Collaborative Project

GeoKnow - Making the Web an Exploratory Place for Geospatial Knowledge

Project Number: 318159 **Start Date of Project:** 2012/12/01 **Duration:** 36 months

Deliverable 4.4.2
Open-social API for GeoKnow Generator

Dissemination Level	Public
Due Date of Deliverable	M34, 30/09/2015
Actual Submission Date	M34, 30/09/2015
Work Package	WP4
Task	T4.4
Type	Prototype
Approval Status	Approved
Version	1.0
Number of Pages	19
Filename	D4.4.2 Open-social API for GeoKnow Generator.pdf

Abstract: In this document we report the software architecture, developments to bring content from Social networks to the GeoKnow Generator. We focused specially how to consume Twitter real-time data for the Supply Chain use case. Some preliminary results are also presented.

Project funded by the European Commission within the Seventh Framework Programme (2007 - 2013)

History

Version	Date	Reason	Revised by
0.1	2013-09-12	Initial structure of this deliverable	Alejandra Garcia Rojas M
0.2	2013-09-19	Worked on architecture section	Alejandra Garcia Rojas M
0.5	2013-09-24	Motivation and Requirements	René Pietzsch
0.8	2013-09-25	Worked on architecture section	Alejandra Garcia Rojas M
0.9	2013-09-29	Internal and peer-review	Claus Stadler
1.0	2013-09-30	Final changes	Alejandra Garcia Rojas M

Author List

Organization	Name	Contact Information
ONTOS	Alejandra Garcia Rojas M.	alejandra.garciarojas@ontos.com
BROX	René Pietzsch	rpietzsch@brox.de

Executive Summary

With the appearance of Social Networks, we are now to learn about any type of events in real time. These events have an impact to the every day business in different domains. This report focuses on the Supply Chain use case of GeoKnow project. This use case requires to have access to information shared in Social Networks in order to adapt their business. In this report we focused on consuming Twitter information relevant to our use case, and we have followed a scalable approach using Big Data technologies that enable us to perform real-time analytics.

Table of Contents

1 Introduction

For several application domains it is important to be aware of different external events that could have an impact on every day business. Social Networks are now an important source of information when it comes to rapidly communicate and learn about events happening in the world.

The main goal of the *Open-social API for GeoKnow Generator* task is to get updates about specific events from the most popular Social Networks. The original intention of this task was to implement some standards that could transparently access any source of information. Therefore, we investigated activities in the Social Web Working Group [1], which aims to define standards and APIs to facilitate access to social functionality as part of the Open Web Platform. However, the activities in these standards definition are still work in progress. To be able to get actual data, we required to implement a source specific development in order to consume data form the different Social Networks.

Thus, the work performed in this task was to create a prototype with technologies for fulfilling the Supply Chain use case requirements (see Section 2). The Supply Chain use case is interested in learning about external events that can influence the logistics in the Supply Chain. Social Networks, specifically Twitter, can provide unknown information to the Supply Chain. Extracted data from Tweets has to be processed and analysed in order to find out relevant eventualities in real time. Therefore, the approach followed includes Big Data consumption and processing based on Apache Spark and Apache Kafka technologies. The system architecture and prototype that was created to fulfil the requirements is presented in Section 3. The preliminary results of the working prototype are presented in Section 4, and this is followed by the conclusions and future work in Section 5.

2 Motivation

Supply chain managers often complain about a lack of transparency and information along their information chain and support systems. This lack of information has an internal perspective denoting the lack and latency of internal information flows and an external perspective.

The motivation of a twitter and social media integration targets the second perspective: the integration and preparation of relevant external background knowledge. External background knowledge sources are viable candidates to be integrated and related (linked) to suppliers to gain new insights and aid decision making in supply chain management. Disasters, accidents, political instabilities or military activities for instance influence strategic spatial sourcing decisions. Likewise do company related information like hiring or release waves, stock volatility, product releases or financial reports about a supplier influence the sourcing strategy.

Nowadays an incredible amount of timely and (often) accurate news are generated in social media channels like Twitter and Facebook. Thus, these sources are valid sources for real-time background knowledge that can be used to show related information for selected supplier or supplier locations. Furthermore a topic and/or sentiment analysis of these information streams can further help supply chain managers to take well informed decisions.

2.1 Requirements

The following requirements are to be meet by the prototype:

[1] https://www.w3.org/wiki/Socialwg

- Ability to configure relevant search terms like company name (rdfs:label, skos:altLabel), location (city, state, country) or language.
- Ability to analyse in real time twitter and/or facebook feeds.
- Ability to extract the article/feed/tweet text for further analysis.
- Ability to process content in English and German.
- Ability to perform named entity recognition (NER) on the feed data and to annotate the data accordingly.
- Representation of the data, analysis and annotation results in RDF.
- Storage of RDF in a graph in a SPARQL endpoint.
- Integration of the content extraction tool stack in the GeoKnow Generator.

3 GeoKnow Social API

Traditional API consumption form social networks like Twitter, Facebook, LinkedIn, are made via REST API, and we can find several libraries that can be reused for extracting the content. However, we are particularly interested in consuming real time data, and particularly we want to get content from Twitter. The content shared on Twitter is commonly related to ongoing events, and is also more organisational oriented than the other Social Networks (i.e. Facebook, LinkedIn).

