DELIVERABLE

Project Acronym:	**LoCloud**
Grant Agreement number:	**325099**
Project Title:	**Local content in a Europeana cloud**

D2.1: Core infrastructure

Revision: final

Authors:

Costis Dallas (DCU)
Dimitris Gavrilis (DCU)
Stavros Angelis (DCU)
Dimitra Nefeli Makri (DCU)
Eleni Afiontzi (DCU)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	x
C	Confidential, only for members of the consortium and the Commission Services	

Revision History

Revision	Date	Author	Organisation	Description
1.0	31/01/2014	Dimitris Gavrilis	DCU	Initial draft
1.1	11/02/2014	Marcin Werla	PSNC	Corrections throughout the document
1.2	12/02/2014	Dimitris Gavrilis	DCU	More details on infrastructure specs
1.3	13/02/2014	Walter Koch	AIT	Added business process models

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Contents

Introduction

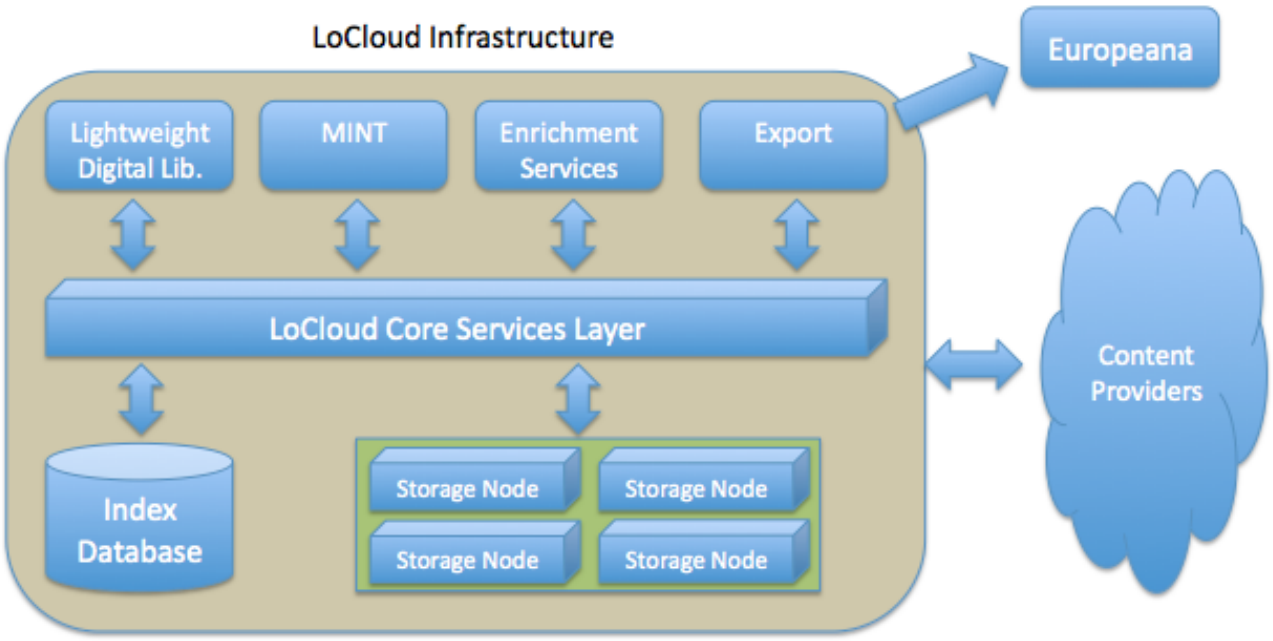
The present document introduces the core technical infrastructure specifications of the LoCloud project. The various technical aspects of the infrastructure are presented in detail, focusing on their advantages and limitations. The challenges of implementing this infrastructure are also described throughout this deliverable.

The primary mission of the LoCloud project is to aggregate, enrich and deliver from a large number of providers to Europeana. The main differences from other Europeana aggregation projects are:

- the number of providers could scale up
- multiple intermediate schemas are supported (instead of one)
- content is enriched through micro-services
- binary content could be ingested into the infrastructure through the Lightweight Digital Library (instead of just metadata)

The main problems that have to be tackled are:

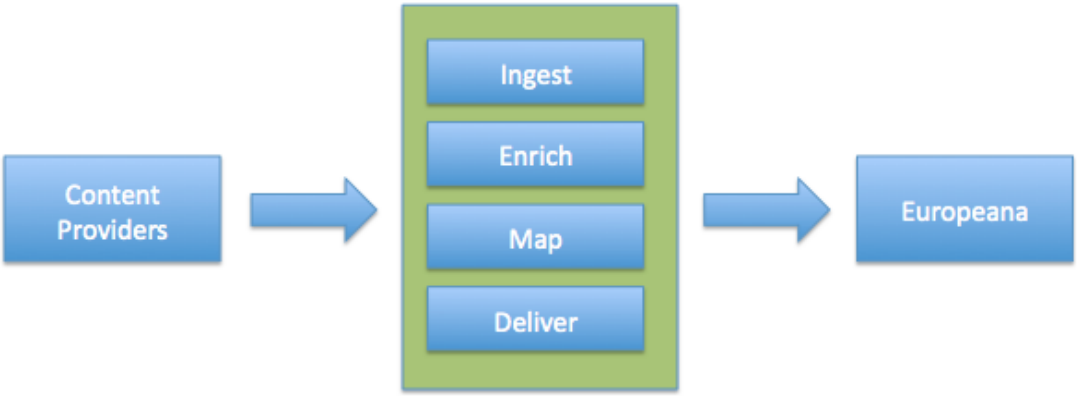
- handling of the increasing complexity
- keeping the cost at low levels



The main architectural diagram of the LoCloud infrastructure as it has been presented can be seen in the above picture. The main components of the system can be seen clearly:

- The lightweight digital library which provides the means for end users with no technical capabilities to start cataloguing and delivering content.
- The MINT tool which is responsible for mapping from native schemas to one of the intermediate schemas available.
- The enrichment micro services which are responsible for enriching metadata
- The export services which are responsible for exporting metadata to Europeana and possibly other providers (e.g. Wikis).

All the above services are orchestrated using a cloud core services layer which requires an index database to maintain important information about the operations described. Metadata are stored in a cluster of storage nodes.



