DIGIT
Unit B3

# EU-FOSSA 2 - SR80 - Deliverable D07.02

# An updated inventory and analysis of the FOSS (Free and Open Source Software) in use at the European Commission

## Report for external publication

Date: 08/05/2020
Doc. Version: Final

## TABLE OF CONTENTS

## TABLE OF FIGURES

## TABLE OF TABLES

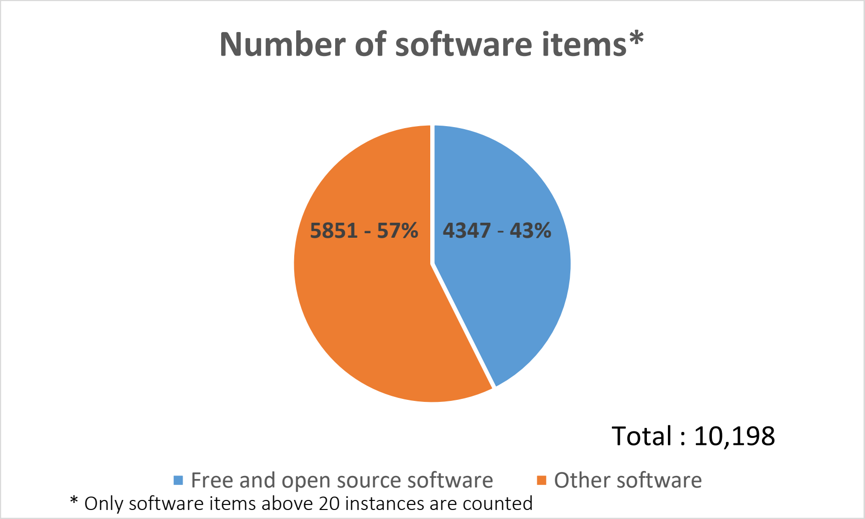
# 1. Executive Summary

WP2 of the EU-FOSSA 2 project commissioned an updated[1] inventory of Free and Open Source Software (FOSS) in use at the European Commission[2]. The methodology and architecture used to collect and analyse the inventory data, was developed in the EU-FOSSA pilot project.  Extracts of software on Servers, AppV, Workstations and the Nexus software repositories was provided to the project in December 2018.

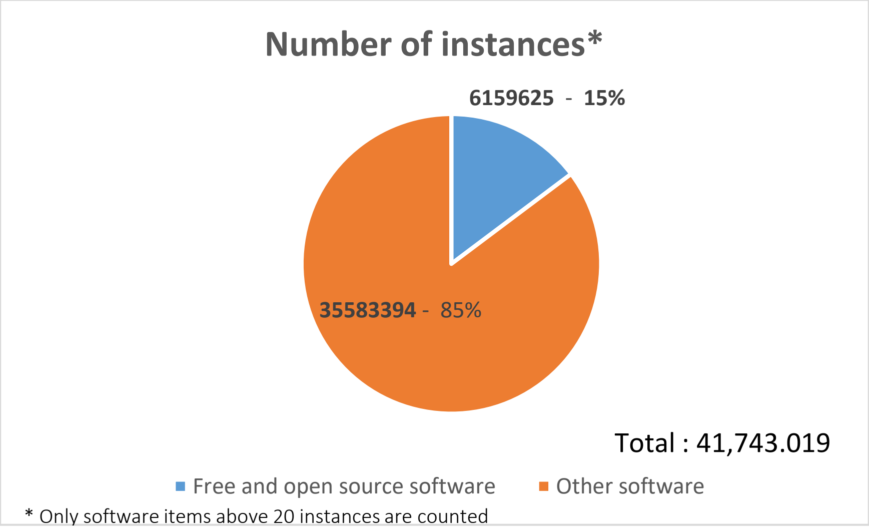
The **inventory scope** covered the infrastructure managed by DIGIT, including Datacentre applications and Desktop applications. The inventory provided results in two main respects: FOSS screening / counting and critical OSS assessment:

## 1. Open Source Software screening / counting results

The screening and counting of software items and of their instances to weight the FOSS on the respective totals provided the following data:



* Only software items above 20 instances are counted



* Only software items above 20 instances are counted

[1] The last inventory was of the January 2016 data conducted by the EU-FOSSA Pilot project.

