KANN ICH ECL (aus etosha-core) auch in spark-shell nutzen? Falls ja, dann wird ECL.sh entfernt. 1

Fact Creation 1

Direct Fact Persistence 1

Flexibility of the Graph Model 2

Metadata and Dataset Management: An Example 4

Collect Dataset Profiles, and Flow Descriptors 4

Dataset description: Contribution to the Data-Catalog The raw data has a schema: AVRO 4

Describe the Indexing Procedure: AVRO files to Solr-Collection 5

Expose Metadata using Etosha CLI 6

How do we organize Context Information for Public Exposure? 7

The Turtle Syntax 7

Knowledge Integration 7

Expose Metadata using Etosha DSPM 8

Appendix 9

Start a Fuseki-Server to manage the Knowledge graph 9

# KANN ICH ECL (aus etosha-core) auch in spark-shell nutzen? Falls ja, dann wird ECL.sh entfernt.

# Fact Creation

|  |  |
| --- | --- |
| **Name** | **Description** |
| Triple Creation | Direct definition of facts within user context |
| Object model to triples | Object structure is translated to triple set based on a set of direct definitions |
| CSV to triples | Recurring procedure applied per record; each line of a CSV file represents a small subgraph with well defined objects and relations. |

Table 1 : Types of Triplification

### Direct Fact Persistence

In a data analysis session, a lot of variables are known and datasets can be accessed through a variety of access patterns using multiple APIs. This allows extraction of more facts and key value pairs (variables with a value represent key value pairs) are the result. Since we plan to store the data in a semantic graph, we have to provide triples of the following structure: (S,P,O)

The variable name represents the predicate P and the value becomes the object O.

(K,V) => (P,O)

The current context – such as the actual processing session we use for fact extraction – is labeled as C in our example. A triple consists of (S,P,O). In order to handle the context C we can either use C as subject directly. Here is another example: for a daily dataset, one counts the nr of bad records per day. Our analysis session is done on Saturday for the previous 5 week days. The day of the session is not relevant. Thus, the subject should rather be each weekday for which the number of bad records is determined.

(S, P, O)

(day1, number\_of\_bad\_records, 4)

(day2, number\_of\_bad\_records, 4)

(day3, number\_of\_bad\_records, 4)

This example needs improvements. The context is not 100% clear yet. We do not get the information about the origin of the extracted facts. There is no notion of the source of the data from which this metadata was taken from, nor can we see any detail about the extraction procedure, e.g., it could have been a complete counting or simply an estimation.

We can solve this problem by adding more information to the predicate, or by adding a fourth component to the triple which transforms it to a quad.

Finally we get:

|  |  |
| --- | --- |
| Extended predicate | Quad |
| (S, ext\_P, O)  (day1, number\_of\_bad\_records\_in\_DS1, 4) | (C,S,P,O)  (DS1,Day1, number\_of\_bad\_records, 4) |

### Flexibility of the Graph Model

One of the huge advantages of the semantic graph is its ability to be transformed into other representations. The creation of a wide table containing multiple facts about a set of entities which can be stored as CSV file is done by joining lots of triples on their subject. Every predicate becomes a column, in which the object is presented as cell-content.

Furthermore, from such CSV files we can take the data which populates the edge list and the node list which both are used to populate a property graph.