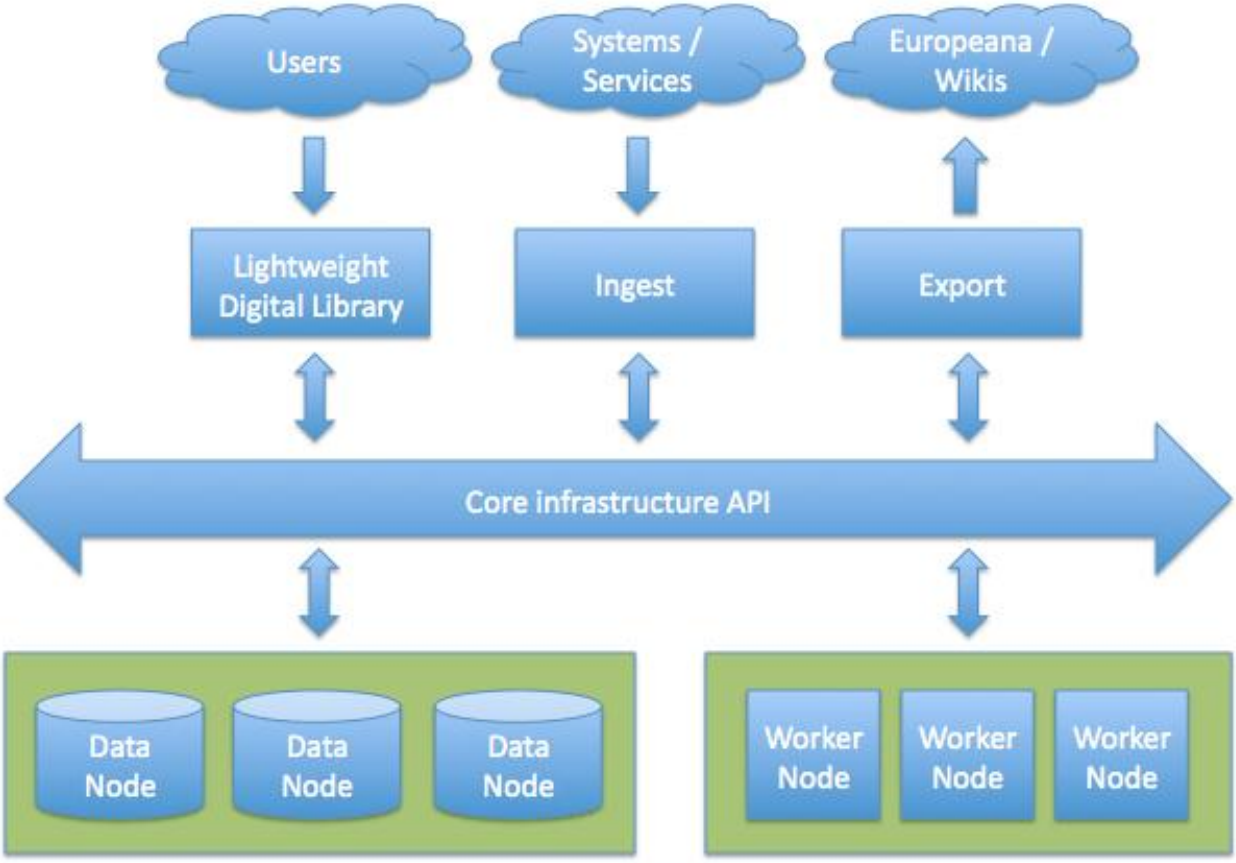
The overall streamline of the above tasks can be seen in the above picture and fall into four main steps: a) content is first ingested into the infrastructure, b) metadata records are enriched through one ore more of the enrichment micro-services, c) metadata are mapped from one of the intermediate schemas supported to the target schema (EDM) and finally d) metadata are exported to Europeana or other interested parties (e.g. Wikis owners who can get an enriched version of their content back).

1. Infrastructure overview

Viewing the core technical infrastructure at the top level, one could see three types of interactions:

- With individual users though the lightweight digital library (or similar out-of-the-box repositories). Users ingest content into the infrastructure.
- With other systems and services (e.g. content providers' systems). These systems ingest content into the infrastructure.
- With Europeana or Wikis that get content out of the infrastructure. Europeana because it wants to integrate new content into the European Digital Library, Wikis that want to get enriched content back from the infrastructure.

Three obvious cases that will be used throughout the project are presented by the above three main interactions. It is possible that through the services that will be provided other uses and interactions may surface.



The three interaction points interface with the infrastructure using an API. This core infrastructure API is responsible for maintaining data integrity (e.g. organizing data in packages, per provider/user, etc.), ensuring integrity among versions, maintaining information related with the tasks assigned to each object (e.g. enrichment, mapping, etc.).

In order for the infrastructure to become operational, a number of components are required. These components will have to handle: binary objects, metadata, indexes, state information, assets, services, etc. Among these components, there are two that are critical in terms of scaling up:

a) the binary object store and
b) the metadata object store

These two components have obviously different requirements but both require scaling, tackling issues such as backups, versioning, etc. Another major requirement is the scalability of worker nodes. Worker nodes are a generic term that refers to micro-services that operate on data or usually on metadata. Worker nodes typically perform tasks such as indexing, enriching, searching, relating, etc. The main characteristics of these worker nodes are:

- they operate on different types of metadata (intermediate schemas)
- they operate on either single objects or on collections of objects
- they usually operate asynchronously (because of the above)
- they have to scale (e.g. use multiple worker nodes to finish each task)

The above two main requirements (storage and worker nodes) need to have access to a certain set of information such as:

- an object registry
- a service registry
- a workflow registry
- a state registry
- a schemas registry

The object registry is required so that a service can have access to a specific object, a collection of objects (e.g. belonging to a provider, collection, etc.). The service registry needs to be able to distinguish among the different services available, which intermediate schemas each can operate on, where they reside on, etc. The workflow registry needs to have all possible and available paths that can be used on a digital object (e.g. first de-duplicate and then thematically enrich). The schemas registry needs to have access on the specifics of each intermediate schemas[1].

[1] D.1.2 Definition of metadata schemas, http://www.locloud.eu/Media/Files/Deliverables/D1.2-Definition-of-Metadata-Schemas

2. Technical specifications

2.1 Infrastructure services

The proposed technical infrastructure specifications comprise of services that are responsible for accomplishing various tasks required for achieving the goals of the project. These services are described in the table below.

Service	Description
Storage service	This service provides a mechanism for persistent storage of digital objects.
Ingest service	This service is responsible for getting metadata in the LoCloud infrastructure.
Validation service	The validation service checks all intermediate schemas for validation errors.
Retrieval service	The retrieval service is responsible for retrieving records or batches of records.
Indexing service	The indexing service is responsible for indexing parts of the ingested records (in any of the intermediate schemas) so that they can be accessed by the infrastructure.
Mapping service	The mapping service is responsible for mapping from the intermediate schemas to the output schemas (EDM).
Export service	The export service is responsible for exporting data sets through OAI-PMH.
Messaging service	The messaging service is responsible for managing inter-services message exchange.
Logging service	The logging service is responsible for keeping track of a log for all services.
Statistics service	The statistics service is responsible for keeping track of various statistics on both the contents of packages and their state information (e.g. how many are ingested/published, etc.).
Enrichment service	The enrichment service is responsible for handling enrichment. It communicates through the various enrichment micro-services.
Preservation service	The preservation service is responsible for maintaining versions, and a PREMIS log for all record actions.
Authentication/authorization service	The authentication service is responsible for authenticating end users / services and provides access to the rest of the services and the infrastructure.

2.2 Data specifications

The services should be implemented using a unified environment and exchange data with specific formats. These formats are described in a previous deliverable[2] and are consisted of the set of the Intermediate Schemas:

- CARARE
- EAD
- ESE / Dublin Core
- LIDO

Each service that needs to perform a task associated with the context of a record needs to be context aware. That means that it must be independent in terms of how to access/modify the content of each record. The infrastructure is and should be schema agnostic.

Each digital object should be available through a single URI.

2.3 Service execution

The core services layer should provide a specification for executing these services. This should facilitate the use of web services and linked data. The proposed approach should make use of REST services and formats such as: XML and JSON for exchanging information and performing execution.

The service execution environment should allow each service to reside on different physical machines / networks thus providing a fully distributed processing environment. The service execution environment,

2.4 Core Services description

The core services of the LoCloud platform include all the services that are tied into the core of the infrastructure and are required in order to complete the basic workflows which are: to ingest, transform and deliver content. These services include the authentication/authorization service, the logging and messaging services, the unique identifier services, the ingest, validation and preservation services and finally the ingest, mapping and export services.

2.4.1 Authentication/authorization

The authentication and authorization service has two main responsibilities:

- to authenticate end users (content providers) first against the core infrastructure and secondly to the individual services.
- to provide an authentication mechanism for all the web services that will comprise the infrastructure. This is necessary due to the distributed nature of these services.

