D2.4 - EW-Shopp Platform – v2

Deliverable n: 2.4
Date: 31 December 2019
Status: Final
Version: 1.0
Authors: Nikolay Nikolov (SINTEF), Aljaž Košmerlj (JSI), Lorenzo Sutton (ENG)
Contributors: SINTEF, JSI, ENG, UNIMIB
Reviewers: Matej Zvan (BT), Michele Ciavotta (UNIMIB)

Distribution: Public
Type: OTHER
# History of Changes

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>10.10.2019</td>
<td>Definition of Table of Contents</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.2</td>
<td>21.10.2019</td>
<td>Finalized deliverable structure</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.3</td>
<td>04.12.2019</td>
<td>Initial draft ready for input</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.4</td>
<td>06.12.2019</td>
<td>Added partial content for section 2.2.1</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.41</td>
<td>09.12.2019</td>
<td>Updates to section 2</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.5</td>
<td>10.12.2019</td>
<td>Added implementation details and analytics workflow regarding the Data Analytics tool</td>
<td>Aljaž Košmerlj</td>
</tr>
<tr>
<td>0.6</td>
<td>11.12.2019</td>
<td>Added details for data preparation and enrichment deployments</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.61</td>
<td>12.12.2019</td>
<td>Added details about scalable data preparation and enrichment tool</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.7</td>
<td>13.12.2019</td>
<td>Added information about data visualization tool</td>
<td>Lorenzo Sutton</td>
</tr>
<tr>
<td>0.8</td>
<td>14.12.2019</td>
<td>Version ready for internal review</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>0.81</td>
<td>19.12.2019</td>
<td>Updated sections 2.2 and 3.2 based on review feedback</td>
<td>Lorenzo Sutton</td>
</tr>
<tr>
<td>0.82</td>
<td>20.12.2019</td>
<td>Updated section 2.3 based on review feedback</td>
<td>Aljaž Košmerlj</td>
</tr>
<tr>
<td>0.83</td>
<td>21.12.2019</td>
<td>Added executive summary</td>
<td>Nikolay Nikolov</td>
</tr>
<tr>
<td>1.0</td>
<td>23.12.2019</td>
<td>Updated Partners’ table</td>
<td>Michele Ciavotta</td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable releases the final implementation of the EW-Shopp toolkit after the latest changes in the second part of the project. The toolkit consists of a set of loosely-coupled components that enable end-to-end data preparation and enrichment, analytics and visualization, and its release has replaced, in the EW-Shopp vision, the formerly scheduled release of an integrated platform. This document complements the description of the individual toolkit components (provided in Deliverable D3.5) with the description of support services required to orchestrate the components and the methodology to set up a successful data pipeline at scale. This document focuses on describing the interactions between the tools and provides usage examples in practical scenarios.

The EW-Shopp toolkit consists of three different kinds of tools - the data preparation and enrichment tools, a data visualization tool and the data analytics tools. The key data preparation and enrichment tools consist of Grafterizer 2.0 (part of the DataGraft platform) and ASIA, which are supported by a scalable back-end for executing data preparation and enrichment at scale. The tools can be deployed as a web-based platform or locally within the intranet of an organization or on a single node with different configurations of the microservice architecture implementation using Docker. The scalable data preparation and enrichment back end can be used to implement data preparation and enrichment pipelines at scale by running multiple parallel instances of the pipelines defined in the interactive UI provided by Grafterizer and ASIA. Apart from the data preparation and enrichment pipelines, the scalable back-end implements other pre- and post-processing actions and, via a common interface, it allows the user to compose more complex workflows (whose steps are provided in the form of containers that are also detailed in the deliverable appendix). Results are published as files but can also be uploaded to databases and the tool provides support for both SQL and NoSQL (ArangoDB) database management systems. The data visualization tool is implemented by the Knowage open-source platform. It provides various deployment variants and can be customized to suit user’s needs in different settings. The tool has a community and enterprise edition. The data visualization tool can be used to visualize data that results from both data preparation and analytics tools by using database connectors, API calls or direct file load. The data analytics tools provide support for setting up analytics pipelines that can be used to build predictive models based on sets of features. The tools are implemented by QMiner and provide integration with files, database management systems and APIs. The resulting predictive models can be deployed as web services or as internal applications that process files directly on the host machine.