Twitter provides two different APIs for extracting content. The REST API [2] allows to perform queries to the Twitter data under some restrictions (i.e. 180 queries per 15 minute window for the time being). This method has some latency from the moment when a tweet is generated until it can be retrieved, moreover one may have to deal with duplicated tweets and define a cutoff of extraction. Therefore, Twiter also provides the Streaming API [3], this method will allow consuming tweets in real time, at the expense of a more complex extraction implementation compared to a simple REST API based implementation.

We investigated existing technologies that could be used for consuming Twitter streaming. The Node.js streaming libraries [4] would be a fast solution for consuming the data, however the scalability may be compromised. More complex platforms that allows scalable stream processing are Apache Storm and Apache Spark. Apache Storm[5] operates on a one-at-a-time processing mode, which means it process individual data items as they arrive in the stream, it is strictly stream processing. Spark Streaming[6] operates on a micro-batch mode, but it can still provide near-real time analytics compared with Storm. Besides the streaming, Spark also targets at speeding up batch analysis jobs, iterative machine learning jobs, interactive query and graph processing. These features are valuable for integrating new analytical or prediction components into the architecture. The choice for a stream processing component really depends on what is the most important requirements to fulfil in the use case [7]. We picked Spark because its analytical features fits better our requirements.

[2] https://dev.twitter.com/rest/public
[3] https://dev.twitter.com/streaming/overview
[4] https://www.npmjs.com/package/node-tweet-stream
[5] https://storm.apache.org/
[6] http://spark.apache.org/streaming/
[7] http://zdatainc.com/2014/09/apache-storm-apache-spark/

The requirements in the Supply Chain use case are not only to consume content from social networks, but also to be able to analyse it in order to support decision making. Therefore, the content extracted from sources needs to be continuously processed and analysed.

3.1 Architecture

We required an architecture capable of processing data from streams and other common data consumption techniques. The acquired data has to be processed through a pipeline to be cleaned, translated, dereferenced, annotated and stored, and this needs to be passed among different systems on a efficient and reliable manner. Thus, the backbone of our architecture to achieve required scalable continuous processing uses one of the most popular messaging broker systems. Kafka[8] is a real-time, fault tolerant, highly scalable messaging system, created at LinkedIn[9] and is widely adopted for a variety of use cases involving real-time data streams, and it is open-source. For using Kafka applications, it is required to implement Kafka consumers and/or producers. A Kafka producer will generate messages that will be posted to Kafka under a specific topic. And, a Kafka consumer will subscribe to one or more topics and will consume messages that are posted by the producers. A schema must be defined for each topic. Maintaining topics and schemata is requires to carfully update producers and consumers Confluent Platform[10] offers to complete Kafka functionality with a Schema Registry. This Registry allows to keep a centralized schema management, and is able to track different versions of the schema. This allows to safely modify schemata without braking the different services.

The Social API conceptual architecture is presented in Figure 1. As mentioned previously, the Confluent Platform, which integrates Kafka and Schema Registry is the main data transport system.

We can see that there could be different types of applications that produce and consume messages form the processing queue. Data Extraction components would only produce messages, we could have for instance, unstructured-content, semi-structured, and structured coming from different sources. The extracted data is then consumed by Data Processing components such as the *ner-extraction*, but one could add other kind of processing components for linking, enriching, cleaning data, etc. Afterwards, Data Analytics components can perform some analytics doing some map-reduce operations. And Storage components could serialize the data in different formats and places (e.g. RDF store or Hadoop file system).

[8] http://kafka.apache.org/
[9] https://www.linkedin.com/
[10] http://docs.confluent.io/1.0.1/platform.html