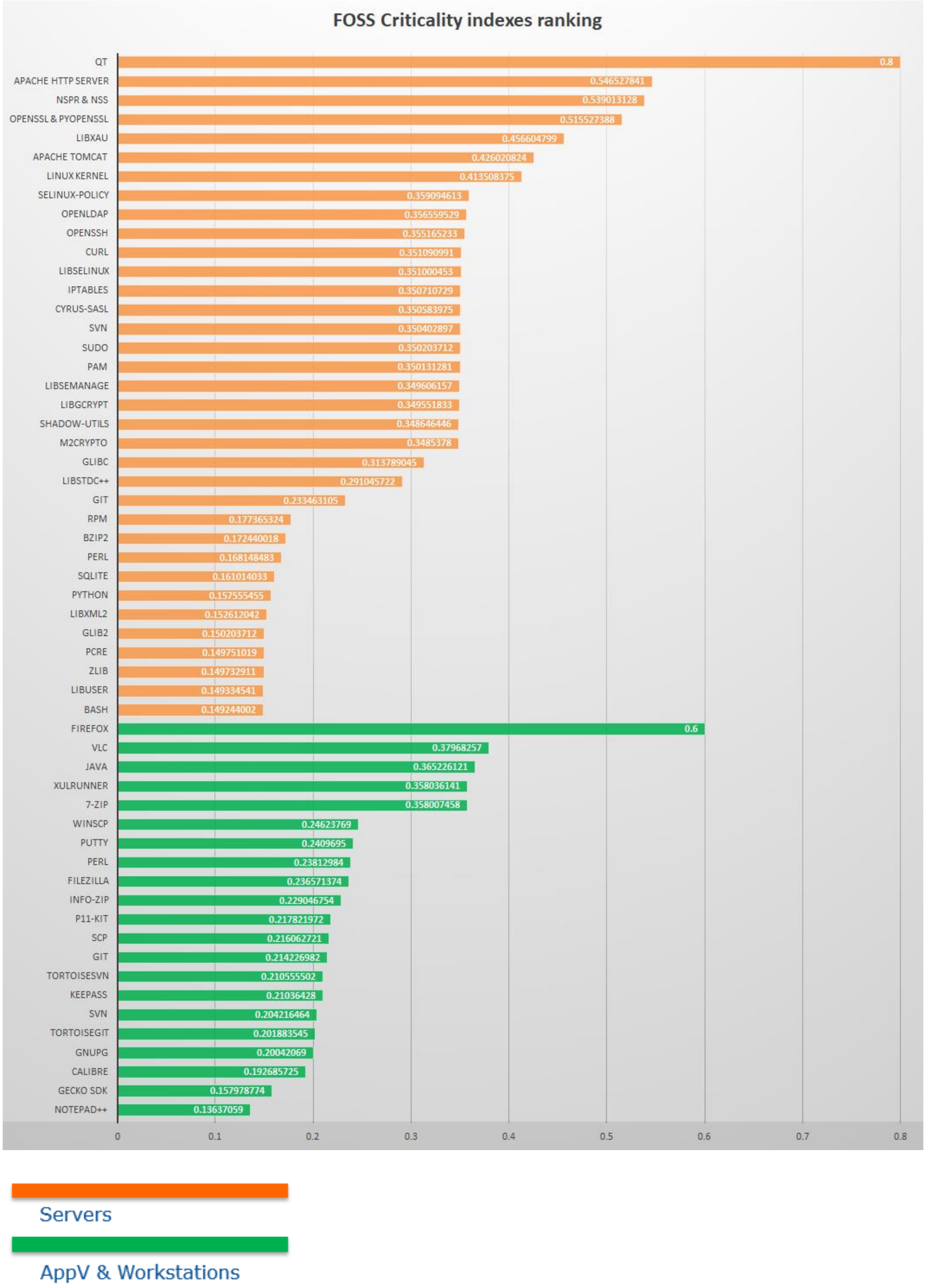
[2] Another similar exercise is being carried out for the European Council, and the results will be published separately.

This means that **<u>a minority, although significant, of all inventoried software in use at the European Commission is FOSS (43% of all software and 15% of installed software instances)</u>**.

## 2. Detailed analysis of Open Source Software

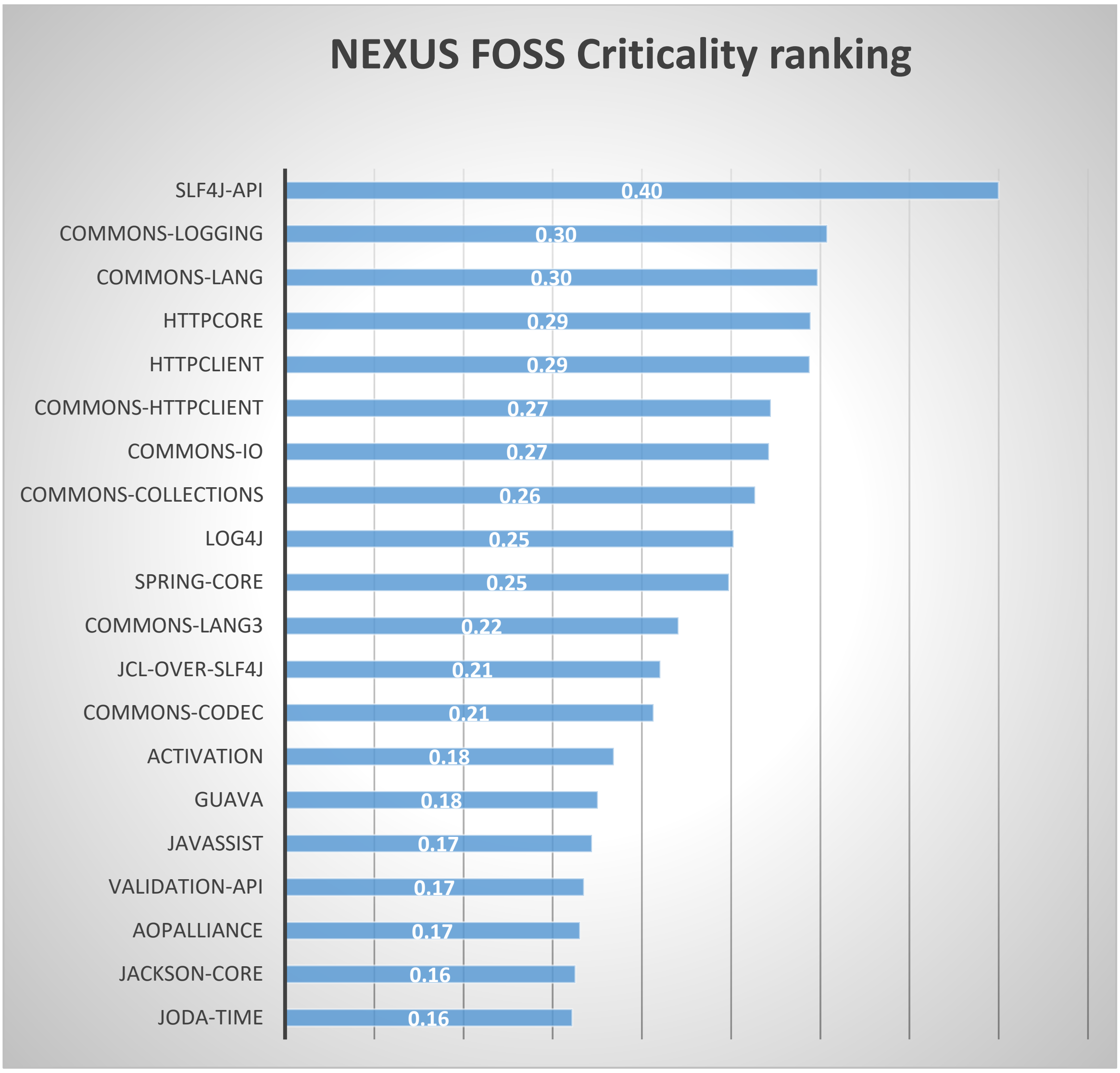
The OSS in use at the European Commission was analysed as follows:

- A <u>business criticality</u> evaluation, based on an analysis of the presence and use of FOSS in the institutions. It took into account parameters such as the number of instances, the exposure to end users and the relation with security. The ranking of FOSS by a <u>Business Criticality Index</u> (BCI), based on such parameters, identified a <u>shortlist of the most critical software in use</u>, encompassing 56 items:



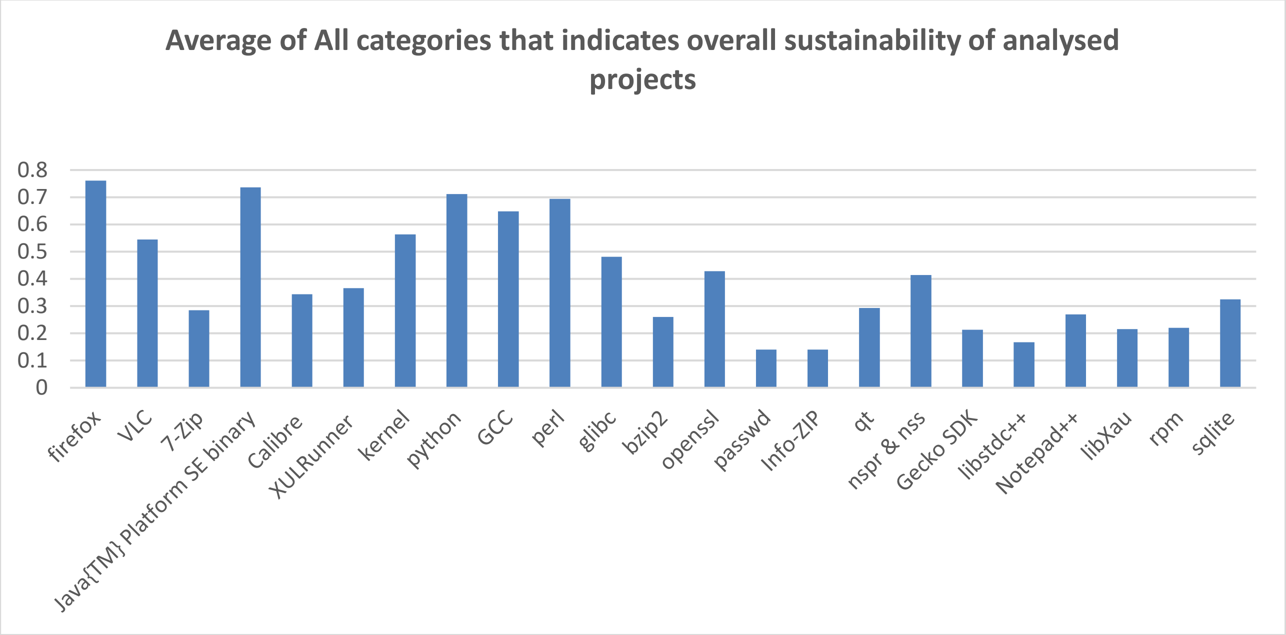
Note: Java is not fully open source software; only source code of some libraries is available, based on a non-OSS compatible license.

