OPAL
OPEN DATA PORTAL

Deliverable D4.3
Prototype index structures and entity recognition

Author: Caglar Demir
Reviewer: Adrian Wilke

Veröffentlichung	Öffentlich
Fälligkeitsdatum	30.06.2019
Fertigstellung	02.07.2019
Arbeitspaket	AP4
Typ	Software Documentation
Status	Final
Version	1.0

Abstract:
This deliverable describes the current architecture and configuration of the OPAL search engine and related index structures for performant retrieval of metadata of datasets in OPAL.

Schlagworte: Elasticsearch, index structures, data storage

mFUND
Das Startkapital für die Mobilität 4.0

Bundesministerium
für Verkehr und
digitale Infrastruktur

Content

1 Introduction

One of the primary goals of OPAL is to extract metadata of datasets from different catalogs and enable easy access to Open Data as specified in Deliverable D1.3 and D4.1. To accomplish these tasks, Deliverable D4.2 highlighted the need of unifying multi formatted datasets. This document aims at defining and implementing a prototype for the storage and retrieval of such extracted and formatted metadata of datasets. Given the existence of extracted large amounts of metadata in the form of RDF triples, an appropriate data storage solution carries great importance at efficiently storing and retrieving RDF triples. Hence, efficient storage and retrieval of RDF data enables easy access to Open Data.

In this work, we devised a storage solution for the storage and retrieval of RDF data. To this end, we firstly stipulate desired properties of a storage solution in Section 2. Consequently, we investigated several technologies that might facilitate accomplishing the task. Section 3 introduces the preferred technology for the storage solution along with the justification of such decision. In Section 4, we evaluate the performance of our new prototype of RDF Triplestore. Section 5 concludes this document and elaborates on future work.

2 Requirements

In this section we define the properties of desired storage solution. It must satisfy the following requirements:

- Scale to store/index large RDF data
- Achieve fast retrieval of RDF data
- Facilitate complex queries

Such requirements are essential for any database systems. The nature of extract metadata of datasets from different catalogs puts constraints on the type of database system. Traditional SQL databases are sufficient for the storage and retrieval of structured data. However, RDF data fundamentally differs from the tabular data as it allows structured and semi-structured data to be mixed, exposed, and shared across different applications. Hence, SQL databases are not sufficient to store/retrieve data consisting of structured and semi-structured data. For instance, DCAT [1] is an RDF vocabulary designed to facilitate interoperability between metadata catalogs published on the Web. Figure 1 illustrates the underlying structure of data catalogs and the contained data.