This deliverable provides also examples of specific relevant use cases for the tools. Firstly, the deliverable shows how to use the scalable data preparation and enrichment back end to set up automatic weather and custom events data download from OpenWeatherMap and an EW-Shopp custom events service and publish it in a MySQL database where it can be consumed by the data visualization tool. Afterwards, the deliverable describes how to use the data visualization tool with the available weather data to display graphical representations of a combination of open data and weather data. Finally, the deliverable demonstrates how we use the analytics tool to produce an analytics model based on the weather data and a dataset with Google AdWords in order to predict the number of impressions of specific keywords.
Table of contents

History of Changes .................................................................................................................. 2
Executive Summary .................................................................................................................. 3
List of figures ............................................................................................................................ 8
Chapter 1  Introduction ............................................................................................................ 11
  1.1  Objectives and Scope .................................................................................................... 11
  1.2  Relationship to Other Deliverables ............................................................................ 11
  1.3  Document Structure .................................................................................................... 12
Chapter 2  EW-Shopp Toolkit Final Implementation ............................................................. 13
  2.1  Data Preparation and Enrichment ................................................................................. 14
  2.1.1  Online Deployment of Data Preparation and Enrichment Tool ......................... 15
  2.1.2  Local Deployment of Data Preparation and Enrichment Tool ......................... 18
  2.1.3  Scalable Data Preparation and Enrichment Back-end ........................................ 19
  2.2  Data Visualisation ...................................................................................................... 23
  2.3  Data Analytics ............................................................................................................. 24
     2.3.1  Pipeline Overview ............................................................................................... 25
Chapter 3  Using the EW-Shopp Toolkit ............................................................................... 28
  3.1  Scalable Data Preparation and Enrichment – the Scenarios of Obtaining Weather Data from OpenWeatherMap and from EW-Shopp Custom Events API ........................................ 28
     3.1.1  Creating the Weather Data Database Schema in MySQL .................................. 29
     3.1.2  Docker Container Set Up .................................................................................... 29
     3.1.3  Rancher Set Up .................................................................................................. 30
  3.2  Setting Up Visualization: an Open Data + Weather Use Case .................................. 32
     3.2.1  Creation of the Data Source and Data Sets in Knowage .................................. 35
     3.2.2  Creation of the Visualization Cockpit and Widgets ......................................... 39
  3.3  Setting Up Analytics .................................................................................................. 53
Appendix A  EW-Shopp Workflow Composable Steps Library ........................................... 59
  A.1  Introduction ................................................................................................................ 59
     A.1.1.  Common ............................................................................................................. 60
     A.1.2.  Common_py ...................................................................................................... 60
     A.1.3.  Building a Container ......................................................................................... 60
  A.2  00_fetch_weather_ecmwf ......................................................................................... 61
A.2.1. Volumes ........................................................................................................61
A.2.2. Arguments .....................................................................................................61
A.2.3. Source ............................................................................................................62
A.3. 01_uploadtoarango ...........................................................................................63
A.3.1. Volumes ........................................................................................................63
A.3.2. Arguments .....................................................................................................63
A.3.3. Source ............................................................................................................63
A.4. 04_extract_weather_ecmwf .............................................................................63
A.4.1. Example of Parameter Settings: .................................................................64
A.4.2. Volumes ........................................................................................................64
A.4.3. Arguments .....................................................................................................64
A.4.4. Source ............................................................................................................65
A.5. 05_uploadtoarango_dual ..................................................................................65
A.5.1. Volumes ........................................................................................................65
A.5.2. Arguments .....................................................................................................66
A.5.3. Source ............................................................................................................66
A.6. 06_fetch_weather_owm ...................................................................................66
A.6.1. Volumes ........................................................................................................66
A.6.2. Arguments .....................................................................................................67
A.6.3. Source ............................................................................................................67
A.6.4. Rancher Setup Example: .............................................................................68
A.7. 07_uploadtomysql.............................................................................................69
A.7.1. Volumes ........................................................................................................69
A.7.2. Arguments .....................................................................................................69
A.7.3. Source ............................................................................................................70
A.7.4. Rancher Setup Example: .............................................................................70
A.8. 10_unzip ...........................................................................................................71
A.8.1. Volumes ........................................................................................................71
A.8.2. Arguments .....................................................................................................71
A.8.3. Source ............................................................................................................71
A.8.4. Rancher Setup Example: .............................................................................71
A.9. 11_tsv2csv .......................................................................................................73
A.9.1. Volumes ........................................................................................................73
A.9.2. Arguments ........................................................................................................... 73
A.9.3. Source...................................................................................................................... 73
A.9.4. Rancher Setup Example: ...................................................................................... 73
A.10. 12_split .................................................................................................................... 75
A.10.1. Volumes ................................................................................................................. 75
A.10.2. Arguments ............................................................................................................. 75
A.10.3. Source..................................................................................................................... 75
A.11. 13_transform .......................................................................................................... 76
A.11.1. Volumes ................................................................................................................. 76
A.11.2. Arguments ............................................................................................................. 76
A.11.3. Source..................................................................................................................... 76
A.11.4. Rancher Setup Example: ...................................................................................... 77
A.12. 14_toarango .......................................................................................................... 78
A.12.1. Volumes ................................................................................................................. 78
A.12.2. Arguments ............................................................................................................. 78
A.12.3. Source..................................................................................................................... 78
A.13. 15_jot_categorize .................................................................................................. 78
A.13.1. Volumes ................................................................................................................. 79
A.13.2. Arguments ............................................................................................................. 79
A.13.3. Source..................................................................................................................... 80
A.13.4. Rancher Setup Example: ...................................................................................... 80
A.14. 16_jot_aggregate_kwds ......................................................................................... 81
A.14.1. Volumes ................................................................................................................. 81
A.14.2. Arguments ............................................................................................................. 81
A.14.3. Source..................................................................................................................... 81
A.14.4. Rancher Setup Example: ...................................................................................... 81
A.15. 17_jot_closest_kwds ............................................................................................. 82
A.15.1. Volumes ................................................................................................................. 82
A.15.2. Arguments ............................................................................................................. 82
A.15.3. Source..................................................................................................................... 83
A.15.4. Rancher Setup Example: ...................................................................................... 83
A.16. 18_fetch_custom_events ......................................................................................... 84
A.16.1. Volumes ................................................................................................................. 84
A.16.2. Arguments ............................................................... 85
A.16.3. Source ................................................................. 85
List of figures

