Building a Data Catalog using   
Semantic Logging

1. Introduction & Motivation 2

2. Features of a Data Catalog 3

2.1 Discovery and Automated Tagging 3

2.2 Crowdsourcing 3

2.3 Ratings and Reviews 3

2.4 Integration Interfaces 3

2.5 Etosha Data Catalog is More than just Search 3

3. Distinguish Implicit and Explicit Triplification 4

4. Recap: Hadoop for Big Data Processing 5

5. A Data Driven Approach for Knowledge Management 6

6. Semantic Logging Framework for Knowledge Workers 7

6.1 A simplified data science process model 8

6.2 Fact Creation 10

6.2.1 Direct Fact Persistence 10

6.2.2 Flexibility of the Graph Model 11

6.3 The Knowledge Graph Approach 12

6.4 First Steps: Contextualizers are Building Blocks for Semantic Logging 12

6.5 Share and Merge 14

6.6 Model Conversion and Multiple Query strategies 15

7. Metadata and Dataset Management: A Complete Example 15

7.1 Collect Dataset Profiles, and Flow Descriptors 15

7.1.1 Dataset description: Contribution to the Data-Catalog 16

7.1.2 Describe the Indexing Procedure: AVRO files to Solr-Collection 16

7.2. Expose Metadata using Etosha CLI 17

7.3 How do we organize Context Information for Public Exposure? 18

7.4 The Turtle Syntax 18

7.5 Knowledge Integration 18

7.6 Expose Metadata using Etosha DSPM 19

8. Conclusion and Outlook 19

8.1 Our achievements 19

8.2 Next Steps: Dynamic Triplification 19

References 20

Appendix 20

Start a Fuseki-Server to manage the Knowledge graph 20

# 1. Introduction & Motivation

In a data driven world we have to navigate along new paths. Big Data technology is celebrated as the magic which opens up new dimensions. This can be a risky challenge besides its huge advantages. Not much was done in the past to map the “data world”. Search technology was driving the usage of the internet but search alone gives us no strategic orientation in the data space. Equally, taking paper and ink does not mean that you have a map for your journey already in your hands.

Like a traditional map needs paper and ink, in our modern world a mobile device needs appropriate apps – but the content is what matters most, followed by usability aspects. This is why data catalogs should not be evaluated by completeness and accuracy only. Some means of fuzzy search and even fuzzy information is sometimes better than having no information at all.

Let’s look deeper into usability. A data catalog should be open in terms of technology and access to it’s content. Since the catalog contains data about the data - not the data itself - we shouldn’t be too picky. But as always – being careful is better than too much of risk, especially in a very open world. With this in mind, we should say: “The data inside the data catalog should be as open as possible for authorized people and systems.” We have to build up the data catalog without technical limitations which could stop us from exporting and merging multiple catalogs.

Time evolution of the content matters as well. Data sets are usually not static as books are – especially in near real time scenarios (driven by an IoT world) we have to consider the life cycle stages of systems and datasets.

The most important design goal is to be independent from a particular programming language and storage technology. Obviously, individual representations of the data model come with particular benefits with regards to accessibility and functionality. The universality of the text based RDF format and the text based wiki/markup languages support our goals. Incorporating ontologies and data transformation tools allows us to integrate into existing data governance infrastructures and to stay independent of any APIs, which change quite fast nowadays. The data in the data catalog is important – not primarily the media it holds.

Since we develop a data driven project it is not a particular software stack which is in our focus, but rather the flexibility to implement the metadata management approach on top of multiple Big Data processing environments, no matter if on premise, cloud based or in hybrid environments.

# 2. Features of a Data Catalog

The Etosha project builds the open source components for open data catalogs. This means that data inspectors and profilers for integration in your own application or simply for interactive use in the Spark-shell are the fundamental results of the project. Full-text search, a graph database, and an RDF store build the backbone for multiple access strategies. RDF was chosen as data export and exchange format. Finally, the success of Git technology motivates us to offer a rich set of publication and integration capabilities.

The following section addresses some issues, raised by Todd Goldman in his blog [1]:

## 2.1 Discovery and Automated Tagging

Due to the large amount of information and data which surrounds us we need an automatic approach beside crowd-sourcing.