The technologies that can be taken into account are:

[2] D1.2 Definition of metadata schema: http://www.locloud.eu/Media/Files/Deliverables/D1.2-Definition-of-Metadata-Schemas

- the OASIS extensible access control mark-up language (XACML) for defining a standard method for describing authentication and authorization requirements.
- the LDAP protocol for maintaining user account information through a standard that will allow for interoperability with heterogeneous technologies.
- the Spring security framework which provides a unified mechanism for authenticating using multiple technologies and protocols.

Some of the requirements of this service include the authentication against multiple sources, authentication of distributed services and the compilation and execution of multiple authentication mechanisms for each asset.

2.4.2 Storage

The storage service is responsible for providing a persistent and scalable storage infrastructure. The storage layer is content agnostic and should provide only CRUD operations. There are various technologies that could be examined and taken into account here, all described in Annex I. Some important aspects of the storage layer are its ability to scale and to handle a large number of concurrent threads (possibly with the introduction of multiple peer nodes that can be queried). The storage layer should also be able to have a simple structure. For example:

- It should be able to separate the digital object from its constituting parts
- It should be able to maintain some basic admin and tech metadata information (e.g. mimetype, size, version, created tstamp, owner information).

2.4.3 Ingest

The ingest service is responsible for getting data and metadata in the infrastructure. Before a digital record becomes part of the LoCloud infrastructure the following steps are required:

- Its provider and collection/package information must be identified
- Its individual parts must be identified
- All its consisting parts must be validated
- It must be assigned a unique identifier

The ingest service should be able to handle lots of requests (possibly concurrently).

During the ingest phase, in case where metadata are not in any of the intermediate schemas the MINT tool will be used to allow content providers to manually map their arbitrary schemas into one of the intermediate ones (see Annex II).

2.4.4 Mapping

The mapping service is responsible for metadata transformations. This service provides a basic yet important functionality as the content maintained by the infrastructure will be encoded in a variety of intermediate schemas (currently 4) and will be required to be exported to EDM. In past projects, the transformation to EDM isn't always trivial as special requirements may result in different transformations with some logic built into them.

2.4.5 Validation

The validation service will be responsible for validating content prior to ingestion. Validation includes a series of integrity checks and will ensure that all content received is well-formed in one of the intermediate schemas.

2.4.6 Retrieval

The retrieval service will be responsible for retrieving content from the LoCloud infrastructure. The retrieval service will perform queries on the content maintained by the infrastructure and will access it through the storage service. The retrieval services' requirements will include the hiding of the storage service details (e.g. all objects will only include their unique identifiers and last version) and to provide an abstraction layer for the distributed cloud storage. The retrieval service will provide metadata in any of the intermediate schemas. The main queries that the retrieval service will perform are:

- to retrieve a single object based on its identifier
- to retrieve the set of objects belonging to a provider
- to retrieve the set of objects belonging to a collection

More complex queries will employ the indexing service.

2.4.7 Export

The export service is responsible for exposing parts of the content of the infrastructure through a variety of ways. The primary method of export is though an OAI-PMH 2.0 provider. Other means of exporting include the export through XML file exports or through REST services (e.g. back to a Wikimedia installation). The main requirements of the export service are to be able to handle large amounts of content and to be able to expose content using a variable scope. The scope what will be used within LoCloud is: a) per provider, b) per collection.

2.4.8 Logging

The logging service will be responsible for maintaining a log of all users and services actions. The logging service will have to support the distributed nature of the infrastructure services and provide a centralized point that will maintain the logs. Regarding the structure requirements of the logging service, most logging frameworks that will be explored (e.g. Log4J) provide: timestamp, source information (service, IP), level and a description. There are also tools and services that allow humans to inspect these logs.

2.4.9 Messaging

The messaging service will be responsible for providing services with messaging functionalities. The messages will be used for inter-service communication (e.g. trigger the execution of a task or indicate the completion of a task). The messaging service basic requirements are its robustness, its capability to handle multiple queues and its ability to integrate easily with the services of the infrastructure. Technologies that will be taken into account are JMS and ActiveMQ.

2.4.10 Unique identifier

The unique identifier service is responsible for providing unique identifiers to the assets created within the cloud infrastructure. When a new digital object is created, a unique identifier is assigned which will be tied to it as long as it lives. This unique identifier will provide a single identity for this object regardless where it physically resides and regardless of its internal identifier assigned by the storage service.

3.4.11 Preservation

The preservation service is responsible for maintaining digital preservation related information for each digital object and datastream. An audit trail log is maintained at the digital object level describing the complete lifecycle of the digital object (and its components – datastreams). In the past, standards such as PREMIS have been used successfully to standardize the description of all write events (these are actions that modify the content of a digital object). The preservation service should also provide the audit log for each file.

2.4.12 Enrichment

The enrichment service will be responsible for providing the infrastructure for metadata enrichment. The enrichment service will invoke the various micro-services responsible for metadata enrichment. These services include:

- Historic place names
- Geo-related information
- Subjects

The enrichment service will be responsible for identifying the part of the metadata that could be enriched, transmitting this information to the respective service and create the enriched version based on the response of the respective service. The enrichment service should be able to "understand" parts of any of the intermediate schemas (e.g. how spatial or thematic information is encoded in each of the intermediate schemas).

3. Workflows

In order to better understand how the different components inter-operate with each other, a number of typical workflow executions are presented.

The main services that are involved are those of MORE plus MINT which can be viewed as a standalone service within the infrastructure.

In these tables shown in the following sections, some typical examples are shown:

3.1 Using a native repository

Workflow	Description
Ingest through existing native repository in intermediate schema.	1. Metadata are ingested using the ingest service through OAI-PMH or by XML file upload. 2. Metadata are validated using the validation service. 3. Metadata are transformed into EDM using the mapping service. 4. Metadata are enriched [optional] using the enrichment service. 5. Metadata are exported to Europeana using the export API.
Ingest through existing native repository in native schema.	1. Metadata are ingested using the ingest service through OAI-PMH or by XML file upload. 2. Metadata are mapped into one of the intermediate schemas using MINT. 3. Metadata are validated using the validation service. 4. Metadata are transformed into EDM using the mapping service. 5. Metadata are enriched [optional] using the enrichment service. 6. Metadata are exported to Europeana using the export API.

3.2 Using a Lightweight Digital Library

Workflow	Description
Ingest through LDL repository in intermediate schema.	6. Metadata are ingested using the ingest service through OAI-PMH or by XML file upload. 7. Metadata are validated using the validation service. 8. Metadata are transformed into EDM using the mapping service.

	9. Metadata are enriched [optional] using the enrichment service. 10. Metadata are exported to Europeana using the export API.

3.3 Through Europeana Local

Workflow	Description
Ingest through existing Europeana Local repository in intermediate schema.	1. Metadata are ingested using the ingest service through OAI-PMH. 2. Metadata are validated using the validation service. 3. Metadata are transformed into EDM using the mapping service. 4. Metadata are enriched [optional] using the enrichment service. 5. Metadata are exported to Europeana using the export API.
Ingest Europeana Local in native schema.	7. Metadata are ingested using the ingest service through OAI-PMH. 8. Metadata are mapped into one of the intermediate schemas using MINT. 9. Metadata are validated using the validation service. 10. Metadata are transformed into EDM using the mapping service. 11. Metadata are enriched [optional] using the enrichment service. 12. Metadata are exported to Europeana using the export API.

4. Business Process Models

4.1 Definition - Business Process Model (BPM)

"A sequential representation of all functions associated with a specific business activity.[3]"

In the LoCloud-Context this is a BPM-Diagram which depicts:

"*how Local Cultural Heritage metadata and content is handled by the 'LoCloud Environment'"*

will show data entry, data transformation, aggregation, and export to "Europeana data aggregation and storage services".

