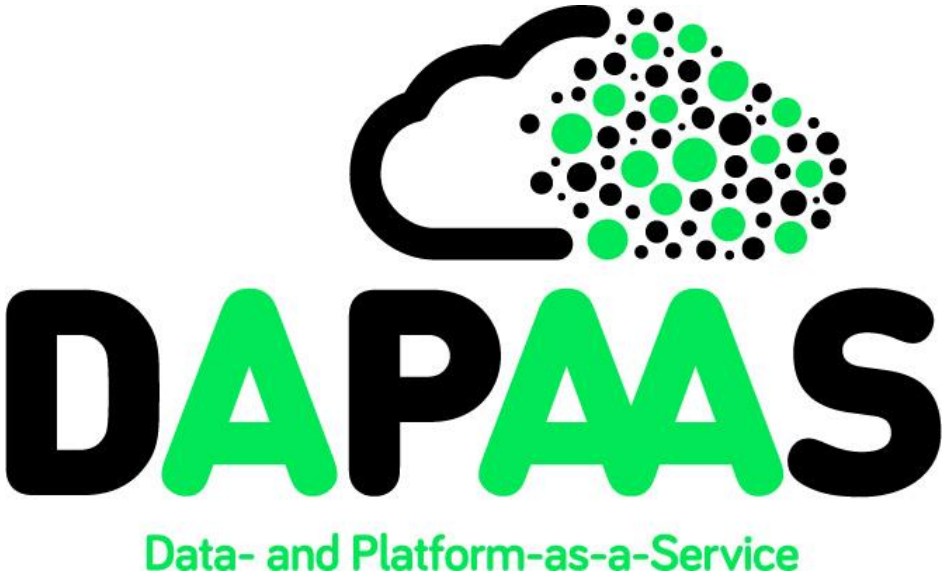
Small or medium-scale focused research project (STREP)

ICT SME-DCA Call 2013
FP7-ICT-2013-SME-DCA

Data Publishing through the Cloud: A Data- and Platform-as-a-Service Approach to Efficient Open Data Publication and Consumption

DaPaaS



Deliverable 1.3

Open DaaS prototype, v.2

Date:	31.07.2015
Author(s):	Marin Dimitrov (Ontotext), Alex Simov (Ontotext), Nikolay Nikolov (SINTEF), Dumitru Roman (SINTEF)
Dissemination level:	PU
WP:	WP1
Version:	1.0

Document metadata

Quality assurors and contributors

Quality assuror(s)	Rick Moynihan (Swirrl), Amanda Smith (ODI), Tom Heath (ODI)
Contributor(s)	DaPaaS Consortium

Version history

Version	Date	Description
0.1	24.07.2015	Initial outline and Table of Contents (TOC)
0.2	27.07.2015	Version for internal review
0.3	28.07.2015	Review comments provided
0.4	29.07.2015	Review comments implemented
1.0	30.07.2015	Final version

Executive Summary

This document describes the final version of the Data-as-a-Service (DaaS) prototype developed by M21 of the project. The deliverable builds upon the previous version D1.2[1], where the main architecture design and implementation decisions are described. Here we focus on the updates since M12, delivering a complete, ready for public use Data-as-a-Service layer as part of the DaPaaS platform.

This deliverable should be read alongside D2.3 which details the platform-as-a-service (PaaS) layer of the platform.

There are three major DaaS Layer improvements reported in this deliverable:

- A Cloud-enabled RDF Database-as-a-Service
- Extended support for non-RDF data
- Updated catalogue services

The structure of the report is as follows:

- Revised general architecture description
- Implementation overview – security, multi-tenancy, statistics and monitoring
- RDF database as a service (DBaaS)
- APIs updates description

[1] Available via: http://project.dapaas.eu/dapaas-reports

Table of Contents

List of Acronyms

API	Application Programming Interface
CSV	Comma Separated Values (format)
DaaS	Data-as-a-Service
DCAT	Data Catalog Vocabulary
JSON	JavaScript Object Notation (format)
PaaS	Platform-as-a-Service
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
DBaaS	Database as a service

List of Figures

1 Introduction

This report describes the second version of the Data-as-a-Service layer, supporting the various data management functionalities of the DaPaaS platform, more recently known as DataGraft and deployed at DataGraft.net. The descriptions in this deliverable rely on familiarity with previous version *D1.2 Open DaaS prototype v1*[2], where the main architecture design and implementation decisions are presented. The content of this document focuses on the individual sub-components of the integrated DaaS platform layer, trying to explain their functionality and to give some insights of the implementation aspects.