*Etosha offers data inspection procedures and routines for automatic execution during data ingestion or directly afterwards, e.g., footprints of individual table columns (structured data sets) are extracted for classification and comparison with future datasets.*

## 2.2 Crowdsourcing

Especially for collaborative projects it is important to share knowledge and to learn from others. This means, integration of knowledge contributed by others is an essential functionality.   
*Etosha allows export and import of metadata and data models using the RDF data format.*

## 2.3 Ratings and Reviews

Automatic data extraction can lead to non accurate or even completely wrong results. Users, especially humans have to evaluate and correct such results. Reviews, ratings provided by humans combined with automatic suggestions, and cross-validation procedures provide the quality assurance we need in an open data catalog.

*Etosha uses an open feedback system – integrated into Github. In the future, we want to establish the Global Data Map (GDM).**GDM is a specific view on a data catalog (or on multiple aggregated data catalogs) which provides context to data, such as dataset usage, dataset ratings, and user feedback.*

## 2.4 Integration Interfaces

Remember: content is what matters. The catalog provides information and knowledge for a variety of tools in multiple categories. From business glossaries to automatic ETL pipeline generation up to data governance and risk analysis – a lot of potential is already in your data catalog.   
*Etosha tools are focused on providing the best open source data catalog for the data driven business. It is on you to request the information. If it is not there yet, we have to find a way to* *provide the fact.*

## 2.5 Etosha Data Catalog is More than just Search

*Since search-like interaction is a common feature. We decided to build Etosha around a multi-layer search interface. The search dashboard is the starting point for your work with the Etosha data catalog. We simply replaced the good old menu bar by some stored searches for quick reports, enriched by multiple search facets and visuals. Furthermore, semantic search using SPARQL and the graph query language Cypher add more flexibility to your search experience. All of this has one goal: make your data catalog as accessible as possible.*

# 3. Distinguish Implicit and Explicit Triplification

Apache Hadoop clusters are used to integrate all types of enterprise data. ‘Any-structured’ data can be handled very flexible. Denormalized data models and indexes allow fast access to any entity while distributed data allows efficient parallel processing. Metadata is managed locally per cluster. SQL like queries are translated into MapReduce jobs via Hive or executed by the high performance Impala framework.

At this time the importance of RDF data stores and seems to be unclear. Some projects provide scalable triple stores on top of Hadoop or any cloud computing system.

Publishing of RDF data can be done in multiple ways. The project [Jena-HBase](http://www.utdallas.edu/~vvk072000/Research/Jena-HBase-Ext/jena-hbase-ext.html) [1] offers a [HBase](http://hbase.apache.org/) backed triple store that can be used with the Jena framework. Thus Jena-HBase provides end-users with a scalable storage and querying solution that supports all features from the [RDF specification](http://www.w3.org/RDF/). The [CumulusRDF](http://www.w3.org/2001/sw/wiki/CumulusRDF) [2] project uses [Apache Casandra](http://cassandra.apache.org/) as storage backend. Cumulus provides a REST-based API with CRUD operations to manage RDF data stored in cloud architectures. Storage connectors for [Apache SOLR](http://lucene.apache.org/solr/) running as SOLR cloud and HBase are under development.

According to [1]: “*Lack of scalability is one of the most significant problems faced by single machine RDF data stores*.” Implementation of scalable triple stores is considered to be a useful but un efficient approach as it requires an explicit triplification of all considered data sets. This leads to tremendous processing requirements and causes increased storage costs due to additional data redundancy. On the other side, using existing tools such as [1,2] makes this approach a candidate for a quick prototype.

A complementary approach is implicit triplification which means, structural information about a dataset is used to translate a SPARQL query into SQL statements, which are executed on traditional database servers or even using Impala. The project [Triplify](http://semanticweb.org/wiki/Triplify) [3] follows this a simple approach to publish RDF and Linked Data based on mapping HTTP‐URI requests onto relational database queries. Triplify transforms the resulting relations into RDF statements and publishes the data on the Web in various RDF serializations, in particular as Linked Data. But it is limited as it not supports SPARQL.