- The business criticality evaluation that took place at the NEXUS software repository, provided the following ranking:



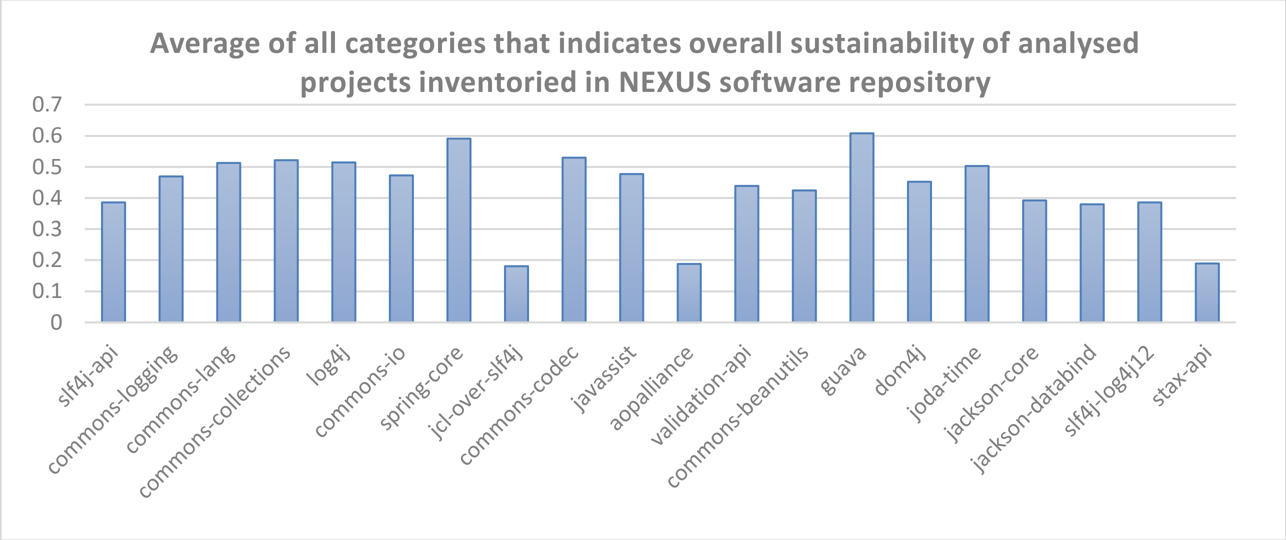
- In addition, an assessment of FOSS sustainability, based on the 34 metrics developed within the pilot EU-FOSSA project took place. Such metrics were used to evaluate the support that each of the Free and Open Source software projects is supposed to get from the community throughout its lifecycle to ensure its long-term success. The elements needed for such evaluation were collected through sources such as project websites, wikis, bug trackers and some reputed platforms of the open source world (e.g. openhub.net or github.com).

  The overall sustainability, calculated as the average of the 34 metrics (100% indicating top sustainability under all metrics), showed that the critical shortlist includes software with a wide range of sustainability, from 20% to almost 80%.



Note: Java is not fully open source software; only source code of some libraries is available, based on a non-OSS compatible license.

Following, the sustainability analysis on the software items retrieved from the Nexus software repository is provided:



## 2. Overview of the Deliverables

The FOSS inventory deliverables were arrived at as follows.

The scope of the inventory exercise was to examine the European Commission data provided to the project in December 2018. Another similar exercise is being carried out for the European Council, and the results will be published separately.

This exercise built on the earlier inventory of data collected in January 2016.

This study describes the various steps through which the *inventory* has been created and analysed, starting from the treatment of the input data to the detailed analysis of the software information. This includes the software rating (both under a business criticality perspective, linked to the use of the software, and under a sustainability perspective, determined according to the metrics developed in the context of the pilot EU-FOSSA project).

Finally, the study analyses the inventory results providing an overview, elaborated on the raw inventory data, of the weight, distribution and typology of the inventoried OSS. Furthermore, it analyses the OSS from the point of view of its sustainability, with particular focus on the critical software shortlist defined.

# 3. Software inventory process

## 3.1. Treatment of input data

The input for the inventory process has been the exports coming from the various data sources, provided in CSV format.

The inputs from Satellite, LANDesk and Nexus software repository have not been submitted to any processing, and have been loaded inside the inventory database by ETL jobs.

For AppV, the input file received was not a CSV file but an Excel table. Consequently, it was transformed in CSV with the format "software name; number of users".

Once in the inventory database, the software list is sorted by descending order of occurrences. All software items from the DB have been listed by SystemType (workstations - source: LANDesk, AppV - source: AppV, and servers - source: Satellite). A separate list was created for the Nexus artefacts as these cannot be directly related/compared to the other software items.

## 3.2. Open source screening & grouping

Open source items have been manually screened in the full list of software, resulting from the previous phase. The screening has been based on the operator's knowledge and on Google searches.

The output of this step is the manually generated "license-override.csv" file, which flags the open source items.

Furthermore, raw inventory data have been elaborated to group all items pertinent to a certain software package and eliminate multiple items referring to the same application, in order to provide a picture where the extent of use of major software packages can be clearly perceived. More specifically:

- Entries like the ones below have been grouped as "Firefox", with the relevant number of instances summed, as they correspond to different installations.
    - FIREFOXPORTABLE
    - Mozilla Firefox ESR, Portable Edition
    - Mozilla Firefox, Portable Edition {Beta}
    - Mozilla Firefox, Portable Edition
    - FirefoxDeveloperEdition
    - Firefox
    - Firefox Nightly
    - Mozilla Firefox Developer Edition, Portable
    - Mozilla Firefox, Portable Edition 2nd Profile
    - Firefox (Mac)
    - Firefox Developer Edition

- Entries like the ones below have NOT been counted towards the sum above as they correspond to the same installation. In other words, they are considered as extensions to the core software and thus their number of instances is ignored.
  - Plugin Hang UI for Firefox
  - Firefox Helper
  - Firefox Software Updater
  - Plugin Container for Firefox

The elaboration has been done manually to obtain a shortlist of software items by instances for each software type.