Figure 1. High-level overview of EW-Shopp toolkit tools ................................................................. 13
Figure 2. Design and run-time separation of concerns in the EW-Shopp Data Preparation and Enrichment tools... 15
Figure 3. Online deployment of data preparation and enrichment tools (design time) ............................... 17
Figure 4. Minimal deployment of data preparation and enrichment tools (design time) ............................. 18
Figure 5. Back-end for running scalable data preparation pipelines (a) on a single host and (b) on multiple hosts .. 20
Figure 6. UI for creating stacks based on service and scaling configuration YAML files ............................ 21
Figure 7. Folder structure mapping for multi-step data workflows ...................................................... 22
Figure 8. General structure of a workflow step procedure .................................................................... 22
Figure 9. Overview of the workflow and structure of the data analytics' pipeline ........................................ 27
Figure 10. Rancher CRON scheduler in the Rancher service catalog ..................................................... 30
Figure 11. Creating a weather ingestion stack in the Rancher UI .......................................................... 31
Figure 12. Open Data and Weather visualisation high level logic ............................................................. 33
Figure 13. Creation of the MySQL data source connecting to the Scalable data preparation and enrichment back-end database ........................................................................................................ 35
Figure 14. Creation of the data set made of a parametrised SQL query to fetch the weather + download data..... 36
Figure 15. Preview of the data set columns ............................................................................................ 36
Figure 16. Creation of a Knownage cockpit for the data visualisation .................................................... 39
Figure 17. Adding the needed data sets to the cockpit ............................................................................ 39
Figure 18. Set-up of the data sets with default parameters ....................................................................... 40
Figure 19. Creating a row association for the date between the two datasets .......................................... 40
Figure 20. Cockpit with WYSIWYG elements (image, text, HTML widget) ............................................. 41
Figure 21. Sheets added to the cockpit ................................................................................................. 41
Figure 22. Addition of the data set to the single chart widget ............................................................... 41
Figure 23. Chart type selection ............................................................................................................. 42
Figure 24. Set-up of the chart structure .................................................................................................. 42
Figure 25. Fine-tuning visual/presentation elements of the chart ............................................................ 43
Figure 26. Chart configuration: legend ................................................................................................ 44
Figure 27. Chart configuration: custom colours ..................................................................................... 45
Figure 28. Final cockpit screenshot: single chart with all data with the different units ............................ 45
Figure 29. Saving the cockpit ............................................................................................................... 46
Figure 30. Adding the data set with the normalised data ......................................................................... 46
Figure 31. Set-up of the chart ............................................................................................................... 48
Figure 32. Normalised chart configuration: custom colours ..................................................................... 48
Figure 33. Screenshot of the chart with normalised data ......................................................................... 48
Figure 34. Zoom to a specific time range in the normalised chart ........................................................... 49
Figure 35. Set-up of the sorted chart ..................................................................................................... 50
Figure 36. Screenshot of the charts with sorted data .............................................................................. 50
Figure 37. Selection of the cross table widget ....................................................................................... 51
Figure 38. Set-up of rows and columns in the cross table widget ............................................................ 51
Figure 39. Set-up of visuals in the cross table .......................................................................................... 52
Figure 40. Screenshot of the cross table (above) and sorted by temperature (below) ............................... 53
Figure 41. The analytics workflow from data ingestion to model building ............................................ 54
Short references may be used to refer to project beneficiaries, also frequently referred to as partners. References are listed in Table 1.