The platform implementation is designed for Cloud deployment and usage which is the reason it integrates and utilizes many services directly from the Cloud service provider. The platform is currently designed to work with Amazon Web Services (AWS) as a leading Cloud services provider, but it can be adapted to other providers supporting similar services.

In the following sections we briefly describe the updated architecture of the data layer and then go into more detailed presentations of the new or updated components since the first platform version (M12).

We have implemented a completely new metadata store based on the Cloud technologies and referred as RDF Database-as-a-Service (DBaaS). The core of the database is still the GraphDB[3] triple store engine but now supporting flexible extensibility based on current demand and offering as-a-service.

Another component which was not matured enough in the first platform version is the content store for non-RDF resources, primarily tabular data files. The component support now the complete set of data management operations: create, read, update, and delete (CRUD).

The integration component of the data layer, the Catalogue service has also been updated to work with the new components and to deliver the new functionalities to external components, both for DataGraft components as well as 3rd party applications.

[2] Available via http://project.dapaas.eu/dapaas-reports
[3] http://graphdb.ontotext.com

2 DaaS Architecture Overview

Figure 1 depicts the latest version of the Data-as-a-Service platform layer. The APIs for accessing the data layer remained almost unchanged - with no change to the functionalities. The significant difference with the previous version is moving the data import adapters to the platform layer (WP2) utilizing the matured data cleaning and transformation capabilities of Grafter[4] (WP4).