*Berners-Lee (2006) outlined a set of 'rules' for publishing data on the Web in a way that all*

*published data becomes part of a single global data space:*

*1. Use URIs as names for things*

*2. Use HTTP URIs so that people can look up those names*

*3. When someone looks up a URI, provide useful information, using the standards*

*(RDF, SPARQL)*

*4. Include links to other URIs, so that they can discover more things*

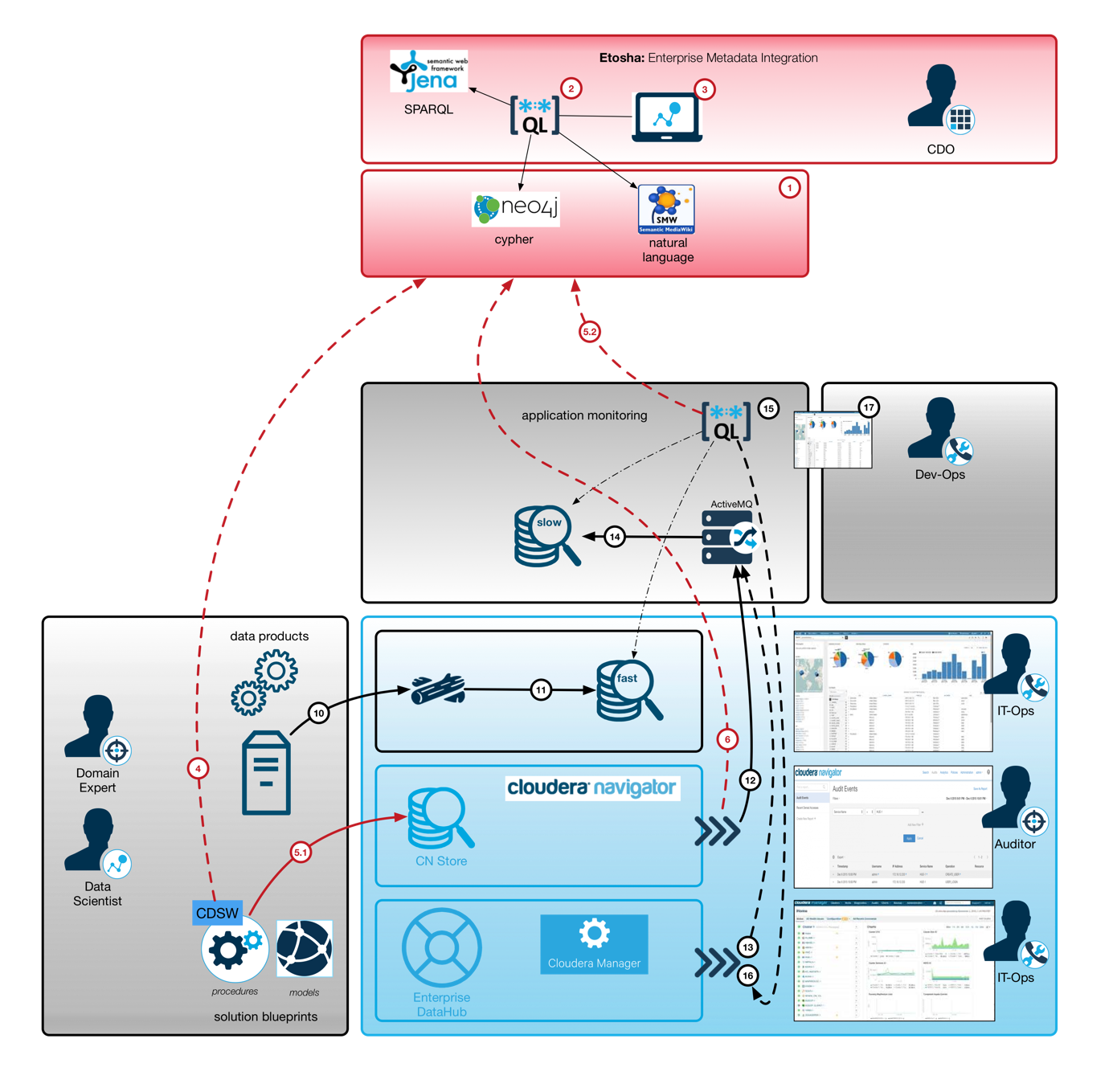
*These have become known as the 'Linked Data principles', and provide a basic recipe for*

*publishing and connecting data using the infrastructure of the Web while adhering to its*

*architecture and standards. We add the following: Provide a context and define a scope to restrict the focus to a given context. Within a context we can apply several normalization procedures.*

# 4. Recap: Hadoop for Big Data Processing

Big Data applications and Data analysis procedures should be done in highly integrated environments rather than in individual isolated systems. Nevertheless, we find a coexistence of multiple Hadoop cluster in many companies. The reasons for running multiple clusters are manifold – but out of our current scope. Obviously, there is a demand for another integration layer beyond data flows and process chains. To start with, one can implemented a four-layer metadata management stack as shown in the following figure:



**Figure 1 :** Enterprise data management platform based on CDH (light blue) with customized data products (gray), data science workbench (dark blue), and metadata management tools (red).

