DELIVERABLE

Project Acronym: **Europeana v3.0**

Grant Agreement number: **620484**

D5.1 - Operations Process Description Manual

Revision: 1 (May 2015)

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

Authors:

Pavel Kats (Europeana Foundation)
Bram Lohman (Europeana Foundation)
Yorgos Mamakis (Europeana Foundation)
Jacob Lundqvist (Europeana Foundation)

Revision	*Date*	*Author*	*Organisation*	*Description*
0.1	*May 2015*	*Pavel Kats* *Bram Lohman* *Yorgos Mamakis* *Jacob Lundqvist*	*EF* *EF* *EF*	*Initial Version*

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This deliverable contains original unpublished work except where clearly indicated otherwise.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.

Introduction

As described in the deliverable *D5.2 Review of Technical and Logical Architecture*, Europeana operates a complex infrastructure for ingesting and publishing digital heritage metadata and content. Managing this infrastructure involves various procedures, needed for deploying new versions of software, publishing new content and running miscellaneous supporting tasks.

As part of Europeana's migration to a Digital Service Infrastructure (DSI), these procedures are standardized, streamlined and documented by the Europeana team. This is an ongoing process because computational resources used by Europeana change quite often; so do software components using these resources.

In the future, we are planning to automate some of the procedures to leave less space to human error and make our operations more efficient.

At this point in time, by the end of Europeana v3, the goal of this deliverable is to share some of the core maintenance procedures.

This deliverable is read best in conjunction with the previous one, *D5.2*.

Production

PaaS Application Deployment

In the new hosting environment (Cloud Foundry PaaS), Europeana's components are divided into three groups and hosted, accordingly, in three different context:

- Applications - custom components interacting with end-users
- Services - standard components, used by applications and managed by the platform (databases, log services)
- VMs - custom components which are still not compatible with the PaaS technology and run in isolation on traditional virtual machines.

The figure below depicts this architecture.

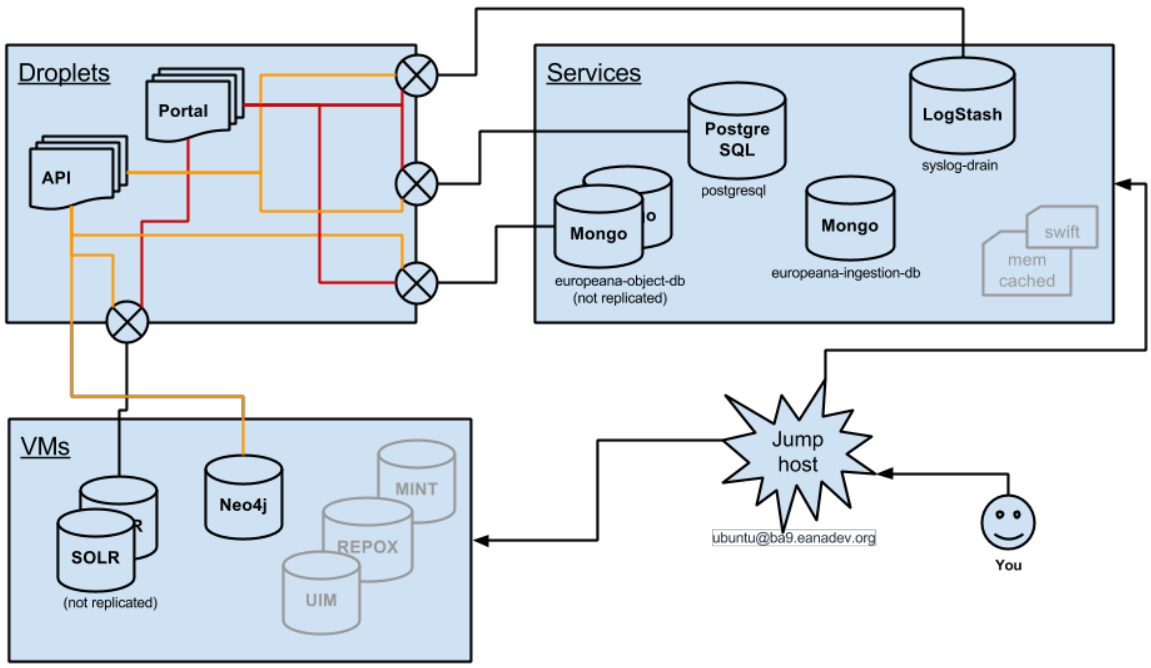

Figure 1: Component Architecture

Setting up services

Applications deployed to a PaaS access external resources via Services. All external dependencies such as databases, messaging systems, files systems and so on are Services. The following commands allow managing the Services:

- `cf services` to check existing services
- `cf marketplace` to check what services can be created

- **`cf create-service`** `to create the required service (e.g. cf create-service postgresql Pluto-free postgres_test)`

Connecting to services

For initialisation, editing outside the application, and other custom neede. direct access to some of the services is sometimes needed. The services are only accessible via the PaaS network; to get there, use the provided jumphost..