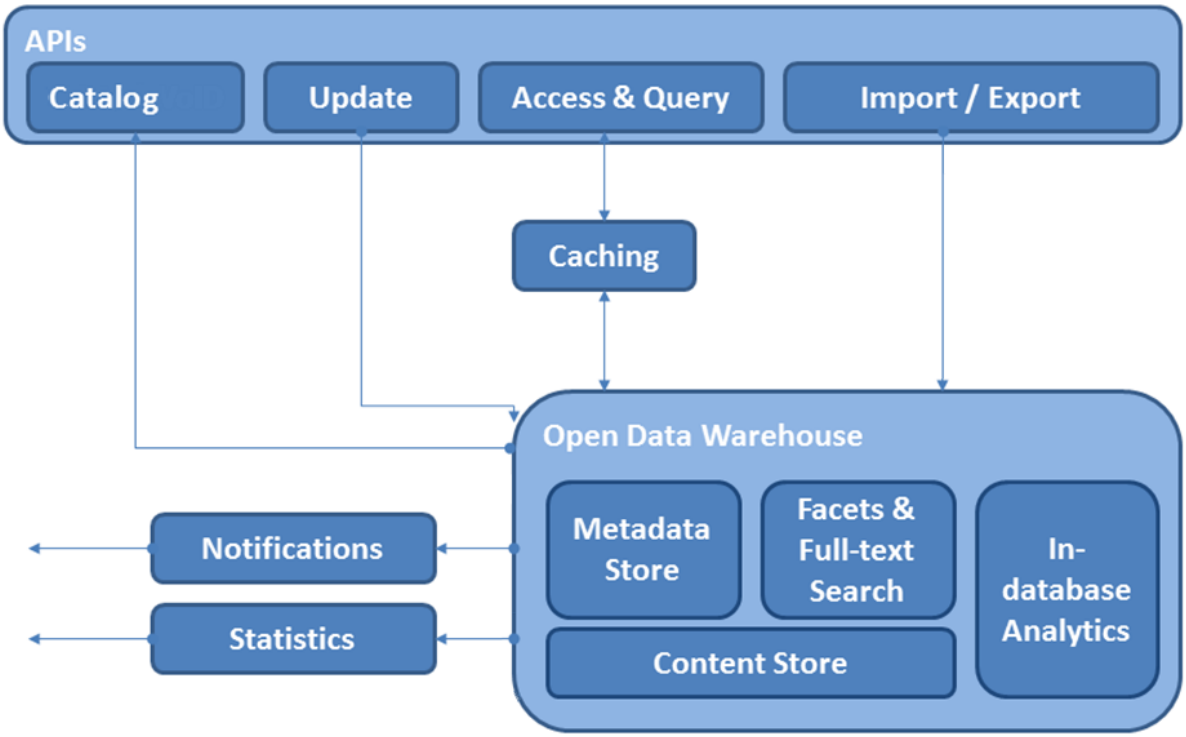


Figure 1. Data Layer Architecture

3 Implementation updates

Significant improvements have been committed to the Open Data Warehouse component, addressing scalability, extensibility and resources provisioning aspects. The metadata store (a.k.a. the RDF database) has been turned into a completely Cloud oriented database as a service (DBaaS). The content store heavily utilizes the Amazon S3[5] storage service to provide reliable and highly scalable storage capabilities for tabular data. The catalogue services have been extended to support the new features provided by the two types of data stores.

3.1 RDF Database as a Service

The fully managed version of GraphDB[6] in the Cloud provides an enterprise-grade RDF database as-a-service (DBaaS). The users do not need to deal with typical administrative tasks such as installation and upgrades, provisioning and deployment, backups and restores. DBaaS also ensures database service availability. The resources consumption is determined by the utilization of the system itself and it is capable to expand and shrink to match the current demand.

[4] http://grafter.org/
[5] http://aws.amazon.com/s3/
[6] http://graphdb.ontotext.com

3.1.1 Architecture and Implementation

From users perspective the RDF DBaaS (implementation of the *Metadata Store* in Figure 1. Data Layer Architecture) supports an API for linked data:

- access
- querying
- management.

The internal architecture of this component is much more complex to meet the requirements for scalability, extensibility and availability on large scale. A detailed and complete architecture is provided on Figure 2 to demonstrate the complexity of the component.