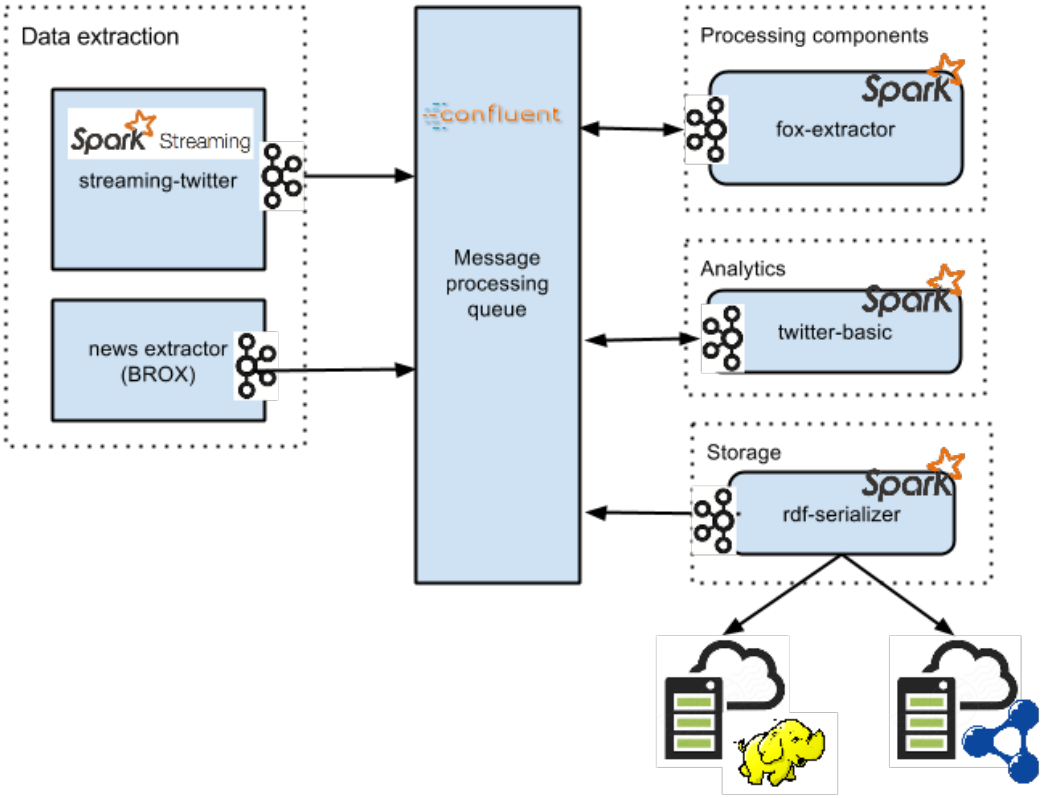


Figure 1: Social API architecture

The implementation of the described architecture for the Supply Chain use case is presented next.

3.2 Social API Prototype

The integration of Confluent with Spark is straightforward. Spark provides a java library that can be used for consuming streams from Kafka [11]. Therefore, for this task we have developed three components. A **streaming-twitter** that consumes streaming tweets of the provided keywords, the (2) **fox-extractor** that uses FOX[12] api to annotate twitter status with Places, Organizations and Persons, and, (3) **rdf-serializer**, that persist the extracted entities by FOX and some information about the tweet to a RDF store. A more detailed description of these components is provided as follows. The binary format of these components are available for testing in the GeoKnow repository at https://github.com/GeoKnow/SocialAPI.

[11] http://spark.apache.org/docs/latest/streaming-kafka-integration.html
[12] http://aksw.org/Projects/FOX.html

3.3 Streaming from Twitter

3.3.1 Description

The **streaming-twitter** is a Spark Streaming application that consumes twitter status based on a provided set of keywords to track. We have defined a schema to send it to the messaging queue, wich is described in Appendix A. This schema has tweet information relevant to the scenario such as content, language, geolocation and some basic user data.

3.3.2 Configuration

The required configuration of this tool is described in the following snippet:

```
# twitter credentials
twitter4j.oauth.consumerKey=**REQUIRED**
twitter4j.oauth.consumerSecret=**REQUIRED**
twitter4j.oauth.accessToken=**REQUIRED**
twitter4j.oauth.accessTokenSecret=**REQUIRED**

# configuration to create org.apache.kafka.clients.producer.KafkaProducer
kafka.producer.bootstrap.servers=localhost:9092
kafka.producer.acks=all
kafka.producer.retries=1
kafka.producer.key.serializer=org.apache.kafka.common.serialization.StringSerializer
kafka.producer.value.serializer=io.confluent.kafka.serializers.KafkaAvroSerializer
kafka.producer.schema.registry.url=http://localhost:8081

# twitter topics separated by comma
app.filter.track=google,alphabet

# kafka topic configuration
app.kafka.topic.out=aTweet
app.kafka.topic.schema=tweet-schema-with-topic.json

# kafka producer pool configuration
app.kafka.poducer.pool.minIdle=4
app.kafka.poducer.pool.maxIdle=10
app.kafka.poducer.pool.validationInterval=20
```

3.4 Name Entity Extraction

3.4.1 Description

The **fox-extractor** application will retrieve all messages that have unstructured content, and will determine the language by using a *language-detector* library[13]. If the language is supported by FOX, it will send this content to a FOX instance, which will extract named entities. FOX return results in RDF format already, therefore, here we transform the data we are interested in into RDF too. The Figure 2 presents the RDF schema we used. This schema is mainly based on the schema.org because it contains most of the vocabulary required, thus we don't need to redefine anything. This, the **fox-extractor** produces new Kafka messages in its corresponding schema format (included in the Appendix A) with the list of extracted named entities in RDF.

[13] https://github.com/optimaize/language-detector

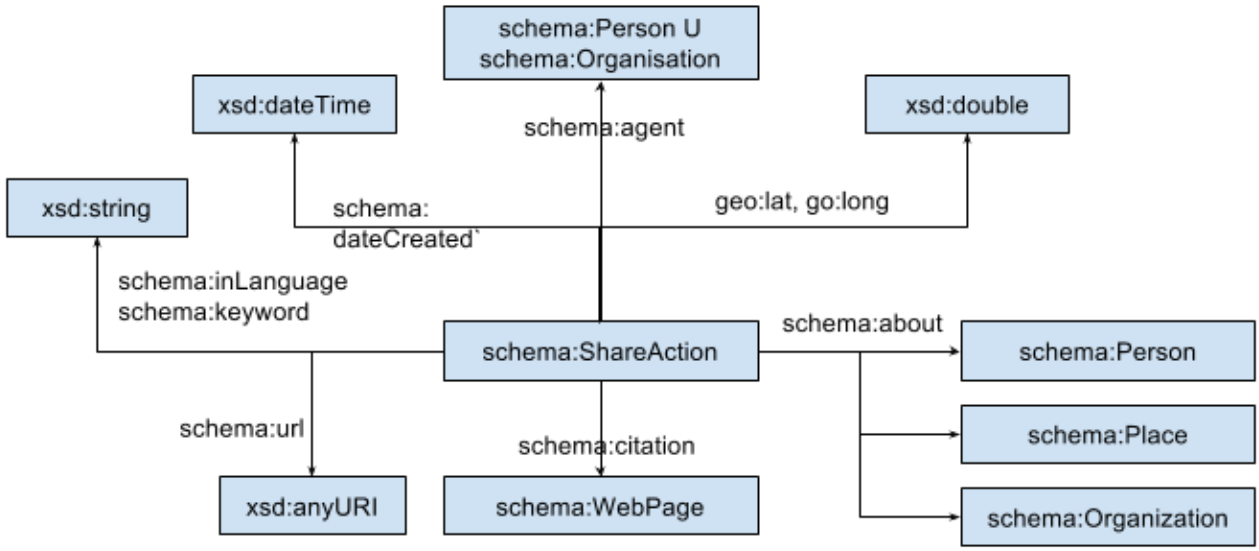


Figure 2: RDF schema of data extracted

3.4.2 Requirements

Besides common requirements of these spark applications, the **fox-extractor** requires to have FOX REST service available.