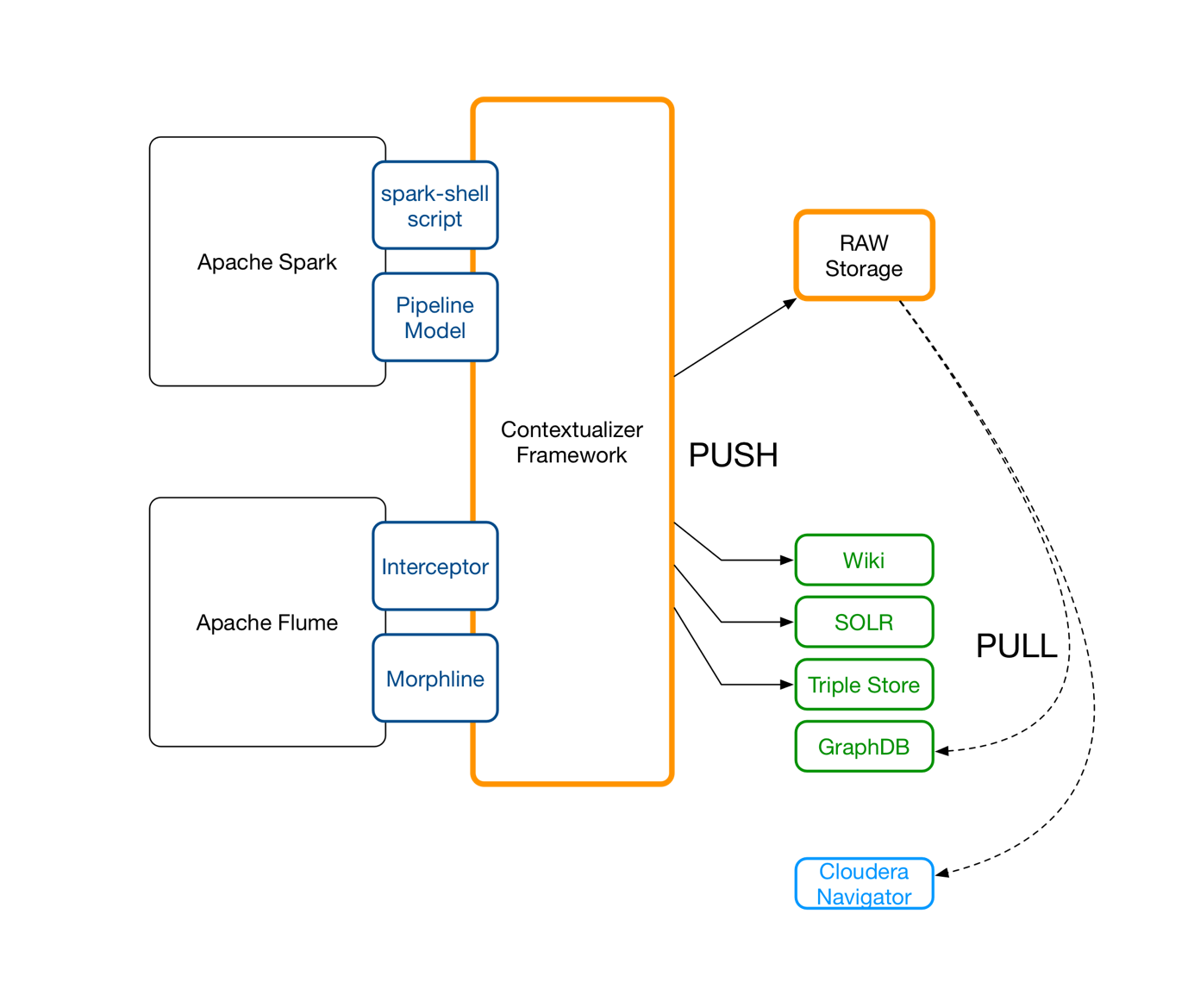
The Cypher query-language is used to extract facts from such a graph, which results in a subset of nodes and edges. Those can all be stored in node- and edge-lists for a direct import into a graph database, such as Neo4J. Furthermore, the node- and edge-lists can be translated into a triple set directly.

It is pretty easy to manage such context data in spread-sheets manually, e.g., for a particular study or in a specific project. Integration of metadata from multiple projects and multiple teams becomes a translation and transformation challenge. Instead of building a new metadata management solution, Etosha focuses on providing the building blocks which can be used to handle the knowledge graph. We assume, that if funding is available, a permanent metadata service can be established – but this will be the next part of the story.

The Etosha knowledge graph is a distributed data structure organized in a decentralized system. Each participant is responsible for his view. To achieve the highest level od accuracy and completeness one would have to import all known facts from all contributors into its own knowledge store. The initial version of the system has no distributed query infrastructure. This design defines clear boundaries. This boundaries lower the legal and operational barrier significantly.