- The Business Process Model includes both IT processes ("non-human actors") and people processes ("human actors").[4]
- Business Process Modelling is cross functional, combining the work and documentation of more than one partner of the project.
- The LoCloud Business Process Modelling is using an Open Source version of the computerized tool "Signavio"[5], applying the BPMN (Business Processing Modelling and Notation) method developed by OMG[6]. A quick guide to BPMN is available at: http://www.bpmnquickguide.com/viewit.html .
- For the LoCloud project the developed model is accessible at: http://test119.ait.co.at:8081/locloud/p/explorer (Folder WP2).
- There is a CIDOC Working Group looking into the possibility of applying the BPMN-method to the SPECTRUM Procedures.[7]

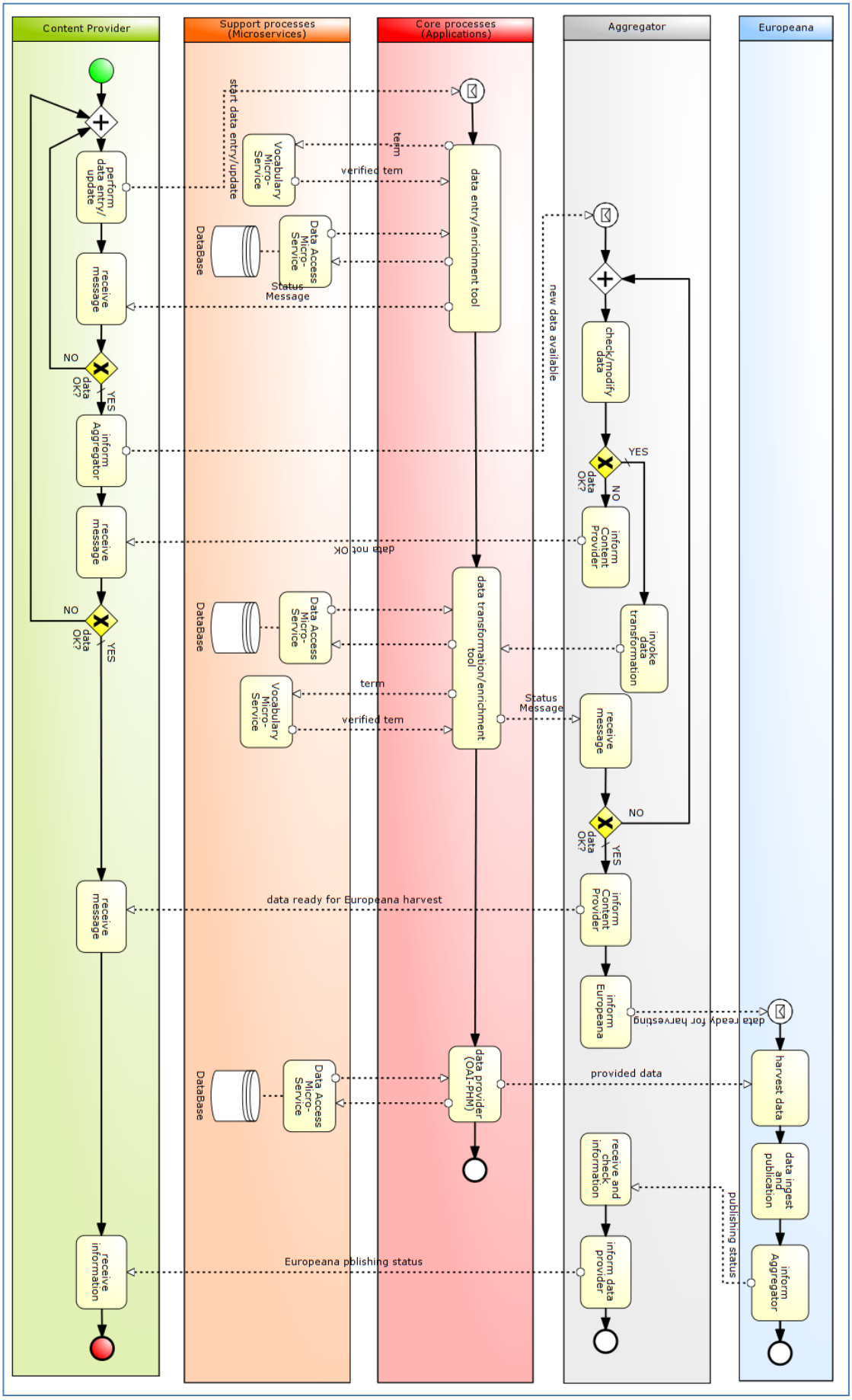
4.2 The LoCloud Business Process Model

The LoCloud Business Process Model which describes the "LoCloud Environment" is divided in 5 different pools containing the activities for:

- Content Provider (human actor),
- Aggregator (human actor),
- Europeana (human actor),
- Applications (Core Processes, non-human actor),
- Micro Services (Support Processes, non-human actor).

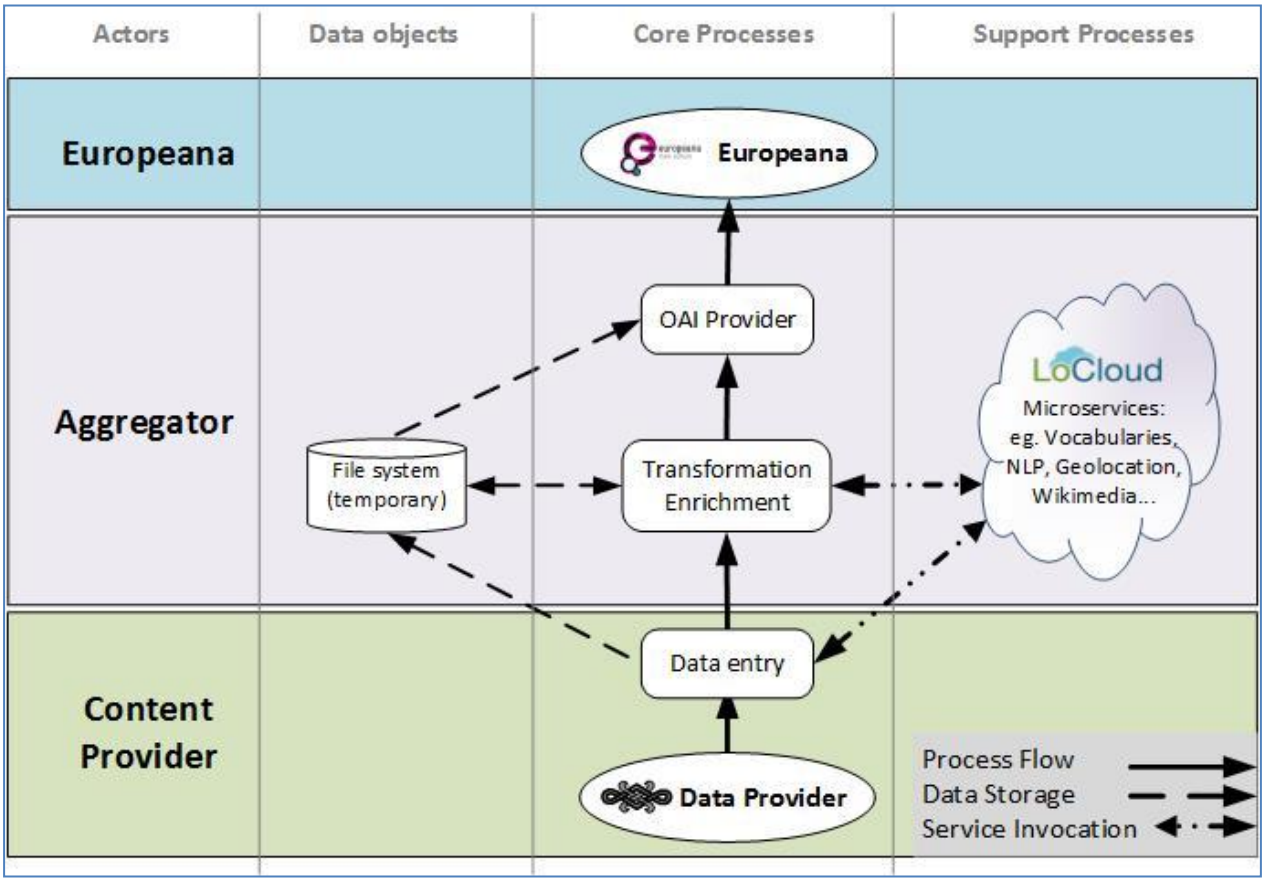
Please note that the diagram (next page) defines three generic applications (data entry/enrichment tool, data transformation/enrichment tool, and data provider). These applications relate to following LoCloud Components (Chapter 2): "Lightweight Digital Library", "Ingest", and "Export". This diagram is a "reference model" for the LoCloud System which is developed within a three year time span.