3.4.3 Configuration

The required configuration for the fox-extractor is presented in the following snippet.

```
# kafka.consumer params has to be set with kafka.consumer
kafka.consumer.bootstrap.servers=localhost:9092
kafka.consumer.zookeeper.connect=localhost:2181
kafka.consumer.group.id=group1
kafka.consumer.schema.registry.url=http://localhost:8081
# maximum number of schema objects created for a given subject
kafka.consumer.schema.registry.identityMapCapacity=10

kafka.producer.bootstrap.servers=localhost:9092
kafka.producer.acks=all
kafka.producer.retries=1
kafka.producer.schema.registry.url=http://localhost:8081

# We can only get schemata by Id from the Schema Registry.
Is to us to find out which id  to set here
app.kafka.consumer.topics=aTweet
app.kafka.producer.topic=annotatedTweet

app.fox.api=http://localhost:4444/api
```

3.5 Data Serialization

3.5.1 Description

The **rdf-serializer** persist Kafka messages into a RDF store and/or the Hadoop file system. The messages coming from fox-extractor provide already RDF content.

3.5.2 Requirements

This component requires one or more accessible endpoint and/or a configured hadoop directory, so the results can be submitted there.

3.5.3 Configuration

```
# Kafka consumer parameters
kafka.consumer.bootstrap.servers=localhost:9092
kafka.consumer.zookeeper.connect=localhost:2181
kafka.consumer.group.id=group1
kafka.consumer.schema.registry.url=http://localhost:8081
# maximum number of schema objects created for a given subject
kafka.consumer.schema.registry.identityMapCapacity=10

# Storage configuration can be endpoint and/or hdfs, but at least one
# topics with RDF content to consume, comma separate
app.kafka.consumer.topics=rdfcontent,rdfcontent2
# specify at least one SPARQL endpoints and credentials
app.endpoint.1.url=http://localhost:9090/sparql
app.endpoint.1.user=dba
app.endpoint.1.pass=dba
app.endpoint.1.graph=http://supplier.geoknow.eu/resource/twitter
# specify 0 - n files to store the data as well
app.file.1=hdfs://localhost:9000/mydatadir
```

3.6 System Setup

For testing the architecture and the applications, we used a single machine: Ubuntu 14.04 with Intel Xeon E5-1650 v3 @ 3.50GHz, 12 cores (24) processors, 128 GB RAM and 4T hard drive. We installed Confluent Framework, Apache Spark, Apache Hadoop and the Kafka Manager[14]. All these applications provide a very comprehensive installation guide. Spark provides different ways of running [15]:

Standalone[16] a simple cluster manager included with Spark that makes it easy to set up a cluster.

Apache Mesos[17] a general cluster manager that can also run Hadoop MapReduce and service applications.

Hadoop YARN[18] the resource manager in Hadoop 2.

For this report we have tested all applications in standalone cluster mode, but cluster modes will be evaluated further. Spark was configured to use 10 cores and 60GB of memory, which let the system work smoothly. To deploy the spark application we use spark-submit[19] command. This spark command allows to submit a self-contained jar file (i.e. all dependent artifacts must be part of this file) in order it can be distributed to the different Spark clusters. Once the applications are submitted, they will be shown in the Spark monitoring web interface. A screen-shot of the deployed component is presented in Figure 3.

For monitoring Kafka topics we installed the Kafka Manager, developed by Yahoo. This UI allows to monitor the status of Kafka brokers, and maintain the Kafka clusters. It also facilitates the visualization of the different Kafka topics and their schemata. Figure 4 shows Kafka Manager interface with the Topics used by the applications described in this section.

[14] https://github.com/yahoo/kafka-manager
[15] http://spark.apache.org/docs/latest/cluster-overview.html
[19] http://spark.apache.org/docs/latest/submitting-applications.html