The contextualizer framework is responsible for fact generation and extraction. Such facts are persisted by pushing the data into the raw storage. This raw storage contains the triples in a generic machine readable format. Alternatively, we can push directly into a Wiki to achieve a human readable format. Pushing the data into the SOLR server (full-text search engine based on Apache Lucene) or into the Fuseki server (triple store based on Apache Jena) increases usability, but both are optional.

Cloudera Navigator is an enterprise metadata management systems for Hadoop. More generic metadata management solutions have been build on Neo4J. Thus, the Etosha toolbox allows pulling the data out of its raw storage in order to inject it into the existing external systems. Such pull operations can also be query based, e.g., using the wiki query language, the Solr query language, SPARQL, or Cypher.



# Metadata and Dataset Management: An Example

Jane is looking for data about products to study the trends in communication about the product and the life cycle of brands.

Ideally, she needs data from multiple online-markets and shop-systems. Useful sources are for example page-click statistics for product pages and adds, website-traces for users including order activity, and social media data, such as tweets from Twitter or Facebook messages to analyze the buzz from from the crows.

First, she starts to collect twitter data using Apache Flume. Technically this is a simple approach to get some initial records and impressions about the topic.

The tutorial about Twitter data collection is provided in appendix 1.

* Add section on Flume data collection about the topics (A) (B) (C)
* <http://www.tutorialspoint.com/apache_flume/fetching_twitter_data.htm>

Next, she starts to index the collected tweets. As shown in the above mentioned tutorial, near real time indexing can easily be implemented using Apache Flume and a Morphline. A first inspection of the index (e.g., the lists of most often used terms and term pairs) shows if our collection procedure and applied filters work as expected. But it seams to be a good idea to also collect some more tweets about related and also about totally unrelated topics for cross-validation and trend comparison purposes.

So far she has two data sets: (a) the raw tweet collection stored as AVRO files in HDFS and (b) a tweet collection which is technically just a Lucene index stored in HDFS as well.

## Collect Dataset Profiles, and Flow Descriptors

Both data sets should be discoverable by other people. Jane uses the Etosha exposure tool to expose the relevant metadata.

NOW WE SHOW HOW SUCH A DESCRIPTOR LOOKS LIKE   
(ONTOLOGY)

### Dataset description: Contribution to the Data-Catalog The raw data has a schema: AVRO

SCHEMA

The collection has a schema: SOLR

SCHEMA

### Describe the Indexing Procedure: AVRO files to Solr-Collection

FLOW DEMO

## Expose Metadata using Etosha CLI

Jane can choose between two description methods. Triples – which describe facts - can be generated without any restrictions using the create and link commands in the Etosha shell.

Alternatively, the DSPM components offer convenient methods, based on a pre-implemented knowledge structure. An instance of the DSPM model is used to interact with the knowledge-model it represents from within custom programs or even in the spark-shell. The EDT component can be used for direct triplification in case the DSPM does not cover the structure one wants to represent.

We start with some examples of direct triplification using the Etosha shell. After starting the shell with the command:

$ **ETOSHA SHELL**

etosha create dataset **tweetsTOPIC1**; // defines an entity (node in the knowledge graph)

Now, she creates a flume-source (not Java code, but in the MD store) to track the configuration of the flume agent, which was used to collect the data:

etosha create datasource **flumeAgent01**; // defines an entity (node in the knowledge graph)

A link is created between both nodes like this:

etosha link **tweetsTOPIC1** ‘**hasSource’** **flumeAgent01;** // links **subject** and **object** via **predicate**

NOTE: **‘hasSource’** plays the role of an URI-shortcut if quotes (‘…**’**) are used. Plain text is interpreted directly as URI as in RDF syntax to store triples in the so called Turtle format.

Furthermore, she adds some free-form tags to the dataset node:

etosha add tag to dataset **tweetsTOPIC1 isAbout TOPIC1**

etosha add tag to dataset **tweetsTOPIC1 range\_start 2017-01-01-00-00-00**

etosha add tag to dataset **tweetsTOPIC1 range\_end 2017-01-21-23-59-59**

etosha add tag to dataset **tweetsTOPIC1 to\_column col1:max=10000**

Now she can persist this data in our knowledge graph by calling the persist methods:

etosha persist ttl

As a result, the current model state is stored as a turtle file in the default location on the machine, where the etosha shell is executed.

