfig;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.common.serialization.Serde;
```

Step 4: Write and test the application code in Java

- **git clone https://github.com/opencore/kafkastreamsdemo**

```
Streamera:~ sbaltagi$ git clone https://github.com/opencore/kafkastreamsdemo
Cloning into 'kafkastreamsdemo'...
remote: Counting objects: 34, done.
remote: Total 34 (delta 0), reused 0 (delta 0), pack-reused 34
Unpacking objects: 100% (34/34), done.
Streamera:~ sbaltagi$
```

- Edit the buzzwords.txt file with your own works and probably one of the twitter terms that you are watching live:

buzzwords.txt — Edited

```
chicago
kafka
meetup
streaming
```

```
bash
total 0
drwxr-xr-x  3 sbaltagi  staff  102 Mar  2 14:00 java
drwxr-xr-x  5 sbaltagi  staff  170 Mar  2 14:00 resources
Streamera:main sbaltagi$ cd resources/
Streamera:resources sbaltagi$ ls -l
total 24
-rw-r--r--  1 sbaltagi  staff   51 Mar  2 14:00 buzzwords.txt
-rw-r--r--  1 sbaltagi  staff  332 Mar  2 14:00 log4j.properties
-rw-r--r--  1 sbaltagi  staff  959 Mar  2 14:00 stopwords.txt
Streamera:resources sbaltagi$ open -e buzzwords.txt
Streamera:resources sbaltagi$
```

Step 4: Write and test the application code in Java

- Edit the pom.xml to reflect the Kafka version compatible with Confluent Data platform/Landoop. See https://github.com/Landoop/fast-data-dev/blob/master/README.md

~/kafkastreamsdemo/pom.xml (functions)

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.opencore.sapwebinarseries</groupId>
    <artifactId>KafkaStreamsDemo</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>KafkaStreamsDemo</name>
    <url>http://maven.apache.org</url>
    <repositories>
        <repository>
            <id>confluent</id>
            <url>http://packages.confluent.io/maven/</url>
        </repository>
    </repositories>
    <dependencies>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-streams</artifactId>
            <version>0.10.2.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.10.2.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro</artifactId>
            <version>1.7.7</version>
        </dependency>
        <dependency>
            <groupId>io.confluent</groupId>
            <artifactId>kafka-avro-serializer</artifactId>
            <version>3.0.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro-maven-plugin</artifactId>
            <version>1.7.7</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
```

Step 5: Run the application

- The next step is to run the Kafka Streams application that processes twitter data.
- First, install Maven http://maven.apache.org/install.html
- Then, **compile the code into a fat jar with Maven.**

$ *mvn package*

```
Streamera:kafkastreamsdemo sbaltagi$ pwd
/Users/sbaltagi/kafkastreamsdemo
Streamera:kafkastreamsdemo sbaltagi$ ls -l
total 40
-rw-r--r--  1 sbaltagi  staff  11357 Mar  2 14:00 LICENSE
-rw-r--r--  1 sbaltagi  staff   2135 Mar  2 14:00 README.md
-rw-r--r--  1 sbaltagi  staff   2910 Mar  2 14:00 pom.xml
drwxr-xr-x  3 sbaltagi  staff    102 Mar  2 14:00 src
Streamera:kafkastreamsdemo sbaltagi$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building KafkaStreamsDemo 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
Downloading: http://packages.confluent.io/maven/org/apache/kafka/kafka-streams/0.10.0.0/kafka-s
treams-0.10.0.0.pom
```

```
Component: org.apache.maven.artifact.handler.ArtifactHandlerejb-client is already defined. Skipping.
Component: org.apache.maven.artifact.handler.ArtifactHandlerrar is already defined. Skipping.
Component: org.apache.maven.artifact.handler.ArtifactHandlerpar is already defined. Skipping.
Component: org.apache.maven.artifact.handler.ArtifactHandlerejb3 is already defined. Skipping.
Component: org.apache.maven.artifact.factory.ArtifactFactory is already defined. Skipping.
Component: org.apache.maven.artifact.resolver.ArtifactCollector is already defined. Skipping.
Component: org.apache.maven.plugin.registry.MavenPluginRegistryBuilder is already defined. Skipping.
Component: org.apache.maven.shared.io.download.DownloadManagerdefault is already defined. Skipping.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 11.475 s
[INFO] Finished at: 2017-03-02T14:24:58-06:00
[INFO] Final Memory: 37M/422M
[INFO] ------------------------------------------------------------------------
Streamera:kafkastreamsdemo sbaltagi$
```