Figure 1 shows the Big Data and AI / ML infrastructure which consists of three different component types. Cloudera Enterprise offers the Hadoop ecosystem. Specific user frontends provide already a lot of details to administrators, operators and even auditors. Besides those out of the box capabilities, the gray boxes show individual extensions, such as custom applications, data science methods implemented in data products and even the brand new data science workbench. Since those scientific tools are not yet integrated completely into the core monitoring and audit stack we implemented some additional handlers to grab and organize relevant metadata.

# 5. A Data Driven Approach for Knowledge Management

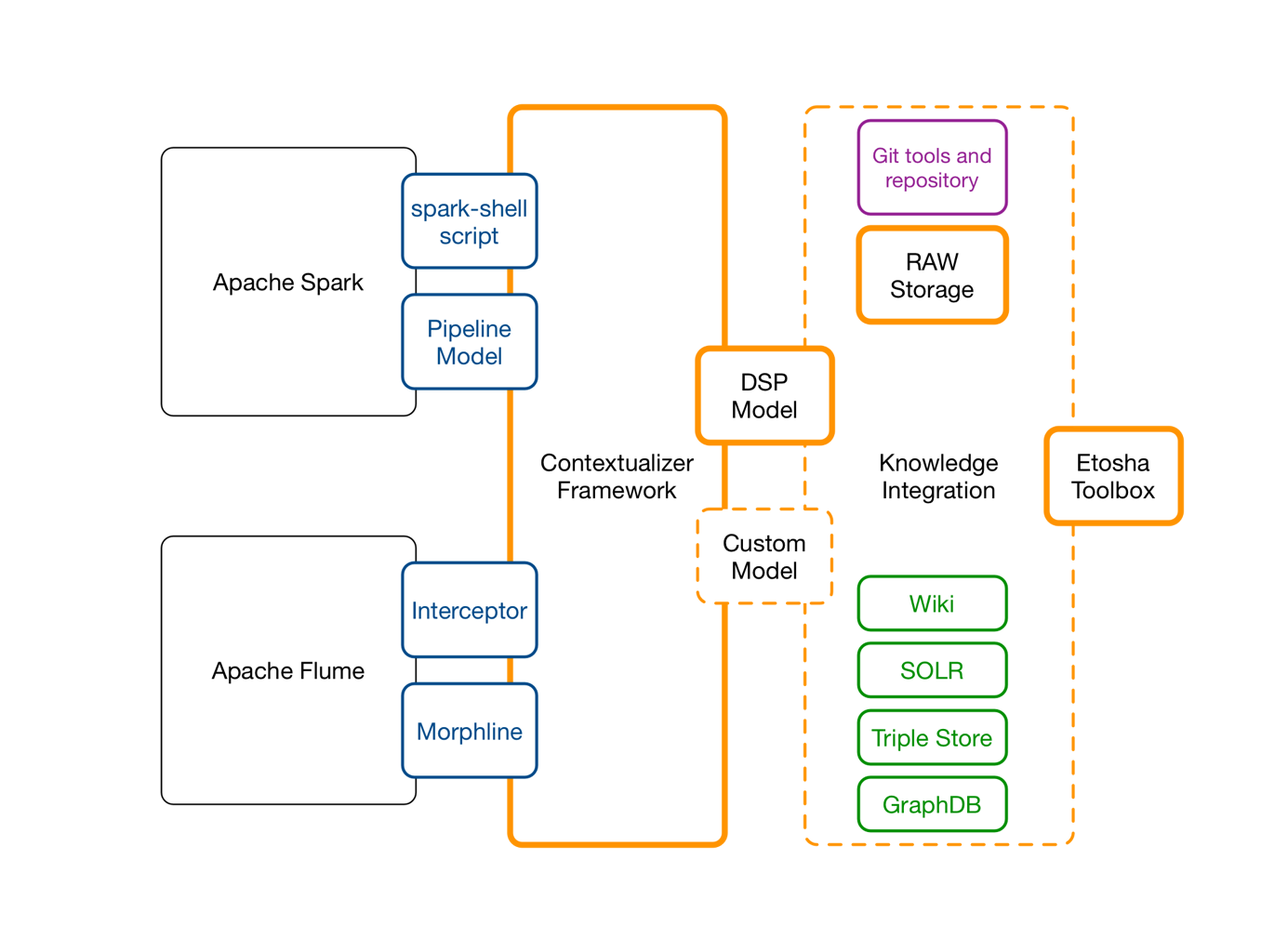
The cluster related monitoring and auditing is not intended to be a shareable asset – but knowledge about datasets and procedures in general could be such an asset. The Etosha tools solve this problem by forming an **open (meta)data capturing layer**. It can be used for automatic fact extraction, but also in a free style mode by creative data workers. The context of the information in our focus is the organizational and the methodological level rather than physical data representation and security aspects such as encryption and access control. Different aspects needs to be provided to different audiences.