Table 1. Short references for project partners

<table>
<thead>
<tr>
<th>No.</th>
<th>Beneficiary (partner) name as in [GA]</th>
<th>Short reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UNIVERSITA’ DEGLI STUDI DI MILANO-BICOCCA</td>
<td>UNIMIB</td>
</tr>
<tr>
<td>2</td>
<td>CENEJE DRUZBA ZA TRGOVINO IN POSLOVNO SVETOVANJE DOO</td>
<td>CE</td>
</tr>
<tr>
<td>3</td>
<td>BROWSETEL (UK) LIMITED</td>
<td>BT</td>
</tr>
<tr>
<td>4</td>
<td>GFK EURISKO SRL.</td>
<td>GFK</td>
</tr>
<tr>
<td>5</td>
<td>BIG BANG, TRGOVINA IN STORITVE, DOO</td>
<td>BB</td>
</tr>
<tr>
<td>6</td>
<td>MEASURENCE LIMITED</td>
<td>ME</td>
</tr>
<tr>
<td>7</td>
<td>JOT INTERNET MEDIA ESPAÑA SL</td>
<td>JOT</td>
</tr>
<tr>
<td>8</td>
<td>ENGINEERING – INGEGNERIA INFORMATICA SPA</td>
<td>ENG</td>
</tr>
<tr>
<td>9</td>
<td>STIFTELEN SINTEF</td>
<td>SINTEF</td>
</tr>
<tr>
<td>10</td>
<td>INSTITUT JOZEF STEFAN</td>
<td>JSI</td>
</tr>
<tr>
<td>11</td>
<td>GfK Italy</td>
<td>GFK(^1)</td>
</tr>
<tr>
<td>12</td>
<td>SINTEF AS</td>
<td>SINTEF(^2)</td>
</tr>
</tbody>
</table>

\(^1\) As a result of change in the GFK EURISKO SRL, the partner GFK Italy has implemented the activities assigned to GFK EURISKO SRL since 1/7/2018. This has been agreed between the EW-Shopp consortium and the EC, leading to an amendment of the Grant Agreement. Since the actions of the two entities are contiguous and not overlapping we use the same acronym GFK for both the entities.