[3] http://www.businessdictionary.com/definition/Business-Process-Model.html 2014-02-13
[4] http://www.businessballs.com/business-process-modelling.htm 2014-02-13
[5] http://www.signavio.com/ 2014-02-13
[6] http://www.bpmn.org/ 2014-02-13
[7] http://network.icom.museum/cidoc/working-groups/mpi-museum-process-implementation/ 2014-02-13


Content Provider
Support processes (Microservices)
Core processes (Applications)
Aggregator
Europeana
perform data entry/ update
start data entry/update
Vocabulary Micro-Service
term
verified tem
data entry/enrichment tool
DataBase
Data Access Micro-Service
Status Message
receive message
NO
data OK?
YES
inform Aggregator
new data available
check/modify data
YES
data OK?
NO
inform Content Provider
data not OK
invoke data transformation
data transformation/enrichment tool
term
verified tem
Status Message
receive message
NO
data OK?
YES
inform Content Provider
data ready for Europeana harvest
inform Europeana
data ready for harvesting
harvest data
data ingest and publication
inform Aggregator
provided data
data provider (OAI-PHM)
receive and check information
publishing status
inform data provider
Europeana polishing status
receive information

This Model does not contain all details that will be found in the final system. E.g. it shows in the "Support Process Pool" only the activity related to the "Vocabulary Micro Service". This pool contains also a "Data Access Micro Service" which might be useful when putting local cultural heritage or sensitive data into a private cloud while all other processes are included in a public cloud. This feature might not be implemented in the LoCloud Project.

The figure below shows the situation on a higher abstraction level.



4.3 Further use of Business Process Modelling in LoCloud

The BPMN method used for describing the overall "LoCloud Environment" will also be applied to the Test-Environment set up for testing the "Vocabulary Micro Service" (WP3). The "Vocabulary Micro Service Test System" will be generated using the "Xataface GUI-Builder"[8], and connection to the Vocabulary Micro Service will be implemented by a "Widget" which also provides access to a collaborative platform in case communication to thesaurus experts is needed when adding a candidate term to a thesaurus. Such a process is more detailed described in the actual ISO Standard 25964-1, Chapter 13 "Managing thesaurus construction and maintenance"[9]. The implementation of this test system using a native "BPMN Execution Engine" as offered e.g. by Bonitasoft[10] is not foreseen but will be investigated.

[8] http://www.xataface.com/ 2014-02-13
[9] http://www.iso.org/iso/catalogue_detail.htm?csnumber=53657 2014-02-13
[10] http://www.bonitasoft.com/ 2014-02-13

References

Akka 2.0: http://doc.akka.io/docs/akka/2.1.1/Akka.pdf

Alexander Zahariev (2009), *Google App Engine*, Helsinki University of Technology

Amazon Web Services: http://aws.amazon.com/ (last access 31/01/2014)

Balla Wade Diack, Samba Ndiaye, Yahya Slimani, *CAP Theorem between Claims and Misunderstandings: What is to be Sacrificed?,* International Journal of Advanced Science and Technology, Vol. 56, 2013

Broberg J., Buyya, R., and Goscinski A., Cloud Computing: Principles and Paradigms, Wiley Press, USA, 2011

CAP Theorem: http://www.slideshare.net/larsga/nosql-databases-the-cap-theorem-and-the-theory-of-relativity (last access 31/01/2014)

Cassandra: http://cassandra.apache.org/

DuraCloud: http://www.duracloud.org/ (last access 31/01/2014)

Eeva Savolainen (2012), *Cloud Service model,* Univeristy of Helsinki, Department of Computer Science, 2012

ElasticSearch: http://www.elasticsearch.org/ (last access 31/01/2014)

Eucalyptus: http://www.eucalyptus.com (last access 31/01/2014)

GoGrid: http://www.gogrid.com/ (last access 31/01/2014)

Google App Engine: https://developers.google.com/appengine/?csw=1 (last access 31/01/2014)

HBase: http://hbase.apache.org/

Microsoft Azure Platform: http://www.windowsazure.com/en-us/ (last access 31/01/2014)

MongoDB: http://www.mongodb.org/

Open Nebula http://opennebula.org (last access 31/01/2014)

Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal (2008), *Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilitie,* High Performance Computing and Communications, HPCC '08, 10th IEEE International Conference

Redis: http://redis.io/

Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung (2003), *The Google File System*, SOSP'03, Bolton Landing, New York, USA.

Types of cloud computing : Private, public and Hybrid clouds : http://blog.appcore.com/blog/bid/167543/Types-of-Cloud-Computing-Private-Public-and-Hybrid-Clouds (last access 31/01/2014)

Yunhee Kang, Kyung-Woo Kang (2013), *An Empirical Study of Hadoop Application running on Private Cloud Environment,* Advanced Science and Technology Letters, Vol.35, pp.70-73

Qi Zhang, Lu Cheng, Raouf Boutaba (2010), *Cloud computing: state-of-the-art and research challenge,* Journal of Internet Services and Applications, Vol.1, pp.7-18

VMWare vSphere (vCloud): http://www.vmware.com/ (last access 31/01/2014)

Ganeti: https://www.usenix.org/legacy/event/lisa07/tech/trotter_talk.pdf

Lauce Albertson, *Comparing Open Source Private Cloud (IaaS) Platforms* (http://www.slideshare.net/OReillyOSCON/comparing-open-source-private-cloud-platforms)

5. Annex I

5.1 Cloud models and types

The rates of the development of processing and storage technologies in combination with the success of the Internet have led to more powerful, reliable and available computing resources. Cloud computing leverages virtualization technology to achieve the goal of providing computing resources as a utility. This section first displays the cloud service models, explaining the characteristics of its model ant the differences among these. The second section displaying the cloud types describes the possible ways in which LoCloud could be operated. It should be noted that the described models and types do not provide a complete list of all platforms and services; however it is an indicative list which should be considered and can be combined by LoCloud infrastructure.

5.1.1 Cloud service models

The architecture of a cloud computing can be divided into 4 layers: the hardware layer, the infrastructure layer, the platform layer and the application layer. Each layer of the architecture can be implemented as a service to the layer above. Cloud computing services are divided into four classes, according to the abstraction level of the capability provided and the service model of providers, namely: (1) Infrastructure as a Service, (2) Platform as a Service, (3) Software as a Service and (4) Community as a Service.

Figure 1 defines the layered structure of the cloud stack from physical infrastructure to applications. These service model levels can also be viewed as a layered architecture where services of a higher layer can be composed from services of the underlying layer.