- `ssh <jumphost>`
- `less service-access-readme`
- Port-forward (using an SSH tunnel) the following databases
  - `Ingestion Mongo`
  - `Portal Mongo`
  - `PostgreSQL`
  - `Neo4j`
- Solr can be accessed by SSHing via the jumphost

Deploying applications

Step 1: Edit the YAML file

Applications are described in configuration files `manifest.yml`. It has the following fields:

- applications
  - name: needs to be unique within the space
  - host: needs to be unique across all organisations (and a valid DNS address)
  - domain: `de.a9sapp.eu`
  - path: location of the war file, e.g. path: **`api2-war/target/api.war`**
  - services: connections to existing services, e.g. - **`postgres_test`**

Step 2: Add the application credentials to any required services

As we are using a customised setup, the Cloud Foundry Orchestrator doesn't automatically add the randomly created application user/password to the requested services. To do this automatically, set up a port-forward to the required database, log in as admin user (see service-access-readme), and create the user with read/write roles, e.g. for MongoDB:

- **`db.createUser({user:"<USER_ID>",pwd:"<USER_PASSWORD>", roles:["readWrite"]})`**

The username/password can be found in the logs once the application has started, e.g.

- **`cf files api-test logs/env.log | grep username`**

Step 3: Push the application

Issue the following command:

- `cf push`

Portal/API Deployment

This section provides instruction on how to deploy a new instance of the Portal and the API application. Europeana employs the blue-green deployment approach for deploying and running its applications. Since any change takes an environment offline, should first push changes to the inactive instance, then swap the active environment.

Step 1: Generating war files

The war files are normally generated on the buildhost using the build user. If you need to ensure you are building the same binaries as in production, find the revision number in `http://www.europeana.eu/portal/build.txt` or `http://www.europeana.eu/api/build.txt` and then check out the same revision, e.g. "Revision d6d53f08c8 built at..." would mean you should do:

- `git checkout 4fbd854141` before building it
- `portal build command mvn clean install -DskipTests -P production`

API does not need a separate build, the one generated normally for a9builder is enough to use.

Step 2: Access the buildhost

Login to the buildhost with a user suited for building the production environment.

Step 3: Pick up WAR files

Run the following commands:

- `cp -av ~a9builder/proj/alpha/api2-alpha/api2-war/target/api.war ~/warfiles`
- `cp -av ~a9builder/proj/alpha/portal-alpha/portal2/target/portal.war ~/warfiles`

Step 4: Deploy WAR files and resources

The following resources need to be deployed

- `~/warfiles` Before deploying you need to move war files to be deployed to this location
- `~/webroot` Static content that should be served by apache

- `~/instances` Yaml files for tomcat projects, apache yaml config files need to be in a special file hierarchy
- `~/backups` All deployed content is saved here

Step 4: Checking what environ is currently active

Issue the following command:

- `cf routes`

If blue is active, do your changes to green, or vice versa.

Step 5: Activating blue (for green swap all green/blue refs)

Use the following commands to depoy the applications you need:

- `web-deploy blue`
- `api-deploy blue`
- `portal-deploy blue`

Wait several mins after deploy is done to ensure that all initial startup processing is done by checking that cpu usage is close to 0%:

- `cf app blue-portal`
- `cf app blue-api`
- `cf app blue-web`

Step 6: verifying the new instances

Run queries on the environment you deployed to make sure that it works properly.

Step 7: Activate environments

- `activate-environ blue`
- `activate-environ green`

Ingestion

UIM is Europeana ingestion workflow engine. The section below provides instruction on (re)starting its main components.

Restarting UIM

Depending on the type of issue a number of procedures must be followed.

If UIM crashed because of java.lang.OutOfMemoryError (affecting MongoDB as well):

- `ssh <uim-host>`
- `ps uxa|grep mongo -> No Mongo instance reported`
- `/etc/init.d/mongodb start`
- `tail -f /var/log/mongodb/mongodb.log` (Wait till the accepting connections message is shown in the logs)
- `Ctrl+C`
- `su - <uim-user>`
- `ps uxa|grep karaf` (check if UIM is still up in crashed state)
- `1.8 kill -9 pid`
- `1.9 cd apps/apache-karaf-2.11`
- `bin/start`

In case only the UIM crashed follow the last five steps of this procedure.
In case the neo4j server has an issue (the ingestion portal does not show hierarchies)

- `ssh <neo4j-host>`
- `ps uxa|grep neo4j -> Check if there is a running instance of Neo4j`
- `cd /data/neo4j-community-2.1.2/`
- `1.14 bin/neo4j restart`

In case the enrichment server has an issue (running enrichment on a dataset produces logs related to EnrichmentDriver or connection refused)

- `ssh <enrichment-server>`
- `ps uxa|grep mongo` (Check if Mongo is affected if not go to the last command)
- `/etc/init.d/mongodb start`
- `/etc/init.d/tomcat stop`
- `/etc/init.d/tomcat start`
- (if the last one fails go to `/usr/local/tomcat` and issue `bin/catalina.sh start`)

In case the remote MongoDB fails:

- `ssh <mongodb-server>`
- `ps uxa|grep mongo`
- `kill -9 pid`
- `/etc/init.d/mongodb start`
- `Wait several minutes`
- Restart the UIM (above)

In case Apache Solr fails:

- `ssh <solr-server>`

- `ps uxa|grep catalina`
- `kill -9 pid`
- `/etc/init.d/solr start`
- Restart the UIM (above)

In case preview ingestion portal crashes:

- `ssh <ingestion portal server>`
- `/etc/init.d/tomcat stop`
- `ps uxa|grep catalina`
- `/etc/init.d/tomcat start`

Restarting REPOX

Log into the server running REPOX.

- `su uim`
- `cd $home`
- `ps aux | grep catalina`
- `kill -9  <process name>`
- `sh apache-tomcat-6.0.36/bin/startup.sh`

Restarting MINT

Log into the server running MINT.

- `su uim`
- `cd $home`
- `ps aux | grep catalina`
- `kill -9  <process name>`
- `sh apache-tomcat-6.0.36/bin/startup.sh`

Summary

The deliverable above describes several core operational procedures Europeana employs.