\(^2\) SINTEF AS has partially taken over STIFTELEN SINTEF activities, and in particular, it took over the activities since 01/01/2018, as agreed between the EW-Shopp consortium and the EC leading to an amendment of the Grant Agreement. Since the actions of the two entities are contiguous and not overlapping we use the same acronym SINTEF for both the entities.
Chapter 1  Introduction

This deliverable describes the final release of the EW-Shopp toolkit (previously ‘Platform’) that integrates the core EW-Shopp services, hosting and support services, and is grounded on the feedback and recommendations from business case evaluations. Initially, the Description of Action (DoA) envisioned this deliverable as accompanying a platform that closely integrates the various tools. The concept has since been adjusted by implementing a lighter and more agile integration, and (meeting the reviewers’ recommendation at M18) the platform has been set up as a toolkit, whereby the various elements can communicate by different interfaces. This approach allows to pick the best tool to perform a certain task and this should lead to a better adoption of the toolkit. This document presents the implementation of the main elements of the platform presented in D3.1 but omits advanced integration features.

1.1 Objectives and Scope

The objective of the deliverable is to describe the implementation of the technical infrastructure, methods, tools and services that comprise the EW-Shopp toolkit. The content puts particular focus on the aspects of the usage of the different tools in different contexts.

1.2 Relationship to Other Deliverables

The deliverable serves as a complement to a number of other deliverables in describing the technical implementation of the EW-Shopp toolkit.

Deliverable "D1.4 Event, weather and multilingual data services" describes the enrichment data that is used to enrich input business data through the data preparation and enrichment tool described in section 2.1. The scalable data preparation and enrichment back-end allows for both scaling up the process of enrichment with that data, as well as a platform to continuously update the enrichment source data itself.

Deliverables "D2.3 EW-Shopp Platform evaluation assessment" and "D3.4 EW-Shopp components evaluation assessment" provide technical assessment of the different components of the toolkit with respect to the business cases of EW-Shopp, provide recommendations in terms of functionalities, and outline the limitations of the toolkit tools.

Deliverable "D3.3 EW-Shopp components as a service: data visualization, navigation and quality assessment" provides details on the implementation of the visualization and navigation tools, as well as the data quality assessment services provided by the toolkit (especially the data preparation and enrichment tool).
Deliverable "D3.5 EW-Shopp components as a service: final release" describes the final release of the individual reconciliation, transformation, analytic and visualization tools from an architectural and functional standpoint (including updates to the different features). In this document, the focus is on the orchestration of the services into data pipelines and their deploy on a scalable data infrastructure, including technical details to set up the tools.

**1.3 Document Structure**

The rest of the document has been divided as follows:

- Chapter 2 provides an overview of the EW-Shopp toolkit and describes how it can be deployed in different configurations
- Chapter 3 provides examples of the use of the EW-Shopp toolkit
- Appendix A provides a detailed description in terms of functionalities, inputs and outputs of the EW-Shopp workflow composable steps library that can be used to implement complex data pipelines.
Chapter 2  EW-Shopp Toolkit Final Implementation

EW-Shopp toolkit consists of three sets of tools a) the data preparation and enrichment tools, b) the data visualization tools and c) data analytics tools. The tools provide support for the different aspects of the dataflow model as shown in Figure 1.

The toolkit is organized in a modular way that allows for the flexible usage of the tools in different scenarios. The main output for the different tools is marked in green and the user input - in blue in Figure 1. The data preparation and enrichment tools are implemented by the Grafterizer tool and ASIA service and is used to processes input data and enrich it with data coming from different data sources. The tools' implementation provides explicit support for storing prepared and enriched data in the file system, a SQL, or a NoSQL (ArangoDB) database management system, where it can be consumed by other services.

The tools implement design/run-time separation of concerns, whereby the data preparation and enrichment configuration are set by the user at small scale (on sample data) and, if necessary, applied to the full dataset using a scalable back-end. For this to work properly, users have to provide as input a configuration file describing how the data needs to be prepared and, if relevant for the use case, deploy additional enrichment data from external sources (ECMWF, OpenWeatherMap,
GeoNames, Wikifier, etc.) or internal (custom) events that are defined in accordance with the model described in deliverable D1.4 and the API available on SwaggerHub\(^3\).