More details are provided in the following Annexes:

1. “Annex_1_OSS shortlist ranked by Business Criticality Index”
2. “Annex_2_Open source software by system type.xlsx”

### 3.3. Analysing the software for criticality

The metrics applied for the evaluation of the inventoried OSS are mostly meant to identify and rank the software that is potentially critical due to its presence and use at the European Institutions (what we could call “business criticality”). A shortlist of the software items with the highest occurrences in the inventory has been identified and analysed based on such criteria.

In a subsequent step, the software items identified as critical according to this shortlist have been further evaluated and ranked on the basis of sustainability criteria.

The criticality criteria applied are:

1. Number of instances:
   - Rationale: the more a software is deployed, the more it impacts the infrastructure and/or the user base, and the more damage a vulnerability could cause;
   - Rating: this metric divides the number of instances of an item by the number of instances of the most common item in the list, thus providing a weighted view of the frequency of a software compared to the others of the list; its possible values range from above 0 to 1.
2. Exposure to users:
   - Rationale: a vulnerability in a component exposed to the end user (i.e. that offers an interface to the end users) increases the risk of an exploit attacking the software. This criteria only applies to data centre infrastructure, since workstation users have a direct login to their machines;
   - Rating: a binary rating whose values can be either “1” (exposed to users) or “0” (not exposed to users).

3. Relation with security:
    - Rationale: a vulnerability in component related to a security aspect may increase the damage due to an exploit. Examples of security-related software are the solutions meant to secure communication, to manage authentication, to manage processes and permissions, etc.

    - Rating: a binary rating whose values can be either "0.5" (security-related) or "0" (not security-related).

The total score provided by the sum of the three above scores is then normalized by dividing it by 2.5 (the sum of the highest values of the three criteria), so that the applicable values for the final score (the "Business Criticality Index") range from 0 to 1.

"Annex_1_OSS shortlist ranked by Business Criticality Index.xlsx" shows the Business Criticality Index of the top critical items. The "number of instances" is calculated based on the number of deployments; as for App-V, it must be understood as "the number of users". As a result of the above evaluation, open source items with most occurrences, possibly security related and possibly facing a large user base, are considered critical.

Similarly the top critical items for Nexus libraries can be found in tab "Nexus" of the above Excel file.

The output of this phase goes into the "criteria-Datacenter.csv", "criteria-Landesk.csv", "criteria-AppV.csv" and "criteria-nexus.csv" files, manually generated based on information coming from the analysis of the software items.

### 3.4. Software Type filtering for critical software items

For critical software items, a manual categorisation is applied according to the following categories:

**Table 3-1 - Software clusters**

| |
|---|
| AppSoftware / Tool |
| CustomSoftware |
| MobileSoftware |
| RuntimeSoftwarePlatform |
| OperatingSystem |
| DevelopmentPlatform / Framework |
| DataCenterResources |
| Library |

More details are provided in "Annex_3_Top open source software by software type.xlsx".

The output of this step goes into the “opensource-softwaretype.csv” file.

Nexus artefacts are by default categorized as “Libraries”.

### 3.5. Evaluation of the critical software through sustainability criteria

The shortlisted items resulting from the analysis described in paragraph 2.3 have been submitted to manual investigation to be rated against the sustainability metrics.

Such metrics have been calculated automatically through an Excel file (refer to “Annex_4_Assessment of 100 items in the inventoried software against the criticality mechanism defined in the EU-FOSSA Pilot Project.xlsx”) in which, once entered the parameters used to define the metrics, these are automatically calculated according to the provided methodology.

The parameters used for the calculation of sustainability metrics have been looked for in openhub.net, Wikipedia.org, github.com, and the specific community websites of each software items, their documentation, and their bugtrackers.

The output of this phase goes into the same files “criteria-Datacenter.csv”, “criteria-Landesk.csv”, “criteria-AppV.csv” and “criteria-nexus.csv” containing the criticality criteria evaluated as per paragraph 2.3 above, manually integrated with the information on the sustainability metrics calculated through the file above.

Due to the length of the full names of the metrics, their IDs (M1 to M34) have been used instead in the database.

### 3.6. License identification for critical software items

For critical software items, manual research has been performed to identify the applied licenses. Output of this step also goes in the “license-override.csv” file, generated as per paragraph 2.2 above. In the file, each row corresponds to a pair of “software item/ respective licence”. For software with multiple licenses, several rows for the same software item have been created.

After this step, the shortlisted critical software items present in the file are mentioned with the exact names and versions of their licenses, whereas the other open source items are simply labelled as ‘open source’.

### 3.7. Identification of dependencies for critical software items

Within the critical software shortlist, the items from the datacentre servers have undergone a dependency identification process. Actually, the information of the dependencies can only be found in Linux systems. In Linux, when a software package is installed, the package manager resolves a list of dependencies that are downloaded and installed on the fly, whereas on Windows, the installed software packages contain all their dependencies.

Finding the dependencies of a Linux package can be done, on any RedHat-like machine, using this command:

```
yum deplist httpd | grep provider
```

where “httpd” is the name of the package to get dependencies from.

