DELIVERABLE

Project Acronym:	**Europeana Cloud**
Grant Agreement number:	**325091**
Project Title:	**Europeana Cloud: Unlocking Europe's Research via The Cloud**

D2.3 – Prototype of Metadata Cloud

Revision: 1

Authors:

Pavel Kats, Europeana Foundation

Contributors:
Marcin Werla
Yorgos Mamakis
Markus Muhr

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

Revision History

Revision	Date	Author	Organisation	Description
1.0	12/02	Pavel Kats	Europeana	1st draft

D2.3 – Prototype of Metadata Cloud

Executive summary

The delivered Prototype of Metadata Cloud is the first version of Europeana Cloud's digital infrastructure. It consists of two core services which together allow data providers and aggregators to upload and access datasets of digital objects. The system uses standard open-source technologies for the database and storage components and is exposed via a set of standard APIs. The APIs are currently for experimental usage only.

Introduction

This deliverable describes the first release candidate version (RC1) of the Metadata Cloud Prototype, the technical infrastructure which will underpin the Europeana Cloud project. Built by Work Package 2 (WP2), this infrastructure will provide the project with a cloud-based system for storing, searching and managing and operating cultural-heritage records and datasets, regardless of their format.

The requirements for the infrastructure were initially drafted by the three main aggregators of the Europeana Cloud project: Europeana, The European Library, and the Polish Digital Libraries Federation. They were then prioritised and published in the deliverable D2.2 – Europeana Cloud Architectural Design.

This deliverable also presented the software architecture of the infrastructure and the division into standalone software services that would constitute the core of it. Development of the services is planned along the lifetime of the project.

The main contribution of milestone D2.3, described by the current deliverable, is the two core services which are the basic building blocks on which the future infrastructure will be erected – essentially, the services which allow a data provider to upload digital objects to Europeana Cloud and associate them with the local copies held by the provider.

The next sections present a more detailed description of the delivered services, including the technical specifications, the software architecture of the system, a description of the installation process and an overview of the demonstration model built by The European Library.

Services

The RC1 version of Metadata Cloud Prototype consists of two fundamental services.

The first is the *Unique Identifier Service* (UIS). It assigns persistent identifiers to records stored in Europeana Cloud by data providers. The service maintains a mapping between Europeana Cloud Identifiers and local identifiers used by the provider – this mapping is necessary to ensure continuous interoperability between Europeana Cloud and software systems ran by data providers.

The second is the *Metadata and Content Service* (MCS). It is responsible for storing records in Europeana Cloud and querying them. The service implements structural approach to entities in the data model (see below). Datasets are stored and queried within the context of a data provider, representations are stored and queried within the context of a record and so on.

Together, these two services allow data providers to upload local digital objects which have been organised in records and datasets to Europeana Cloud by establishing a persistent mapping between the local identifier of a record and its Europeana Cloud identifier, and to perform further updates and queries of this record.

Both services reflect the data modelling proposed in D2.2 – Europeana Cloud Architectural Design. This modelling describes the knowledge domain of the system in terms of the following entities:

- Data provider – organisations responsible for providing data to Europeana Cloud
- Dataset – administrative or thematic collection of data records
- Records – unit of data transferred between providers, aggregators and clients
- Representation – a way a record is presented in one of digital formats
- Version – a version of a representation

- Files – a physical file constituting a version

For detailed documentation of the services, browse the online specification here and here.

Architecture

The diagram below illustrates the service architecture of Europeana Cloud (as outlined in D2.2 – Europeana Cloud Architectural Design). Each service has a corresponding delivery milestone, RC1 being the current deliverable.