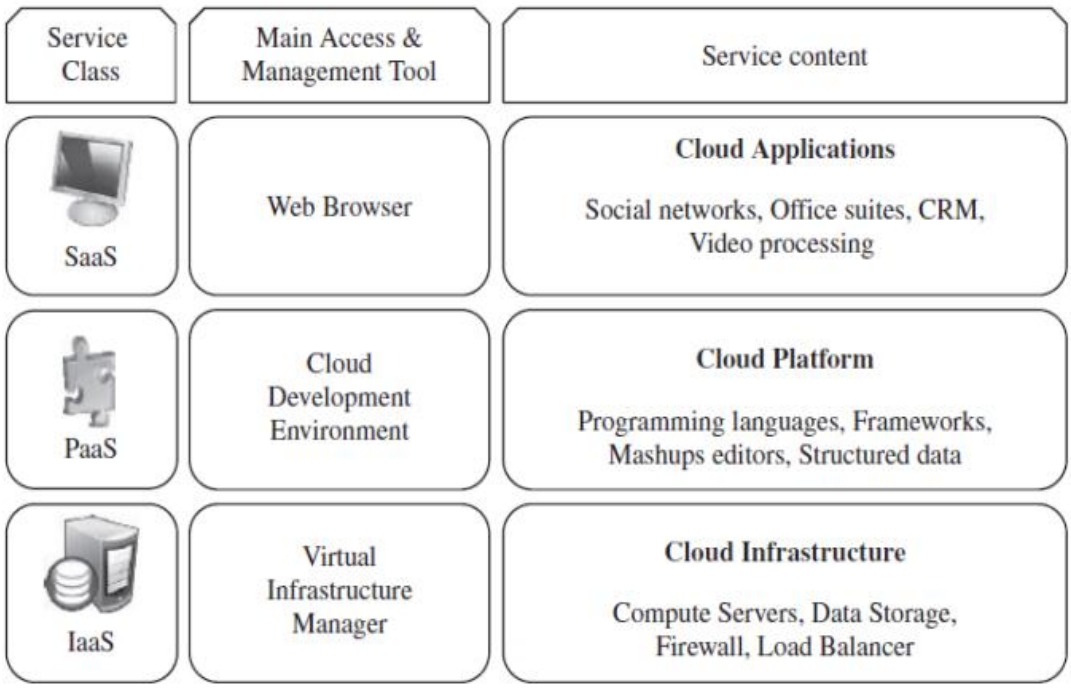


Figure 1: Cloud service model [Broberg et al. , 2011]

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) refers to on-demand provisioning of infrastructural resources as a shared service. IaaS is the hardware and software that powers the servers, storage, networks and operating systems.

In IaaS model cloud consumers directly use infrastructure components (storage, firewalls, networks, and other computing resources) provided by the cloud provider. Virtualization is widely used in order to provide physical resources in an ad-hoc manner to meet current resource demand of cloud consumers. Examples of IaaS providers include the Amazon EC2 service or the GoGrid infrastructure.

Platform as a Service (PaaS)

Platform as a Service (PaaS) refers to providing platform layer resources, including operating system support and software development services. PaaS offers an environment where developers can create and deploy applications, providing a service that can be used to a complete software development lifecycle management, from planning to design to building applications to deployment to testing to maintenance.

PaaS clouds provider higher-level abstractions for cloud applications, which simplifies the application development process and removes the need to manage the underlying software and hardware infrastructure. Examples of PaaS providers include the Google App Engine or the Microsoft Azure Platform.

Software as a Service (SaaS)

Software as a Service (SaaS) refers to providing on-demand applications over the internet. SaaS is a software delivery method that provides access to software and its functions remotely as a Web-based service.

In SaaS model a software provider licenses a software application to be used and purchased on demand. Applications can be accessed through networks from various clients (web browser, mobile phone, etc.) by application users. The most common pricing model is pay per use, which a customer pays a static price for units they use. Example of SaaS providers includes the Salesforce platform.

Community as a Service (CaaS)

Community as a Service (CaaS) is the next layer of cloud computing, economic development and exports and the path to a simpler society. From an information perspective, open source technologies are assembled to organise communities in a web based platform. From an economic development perspective, CaaS organises wisdom and effort and regional and global participation drives export revenue.

In CaaS model several organizations from a specific group share the infrastructure. The goal of a community cloud is to have participating organizations realize the benefits of a multi-tenancy and a pay-as –you-go billing structure with the added level of privacy, security and policy compliance.

5.1.2 Cloud Types

There are different types of clouds that you can subscribe to depending on user needs. The cloud types play a great role in the way that LoCloud infrastructure will be operationalized as the cloud type affects both the organization and the implementation of the cloud services. Figure 2 depicts the 4 cloud types.

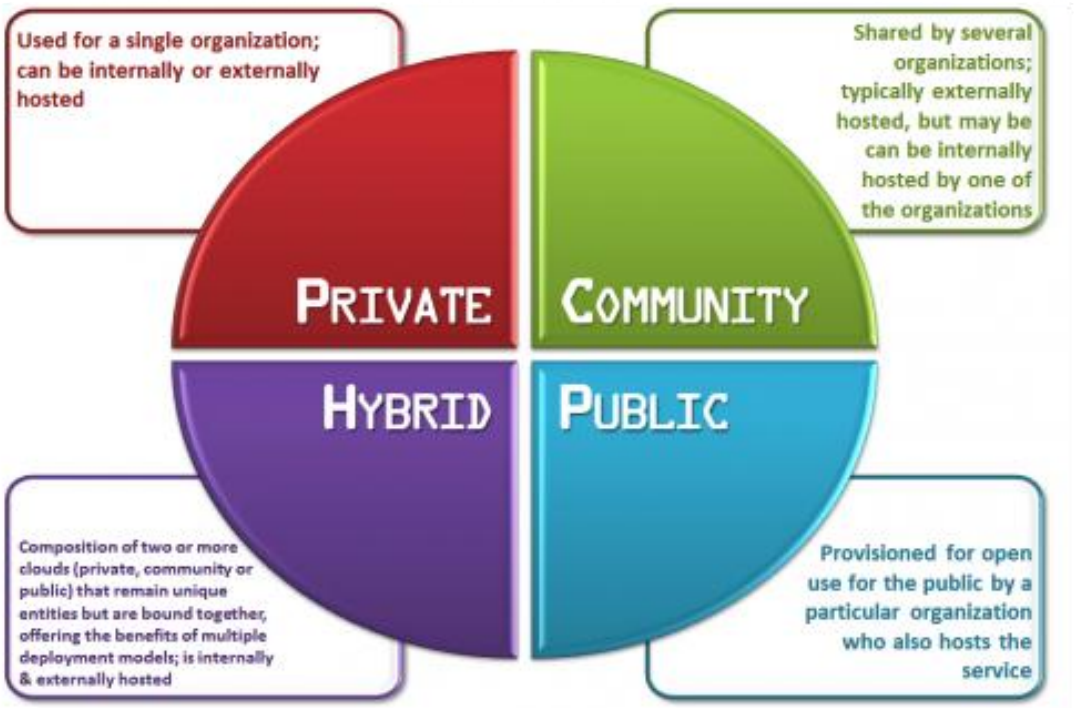


Figure 2: Cloud types [http://blog.appcore.com/blog/bid/167543/Types-of-Cloud-Computing-Private-Public-and-Hybrid-Clouds]

Private Cloud

Private cloud is established for a specific group or organization and limits access to just that group. The private cloud is usually a pool of resource inside a company; however it may be managed by either the company or a third party.

In case of Europeana network and under the LoCloud project, an organization/consortium could be formed that would adopt the principles of the private cloud infrastructure and it would own the hardware hosted in a data center.