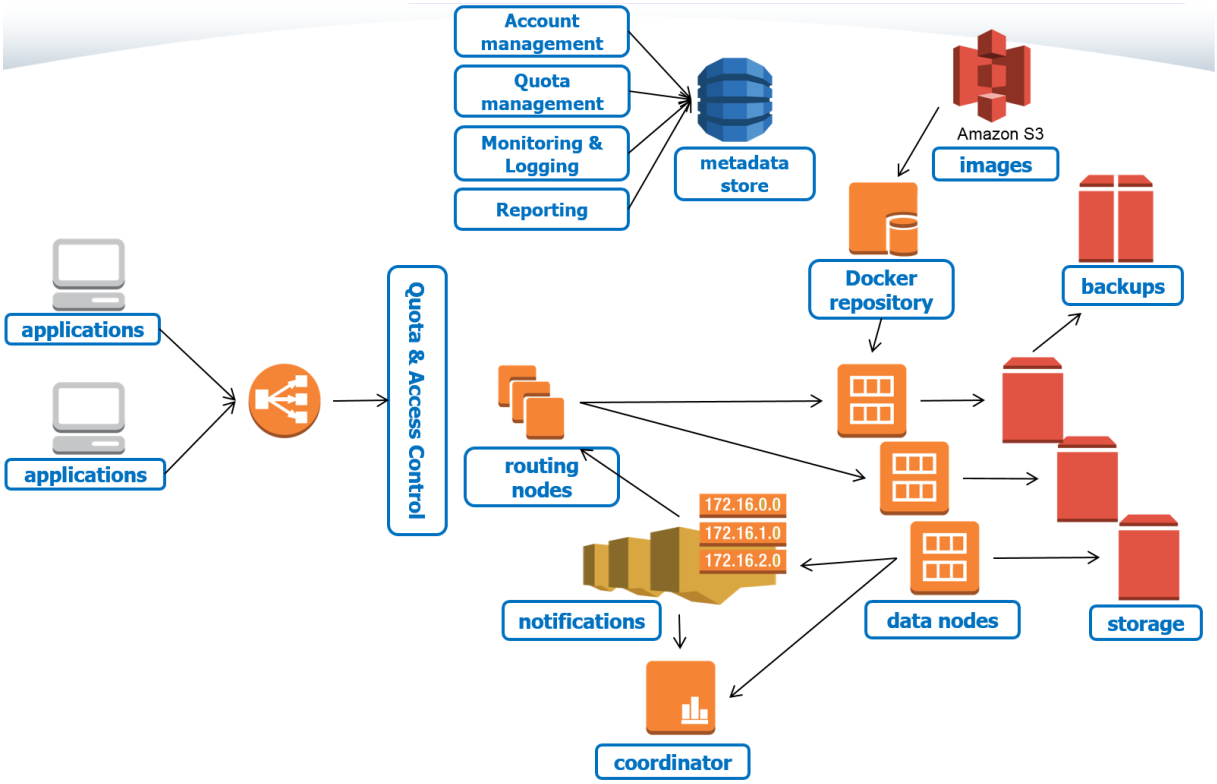


Figure 2. RDF DBaaS Architecture

The RDF DBaaS follows the principles of micro-service architectures and it is comprised of the following main components and layers:

- **Load balancer**[7] – the entry point to the RDF database services is the load balancer of the AWS platform, which will route incoming requests to one of the available frontend nodes. The load balancer can distribute requests even between instances in different datacenters.
- **Frontend routing nodes** – the frontend nodes host various micro-services such as: user authentication, access control, usage metering and quota enforcement for the RDF database-as-as-service layer. All instances host the same set of stateless front-end services and the frontend layer is automatically scaled up or down (new instances added or removed) based on the current system load.
- **Database nodes** – this layer contain nodes running multiple instances of the GraphDB database (packaged as Docker[8] containers). Each user has its own database instance

[7] http://aws.amazon.com/elasticloadbalancing/

(container) and it cannot interfere with the database instance or with the data of the other users of the platform. The data is hosted on Network-attached storage volumes (EBS[9]) and each user/database has its own private EBS volume. Additional OS level security ensures the proper data isolation and access control. Unlike the other layers of the system, each virtual machine in this layer hosts only a subset of all the database containers, e.g. database containers are not replicated across backend servers. Future versions of the system will introduce container replication as well for the purpose of improved throughput, so that read-only queries can be distributed among multiple servers hosting same database replica.

- **Integration services** – a distributed queue and a distributed push messaging service are used for loose coupling between the various frontend and database nodes on the platform. All components use publish-subscribe communication model to be aware of the current state of system. This way, the frontend and the backend layers are not aware of their size and topology and they can be scaled up or down independently.
- **Distributed storage** – all user data is stored on the Network-attached Storage (EBS), whereas static backups and exports are persisted on Amazon S3. Logging data, user data as well as various configuration metadata is stored in a distributed NoSQL database (AWS DynamoDB).
- **Monitoring services** – the AWS cloud provides various metrics for monitoring the service performance. The RDF DBaaS utilises these metrics in order to provide optimal performance and scalability of the platform. The different layers of the platform can be automatically scaled up (to increase system performance) or down (to decrease operational costs) in response to the current system load and utilisation.

3.2 Content store

The content store component is responsible for persistence and management of various user data files in tabular format. This serves two purposes:

1. Publishing of raw or processed data files;
2. Availability of data files as inputs for further processing by Grafter[10], data cleaning and linked data generation tool by WP4.

The latter is important for data reuse in different transformation scenarios. The following Figure 3 is a simplified overview of the content store architecture.