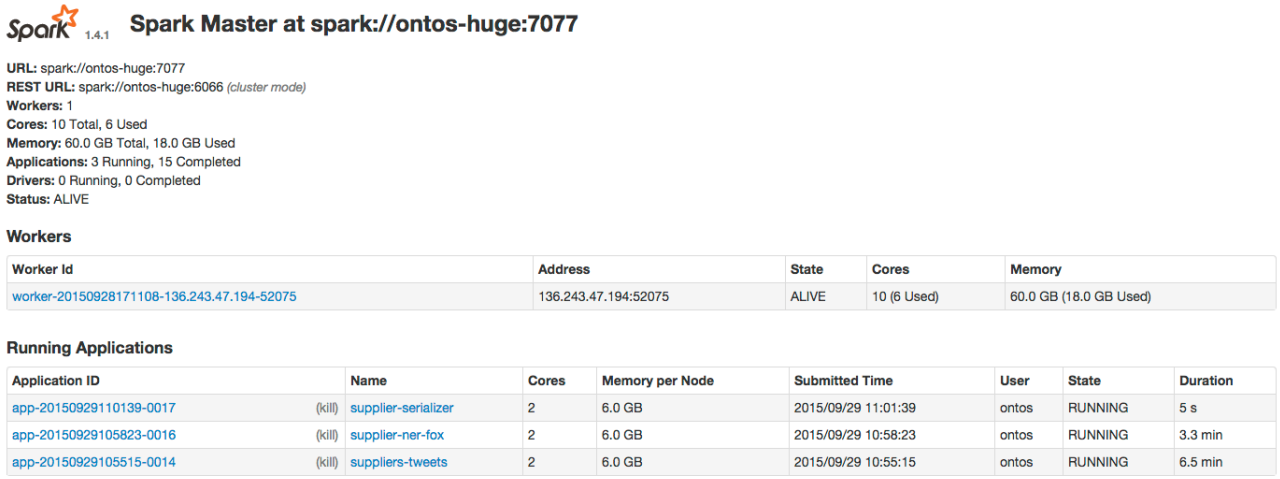


Figure 3: Spark Manager interface with the three processing applications

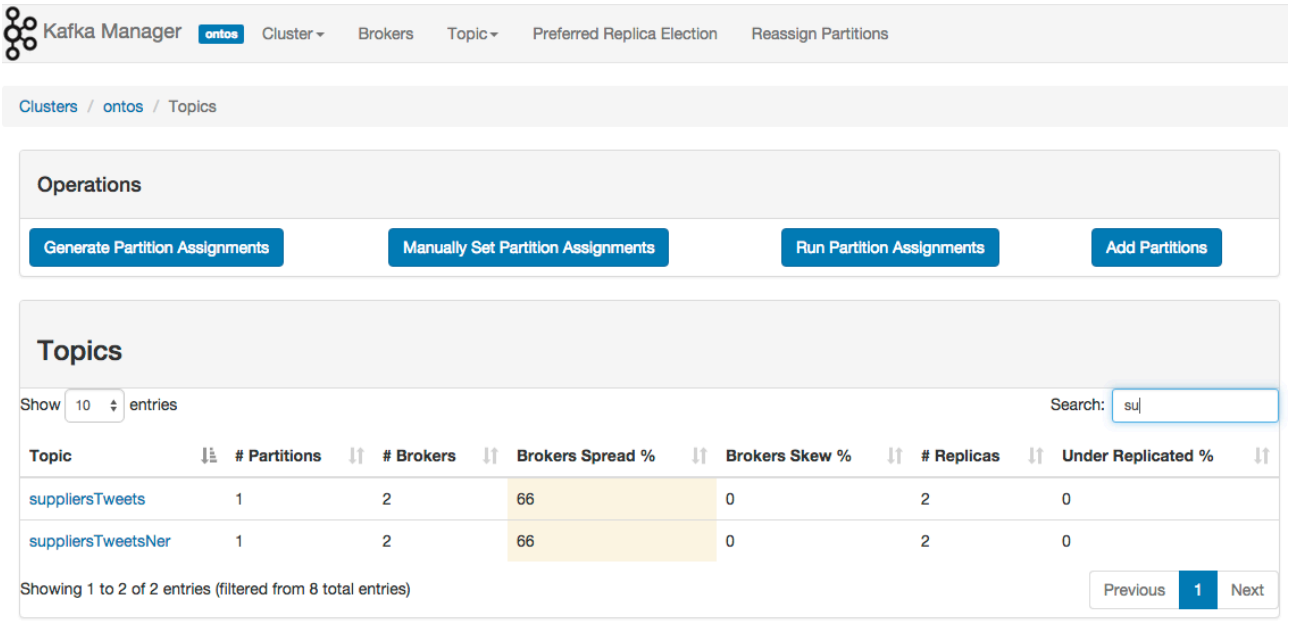


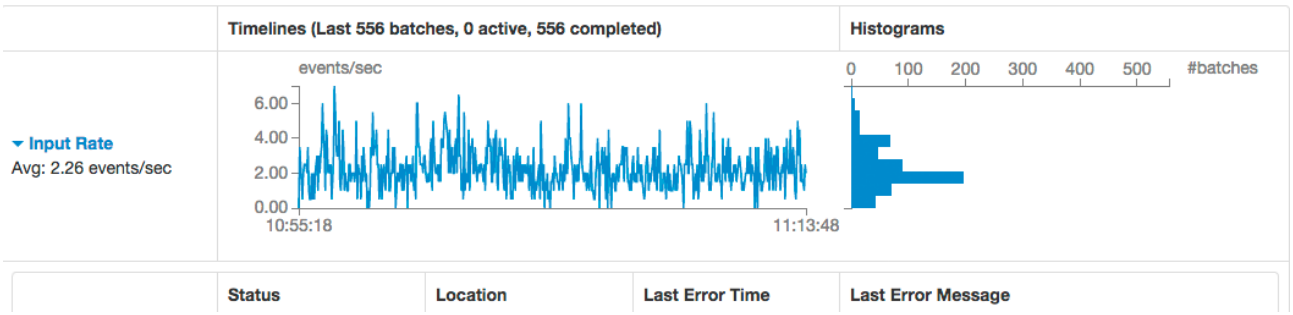
Figure 4: Kafka Manager

Besides of above described tools, we also installed two instances of FOX to be able to process text in English and in German. To be able to have this instance well configured for these languages, we created a container virtualized (docker) version of FOX. The dockerfile to build such container is available at: https://github.com/GeoKnow/FOX-Docker, a prebuild image can be found at: https://hub.docker.com/r/rpietzsch/fox-docker/. Each installed FOX instance required 7G of memory to run.