The output of this step goes into the “dependencies.csv” file, generated from the above command through the transformation of the result into CSV format.

Dependencies for Nexus libraries were retrieved by using OWASP Dependency-Check utility.

More details are provided in the following Annexes:

- “Annex_5_Top open source software dependencies.xlsx”
- “Annex_6_Nexus libraries dependencies.xlsx”

### 3.8. Input of the analytical files in the database

Once those additional input files (“criteria-AppV.csv”, “criteria-Datacenter.csv”, “criteria-Landesk.csv”, “criteria-nexus.csv”, “dependencies.csv”, “license-override.csv”, “opensource-softwaretype.csv”) were generated, new Talend jobs were designed to process them and inject them into the inventory database.

The whole docker-compose stack could then be redeployed.

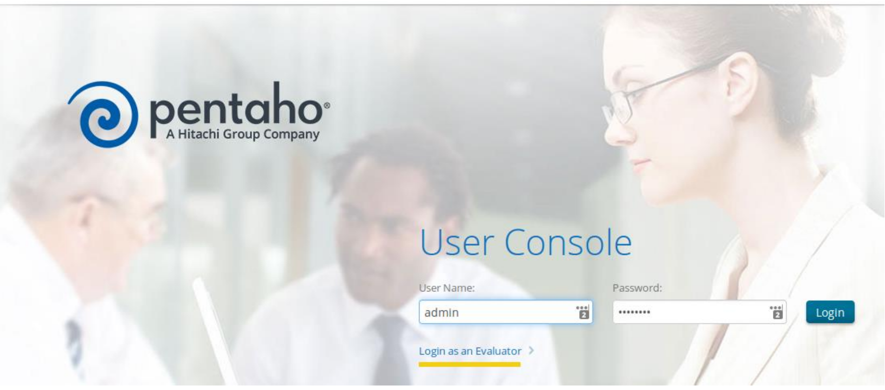
## 4. Creation of dashboards and extraction of inventory reports

Inventory data can be navigated and analysed through Pentaho dashboards. These are collections of other content components displayed together with the goal of providing a centralized view of key performance indicators (KPIs) and other business data movements.

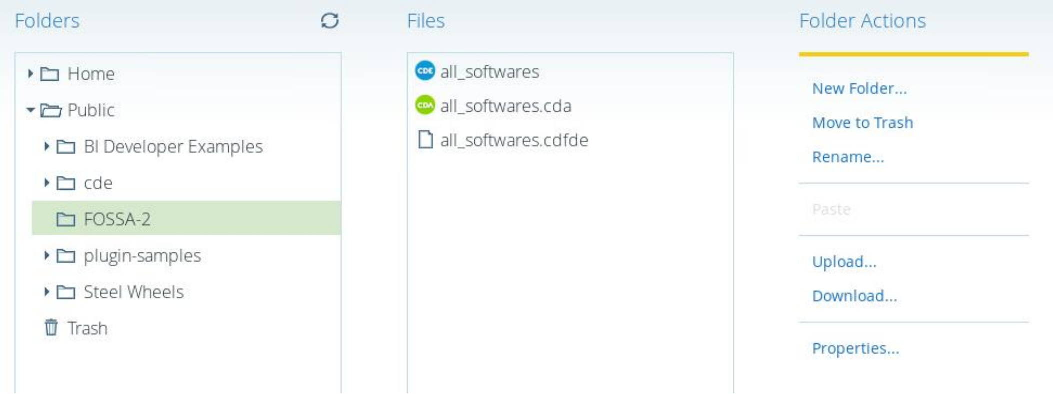
This section gives an overview of the structure of the Pentaho dashboarding feature, to help DIGIT manage and edit the dashboards.

Credentials to connect to the Pentaho User Console (on the docker host, url would be http://localhost:8080/pentaho) are login: “**admin**”; password: “**password**”.

**Figure 1 - Pentaho User Console**

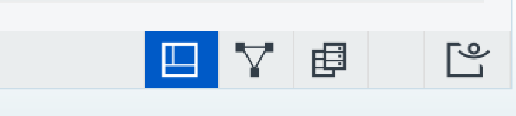


To edit an existing dashboard, click on “Browse files” and go to the folder Public -> FOSSA2:



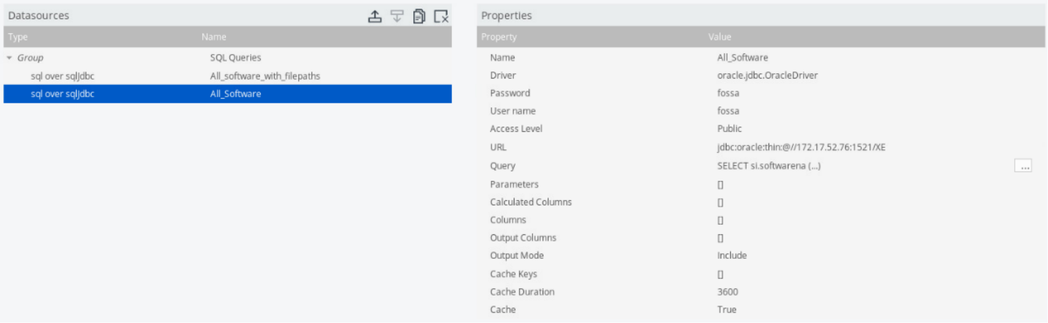
Select the CDE file from the dashboard to edit, and click edit in the right panel. This opens the CDE (Community Dashboard Editor). The editor is split in 3 tabs (on the right hand): layout, components, and datasources.