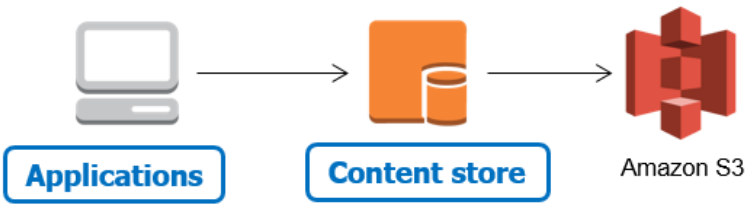


Figure 3. Content store

The data representation comprises of two parts: metadata and the actual data. The metadata describes the actual content as file formats, names, ownership, licensing, publication/modification dates.

The content store support all substantial management operations: create, read, update and delete (CRUD). This functionality is accessible as part of the *Catalogue APIs*, described in the following sections.

[8] https://www.docker.com/
[9] http://aws.amazon.com/ebs/
[10] http://grafter.org/

The actual implementation uses the Amazon S3 service for data storage and dedicated part of the RDF DBaaS service for metadata persistence. This component is primarily a file storage service and it does not do any additional processing on the data.

3.3 Statistics and monitoring

We collect various usage information from all over the system to improve the performance and the quality of service. On the frontend routing nodes (Figure 2) each user request is logged into a database serving two purposes:

1. The reporting service generates aggregates usage reports;
2. The usage quota management service receives its input.

The usage metrics collected from the AWS provide us with infrastructure utilization information which help us to improve the resources consumption management and to identify potential bottlenecks in the system. This ensures the platform and its associated services are scalable and accessible at all times.

Detailed usage statics (CPU/IO/RAM) collected on the level of user databases (Docker containers) allows us to better distribute containers by collocating heavy and low utilized databases on the same hosts.

3.4 Notifications

The notifications component uses a set of predefined metrics and rules to send push notifications when certain events occur. Such events are new user registrations, new datasets and transformations availability, as well as events related to infrastructure utilization – components overloading, system scaling activities, failed or missing components.

As notifications delivery channel the implementation utilizes the Amazon Simple Notification Service[11] which is a fast, flexible, fully managed push notification service. The messages of the platform are delivered mainly as e-mail notifications to the system operators, but the service also supports notifications as SMS messages to mobile devices.

Notifications senders are monitoring components of the infrastructure (AWS CloudWatch[12]) as well as custom components of the DaaS and PaaS layers.

3.5 Security & access control

To ensure adequate protection of users' data we combine different security techniques on the different components of the platform. The user access to the platform is done via secured HTTPS/TLS channels which prevents from sniffing and man-in-the-middle attacks.

The inter-component communication within the platform also deploys HTTPS where it is applicable but mainly it relies on the Amazon Security Groups and IAM services which strictly defines which components can communicate with which services.

As an extra measure for data protection we enable encryption on the storage volumes (EBS) and the S3 service.

The external access to the platform is controlled by API keys which provide access to protected resources. Anonymous read-only access is also supported for resource where the owner explicitly allowed this.

[11] http://aws.amazon.com/sns/
[12] http://aws.amazon.com/cloudwatch/

4 Data Catalogue services

The data catalogue services were enriched with support for the new database-as-a-service implementation as well as the new functionality for CRUD operations over tabular data files. The APIs are still compliant with the DCAT model (Figure 4) and vocabulary.