Step 5: Run the application

- Two jar files will be created in the **target** folder:
  1. KafkaStreamsDemo-1.0-SNAPSHOT.jar – Only your project classes
  2. KafkaStreamsDemo-1.0-SNAPSHOT-**jar-with-dependencies**.jar – Project and dependency classes in a single jar.

```
Streamera:kafkastreamsdemo sbaltagi$ pwd
/Users/sbaltagi/kafkastreamsdemo
Streamera:kafkastreamsdemo sbaltagi$ ls -l
total 40
-rw-r--r--  1 sbaltagi  staff  11357 Mar  2 14:00 LICENSE
-rw-r--r--  1 sbaltagi  staff   2135 Mar  2 14:00 README.md
-rw-r--r--  1 sbaltagi  staff   2910 Mar  2 14:00 pom.xml
drwxr-xr-x  3 sbaltagi  staff    102 Mar  2 14:00 src
drwxr-xr-x  8 sbaltagi  staff    272 Mar  2 14:24 target
Streamera:kafkastreamsdemo sbaltagi$ cd target/
Streamera:target sbaltagi$ ls -l
total 31672
-rw-r--r--  1 sbaltagi  staff  16203673 Mar  2 14:24 KafkaStreamsDemo-1.0-SNAPSHOT-jar-with-dependencies.jar
-rw-r--r--  1 sbaltagi  staff     11537 Mar  2 14:24 KafkaStreamsDemo-1.0-SNAPSHOT.jar
drwxr-xr-x  2 sbaltagi  staff        68 Mar  2 14:24 archive-tmp
drwxr-xr-x  6 sbaltagi  staff       204 Mar  2 14:24 classes
drwxr-xr-x  3 sbaltagi  staff       102 Mar  2 14:24 generated-sources
drwxr-xr-x  3 sbaltagi  staff       102 Mar  2 14:24 maven-archiver
Streamera:target sbaltagi$
```

Step 5: Run the application

- Then

java -cp target/KafkaStreamsDemo-1.0-SNAPSHOT-**jar-with-dependencies**.jar com.opencore.sapwebinarseries.KafkaStreamsDemo

- **TIP**: During development: from your IDE, from CLI … Kafka Streams Application Reset Tool, **available since** Apache Kafka 0.10.0.1, is great for playing around.
  https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+Application+Reset+Tool

3.4. Results of this demo

- Once the above is running, the following topics will be populated with data :
  - **Raw word count** - Every occurrence of individual words is counted and written to the topic *wordcount* (a predefined list of stopwords will be ignored)
  - **5-Minute word count** - Words are counted per 5 minute window and every word that has more than three occurrences is written to the topic *wordcount5m*
  - **Buzzwords** - a list of special interest words can be defined and those will be tracked in the topic *buzzwords* - the list of these words can be defined in the file *buzzwords.txt*

3.4. Results of this demo

- **Accessing the data generated by the code** is as simple as starting a **console consumer** which is shipped with Kafka
- You need first to enter the container to use any tool as you like: ***docker run --rm -it --net=host landoop/fast-data-dev bash***
- Use the following command to check the topics:
  - ***kafka-console-consumer --topic wordcount --new-consumer --bootstrap-server 127.0.0.1:9092 --property print.key=true***
  - ***kafka-console-consumer --topic wordcount5m --new-consumer --bootstrap-server 127.0.0.1:9092 --property print.key=true***
  - ***kafka-console-consumer --topic buzzwords --new-consumer --bootstrap-server 127.0.0.1:9092 --property print.key=true***