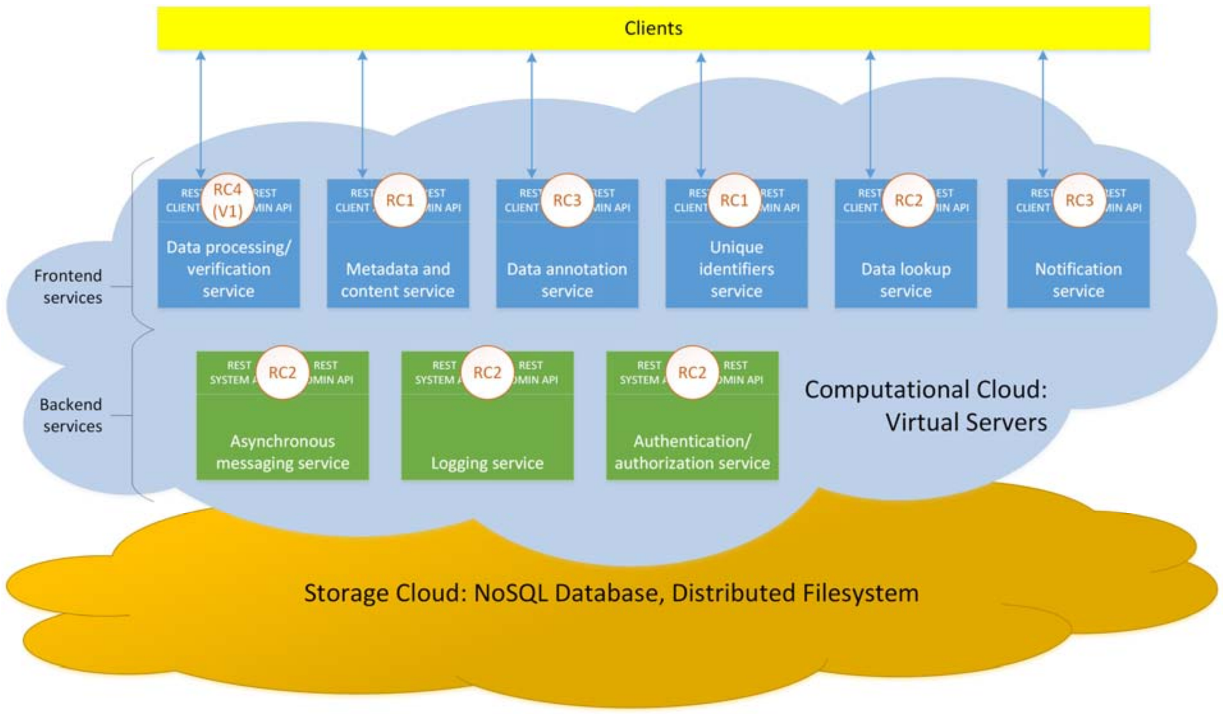
Clients
Frontend services
Backend services
RC4 (V1)
REST CLIENT
REST ADMIN API
Data processing/ verification service
RC1
Metadata and content service
RC3
Data annotation service
RC1
Unique identifiers service
RC2
Data lookup service
RC3
Notification service
RC2
REST SYSTEM
Asynchronous messaging service
RC2
Logging service
RC2
Authentication/ authorization service
Computational Cloud: Virtual Servers
Storage Cloud: NoSQL Database, Distributed Filesystem

Figure 1: Europeana Cloud Service Architecture and Roadmap

During the development phase, the system is installed on the data centres of two partnering institutions: the Poznan Supercomputing and Networking Centre (PSNC), operating the Polish Digital Libraries Federation, and Istituto di Scienza e Tecnologie dell'Informazione of the National Research Council (CNR-ISTI), already operating the Europeana Labs development environment. In addition to hosting the infrastructure itself, the two partners also maintain the 3rd party components constituting its storage layer: OpenStack, distributed file-system is hosted by CNR-ISTI and Apache Cassandra, the distributed database for storing the technical metadata related to each record, is maintained by PSNC. The diagram below illustrates this deployment topology:

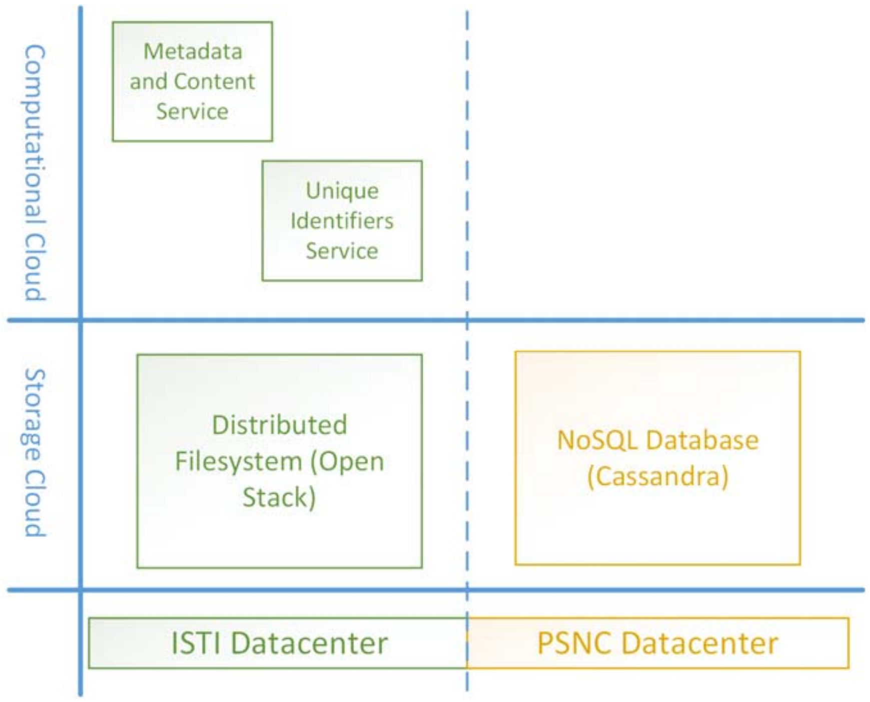
Computational Cloud
Metadata and Content Service
Unique Identifiers Service
Storage Cloud
Distributed Filesystem (Open Stack)
NoSQL Database (Cassandra)
ISTI Datacenter
PSNC Datacenter

Figure 2: RC1 Deployment Topology

Installation

Europeana Cloud uses only open-source components and can be easily installed in any standard environment. This is in line with the Description of Work and the overriding open-source philosophy behind Europeana in general. Appendix A contains detailed installation instructions for a 64bits Linux machine.

Demonstrator

For data providers who wish to use Europeana Cloud, the first step will be ingesting their datasets. Due to the variety of forms in which data can be stored, we expect that several methods will be required to support the individual needs of data providers. The more we can tailor our system to meet these needs, the easier it will be for providers to add data to the cloud. Ease-of-use is a key factor in ensuring that our new infrastructure is widely adopted.

To start addressing these issues, we have created a demonstration model that uses the RC1 prototype to ingest a dataset from The European Library into the cloud. It takes a sample collection from The European Library (a0448 – Reading Europe) and uploads it to Europeana Cloud, where each object is stored in two representations: in the proprietary binary format and in Europeana Semantic Elements (ESE). The ingested collection can be accessed through The European Library portal. The small caption (‘Loaded via Europeana Cloud’) at the bottom of each object page signifies that the object was indeed loaded in real time via the current version of the Europeana Cloud API.

Further Activities

The work package partners will continue to extend the delivered services and to developing new services in the order of priority, as depicted above in Figure 1. In parallel we'll support project partners and external institutions who would be interested in testing the infrastructure. For expressions of interest please contact the author at pavel.kats@europeana.eu.

Europeana Cloud Installation

Europeana Cloud Installation Instructions

Changelog

30-1-2014

[Yorgos Mamakis] Initial version of the document, instruction for Linux

06-02-2014

Ola (Aleksandra) Nowak Solr install instructions for Europeana Cloud ver. > 0.1

This guide has been written for the Linux OS 64-bit version. Versions for MacOS and Windows vary. Linux 32-bit version is not recommended due to memory limitations.