The **data analytics tools** are implemented by QMiner and can be used to set up analytics models. Data for the models are ingested by a data loader and can retrieved from the file system, SQL, or NoSQL database management system, or a web service. The user provides configuration parameters for the data loading and the machine learning model to be used to process the input. Once data are processed, the model can be deployed as a standalone application or web service (also can be done through simple configuration file specification) and used to produce predictions for a set of input features.

The **data visualization tools** are implemented by the Knowage software platform and, as part of the EW-Shopp toolkit, is used to visualize and explore data, which can be both a result of the preparation and enrichment process, and from the analytical models that are produced by the data analytics tools. The visualization inputs are configured by defining data sources and data sets in the Knowage user interface and this data is then fed to the visualization cockpits and widgets that are configured by the user of the tool. Inputs can be configured to consume data directly from databases (both supporting SQL and NoSQL solutions), through upload of tabular files, and through calling REST APIs and visualizing the results (the tool also supports other file formats and communication protocols).

The rest of the chapter provides details on the final implementation of the EW-Shopp toolkit. It describes how users of the toolkit can use the provided toolkit tools - specifically, which parts of the architecture need to be in place in order to take advantage of their functionality and how they can be applied in practical settings.

### 2.1 Data Preparation and Enrichment

The EW-Shopp Data Preparation and Enrichment tools are composed of the Grafterizer 2.0 web-based tool (part of the DataGraft platform\(^4\)), along with the ASIA and ABSTAT services. The tools are implemented through a microservice architecture, whereby each of the sub-components is encapsulated in a Docker\(^5\) container. This allows for the flexible deployment of different parts of the data preparation and enrichment system both locally and in production settings. Furthermore, in order to support scalability of the data preparation and enrichment processes (all loaded data in Grafterizer is kept in the browser and it uses a client-server interaction to produce intermediate results), it implements the separation of design- and run-time aspects as shown in Figure 2. Thereby, at design time, a data scientist may use sample data to describe the clean-up, enrichment, transformation and export format of the data preparation and enrichment pipeline to be implemented, based on which a set of scripts are automatically generated. These scripts can then be used by a data engineer to deploy the transformation on distributed infrastructure and execute the

---

\(^3\) [https://app.swaggerhub.com/apis/EW-Shopp/EW-Shopp_Event_API/2.2.0](https://app.swaggerhub.com/apis/EW-Shopp/EW-Shopp_Event_API/2.2.0)

\(^4\) [https://datagraft.io/](https://datagraft.io/)

\(^5\) [https://www.docker.com/](https://www.docker.com/)
pipeline at scale against the full dataset as a batch process or a service. This scalable infrastructure is referred to as the scalable data preparation and enrichment back-end.

When large-scale data is not involved, it is possible to use only the design-time aspects of the tools, which are deployed as part of the DataGraft platform.

Figure 2. Design and run-time separation of concerns in the EW-Shopp Data Preparation and Enrichment tools

The rest of the section discusses the modes of deployment of the data preparation and enrichment tools as well as their scalable back-end.

### 2.1.1 Online Deployment of Data Preparation and Enrichment Tool

The online deployment of the data preparation and enrichment tools have been designed to scale with the number of users of the different aspects of the data preparation and enrichment process. The microservice components and the dependencies of the online ("production") deployment of the EW-Shopp data preparation and enrichment tool is shown in Figure 3 (note: DataGraft is an existing platform and newly created or updated microservices are marked in green). The microservice containers that comprise the preparation and enrichment stack are as follows:
• **DataGraft portal**: Contains the implementation for the graphical user interface (views) and corresponding user services (user interface controllers and models) that comprises the portal component of the DataGraft Platform.

• **Grafterizer**: Contains the implementation for the Grafterizer web-based graphical user interface for defining data preparation and enrichment pipelines.

• **Grafterizer Dispatch Service**: Contains the implementation for the dispatch service that handles request authentication with the DataGraft platform on behalf of Grafterizer and dispatches requests for input and output across the multiple services.