**Figure 2 - Community Dashboard Editor**



The datasource tab appears as follows:

**Figure 3 - Pentaho CDE Datasource tab**

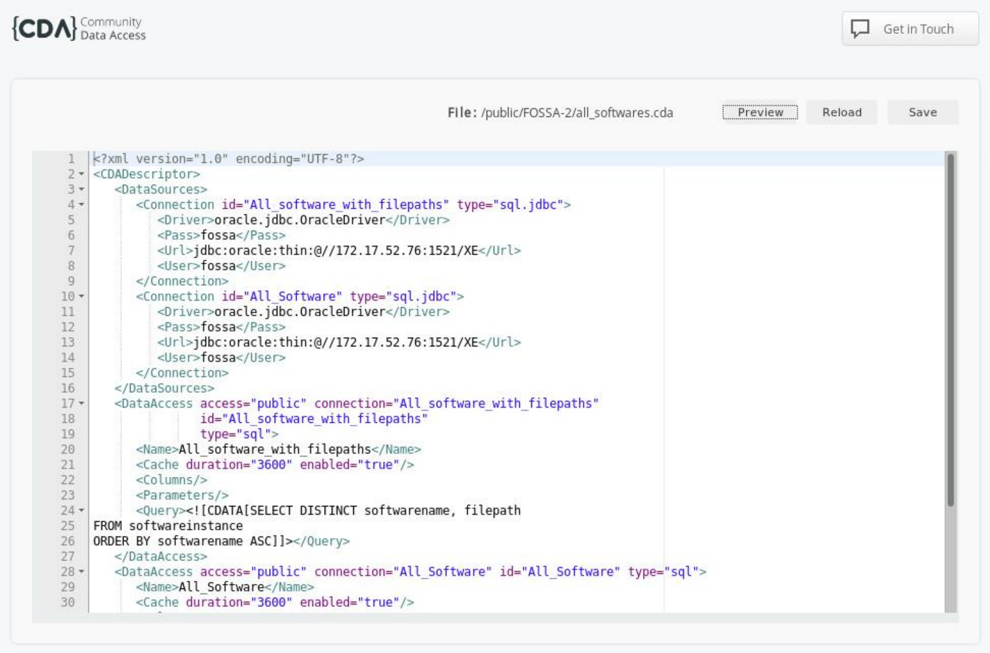


| Type | Name |
|---|---|
| Group | SQL Queries |
| sql over sqlJdbc | All_software_with_filepaths |
| sql over sqlJdbc | All_Software |

| Property | Value |
|---|---|
| Name | All_Software |
| Driver | oracle.jdbc.OracleDriver |
| Password | fossa |
| User name | fossa |
| Access Level | Public |
| URL | jdbc:oracle:thin:@//172.17.52.76:1521/XE |
| Query | SELECT si.softwarena (...) |
| Parameters | [] |
| Calculated Columns | [] |
| Columns | [] |
| Output Columns | [] |
| Output Mode | Include |
| Cache Keys | [] |
| Cache Duration | 3600 |
| Cache | True |

The settings to access the (Oracle) database are shown in the capture above. The “Query” setting handles the SQL request to access the data.

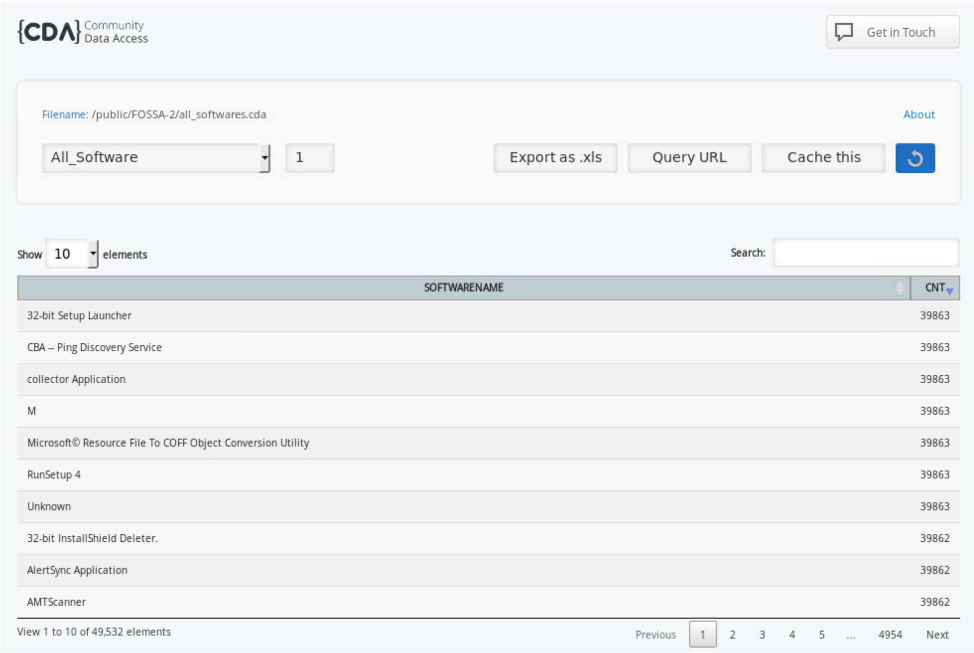
Select the CDA file from the dashboard to edit, and click edit in the right panel. Then click on “Preview”.