Components:

- Apache Tomcat 7.0.42
- Apache Cassandra 2.0.4
- Apache Solr 4.0.5
- Openstack Swift
- Oracle Java 1.7.0_40
- Apache Maven 3.0.5
- git

Component Installation

Oracle Java Installation

- Download Oracle Java 1.7.0_40 from http://download.oracle.com/otn/java/jdk/7u40-b43/jdk-7u40-linux-x64.tar.gz
- Extract
  - $ tar -xvzf 7u40-b43/jdk-7u40-linux-x64.tar.gz
- Set JAVA_HOME and PATH environment to use this version of Java
  - $ export JAVA_HOME=[path_to_java]
  - $ export PATH=$PATH:$JAVA_HOME/bin

Apache Maven Installation

- Download Apache Maven from

  http://apache.mirror1.spango.com/maven/maven-3/3.0.5/binaries/apache-maven-3.0.5-bin.tar.gz

- Extract
  - $tar -xvzf apache-maven-3.0.5-bin.tar.gz
- SET M2, M2_HOME and PATH environment variables
  - $ export M2_HOME= [path to maven]
  - $ export M2 = $M2_HOME/bin
  - $ export PATH=$PATH:$M2

Git Installation (assuming Debian - based Linux version)

- $ sudo apt-get install git

Apache Cassandra Installation

- Download Apache Cassandra from http://www.apache.org/dyn/closer.cgi?path=/cassandra/2.0.4/apache-cassandra-2.0.4-bin.tar.gz
- Extract
    - $ tar -xvzf apache-cassandra-2.0.4-bin.tar.gz
- Make directories for data and logs.
- Modify [cassandra installation folder]/conf/cassandra.yaml file and change the following parameters:
    - data_file_directories
    - commitlog_directory
    - saved_caches_directory

  and point them to actual (different) folders

- Modify [cassandra installation folder]/conf/log4j-server.properties file and change the parameter
    - log4j.appender.R.File to point to previously created logs directory
- Start Cassandra by bin/cassandra

Apache Tomcat Installation

- Download Apache Tomcat from

http://archive.apache.org/dist/tomcat/tomcat-7/v7.0.42/bin/apache-tomcat-7.0.42.tar.gz

- Extract
    - $ tar -xvzf apache-tomcat-7.0.42.tar.gz
- Start Tomcat by
    - bin/catalina.sh run (to see the output)

Apache Solr in Tomcat Installation

- Download modified solr war from

https://www.dropbox.com/s/qy3qagtixwgfnsr/solr.war

- Save it in [tomcat installation folder]/webapps
- Startup tomcat (and fail on the first time)
- modify file webapps/solr/WEB-INF/classes/solr/conf/solrconfig.xml and point

```
<dataDir>/home/solr/data/</dataDir>
```

property to a valid folder in the filesystem

- Restart tomcat

This instance of Solr is already using the configuration required by MCS

Apache Solr in Tomcat Installation (for versions > 0.1)

- Build Europeana Cloud Solr project. It has a dependency to Solr and repacks solr archive with Europeana Cloud specific configuration. Solr requires two parameters to be defined: solr.home and solr.data.dir. If you want to change defalult values of these parameters, you should build this project with -Dsolr.home and -Dsolr.data.dir parameters, eg. mvn install -Dsolr.home=/home/solr/ -Dsolr.data.dir=/home/solr/data/
- Move war file from ecloud-solr/solr-distr/target to [tomcat installation folder]/webapps

OpenStack Swift Installation (on a OpenStack working controller machine running Debian 7 Linux version)

- sudo apt-get install swift swift-proxy swift-account swift-container swift-object memcached xfsprogs