3.4. Results of this demo

5. docker

docker

```
RT @SandraTXAS: Mayor Rahm Emanuel and Democrats allow Chicago citizens to be murdered in their neighborhoods for decades, yet blam… en
SandraTXAS
null    2017-03-02T21:28:48.000+0000Marsha GanzMarshaGanz$Ft. Lauderdale, FLRT @Science4UsSays: Bwhahahahahaha #thursdaythoughts #ICE1
7 #Chicago #davinci  https://t.co/89NqAgfHwn @MarshaGanz https://t.co/324aVsDUXot thursdaythoughts
ICE17Chicagodavinci4pic.twitter.com/324aVsDUXo https://twitter.com/Science4UsSays/status/837413032342999040/photo/1
photo.https://t.co/324aVsDUXobuff.ly/2lyHLAH,http://buff.ly/2lyHLAH.https://t.co/89NqAgfHwn
                                                                                                "Jenna and SciefusScience4UsSaysMarsha GanzMarshaGan
z
null    2017-03-02T21:29:10.000+0000Billy BlokeeBayDownUnder8Australian Capital Territoryjchicago girl writes 'I hate this life' d
ays before death https://t.co/kJ6Zmq7FlF https://t.co/jzvOWF0qwyen4pic.twitter.com/jzvOWF0qwy https://twitter.com/eBayDownUnder/status/837414503738322
944/photo/1
photo.https://t.co/jzvOWF0qwydlvr.it/NWnpJ3*http://dlvr.it/NWnpJ3.https://t.co/kJ6Zmq7FlF
```

docker

```
https://t.co/89nqagfhwn 2
@marshaganz     2
https://t.co/324avsduxo 2
chicago 2263
girl    155
writes  51
'i      9
hate    134
life'   1
days    38
death   18
https://t.co/kj6zmq7flf 1
https://t.co/jzvowf0qwy 1
```

docker

```
Thu Mar 02 15:30:00 CST 2017-democrats  5
Thu Mar 02 15:30:00 CST 2017-allow      5
Thu Mar 02 15:30:00 CST 2017-chicago    99
Thu Mar 02 15:30:00 CST 2017-citizens   5
Thu Mar 02 15:30:00 CST 2017-murdered   5
Thu Mar 02 15:30:00 CST 2017-neighborhoods     5
Thu Mar 02 15:30:00 CST 2017-decades,   5
Thu Mar 02 15:30:00 CST 2017-yet        5
Thu Mar 02 15:30:00 CST 2017-blam…      5
Thu Mar 02 15:30:00 CST 2017-rt 124
Thu Mar 02 15:30:00 CST 2017-#chicago   17
Thu Mar 02 15:30:00 CST 2017-chicago    100
Thu Mar 02 15:30:00 CST 2017-days       3
```

docker

```
Thu Mar 02 15:30:00 CST 2017-chicago    91
Thu Mar 02 15:30:00 CST 2017-chicago    92
Thu Mar 02 15:30:00 CST 2017-chicago    93
Thu Mar 02 15:30:00 CST 2017-chicago    94
Thu Mar 02 15:30:00 CST 2017-chicago    95
Thu Mar 02 15:30:00 CST 2017-meetup     8
Thu Mar 02 15:30:00 CST 2017-chicago    96
Thu Mar 02 15:30:00 CST 2017-chicago    97
Thu Mar 02 15:30:00 CST 2017-chicago    98
Thu Mar 02 15:30:00 CST 2017-meetup     9
Thu Mar 02 15:30:00 CST 2017-chicago    99
Thu Mar 02 15:30:00 CST 2017-#chicago   17
Thu Mar 02 15:30:00 CST 2017-chicago    100
```

3.5. Stopping the demo!

- To stop the Kafka Streams Demo application:
  - **$ ps – A | grep java**
  - **$ kill -9 <PID>**

```
1. tmux
Streamera:~ sbaltagi$ ps -A |grep java
35532 ttys007    0:00.00 grep java
35253 ttys013    0:29.90 /usr/bin/java -cp target/KafkaStreamsDemo-1.0-SNAPSHOT-
jar-with-dependencies.jar com.opencore.sapwebinarseries.KafkaStreamsDemo
Streamera:~ sbaltagi$ kill -9 35253
Streamera:~ sbaltagi$ ps -A |grep java
35536 ttys007    0:00.00 grep java
Streamera:~ sbaltagi$
```

- If you want to **stop all services in fast-data-dev Docker** image and remove everything, simply hit **Control+C.**

Thank you!

Let’s keep in touch!

@SlimBaltagi

https://www.linkedin.com/in/slimbaltagi

sbaltagi@gmail.com