One can see the underlying triplification pattern.   
  
etosha add tag to dataset subject predicate object.

etosha add tag to dataset **tweetsTOPIC1 isAbout TOPIC1**

etosha add tag to dataset **tweetsTOPIC1 range\_start 2017-01-01-00-00-00**

etosha add tag to dataset **tweetsTOPIC1 range\_end 2017-01-21-23-59-59**

etosha add tag to dataset **tweetsTOPIC1 to\_column col1:max=10000**

**DISKUSSION**

Currently, Jane uses low level functions implemented in the etosha-core package to expose knowledge as facts represented by triples to a central location in HDFS. From here, the graph contributions are exported by external tools such as the fact collector services or simply by using git command.

### How do we organize Context Information for Public Exposure?

The folder /etosha/cluster/local contains all Etosha related metadata about entities which belong to the cluster in which this folder is stored. This folder contains files in N-Turtle format. All the files in this folder define the present scope. Further enhancements are possible by time based partitions or by other subfolder structures which reflect domain specific scopes. The initial implementation uses only the preconfigured default folder.

|  |  |
| --- | --- |
| Variable | Value |
| ETOSHA\_DEFAULT\_FOLDER\_LOCALFS | $USER\_HOME/etosha |
| ETOSHA\_DEFAULT\_FOLDER\_HDFS | /etosha/cluster/local |

As result, Jane has generated the RDF file named:   
/home/jane/etosha/model\_LONG-TS.ttl

### The Turtle Syntax

<flumeAgent01> <produces> "tweetsTTIP\_CETA" .

<flumeAgent01> <isOftype> "etosha:datasource" .

<tweetsTTIP\_CETA> <isAbout> "TTIP" .

<tweetsTTIP\_CETA> <isAbout> "CETA" .

<tweetsTTIP\_CETA> <hasColumn> "col1" .

<tweetsTTIP\_CETA> <isProducedby> "flumeAgent01" .

<tweetsTTIP\_CETA> <isOftype> "etosha:dataset" .

<col1> <isColumnIn> "tweetsTTIP\_CETA" .

<col1> <typeOfIdsUsed> "twitterUserAccounts" .

Listing 1: Content of /home/jane/etosha/model\_LONG-TS.ttl

LINK TO TURTLE DEMO.

## Knowledge Integration

She loads this RDF file into Fuseki in order to enable SPARQL queries on the exposed metadata or better to say, on the exposed knowledge graph.

This procedure is called PULL mode because the triples are pulled by an external tool from Etosha temp storage.

$ curl example for FUSEKI

We assume that ruby and curl are installed on the host, from which the turtle files will be loaded into the triple store.

## Expose Metadata using Etosha DSPM

Next, Jane creates a Spark DataFrame from raw AVRO data. Some additional information like max, min, and number of distinct words in a column can easily be calculated for bound datasets. For static datasets it also makes sense to persist such facts as soon as they are available. In case one needs such values more often, the query planer can even use the existing data instead of reprocessing again.

The Etosha contextualizer components can also work in PUSH mode. We use the dataset profiler to illustrate this functionality.

* Spark DataFrame profiler example

* RDD describe function for columns

A detailed tour through Etosha data capturing procedures is provided in this document. How To: Transform Data Into Facts - A Quickstart Guide to Triplification in Hadoop

<https://docs.google.com/document/d/1hPUm9SZAGEmK94854qEqM0dZIfkbYWKLodRyhINiOZk/edit#heading=h.t8q6ccruh16m>

# Appendix

## Start a Fuseki-Server to manage the Knowledge graph

[root@quickstart scripts]# cat run.sh

#!/bin/bash

export FUSEKI\_HOME=/opt/fuseki/scripts/main/apache-jena-fuseki-2.3.0

export JAVA\_HOME=/opt/jdk1.8.0\_101

#

# Here we use a CLI parameter to define a location of the modelfile and the port.

#

CMD=$1

CORE\_MODEL\_FILE=/home/cloudera/.etosha/model.ttl

WEBSERVER\_PORT=9999

PART\_FOLDER=/home/cloudera/.etosha

#

# We need a time stamp

#

####### TS=$(time)