This explains our **multi-layer approach**. The Etosha tools are complementary to existing views. Data can be exported and imported and finally, metadata models from multiple clusters can be integrated into one view. We decided to implement the core data model as an RDF graph to maximize the flexibility and extendibility with at least external dependencies as possible.

**Semantic logging** allows us to expose structured and unstructured information with a high degree of reusability. Log analysis is common sense today, but a lot of effort goes into log parsing. Standardized log messages are also an important step forward towards knowledge integration[[1]](#footnote-1). Furthermore, the huge amount of technical logs generated during data ingestion or by automatic procedures needs a different handling than models, generated by data scientists. Creative approaches with new algorithms can not be covered by standard log formats but using RDF representations and domain specific ontologies offer such flexibility.

# 6. Semantic Logging Framework for Knowledge Workers

The following figure shows the main components of the Etosha framework (orange).



**Figure 2 :** Loosely coupled software tools (orange boxes) together with established standard software (black: Big Data environment, purple: data integration and modularization, green: query engines) allow the realization of a continuous data capturing procedure to collect process knowledge.

Contextualizers can be embedded into multiple places (dark blue), e.g., for ad-hoc capturing during data science procedures (black, Apache Spark), or for automatic extraction of facts in data ingestion pipelines (black, Apache Flume). The components in which semantic logging is used are blue. The core framework offers some software components (orange line) and a methodology (dashed orange line). The knowledge graph is stored in text files using the RDF format. This allows us to handle facts like source code and git is used (see purple box) for sharing and integration of this data. For reliability and scalability, we store raw facts in HDFS Multiple optimization strategies exist and can be applied in a transparent way. Finally, the end user functionality is composed by orchestration of multiple “state of the art” solutions. We use a Mediawiki for user interaction and feedback. SOLR cloud offers facetted search and full-text search capabilities. Apache Jena is used as triple store and SPARQL query engine. The graph database Neo4J offers the cypher query language. SPARQL and Cypher are competing and complementary at the same time. Both can be used for analysis and exploration of graph data. While SPARQL is a natural fit for semantic projection and integration of multiple ontologies we use Cypher on the property graph for topological analysisof the knowledge graph.

Semantic logging as a data capturing method leads to a continuously growing data set: the knowledge graph. The tooling around it, in this case the Etosha framework is related to it initially but not an inherent part as in traditional application centric solutions.

Traditional IT systems use a very specific, often internal and not completely documented data format. Export is often not easy and sometimes not wanted. An open data format on the other hand requires individual tool adoption, but it doesn’t dictate the technology which needs to be used to work with the data. This degree of freedom is an important design goal in our project.

## 6.1 A simplified data science process model

The following screenshot shows a simplified process model for data science (DSPM). The green nodes represent users who are also the responsible persons for particular procedures (red nodes) and thus need access to certain datasets (blue nodes). Only such a knowledge graph can show and illustrate the interdependencies between the procedures, datasets, and people which all collaborate in one particular context – learning a using a classification model within a particular business context.