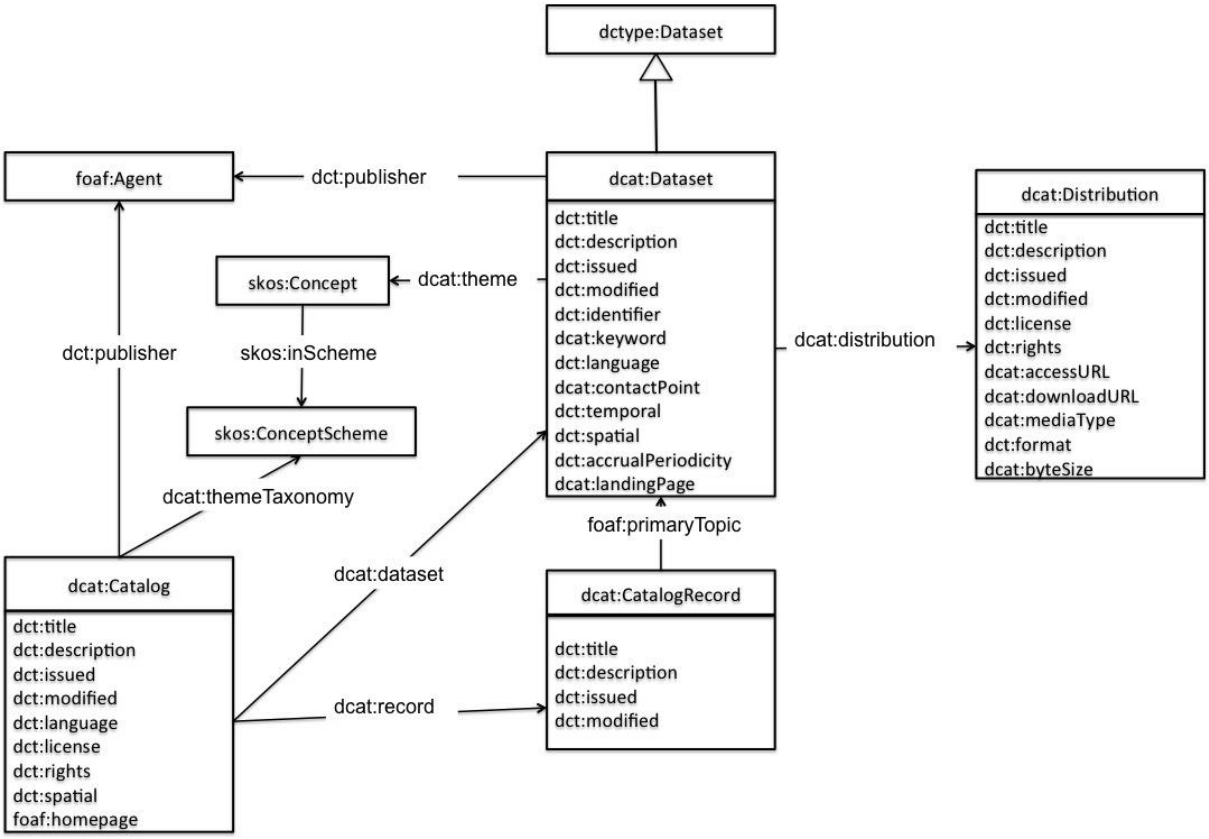


Figure 4: DCAT model

The first version of the data catalogue services used the DCAT model to maintain only RDF repositories mapped to *dcat:Distribution*. An RDF repository is an autonomous container for RDF data within the DBaaS. The extended version of the data catalogue services added support for working with raw tabular data files as well.

RDF repositories, part of DBaaS exploit the *dcat:accessURL* to provide access to its SPARQL endpoint. Data files rely on properties like: *dcat:downloadURL, dcat:mediaType, dcat:byteSize* to represent their specific aspects. All of them share the common metadata properties like names, descriptions, dates, etc.

4.1 Catalogue APIs update

The Catalogue APIs appearance have been refactored completely since the previous version (M12), however the major functionalities are preserved. Therefore the complete *Data Catalogue API* description is included in the *Appendix A: API Documentation* in this document. Here we focus only on the essential updates.

4.1.1 On-demand RDF database provisioning

Although the catalogue API hides the complexity of RDF database provisioning, there needs to be a mechanism to control the lifecycle of a database in terms of resources allocation and release depending on the current needs.

Two new API HTTP methods are introduced on the level of *distributions* which prepare the proper requests to the DBaaS service and update the catalogue metadata accordingly.