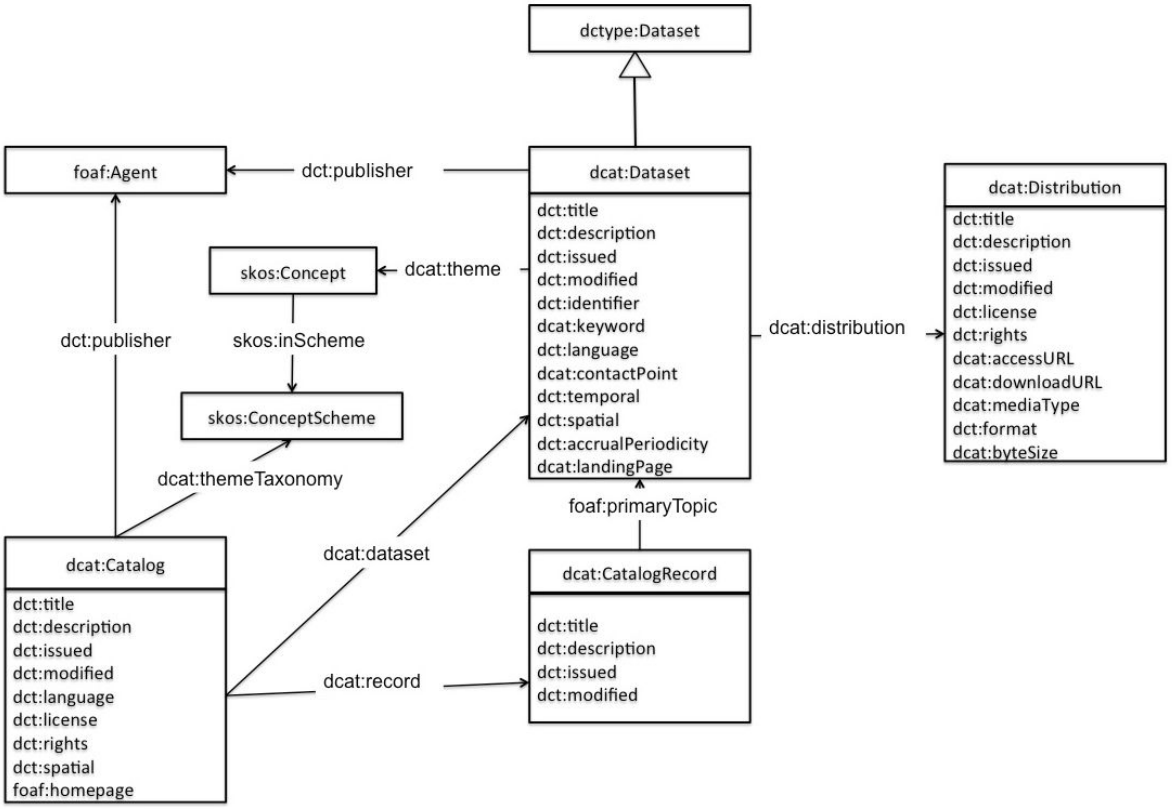

Figure 1: DCAT schema [1]

3 Elasticsearch as a storage solution

The nature of extracted metadata of datasets convinced us to use a technology/engine that facilitates full-text search and structured search while performing such operation in real-time. To this end, we employ Elasticsearch as the desired storage solution. In this section, we introduce Elasticsearch and its usage as the storage solution in OPAL.

Elasticsearch is an open-source, RESTful, distributed search and analytics engine built on Apache Lucene. Since 2010, Elasticsearch has quickly become the most popular search engine, and is commonly used for log analytics, full-text search, security intelligence, business analytics, and operational intelligence use cases [2]. Importantly, Elasticsearch offers various operators to be performed in near real-time. Such operations includes reading/writing data, full-text-search and so forth [3]. Figure 2 illustrates three main reasons of utilizing Elasticsearch as the storage solution in OPAL.

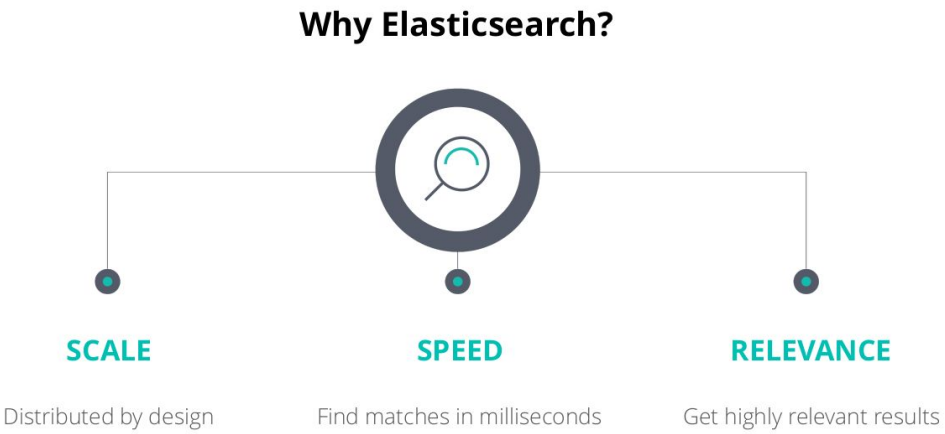

Figure 2: Elasticsearch characteristics [4]

Elasticsearch enables to store extracted metadata of datasets as documents/objects while the contents of each document being index. Indexing the contents of each document makes each content searchable. To elucidate such important feature please consider the following scenario: Assume we have a list of extracted metadata of datasets where each item corresponds to dcat:Dataset table in Figure 1. Each item in such list coincides with a document in Elasticsearch. Therefore, we store each document (the metadata) in Elasticsearch and define the contents of each document (title, description, issued etc). Storing hundreds of thousands of such documents along with the respective contents occurs in seconds. The act of storing data in Elasticsearch is called *indexing* [3].

4 Related works

4.1 Entity recognition

The recognition and integration of named entities has been integrated to OPAL and is described in Deliverable D3.3 *metadata extraction* [5].

4.2 Linguistic ambiguities

The generation of synonyms into Elasticsearch is described in OPAL Deliverable D7.1 *search component* [6]. It has been integrated using the Elasticsearch *synonym token filter* [7].

5 Customization

We designed three index structures for dcat:Dataset, dcat:Distribution and Measurements. The fields of documents correspond to the respective predicates/properties defined in DCAT. To store a document in an index structure, we use its URI and ID. As the index structures are created, we explicitly defined mappings and disabled dynamic indexing. Mapping is the process of defining how a document and its fields are stored. Mapping carries a great deal of importance at the retrieval of documents as mapping defines the searchable fields of documents. Dynamic mapping eases the process of mapping as it automatically detects the fields of document by indexing document. Therefore, dynamic mapping allows to add new fields (hence make them

searchable) at indexing time. However this convenience comes with the cost of mapping explosion [8]. Consequently, we created an explicit mappings for each index structure. Explicit mapping for the Measurements index contains only two fields as seen in Figure 3.

```
"mappings": {
    "dynamic":"false",
     "properties": {
        "http://www.w3.org/ns/dqv#isMeasurementOf": {"type": "keyword"},
        "http://www.w3.org/ns/dqv#value": {"type": "text"},
     }
}
```