Europeana Cloud configuration

Apache Tomcat Configuration

MCS and UIS use Tomcat environmental variables to configure and setup the services. These are stored in [tomcat installation folder]/conf/server.xml and are

```
 <!-- UIS local specific configuration -->
       <Environment name="uis/cassandra/host" type="java.lang.String" value="localhost"/>
       <Environment name="uis/cassandra/port" type="java.lang.Integer" value="9042"/>
       <Environment name="uis/cassandra/user" type="java.lang.String" value=""/>
       <Environment name="uis/cassandra/password" type="java.lang.String" value=""/>
       <Environment name="uis/cassandra/keyspace" type="java.lang.String" value="ecloud_uis"/>
       <!-- MCS local Cassandra specific configuration -->
       <Environment name="mcs/cassandra/host" type="java.lang.String" value="localhost"/>
       <Environment name="mcs/cassandra/port" type="java.lang.Integer" value="9042"/>
       <Environment name="mcs/cassandra/user" type="java.lang.String" value=""/>
       <Environment name="mcs/cassandra/password" type="java.lang.String" value=""/>
       <Environment name="mcs/cassandra/keyspace" type="java.lang.String" value="ecloud_test"/>
       <!-- MCS Swift specific configuration -->
       <Environment name="mcs/swift/provider" type="java.lang.String" value="swift-keystone"/>
       <Environment name="mcs/swift/container" type="java.lang.String" value="ecloud"/>
       <Environment name="mcs/swift/user" type="java.lang.String" value=""/>
       <Environment name="mcs/swift/password" type="java.lang.String" value=""/>
       <Environment name="mcs/swift/endpoint" type="java.lang.String" value="http://localhost:5000/v2.0"/>
       <!-- MCS Solr specific configuration -->
       <Environment name="mcs/solr/url" type="java.lang.String" value="http://localhost:8080/solr/"/>
       <Environment name="mcs/uis-url" value="http://localhost:8080/ecloud-service-uis-rest-0.1-SNAPSHOT/"
type="java.lang.String" override="false"/>
```

These parameters should be placed in the <GlobalNamingResources> directive

Apache Cassandra Configuration

Execute [cassandra installation folder]/bin/cqlsh and issue the following commands for MCS setup

```
 CREATE KEYSPACE ecloud_test
         WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 1};
  USE ecloud_test;
CREATE TABLE data_sets (
       provider_id varchar,
       dataset_id varchar,
       description text,
       creation_date timestamp,
       PRIMARY KEY (provider_id, dataset_id)
);
```

```
CREATE TABLE data_set_assignments (
        provider_dataset_id varchar, /* concatenation: provider_id | dataset_id */
        cloud_id varchar,
        schema_id varchar,
        version_id timeuuid,
        creation_date timestamp,
        PRIMARY KEY (cloud_id, schema_id, provider_dataset_id)
);
CREATE INDEX data_set_assignments_provider_dataset_id ON data_set_assignments (provider_dataset_id);

CREATE TABLE representation_versions (
        cloud_id varchar,
        schema_id varchar,
        version_id timeuuid,
        provider_id varchar,
        persistent boolean,
        files map<varchar, text>, /* fileName -> json object: (mime_type, content_md5, content_length, last_modification_date)
*/
        creation_date timestamp,
        PRIMARY KEY (cloud_id, schema_id, version_id)
);
CREATE INDEX representations_provider_id ON representation_versions (provider_id);
```

For UIS setup issue:

```
CREATE KEYSPACE ecloud_uis

          WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 1};
```

OpenStack Swift Configuration

These configuration refers to a Swift installation running on a OpenStack working controller machine, using a local disk array (/dev/cciss/c0d1) with no file redundancy.
The instructions must be executed with 'su' permissions.