URL	METHOD	INPUT	DESCRIPTION
/distributions/repository	PUT	Header param: **distrib-id** - uri of the distribution **repository-id** - identifier for the new repository	Provisions RDF repository for distribution
/distributions/repository	DELETE	Header param: **distrib-id** - uri of the distribution **repository-id** - identifier of the repository to be deleted	Releases the RDF repository allocated for the distribution

4.1.2 Working with raw data files

Each data file containing tabular data (csv, tsv, xls) is represented by a *dcat:Distribution* object carrying its metadata properties (size, format, name, …) plus the actual file. Thus the management of raw data files is brought to operation over DCAT distributions. This reflects on the API which expects data and metadata on distribution creation or update. For accessing the stored objects the API offers independent access to the data and metadata.

Below is a summary of the new API methods:

URL	METHOD	INPUT	DESCRIPTION
/distributions	POST	**dataset-id** - uri of the dataset containing the distribution **Content-Type** - "multipart/mixed" Attachments: • **meta** – distribution's metadata (as RDF or JSON-LD) • **file** - the raw file content The 'file' attachment is optional (not applicable for RDF distributions). The metadata should contain the file metadata (content type, size, ...)	Creates a new distribution object belonging to a dataset
/distributions	PUT	(same as above)	Updates an existing distribution allowing metadata update or data file replacement
/distributions	GET	**distrib-id** - uri of the distribution	Retrieves the metadata for the distribution
/distributions/file	GET	**distrib-id** - uri of the distribution	Retrieves the actual data file
/distributions	DELETE	**distrib-id** - uri of the distribution to be deleted	Removes the data and the metadata of the distribution

5 Deployment Details

5.1 Hardware, OS

The DaaS platform layer is a completely Cloud-based system deployed on Amazon Web Services infrastructure. It is designed to flexibly expand or shrink depending on the current resources demand. The amount of resources required for the data layer depends mainly on the data storage capacities – amount of data which is hosted.

The following table provides summary the resources (computing nodes) consumed by the platform with a relative price categories.

Component	CPU requirements	Memory requirements	Storage	Price
Catalogue & Access control	*	*	*	$
DBaaS frontend	**	*	-	$$
DBaaS data node	***	***	***	$$$
DBaaS coordinator	*	*	-	$
DBaaS Docker registry	*	*	*	$

Note: the data nodes are the main items of expenditure but on the other hand they are capable of hosting (collocate) resources of different users on single machines leading to better resources utilization.

5.2 Software (3rd party)

The following third party components are supporting the platform:

- Docker platform (containerization)
- Apache Tomcat 7.0 (applications server)
- OpenRDF Sesame framework (RDF management middleware)
- AWS SDK for Java (java libraries for working with the Amazon services)
- Apache CXF framework (web services development framework)

5.3 DataGraft (by DaPaaS) components

The data layer of the platform is organized into several components, most of them packaged as web applications and exposed as RESTful web services:

- Data access services
- Data import services
- Catalogue services
- User management and access control

6 Final Remarks

This document describes the final prototype of the data-as-a-service layer, mature enough for public offering as a part of the DaPaaS's deployment on DataGraft.net. In the course of the project the priorities have been adapted to technological evolution which resulted in some changes of the direction of platform development. Some components were completely dropped, others turned to be key factors for the platform and developed as separate products.

The data import adapters were removed from the data layer because of the shift to tabular data processing and the advances of the Grafterizer/Grafter development. The automated interlinking functionality was dropped because of the immaturity of the tools initially planned for reuse.

Compliance with emerging standards for Linked Data oriented applications interoperability is still on focus and we will be working in this direction in the remaining time of the project. We are planning to put a layer on top of the RDF DBaaS services to expose content as linked data rather than RDF statements. Currently we are targeting implementations of LDP[13].

This report should be read alongside D2.3 which details the platform-as-a-service (PaaS) layer of the platform.

[13] http://www.w3.org/TR/ldp/

7 Appendix A: API Documentation

A user friendly version of the API documentation is publicly available at: http://dapaas.github.io/api/.