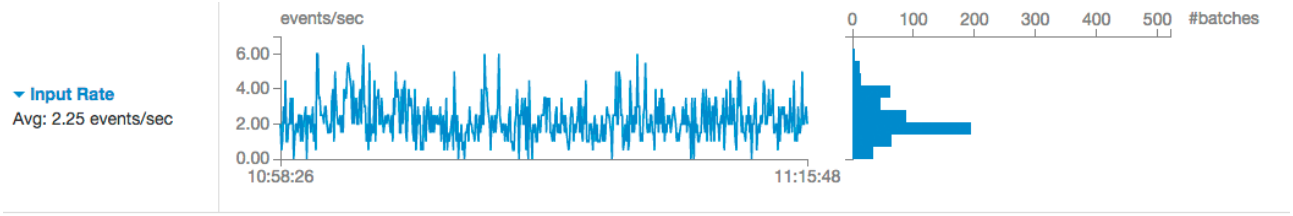
4 Experimental Results

We configured the **streaming-twitter** with the list of suppliers names as shown in Appendix B with 80 supplier's company names. Figure 5 shows the streaming statistics in the different components. the

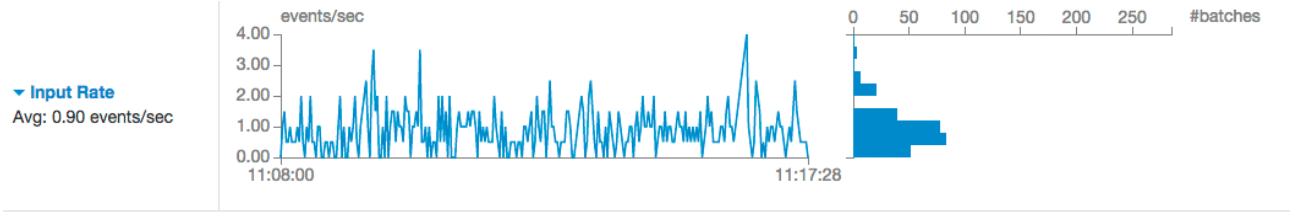
streaming-twitter application has in average 2.2 events per second, this is similar to the **fox-extractor**, but the **rdf-serializer** has less than 1.0, this is because **fox-processor** filters out all languages that it cannot process.



(a) twitter streaming



(b) fox-extractor streaming

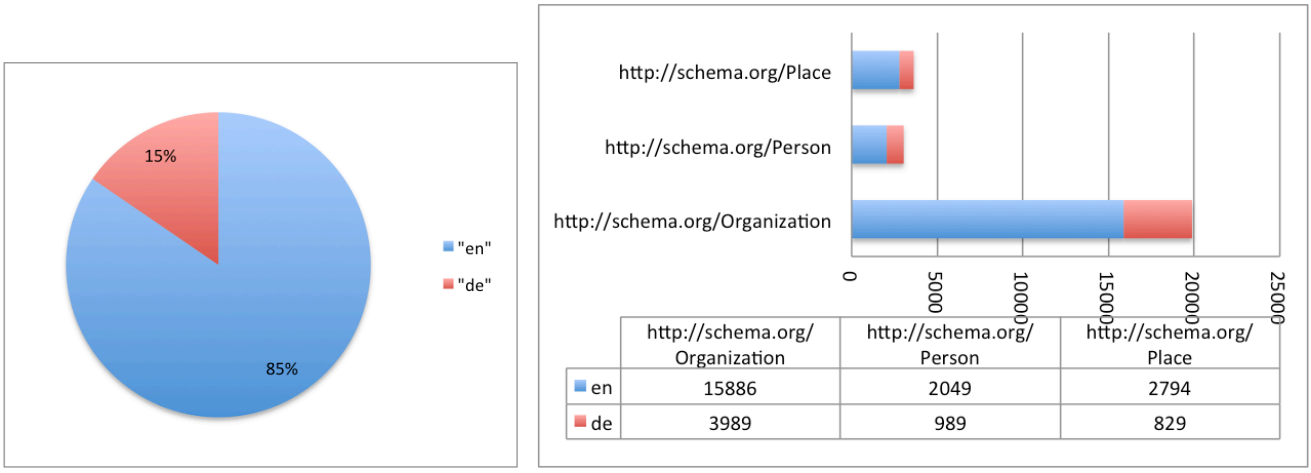


(c) rdf-serializer streaming

Figure 5: Streaming statistics of the three applications

4.1 Exploratory Analytics

The data analysed correspond to 24 hours of data collection Figure 6 depicts some initial figures of the obtained data: the percentage of tweets by language, and the number of tweets by named entities extracted.



(a) Tweets by language

(b) Tweets topics by type

Figure 6: Preliminary results

Following tables present top 20 named entities identified on the tweets in the two different languages. Our results are showing the current trending topic about the VW Diesel scandal [20].