**File:** /public/FOSSA-2/all_softwares.cda

```
<?xml version="1.0" encoding="UTF-8"?>
<CDADescriptor>
   <DataSources>
      <Connection id="All_software_with_filepaths" type="sql.jdbc">
         <Driver>oracle.jdbc.OracleDriver</Driver>
         <Pass>fossa</Pass>
         <Url>jdbc:oracle:thin:@//172.17.52.76:1521/XE</Url>
         <User>fossa</User>
      </Connection>
      <Connection id="All_Software" type="sql.jdbc">
         <Driver>oracle.jdbc.OracleDriver</Driver>
         <Pass>fossa</Pass>
         <Url>jdbc:oracle:thin:@//172.17.52.76:1521/XE</Url>
         <User>fossa</User>
      </Connection>
   </DataSources>
   <DataAccess access="public" connection="All_software_with_filepaths"
               id="All_software_with_filepaths"
               type="sql">
      <Name>All_software_with_filepaths</Name>
      <Cache duration="3600" enabled="true"/>
      <Columns/>
      <Parameters/>
      <Query><![CDATA[SELECT DISTINCT softwarename, filepath
FROM softwareinstance
ORDER BY softwarename ASC]]></Query>
   </DataAccess>
   <DataAccess access="public" connection="All_Software" id="All_Software" type="sql">
      <Name>All_Software</Name>
      <Cache duration="3600" enabled="true"/>
```

The final output is the table below with the results of the query:



Filename: /public/FOSSA-2/all_softwares.cda

All_Software 1

Show 10 elements

Search:

| SOFTWARENAME | CNT |
|---|---|
| 32-bit Setup Launcher | 39863 |
| CBA -- Ping Discovery Service | 39863 |
| collector Application | 39863 |
| M | 39863 |
| Microsoft© Resource File To COFF Object Conversion Utility | 39863 |
| RunSetup 4 | 39863 |
| Unknown | 39863 |
| 32-bit InstallShield Deleter. | 39862 |
| AlertSync Application | 39862 |
| AMTScanner | 39862 |

View 1 to 10 of 49,532 elements

## 5. Analysis results of critical software items

- The majority of the critical Open source software items is installed on Datacentre servers (246), while 16 are installed on workstations and only 5 managed by AppV (Firefox, 7-zip, Notepad++, VLC and Java[3].

  The analysis of OSS by software types, covering the inventoried items with the highest number of instances, has shown that the vast majority of OSS belongs to the "Application software / Tool" type (127), followed by the "Library" type (111). No software has been identified as belonging to the types "DatacenterResources" and "MobileSoftware". Regarding the latter, this is due to the fact that mobile applications are out of the scope in the present inventory.

- A further, in-depth analysis has been performed on the critical software shortlist defined on the basis of the Business Criticality Index.

  The first analysis, related to the sustainability of OSS projects evaluated based on 34 metrics, has shown some limits in the actual availability and usability of data related to some of such metrics, particularly concerning the area of performance (e.g. time spent in code reviews). Such lack of information has lowered the average rating of the projects in that area.

  The overall Sustainability Index (SI), calculated as the average of the 34 metrics, shows that the critical shortlist includes software with a wide range of SI values.

- The inventory analysis has also covered one area that may amplify significantly the risks occurring in one of the inventoried OSS components: the dependencies analysis performed on the Critical OSS shortlist.
  The following components have more than 1 dependency upon the shortlisted items (extract from "Annex_5_Top open source software dependencies.xlsx" mentioned in section 3.7):

| Components | Number of dependencies |
|---|---|
| **Python** | 21 dependencies |
| **Firefox** | 20 dependencies |
| **gtk2** | 19 dependencies |
| **XULRUNNER** | 17 dependencies |
| **nfs-utils** | 15 dependencies |
| **initscripts** | 14 dependencies |
| **VLC** | 13 dependencies |
| **rpm** | 12 dependencies |
| **autofs** | 10 dependencies |
| **subscription-manager** | 9 dependencies |
| **Perl** | 2 dependencies |

The analysis shows a relative fragmentation of the dependencies, apart from Glibc and Bash, which relate to the software shown below (extract from "Annex_5_Top open source software dependencies.xlsx" mentioned in section 3.7):

[3] Java is not fully open source software; only source code of some libraries is available, based on a non-OSS compatible license.

| Components | Number of dependant software |
|---|---|
| glibc | 205 |
| bash | 105 |
| zlib | 42 |
| coreutils | 33 |
| libselinux | 32 |
| libX11 | 30 |
| systemd | 28 |
| info | 25 |
| openssl-libs | 20 |

# 6. Appendix – Abbreviations and Acronyms

| | |
|---|---|
| FOSSA | Free and Open Source Software Auditing |
| WP | Work Package |
| DLV | Deliverable |
| csv | Comma-Separated Values (file format) |
| CMDB | Configuration Management Data Base |
| ETL | Extract, Transform, Load |
| OSS | Open Source Software |
| CDE | Community Dashboard Editor |
| SQL | Structured Query Language |