Public Cloud

A public cloud can be accessed by any subscriber with an internet connection and access to the cloud space. The cloud infrastructure is made available to any organizations while both service providers and company are benefit from economies of scale. With public cloud services, users don't need to purchase hardware, software or supporting infrastructure, as it is owned and managed by providers.

In case of Europeana network and under the LoCloud project, the services would be developed and deployed in the public infrastructure of existing providers. Partners, in order to run the cloud

infrastructure, do not need any existing storage or computational resources as these are provided by the public cloud.

Hybrid Cloud

A hybrid cloud is essentially a combination of at least two clouds, where the clouds included are a mixture of public, private, or community. A hybrid cloud uses a private cloud foundation combined with the strategic use of public cloud services. Most companies with private clouds will evolve to manage workloads across data centers, private clouds and public clouds—thereby creating hybrid clouds.

In case of Europeana network and under the LoCloud project, the hybrid cloud would benefit the pros of both the public and the private public, making it clear which services and when run from public and private component.

Community Cloud

A community cloud is shared among two or more organizations that have similar cloud requirements. A community cloud is a is a multi-tenant cloud service model that is shared among several or organizations and that is governed, managed and secured commonly by all the participating organizations or a third party managed service provider.

In case of Europeana network and under the LoCloud project, the resources would be utilized by the number of partners participating to the project so as to build the shared cloud infrastructure. In this way, partners could benefit from both their services and resources and other partners' resources, some of which provide them with unique offerings.

5.2 Cloud technologies

In this section the state-of-the art implementations of cloud computing are presented. Firstly, a variety of services are described, which are distinguished into compute services, storage services and software and other services. Moreover, technologies and frameworks for building cloud environments are displayed.

5.2.1 Compute services

Amazon Elastic Compute Cloud (EC2) provides a virtual computing environment that enables a user to rent virtual servers and deploy on them their own services. The user can either create a new Amazon Machine Image (AMI) containing the applications, libraries, data and associated configuration settings, or select from a library of globally available AMIs. The user needs to upload the created or selected AMIs to Amazon Simple Storage Service (S3) and monitor instances of the uploaded AMIs. Amazon EC2 charges the user for the time when the instance is alive, while Amazon S3 charges for any data transfer (both upload and download). Amazon EC2 is one of the most considerable services in the IaaS field, while other important players in this field are mentioned, for example, in the following list: http://www.clouds360.com/iaas.php.

Google App Engine is a platform which allows users to run and host their web applications on Google's infrastructure, which are easy to build, to maintain and to scale whenever traffic and data storage needed. Google App Engine does not require any server to be maintained and no administrators needed. The basic idea is that the user just uploads his application and then its own customers are ready to be served. The user, by Google App Engine, has the option to limit the access of the application within the members of his organization or to share it with the rest of the world. The starting packet is free of charge and additional obligation. Implementation of Google App Engine applications is done under Python programming language. Google App Engine is able to distribute application's web requests across various servers, which allows starting and stopping the servers to meet traffic demand. However, applications uploaded to engine are not portable to other platforms. Other PaaS providers are listed in the following list: http://www.clouds360.com/paas.php.

Windows Azure is a cloud computing platform and infrastructure, created by Microsoft, for building, deploying and managing applications and services through a global network of Microsoft-managed data centres. It provides both PaaS and IaaS services and supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems. Windows Azure presents a .NET-based hosting platform that is integrated into a virtual machine abstraction; however, it achieves flexibility via the wide range of language supports other than .NET framework, such as Java and PHP.

5.2.2 Storage services and software

5.2.2.1 Storage services

. Amazon provides three remarkable public cloud storage services designed for different purpose: ESB, S3 and Glacier.

Amazon Elastic Block Service (EBS) provides persistent block storage for use with Amazon EC2 instances. Each Amazon EBS volume is automatically replicated to protect you from component failure, offering high availability and durability, while it offers the consistent and low-latency performance needed to run your workloads.
Amazon Simple Storage Service (S3) provides a simple web-service interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. The service aims to maximize benefits of scale and to pass it on to developers. It has a slightly higher latency than EBS and is intended for storing data of any size.
Amazon Glacier is an extremely low-cost storage service that provides secure and durable storage for data archiving and backup. In order to keep costs low, Amazon Glacier is optimized for data that is infrequently accessed and for which retrieval times of several hours are suitable. It is claimed to be up to 90% cheaper than S3.
DuraCloud is an interesting service in the domain of digital libraries designed for storage and preservation aiming to maximize availability and durability. This means that data are not only distributed into multiple geographical locations managed by one vendor but also to infrastructures of different vendors. DuraCloud uses Amazon S3, Amazon Glacier and Rackspace as their underlying storage services.
The list of other important cloud storage providers is given here: http://www.clouds360.com/storage.php.

5.2.2.2 Frameworks for distributed file systems

Apart from the storage services, there are interesting projects providing software for building distributed file systems on the market.

Google File System (GFS) – is a distributed file system that has been developed in order to meet the needs of Google. GFS shares the same goals with all the previous distributed file systems such as performance, scalability, reliability, availability. Google File System consists of a great number of storage machines, accessed by many client machines. The files are organized hierarchically in directories and identified by path names. A GFS cluster consists of a single master and multiple chunkservers. A single master process and maintains the metadata while a chunkserver stores the data in units, called chunks. The GFS provides fault tolerance as for fast recovery and chunk replication.

Hadoop Distributed File System (HDFS) – is part of the Apache Hadoop software framework that has been developed for large scale data analytics across clusters of computers. Hadoop Distributed File System stores large files across multiple machines. It achieves reliability by replicating the data across multiple servers. Similarly to Google File System, data is stored on multiple geo-diverse nodes. The file system is built from a cluster of data nodes, each of which serves blocks of data over the network using a block protocol specific to HDFS. Data is also provided over HTTP, allowing access to all content from a web browser or other types of clients. Data nodes can talk to each other to rebalance data distribution, to move copies around, and to keep the replication of data high. Hadoop provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce paradigm.

OpenStack (Swift/Cinder) – is a collection of open source components to deliver public and private clouds. OpenStack utilizes Python as a development language for open source project to build a private cloud computing environment. It is composed of such components as Nova, Swift, Cinder,

KeyStone, etc. For instance, Nova supports the virtualization of computing resources and manages the virtual machine instance. Swift project provides storage service like S3 service of Amazon as an object storage project of OpenStack. OpenStack provisions the computing resources dynamically as a tool of IaaS (Infrastructure as a Service). It is provided as a service that forms the sub-domain structure of computing resources of the virtualized computing, storage and network.

5.2.3 Other services

Apart from computing and storage services, there are many other cloud services including searching in the cloud and distributed databases that are useful and necessary in order to deploy a cloud infrastructure.

5.2.3.1 Searching in the cloud

In terms of search in the cloud, one interesting offering is the ElasticSearch hosting. ElasticSearch is a flexible and powerful open source, distributed, real-time search and analytics engine, giving you the ability to move easily beyond simple full-text search. ElasticSearch is implemented in:

6. Bonsai (https://addons.heroku.com/bonsai),
7. QBox (https://qbox.io/)
8. Found (http://www.found.no/)
9. Amazon CloudSearch (http://aws.amazon.com/cloudsearch/pricing/) service.

Considering the above options, it could be concluded that search as a service is enough expensive, despite the fact that it is often guaranteed by the providers that the indexes are stored in main memory (RAM), which in the case of search is the key for achieving optimum performance. However, the biggest issue of the current search as a service offering is that many of the available solutions do not provide satisfying index storage. Amazon CloudSearch seems to be a possibly cheaper option suitable also for large indexes. CloudSearch costs are calculated based on data out (retrieved items), data in are free. This makes CloudSearch potentially attractive for large indexes with relatively low amount of retrieved items, which is a description that can fit cultural heritage institutions.