7.1.1 Datasets descriptions access and management

7.1.1.1 Get dataset description

URL	/datasets
HTTP Method	GET
Description	Get a dataset description (metadata) by id
Inputs	HTTP headers: **dataset-id** - URI of the dataset, taken from the catalogue **Accept** - result serialization format (*application/ld+json*, *application/rdf+xml*, … any standard RDF type)
Response	Complete dataset description using the DCAT vocabulary in RDF or JSON-LD

7.1.1.2 Create dataset description

URL	/datasets
HTTP Method	POST
Description	Create a new dataset description
Inputs	HTTP header: **Content-Type** - format of the metadata supplied Message body: dataset description as RDF or JSON-LD Note: if the description contains no dataset identifier, the system will generate one
Response	URI of the new dataset in the format: { "@id" : "http://dapaas.eu/dataset/4" }

7.1.1.3 Update dataset description

URL	/datasets
HTTP Method	PUT
Description	Update an existing dataset description
Inputs	HTTP header: **Content-Type** - format of the metadata supplied Message body: dataset description as RDF or JSON-LD
Response	Operation completion status (HTTP response code)

7.1.1.4 Delete dataset description

URL	/datasets
HTTP Method	DELETE

Description	Delete a dataset description with all of its distributions
Inputs	HTTP header: **dataset-id** - URI of the dataset to be removed
Response	Operation completion status (HTTP response code)

7.1.2 Distributions management

7.1.2.1 Get distribution description

URL	/distributions
HTTP Method	GET
Description	Get distribution description by id (contained in the corresponding dataset description)
Inputs	HTTP header: **distrib-id** - URI of the distribution, taken from the dataset description **Accept** - result serialization format (JSON-LD or RDF)
Response	Complete distribution description using the DCAT vocabulary in RDF or JSON-LD

7.1.2.2 Create distribution description

URL	/distributions
HTTP Method	POST
Description	Create a new distribution description. The request should contain the metadata for the distribution and optionally the data file of the distribution
Inputs	HTTP headers: **dataset-id** - URI of the dataset containing the distribution **Content-Type** – request message format: *multipart/mixed*, or *multipart/form-data* Request body attachments: **meta** - distribution metadata as RDF of JSON-LD. Use *Content-type* of the attachment to specify the metadata format **file** - the raw file content (CSV, TSV, XLS, …), This attachment is optional (not applicable for RDF distributions). The metadata should contain the file metadata (content type, size, ... see DCAT model for details) Note: if the meta description contains no distribution identifier, the system will generate one
Response	URI of the new distribution in the format: { "@id" : "http://dapaas.eu/distributions/abc" }

7.1.2.3 Update distribution description

URL	/distributions
HTTP Method	PUT
Description	Update an existing distribution description. This API method can be used either for metadata update or the actual data replacement or both.
Inputs	HTTP header:

	Content-Type – request message format: *multipart/mixed*, or *multipart/form-data* Request body attachments: **meta** - distribution metadata as RDF of JSON-LD. Use *Content-type* of the attachment to specify the metadata format **file** - the raw file content (CSV, TSV, XLS, …), If provided, the metadata should contain the file metadata (content type, size, ... see DCAT model for details) Note: if provided any new file overwrites the previous one.
Response	Operation completion status (HTTP response code)

7.1.2.4 Delete distribution description

URL	/distributions
HTTP Method	DELETE
Description	Delete a distribution description
Inputs	HTTP header: **distrib-id** - URI of the distribution to be deleted
Response	Operation completion status (HTTP response code)

7.1.3 RDF repository management

URL	/distributions/repository
HTTP Method	PUT
Description	Provision RDF repository for a distribution
Inputs	HTTP headers: **distrib-id** - URI of the distribution to contain the new repository **repository-id** - identifier for the new repository
Response	URL to the newly allocated repository in the form of: *{"access-url":<repository url>}*

URL	/distributions/repository
HTTP Method	DELETE
Description	Release the RDF repository allocated for the distribution
Inputs	HTTP headers: **distrib-id** - URI of the distribution containing the repository **repository-id** - identifier for the repository
Response	Operation completion status (HTTP response code)

7.1.4 Distributions data files access

URL	/distributions/file
HTTP Method	GET
Description	Accessing the data files of distributions
Inputs	HTTP header:

	distrib-id - URI of the distribution containing the file
Response	The data file in its original format and encoding