<https://www.youtube.com/watch?v=ZE7Gcanv90s&feature=youtu.be>

**--- Formalizing the approach ---**

****

Problem:  
 Classification of unknown documents. (Background)

Procedure to solve the problem:

Data ingestion (Flow)  
Docment collection (Dataset)  
Model training (Modelpipeline)

Solution:

LDA or KMeans (Model)

Model evaluation (Flow)

Technical Outcome:

Real time classification pipeline: (Flow)

Business impact:

Inspection and analysis of metadata (Query on Metadata graph)

This example shows how four concepts (datasets, dataflows, modelpipelines, and models) are combined to describe the relevant aspects in a text classification use case. Properties of the datasets, dataflows, modelpipelines, and models are the facts and thus are expressed as triples.

## 6.2 Fact Creation

Facts are extracted by contextualizers (triple creation and object model translation) or imported from structured data (CSV files or query results)

|  |  |
| --- | --- |
| **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

### 6.2.1 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 are the result. Variables with a value represent key value pairs. 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. There is another approach: for a daily dataset, one counts the number 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, 8)

(day3, number\_of\_bad\_records, 0)

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 on all records or simply an estimation using samples.

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) |

### 6.2.2 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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | ID | Color | Age | | Car1 | Red | 12 | | Boot2 | Blue | 32 | | Bike3 | Black | 17 |   CSV file | |  |  |  | | --- | --- | --- | | S | P | O | | Car1 | hasColor | Red | | Car1 | wasBuilt | 2005 | | Boot2 | hasColor | Blue | | Boot2 | wasBuilt | 1985 | | Bike3 | hasColor | Black | | Bike3 | wasBuilt | 2000 |   Triples |

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 property graph graph. SPARQL queries add the ability for semantic reasoning. Queries lead to subgraphs, which are simply subsets of the 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. This shows how the individual representations are exchangeable and complementary.

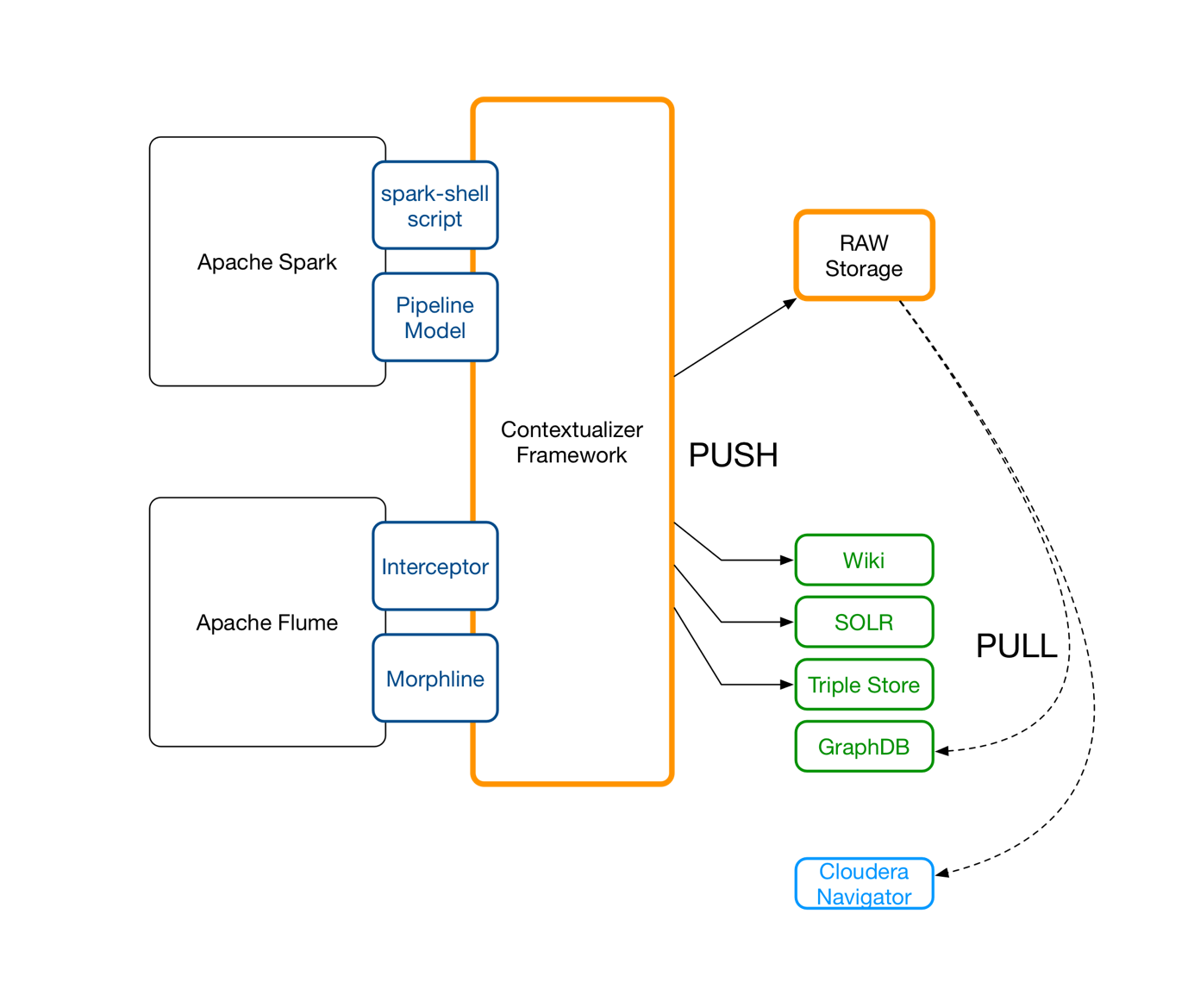
It is pretty easy to manage context data in spread-sheets manually, e.g., for a particular study or in a small project. Integration of metadata from multiple projects and multiple teams becomes a translation and transformation challenge. Instead of building a new metadata management solution, Etosha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