Modify swift-hash section in /etc/swift/swift.conf:

```
...
[swift-hash]
swift_hash_path_suffix = "AnyString"
...
```

execute the following commands as root user:

```
$ fdisk /dev/cciss/c0d1
$ mkfs.xfs /dev/cciss/c0d1p1
```

Modify /etc/fstab adding:

```
/dev/cciss/c0d1p1 /srv/node/c0d1p1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0
```

Execute following commands as root

```
$ mkdir -p /srv/node/c0d1p1
$ mount /srv/node/c0d1p1
$ chown -R swift:swift /srv/node
```

Modify the file /etc/rsyncd.conf:

```
...
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 192.168.48.113
[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock
[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock
[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
...
```

Modify the file /etc/default/rsync:

```
RSYNC_ENABLE = true
```

Execute the following instructions:

```
$ service rsync start
$ mkdir -p /var/swift/recon
$ chown -R swift:swift /var/swift/recon
$ cd /etc/swift
$ openssl req -new -x509 -nodes -out cert.crt -keyout cert.key
```

Modify /etc/memcached.conf and restart the service:

```
-l 192.168.48.113

$ service memcached restart
```

Modify the file /etc/swift/proxy-server.conf:

```
 ...
[DEFAULT]
bind_port = 8888
user = swift
[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth proxy-server
[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
delay_auth_decision = true
signing_dir = /home/swift/keystone-signing
auth_protocol = http
auth_host = 192.168.48.113
auth_port = 35357
admin_token = "AdminToken"
admin_tenant_name = services
admin_user = swift
admin_password = "SwiftPass"
[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = Member,admin,swiftoperator
[filter:healthcheck]
use = egg:swift#healthcheck
```

```
[filter:cache]
use = egg:swift#memcache
[filter:catch_errors]
use = egg:swift#catch_errors
...
```

The following commands must be executed to complete the configuration and start the Swift service

```
$ chown -R swift:swift /etc/swift
$ mkdir -p /home/swift/keystone-signing
$ chown -R swift:swift /home/swift/keystone-signing
$ swift-ring-builder account.builder create 7 1 1
$ swift-ring-builder container.builder create 7 1 1
$ swift-ring-builder object.builder create 7 1 1
$ swift-ring-builder account.builder add z1-192.168.48.113:6002/c0d1p1 100
$ swift-ring-builder container.builder add z1-192.168.48.113:6001/c0d1p1 100
$ swift-ring-builder object.builder add z1-192.168.48.113:6000/c0d1p1 100
$ swift-ring-builder account.builder
$ swift-ring-builder container.builder
$ swift-ring-builder object.builder
$ swift-ring-builder account.builder rebalance
$ swift-ring-builder container.builder rebalance
$ swift-ring-builder object.builder rebalance
$ chown -R swift:swift /etc/swift
$ service swift-proxy start
$ swift-init main start
$ service rsyslog restart
$ service memcached restart
$ keystone tenant-create --name services
$ keystone user-create --tenant-id 7a6b7b67db004621a5cffc38b2977015 --name swift --pass "SwiftPass"
$ keystone service-create --name swift --type object-store --description "Swift Storage Service"
$ keystone endpoint-create --region regionOne --service-id=0198cdf93d0a4d138bd2068e5e869b97 \
--publicurl 'http://146.48.82.113:8888/v1/AUTH_%(tenant_id)s' \
--adminurl 'http://192.168.48.113:8888/v1' \
--internalurl 'http://192.168.48.113:8888/v1/AUTH_%(tenant_id)s'
```

Build and Install services

In order to install the services, checkout the code from Git

```
$ mkdir ecloud-source

$ cd ecloud-source

$ git pull http://github.com/europeana/Europeana-Cloud

$ cd Europeana-Cloud

$ mvn install
```

After this step two web applications will have been created in

- services/mcs/rest/target/ecloud-mcs*.war
- services/uis/rest/target/ecloud-uis*.war

At this point, these web applications use the in-memory implementation of the services

Install the services by copying them to [tomcat installation folder]/webapps

In order to use the persistent implementation of these services, then the following should be executed

```
$ cd services/mcs/rest

$ mvn clean install -p persistent

$ cd ../../uis/rest

$ mvn clean install -p persistent
```

After that both of the services will use the persistent implementation (Cassandra, Openstack Swift)

The location of the services is the same as above