Topic	Tweets
http:\dbpedia.org/resource/Volkswagen	11444
http:\dbpedia.org/resource/Australia	641
http:\dbpedia.org/resource/Urban_Symphony	300
http:\dbpedia.org/resource/United_Kingdom	252
http:\dbpedia.org/resource/Ethelbert_Blatter	223
http:\scms.eu/U.S.	219
http:\dbpedia.org/resource/Germany	197
http:\dbpedia.org/resource/Reuters	195
http:\dbpedia.org/resource/United_States_Environmental_Protection_Agency	188
http:\dbpedia.org/resource/U.S._Route_219	169
http:\scms.eu/Volkswagen_Diesel_Emission_Legal_Claims	161
http:\dbpedia.org/resource/Audi	128
http:\scms.eu/VOLKSWAGEN_POLO	119
http:\scms.eu/WASHINGTON	111
http:\scms.eu/BERLIN	110
http:\dbpedia.org/resource/Atlantic_Philanthropies	106
http:\dbpedia.org/resource/Volkswagen_Group	96
http:\scms.eu/Hitle	94
http:\dbpedia.org/resource/BP	92

[20] https://en.wikipedia.org/wiki/Volkswagen_emissions_violations

http:\dbpedia.org/resource/France_2	83

Table 1: Top 20 named entities in english

Topic	Tweets
http://de.dbpedia.org/resource/Volkswagen	1391
http://de.dbpedia.org/resource/R%C3%BCckw%C3%A4rtsterminierung	583
http://de.dbpedia.org/resource/Volkswagen_R	357
http://de.dbpedia.org/resource/Deutschland	127
http://scms.eu/Volkswagen_Scandal	96
http://de.dbpedia.org/resource/Australien	91
http://scms.eu/VOLKSWAGEN	88
http://de.dbpedia.org/resource/RT_%28Fernsehsender%29	79
http://de.dbpedia.org/resource/Turbokompressor	63
http://scms.eu/VW_Volkswagen	63
http://de.dbpedia.org/resource/VW_Polo	61
http://scms.eu/VW_Passat_Variant_GL_BJ.	59
http://scms.eu/ZOLL	59
http://scms.eu/VW_PASSAT	58
http://scms.eu/PHILIPPINES	57
http://scms.eu/WAY	57
http://de.dbpedia.org/resource/Guy	56
http://de.dbpedia.org/resource/Vereinigte_Staaten	55
http://de.dbpedia.org/resource/Der	51
http://de.dbpedia.org/resource/VW_Gol	43

Table 2: Top 20 named entities in German

The following last example shows the most mentioned Person type entities found in the sampled data.

Person	Tweets
subject type tweets http://dbpedia.org/resource/Ethelbert_Blatter	223
http://scms.eu/Hitle	94
http://dbpedia.org/resource/Nitin_Saxena	63
http://scms.eu/WAY	57
http://de.dbpedia.org/resource/Guy	56
http://scms.eu/Hans_Dieter_Poetsc	53
http://de.dbpedia.org/resource/Der	51

http://dbpedia.org/resource/Stephen_Harper	38
http://dbpedia.org/resource/MES	37
http://dbpedia.org/resource/Christian_Wolff_%28philosopher%29	33
http://scms.eu/BERLI	28
http://scms.eu/DEL_CASO	26
http://scms.eu/Der_Abgas-Skandal	25
http://scms.eu/ALAN_COWELL	22
http://dbpedia.org/resource/Elon_Musk	22
http://dbpedia.org/resource/Thompson%2C_Manitoba	22
http://scms.eu/Monagha	22
http://dbpedia.org/resource/Wonga_Beach%2C_Queensland	21
http://dbpedia.org/resource/Steve_Dawson	20
http://de.dbpedia.org/resource/Kurt_Link_%28Badminton%29	19

Table 3: Popular topics in German

The SPARQL queries used for exploring the data are included in the Appendix B.

5 Conclusions and Future Work

In this report we have presented a scalable architecture for consuming data from Social Networks. This architecture uses technologies that support real-time data analytics, which is a requirement from the Supply Chain use case.

In the implementation of the prototype, we have focused on consuming data from Twitter, but the architecture allows for adding other data sources (e.g. news collector). We developed three applications that are part of the data processing pipeline, that allows to perform an initial exploratory analysis of data gathered from Twitter. The developed components require to be tested longer in order to stabilize the environment. We want to setup a managed cluster using several hardware clusters, so we can dynamically scale the application.

With respect to the Spark applications, we want to extend the content extractions from Tweets. Since tweets share also web pages, we want to extract their content and relate it to the original source. Also we have to improve the FOX annotation, we noticed that FOX can fail in annotating wrongly the entities. Another feature we want to focus on, is the current geospatial information. The data gathered is limited to longitude and latitude, we want to extend the information to generalise this data to some geospatial regions, for instance w.r.t. the Supply Chain, administrative regions in Germany.

Concerning the use case we need to realize a more comprehensive data analysis to know if we are collecting required data, for instance, we need to find all *hashtags* related to the Suppliers. Also we need to assess the data we are receiving to filter out noise data. This analysis will be the basis to define an analytical component where we can provide a real-time results that can be used by the Supply Chain managers.