## 6.3 The Knowledge Graph Approach

The Etosha knowledge graph is a distributed data structure. It is organized in a decentralized system. Each participant is responsible for his own view. To achieve the highest level of 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 (see section … ) 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 system for Hadoop. Multiple 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.



**Figure 3:** Push and Pull operations allow a decoupled distributed operations model.

## 6.4 First Steps: Contextualizers are Building Blocks for Semantic Logging

Contextualizers extract relevant information from datasets (at a given point in time) and data processing components (during a period of time). Furthermore, developers and data scientists use contextualizers to capture information exactly at this point in time, when most facts and the context are obvious. This reduces the complexity for data capturing and increases precision dramatically, especially for ad-hoc data capturing during creative procedures. A contextualizer can be compared with a logger component which is widely used in software development. Like in the case of loggers, multiple output paths and output handling strategies exist. We develop contextualizers to capture special facets of an algorithm, a dataset, or a full processing pipeline.

Initial set of Etosha contectualizers:

**DatasetContextualizer**

Non technical metadata,

Usage statistics,

Content statistics: column & partition profiles

**ModelContextualizer**

Description of model properties and evaluation results

**ModelpipelineContextualizer**

Description of arbitrary ML procedures (model creation)

**FlowContextualizer**

Data ingestion & gf\_processing (data preparation, model application)

6.4.1 Contextualizers in Action

<< show examples of what is captured by a contextualizer and how a contectualizer used >>

|  |  |
| --- | --- |
| Code | RDF output |

## 6.5 Share and Merge

Extracted metadata can be pushed into multiple systems in different formats. To begin, we simply store the facts in a git project. Git offers versioning, merging, and issue tracking functionality together with a decentralized infrastructure. Collaboration across multiple teams and multiple technology stacks becomes easy this way.

<< show model combination >>   
using multiple loads into one triple store or one Neo4J instance ….

A git project typically contains source code and configuration files for software projects. Documentation can be part of the project as well. Where can we store the analysis metadata? We can identify multiple scopes: (a) project scope, (b) organizational scope, (c) cluster scope. Project and organizational scopes can be distributed across multiple clusters or bound to one cluster or a even a particular workstation. The cluster scope is bound to one Hadoop cluster and can cover multiple projects and multiple organizations.

Furthermore, we should figure out if the process for which we want to track the context is bound to a particular software project. Usually, the artifacts of multiple development processes are combined in an analysis scope. This is why we create a new git project just for tracking our process or use case specific metadata. This git project can be used in multiple context by multiple users as submodule.

Mounting multiple submodules allows us also to generate an aggregated view containing multiple processes or use cases for deeper dependency analysis. Projects with an “etosha.ini” file are called context aware. Starting in a particular root folder we can identify all context aware subfolders / subprojects an load contributions to the overall knowledge graph into one triple store. As a consequence, a merged knowledge-graph emerges.