5.2.3.2 Distributed Databases

One of the basic principles of distributed databases has been described by the CAP theorem. The CAP theorem (CAP standing for *Consistency, Availability* and *Partition Tolerance*) states that a distributed system is impossible to simultaneously provide all three of the following guarantees:

- *Consistency*, which means that all nodes give always the same answer
- *Availability*, which means that nodes always answer queries and accept updates
- *Partition tolerance,* which means that the system continues working even if one or more nodes go quiet.

There is a great variety of distributed databases, each one of which has different features. For example, some distributed databases, like key-value stores, offer the highest levels of scalability in terms of data size but are less suitable for expressing complex data while others, like graph databases, can express more complex data relationships, but are less scalable. [http://en.wikipedia.org/wiki/NoSQL#Classification_based_on_feature]. Some examples of distributed databases are the following:

Redis - is an open source, BSD licensed, advanced key-value store. The user can run atomic operations, like appending to a string; incrementing the value in a hash; pushing to a list; computing

set intersection, union and difference; or getting the member with highest ranking in a sorted set. Redis works with an in-memory dataset and it supports trivial-to-setup master-slave replication, with very fast non-blocking first synchronization, auto-reconnection on net split and so forth. Moreover, there is a useful implementation of Redis, Redis cloud, which is a fully-managed cloud service for hosting and running redis dataset in a highly-available and scalable manner, with predictable and stable top performance.

Cassandra - is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. One of the most significant features of Cassandra is the linear scalability and the proven fault-tolerance on commodity hardware. Cassandra also supports the replica across multiple data centres, providing lower latency for the users.

Apache HBase - is an open source, non-relational, distributed database system. It is the Hadoop database and it is used to provide real-time read and write access to the user's data. Some of the features of HBase are linear and modular scalability, consistency in reads and writes or support for exporting metrics via the Hadoop metrics subsystem.

MongoDB – is an open-source document database system. Classified as a NoSQL database, MongoDB is based on the relational database structure in favor of JSON-like documents with dynamic schemas, making the integration of data in certain types of applications easier and faster. Interesting features of MongoDB are the support of full indexing, the replication and high availability, the flexible aggregation and data processing through map/reduce

5.2.4 Frameworks and software for building cloud infrastructures

There is a plenitude of tools supporting the development and the deployment of cloud infrastructures. In the field of frameworks for building cloud environments, the following projects should certainly be noted:

OpenStack is designed to create freely available code, standards, and common ground for the benefit of both cloud providers and cloud customers. The goal of OpenStack is to allow organization to create and offer cloud computing capabilities using open source software running on standard hardware. The project boasts of compute, storage and image service component. OpenStack Compute is open source software designed to provision and manage large networks of virtual machines, creating a redundant and scalable cloud computing platform. OpenStack Storage is software for creating redundant, scalable object storage using clusters of commodity servers to store terabytes or even petabytes of data. All of the code for OpenStack is freely available under the Apache 2.0 license. OpenStack is aiming at Virtualization Portability where user will be able to move from virtualization technologies including those hosted in the cloud and will be able to migrate seamlessly.

VMWare vSphere (vCloud) Suite is an integrated solution for building and operating a private cloud based on VMware vSphere that leverages the software-defined data center architecture. VCloud can integrate virtualized IT services with analytics-based, highly automated operations management. Moreover, VMWare vSpere deploys infrastructure services in a fast pace with full command of critical business and IT policies.

Ganeti - is an open source virtualization management platform developed at Google. Ganeti is a cluster virtual server management software tool built on top of existing virtualization technologies and it is used internally in Google to serve in total its infrastructures. It is based primarily on Xen virtualization platform and it makes it simple to manage many nodes and much more instances at a time. The basic goals of Ganeti is to increase the availability, to reduce hardware cost using cheap commodity hardware, to increase flexibility, to provide transparency and to reduce hardware fault-tolerance. It supports different host systems and it is independent on specific hardware. Ganeti components are the master daemon that controls overall cluster coordination, the node daemon which controls node functions, the conf daemon which provides a fast way to query configuration, an API daemon which provides a remote API and Htools, that means tools responsible for auto-allocation and rebalancing.

Other well-known tools and frameworks in this filed include:

- ✓ **AKKA**, a JVM-based toolkit and runtime that implements the actor model concurrency;
- ✓ **EUKALYPTUS** is an open source software that promises the creation of on-premise private clouds, with no requirements for retooling the organization's existing IT infrastructure or need to introduce specialized hardware;
- ✓ **OpenNebula**, an open-source cloud computing toolkit for allowing integration with different storage and network infrastructure configurations and hypervisor technologies.

6. Annex II

6.1 Mint

MINT is a web based platform designed and developed to facilitate aggregation initiatives for cultural heritage content and metadata in Europe. It is employed from the first steps of such workflows, corresponding to the ingestion, mapping and aggregation of metadata records, and proceeds to implement a variety of remediation approaches for the resulting repository. The platform offers a user and organization management system that allows the deployment and operation of different aggregation schemes (thematic or cross-domain, international, national or regional) and corresponding access rights. Registered organizations can upload (http, ftp, OAI-PMH) their metadata records in xml or CSV serialization in order to manage, aggregate and publish their collections.

A reference metadata model serves as the aggregation schema to which the ingested (standard or proprietary) schemata are aligned. Users can define their metadata crosswalks with the help of a visual mappings editor for the XSL language. Mapping is performed with simple drag-and-drop or input operations, which are then translated to the corresponding code. The mappings editor visualizes both the input and target XSD, in an intuitive interface that provides access and navigation of the structure and data of the input schema, and the structure, documentation and restrictions of the target one. It supports string manipulation functions for input elements in order to perform 1-n and m-1 (with the option between concatenation and element repetition) mappings between the two models. Additionally, structural element mappings are allowed, as well as constant or controlled value (target schema enumerations) assignment, conditional mappings (with a complex condition editor) and value mappings between input and target value lists. Mappings can be applied to ingested records, edited, downloaded and shared as templates between users of the platform.

Preview interfaces present to users the steps of the aggregation including the current input xml record, the XSLT of their mappings, the transformed record in the target schema, subsequent transformations from the target schema to other models of interest (e.g. Europeana's metadata schema), and available html renderings of each xml record. Users can transform their selected collections using complete and validated mappings in order to publish them in available target schemas for the required aggregation and remediation steps.

The metadata ingestion workflow that takes place in LoCloud framework, as illustrated in *Figure*, consists of three main steps. First is the Import of provider's metadata using common data delivery protocols, such as OAI-PMH, HTTP and FTP. Following is the Schema Mapping procedure, during which the imported metadata can be mapped to LIDO, Carare2.0.1 or EDM that serve as the intermediate metadata standards between provider's metadata and Europeana Data Model. A graphical user interface assists content providers in mapping their metadata to the selected schema, using an underlying machine-understandable mapping language. Furthermore, it provides useful statistics about the provider's metadata also supporting the share and reuse of metadata crosswalks and the establishment of template transformations. The third step is the Transformation procedure, during which provider's metadata is transformed to the selected schema by using the mapping they made in the previous step.

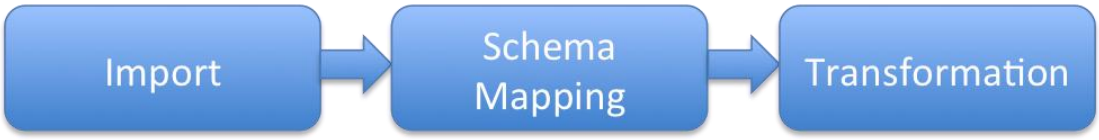


Figure MINT Ingestion workflow