Figure 3: The explicit indexing of Measurements

Figure 4 illustrates the explicit mapping of the mapping of dcat:Dataset.

```
"mappings": {
    "dynamic":"false",
     "properties": {
        "http://purl.org/dc/terms/title": {"type": "text"},
        "http://purl.org/dc/terms/description": {"type": "text"},
        "http://www.w3.org/ns/dcat#keyword": {"type": "text"},
        "http://purl.org/dc/terms/issued": {"type": "text"},
        "http://purl.org/dc/terms/modified": {"type": "text"},
        "http://purl.org/dc/terms/publisher":{"type":"keyword"},
        "http://www.w3.org/ns/dqv#hasQualityMeasurement":{"type":"keyword"},
        "http://www.w3.org/ns/dcat#distribution":{"type":"keyword"},
        "http://purl.org/dc/terms/accrualPeriodicity":{"type":"keyword"},
        "http://purl.org/dc/terms/spatial":{"type":"text"},
        "http://purl.org/dc/terms/identifier":{"type":"keyword"},
        "http://xmlns.com/foaf/0.1/isPrimaryTopicOf":{"type":"text"}
     }
}
```

Figure 5: The explicit indexing of dcat:Dataset

More details pertaining mappings of index structures and a running example of using Elasticsearch as a storage solution can be found in the GitHub repository of OPAL [9].

6 Evaluation

The goal of our evaluation was to measure the runtime performance of Elasticsearch in comparison to the runtime performance of TripleStore implementations like Virtuoso, which is one of the used OPAL TripleStores. To this end, we analyzed SPARQL queries, which are required to ensure a performant data flow related to the OPAL frontend. At the construction of appropriate SPARQL queries, we focused on text queries. A template for typical SPARQL queries is given in Figure 6. To obtain runtimes, we use the OPAL SPARQL sparql endpoint and an Elasticsearch instance on an Ubuntu server 18.04 with 126 GB RAM that contains 16 Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors. Each text query is performed 3 times. The respective queries as well as the related arithmetic means of runtimes are presented in Table 1.

```
PREFIX  dcat: <http://www.w3.org/ns/dcat#>
PREFIX dct: <http://purl.org/dc/terms/>

SELECT (COUNT(DISTINCT ?s) as ?num)
WHERE
 { GRAPH ?g
        {
        ?s a dcat:Dataset .
        --query below--
        }
 }
```

Figure 6: SPARQL query template

Query	Virtuoso runtime in seconds	Elasticsearch Runtime in seconds
?s dct:title ?x . FILTER CONTAINS (STR(?x), "Berlin") .	3.1	0.033
?s dct:title ?x . FILTER CONTAINS (STR(?x), "Paderborn") .	4.0	0.003
?s dcat:keyword ?x . FILTER CONTAINS (STR(?x), "Bahn") .	6.4	0.026
?s dct:description ?x . FILTER CONTAINS (STR(?x), "Strasse") .	5.6	0.014
?s dct:description ?x . FILTER CONTAINS (STR(?x), "Construction") .	5.1	0.032

Table 1: Performed queries

7 Conclusions and Future work

In this document, we provided the current architecture and configuration of a prototype using index structures for the storage of metadata of datasets in OPAL. Elasticsearch is employed as a storage solution that scale to large numbers of metadata of datasets (*documents* in the terminology of Elasticsearch) while allowing flexible queries at retrieval of documents.

In our future work, we will focus on integrating Elasticsearch into the OPAL RabbitMQ pipeline and aim to improve retrieval quality of documents.

8 References

[1] Fadi Maali, John Erickson: Data Catalog Vocabulary (DCAT). W3C Recommendation 16 January 2014. https://www.w3.org/TR/vocab-dcat/
[2] What is Elasticsearch? – Amazon Web Services. https://aws.amazon.com/elasticsearch-service/what-is-elasticsearch/
[3] Gormley, Clinton and Tong, Zachary. 2015. Elasticsearch: The Definitive Guide.
[4] Elasticsearch. https://www.elastic.co/
[5] Adrian Wilke, Michael Röder: OPAL D3.3 Erste Metadatenextraktionskomponente.
[6] Adrian Wilke, Caglar Demir: OPAL D7.1 Suchkomponente.
[7] Elasticsearch Reference 7.2: Synonym Token Filter. https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-synonym-tokenfilter.html
[8] Elasticsearch Reference 7.2: Mapping. https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html
[9] OPAL GitHub repository ElasticRDF. https://github.com/projekt-opal/ElasticRDF