Appendices

A Kafka Messages Schemata

Schema of the message sent to Kafka by **streaming-twitter** component.

```
{
    "namespace": "ontos.com",
    "type": "record",
    "name": "atweet",
    "fields": [
        {
            "name": "date",
            "type": "string"
        },
        {
            "name": "content",
            "type": "string"
        },
        {
            "name": "key",
            "type": "string"
        },
        {
            "name": "kafkaTopic",
            "type": "string"
        },
        {
            "name": "geolocation",
            "default": null,
            "type": [
                "null",
                {
                    "name": "geolocationtype",
                    "type": "record",
                    "fields": [
                        {
                            "name": "lat",
                            "type": "double"
                        },
                        {
                            "name": "long",
                            "type": "double"
                        }
                    ]
                }
            ]
        },
        {
            "name": "source",
            "type": [
                "null",
                "string"
            ],
            "default": null
        },
        {
            "name": "retweetcount",
            "type": "int"
        },
        {
            "name": "isfaved",
            "type": "boolean"
        },
        {
            "name": "user",
            "default": null,
            "type": [
                "null",
                {
                    "name": "usertype",
                    "type": "record",
                    "fields": [
```

```
                        {
                            "name": "id",
                            "type": "long"
                        },
                        {
                            "name": "name",
                            "type": [
                                "null",
                                "string"
                            ]
                        },
                        {
                            "name": "screenname",
                            "type": [
                                "null",
                                "string"
                            ]
                        },
                        {
                            "name": "followers",
                            "type": "int"
                        },
                        {
                            "name": "following",
                            "type": "int"
                        },
                        {
                            "name": "description",
                            "type": [
                                "null",
                                "string"
                            ]
                        },
                        {
                            "name": "tweetcount",
                            "type": "int"
                        },
                        {
                            "name": "lang",
                            "type": [
                                "null",
                                "string"
                            ]
                        }
                    ]
                }
            ]
        }
    ]
}
```

Schema used for messaging **fox-extractor** results.

```
{
    "namespace": "com.ontos",
    "type": "record",
    "name": "rdfcontent",
    "fields": [
        {
            "name": "rdf",
            "type": "string"
        },
        {
            "name": "key",
            "type": "string"
        },
        {
            "name": "kafkaTopic",
            "type": "string"
        }
    ]
}
```

B Supply Chain use case set up

Next we present the list of terms used to feed the twitter stream consumption.

```
August Friedberg GmbH
August Friedberg
Autoliv BV & Co. KG
Berg & Co. GmbH Bielefeld
Bergler GmbH & Co. KG
Bergmann Maschinenbau
Blechformwerke Bernbach GmbH
Blomberger timber industry
Brangs & Heinrich GmbH
Con Garcia Industrial Coating
Continental AG
Conti
Continental
DE0005439004
ContiTech
Continental Teves AG & Co. OHG
ContiTech Power Transmission Systems GmbH
ContiTech Vibration Control GmbH
DBW Metallverarbeitungs GmbH
DKM Dodendorfer plastic
German Pentosin-Werke GmbH
Dichtungstechnik G.Bruss
ESKA Automotive GmbH
Filzfabrik
Fischer + Plath GmbH
FlammMOTEC GmbH & Co. KG
Frankenfeld Service GmbH
Friedr. Schmücker GmbH
Galvanotechnik Kessel GmbH & Co. KG
Gutbrod Schmölln GmbH
Hella Vehicle Components GmbH
Hilgendorf GmbH + Co.
Ina Schaeffler KG
Jama plastics processing
Küster ACS GmbH
KKF Fels GmbH & Co.KG
KUKA toolmaking
KWD Automobiltechnik GmbH
Karl Schöngen KG
Karosseriewerke Dresden GmbH
Kautex Textron GmbH & Co. KG
Kayser A. GmbH & Co. KG
KmB technology
Plastics processing
Linde & Wiemann
Magna International. Stanztechnik GmbH
Mann & Hummel Automotive GmbH
Nedschroef Plettenberg GmbH
Oris Fahrzeugteile
Pierburg Pump Technology GmbH
Procter & Gamble Intern.Operations SA
Profas GmbH
Wheel system GmbH
Robert Bosch Elektronik GmbH
Robert Bosch GmbH
SMH steel magazine
SMP Germany GmbH
Sachsentrans
Salzgitter Automotive
Schnellecke Logistics
Schnellecke Transport Logistik GmbH
Quick Corner
Schnellecke logistics
Sika Automotive
Sumitomo Electric Bordnetze GmbH
Sumitomo
TI Automotive (Gifhorn) GmbH
TRW Germany GmbH
Theysohn Kunststoff GmbH
Volkswagen AG
DE0007664005
VOW3
VW
Volkswagen
```

```
Wolpers GmbH
ZF Friedrichshafen AG
ZF
ZF Friedrichshafen
ZF Group
```

Following we present a lost of queries to explore the data.

```
# tweets per language
SELECT  ?language (count (?t) as ?tweets)
WHERE {
  ?t a <http://schema.org/ShareAction> .
  ?t <http://schema.org/inLanguage> ?language
}
group by ?language

# found entities by type
SELECT ?type (count(?s) as ?subjects)
WHERE {
  ?s a ?type .
}
group by ?type

# most mentioned entities by language
SELECT ?subject ?lang (count(?tweet) as ?tweets)
WHERE {

  ?tweet a <http://schema.org/ShareAction>.
  ?tweet <http://schema.org/about> ?subject .
  ?tweet <http://schema.org/inLanguage> ?lang .
}
group by ?subject ?lang
order by DESC(?tweets)

# most popular persons
SELECT ?subject (count(?tweet) as ?tweets)
WHERE {
  ?tweet a <http://schema.org/ShareAction>.
  ?tweet <http://schema.org/about> ?subject .
  ?subject a <http://schema.org/Person> .
}
group by ?subject
order by DESC(?tweets)
limit 20
```