######################################

#

# Etosha Triple Collector Service

#

case $CMD in

(start)

clear

echo " FUSEKI\_HOME: $FUSEKI\_HOME"

echo " JAVA\_HOME: $JAVA\_HOME"

echo " PORT: $WEBSERVER\_PORT"

echo " CORE\_MODEL\_FILE: $CORE\_MODEL\_FILE"

echo " ETOSH-GRAPH-FOLDER: $PART\_FOLDER"

echo ">>> Starting the Fuseki-Server on port [$WEBSERVER\_PORT] (default: 3030)"

# clean

# ### BE CAREFULL ### rm -r $PART\_FOLDER -f

#

######## mv $PART\_FOLDER /tmp/$TS-ETOSHA-PARTITION-OLD

# load files to local from HDFS default location

######## hdfs dfs -get /etosha/snap/\* $PART\_FOLDER

exec $FUSEKI\_HOME/fuseki-server --file=$CORE\_MODEL\_FILE --update --port=$WEBSERVER\_PORT /ETCS &

sleep 5

FILES=$PART\_FOLDER/\*

for f in $FILES

do

echo "> LOAD ETOSH-GRAPH-SNAPHSOT: $f ..."

# take action on each file. $f store current file name

$FUSEKI\_HOME/bin/s-post http://localhost:$WEBSERVER\_PORT/ETCS/data default $f

done

;;

(list)

$FUSEKI\_HOME/bin/s-query --service http://localhost:$WEBSERVER\_PORT/ETCS/query 'SELECT \* {?s ?p ?o}'

;;

(reload)

;;

(\*)

echo "Don't understand [$CMD]"

;;

esac

root@quickstart scripts]# ./run.sh start

FUSEKI\_HOME: /opt/fuseki/scripts/main/apache-jena-fuseki-2.3.0

JAVA\_HOME: /opt/jdk1.8.0\_101

PORT: 9999

CORE\_MODEL\_FILE: /home/cloudera/.etosha/model.ttl

ETOSH-GRAPH-FOLDER: /home/cloudera/.etosha

>>> Starting the Fuseki-Server on port [9999] (default: 3030)

[2016-09-29 15:37:30] Server INFO Dataset: in-memory: load file: /home/cloudera/.etosha/model.ttl

[2016-09-29 15:37:31] Server INFO Fuseki 2.3.0 2015-07-25T17:11:28+0000

[2016-09-29 15:37:31] Config INFO FUSEKI\_HOME=/opt/fuseki/scripts/main/apache-jena-fuseki-2.3.0

[2016-09-29 15:37:31] Config INFO FUSEKI\_BASE=/var/run/cloudera-scm-agent/process/85-fuseki-FUSEKI\_SERVICE/scripts/run

[2016-09-29 15:37:31] Servlet INFO Initializing Shiro environment

[2016-09-29 15:37:31] Config INFO Shiro file: file:///var/run/cloudera-scm-agent/process/85-fuseki-FUSEKI\_SERVICE/scripts/run/shiro.ini

[2016-09-29 15:37:31] Config INFO Register: /ETCS

[2016-09-29 15:37:31] Server INFO Started 2016/09/29 15:37:31 PDT on port 9999

> LOAD ETOSH-GRAPH-SNAPHSOT: /home/cloudera/.etosha/model.ttl ...

[2016-09-29 15:37:34] Fuseki INFO [1] POST http://localhost:9999/ETCS/data?default

[2016-09-29 15:37:34] Fuseki INFO [1] POST /ETCS :: 'data' :: [text/turtle charset=UTF-8] ? default

[2016-09-29 15:37:34] Fuseki INFO [1] Body: Content-Length=12219, Content-Type=text/turtle, Charset=utf-8;charset=utf-8 =>

Turtle : Count=120 Triples=120 Quads=0

[2016-09-29 15:37:34] Fuseki INFO [1] 200 OK (60 ms)