## 6.6 Model Conversion and Multiple Query strategies

Individual storage components support specific access patterns. In our case we need a very flexible integration layer and scalable long term storage, combined with analytical power. Thus, we use HBase for raw data and images, and SOLR for keeping multiple indexes. Neighborhood exploration is based on the RDF graph and the property graph (Neo4J). Finally, a human readable representation is provided by a Wiki, such as Confluence or Semantic Media Wiki (SMW). Depending on the project we are able to select the right components and also to change them flexibly.

<< show model trafo >>

Indexing to SOLR

Collection to CSV to Neo4J  
XSLT and Velocity

# 7. Metadata and Dataset Management: A Complete 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 crowds.

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) and process logging
* <http://www.tutorialspoint.com/apache_flume/fetching_twitter_data.htm>

Next, she wants 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 types of data sets: (a) the raw tweet collection stored as AVRO files in HDFS and (b) a tweet collection which is technically just an Apache Lucene index stored in HDFS as well.

## 7.1 Collect Dataset Profiles, and Flow Descriptors

The 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)

### 7.1.1 Dataset description: Contribution to the Data-Catalog

The raw data has a schema: AVRO

SCHEMA

The collection has a schema: SOLR

SCHEMA

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

FLOW DEMO for Spark or MR-Job.

To create a description of the NRT indexing pipeline we use a custom Flume interceptor. The interceptor is statefull and tracks the properties over a a sliding window. This state information can be indexed as well and finally a query bean is used for further analysis. This kind of metadata extraction is relevant especially for unbounded growing datasets.

## 7.2. 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. Fundamental skills in semantic modelling are required to achieve consistency.

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 Etocha context logger component (ECL) 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 context tool in an Apache Spark shell. After starting the spark-shell with the additional package:   
  
<<<ADD ETOSHA ECL JAR>>>

The first fact which is tracked is the topic Jane works on:

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

Now, she creates a flume-source (no Java code, but an entry in the MD store) to attach and 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 her 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. This folder should be inside the analysis project as described in section (……).

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 >>> LINK TO THEORY PART**

This shows, how Jane uses low level functions implemented in the etosha-core package to expose knowledge as facts represented by triples to a local git project (for a project scope) or in HDFS (for a cluster wide scope). From here, the graph contributions are exported by external tools such as the fact collector services or simply merged using git.

### 7.3 How do we organize Cluster wide Context Information for Public Exposure?

The HDFS folder /etosha/cluster/local contains all Etosha related metadata about entities which belong to a particular 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 then 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 |

### 7.4 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.

## 7.5 Knowledge Integration

She loads this RDF file into an embedded Fuseki server 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.

We assume that the curl command is installed on the host, from which the turtle should be loaded into the triple store.

$ curl example for FUSEKI

Alternatively she can push directly into the embedded or even a remote triple store.

<<< SHOW BOTH CONFIGURATIONS >>>

## 7.6 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 the bound dataset again.

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

* Spark DataFrame profiler example >>> PUSH to triple store

* RDD describe function for columns >>> PUSH to Wiki

# 8. Conclusion and Outlook

## 8.1 Our achievements

8.2 Next Steps: Dynamic Triplification

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

In our example we used Twitter tweets for a communication analysis study. Besides operational metadata we can extract more facts from the raw dataset using full text queries and faceted search capabilities of Solr. Triplification can be seen as a knowledge extraction technique which goes beyond traditional text search.

Based on this approach, it is possible to incorporate search on documents, source code, blogs and other social media. We simply translate the query results of a Solr query into an RDF graph instead of exporting the full index to the knowledge graph - which would be overloaded pretty soon. A combination of the semantic graph and multilingual full text search lowers the barriers and improves the creative flow of knowledge workers.

# References

[1] <http://blog.waterlinedata.com/blog/search-is-not-a-data-catalog?utm_content=52869351&utm_medium=social&utm_source=linkedin)>

# 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)

In this example, only a single Fuseki-server instance has been used. The Fuseki-Cloud proposal is under construction and needs some more work.

<https://docs.google.com/document/d/1gs1ftj9GuMLx7fdN5X5xTGxe9CLUNG0NMq9Fl3HN-ec/edit#heading=h.shertkpcttdg>

1. See presentation about structured logging. [↑](#footnote-ref-1)