• **Jarfter**: Contains the implementation for the Jarfter web service that generates executable Java Archives (JARs) for running Grafterizer transformation and enrichment. JARs all contain the ASIA client library, which is used to connect to a pre-specified ASIA service for enrichment.

• **Vocabulary Manager**: Contains the implementation for the RDF vocabulary manager used by Grafterizer.

• **Graftwerk**: Contains the Clojure implementation of Graftwerk, which is a RESTful execution service for running Grafter-based data transformation pipelines that are defined using Grafterizer.

• **Graftwerk Load Balancer**: Contains the implementation of a load balancer service that implements the Graftwerk API and allows horizontal scaling.

• **Graftwerk Cache**: Contains the implementation for a service that caches Grafterizer transformation results obtained from Graftwerk.

• **ASIA backend**: is the main access point to reconciliation and enrichment functionalities. It mainly works as an API gateway intercepting and forwarding requests from the ASIA graphical interface implemented in Grafterizer. Moreover, it implements directly the weather and events enrichment functionalities interacting with ArangoDB database.

• **ASIA suggester**: this microservice is in charge of providing cross-lingual annotation suggestion to the user (to perform the schema-level linking activity) by interacting with remote translation services (Azure Translator Text API), vocabulary services (LOV), and RDF summary services (ABSTAT).

• **Conciliator**: implements several reconciliation and extension functionalities and exposes them via the OpenRefine API. In particular, it interacts with ArangoDB, Virtuoso, ElasticSearch, and remote services (ex. Wikifier) to provide support for reconciliation using GeoNames, Google GeoTargets, Wikidata, Wikifier. Using a service (called Keyword2Category) developed by JSI and described in detail in D3.5, conciliator can also provide a semantic matching functionality that maps keywords to Google categories. Finally, conciliator provides other reconciliation functionalities that pre-dates the project (VIAF, ORCID, Open Library, Solr).

---

6 [https://github.com/datagraft/datagraft-portal](https://github.com/datagraft/datagraft-portal)  
7 [https://github.com/datagraft/grafterizer](https://github.com/datagraft/grafterizer)  
8 [https://github.com/datagraft/grafterizer-dispatch-service](https://github.com/datagraft/grafterizer-dispatch-service)  
9 [https://github.com/datagraft/jarfter](https://github.com/datagraft/jarfter)  
10 [https://github.com/datagraft/vocabulary-manager](https://github.com/datagraft/vocabulary-manager)  
11 [https://github.com/datagraft/graftwerk](https://github.com/datagraft/graftwerk)  
13 [https://github.com/datagraft/graftwerk-cache](https://github.com/datagraft/graftwerk-cache)  
14 [https://github.com/UNIMIBInside/asia-backend](https://github.com/UNIMIBInside/asia-backend)  
15 [https://github.com/UNIMIBInside/ASIA-suggester](https://github.com/UNIMIBInside/ASIA-suggester)  
16 [https://github.com/UNIMIBInside/conciliator](https://github.com/UNIMIBInside/conciliator)
The DataGraft platform is hosted using resources and functionalities provided by Amazon Web Services (AWS) and available at [https://datagraft.io/](https://datagraft.io/). The service is roughly divided into three deployable partitions - the Grafterizer/DataGraft stack (DataGraft platform, Grafterizer, Grafterizer back-end services), the ASIA stack (ASIA backend API, Suggester, Conciliator and enrichment databases) and the Graftwerk load-balanced scaling group. The division is made to enable independent scaling of the different groups of services, which can be used during high loads. The DataGraft/Grafterizer and ASIA stacks are deployed using Amazon Elastic Container Service (ECS), which allows for automatically deploying Docker-based services, maintaining operational capacity and scaling up sets of related Docker services. The Grafterizer/DataGraft stack contains the user interface to the DataGraft asset management service and the data transformation script generation as well as supporting services related to their operation (caching of data transformation results, security, management of vocabularies for data modelling and generation of executable JAR files). Data about users and their assets are stored in a relational database, which is hosted in Amazon Relational Database Service (RDS). Files used in the Grafterizer UI (and managed by the DataGraft asset management system) are stored in block storage provided by Amazon Simple Storage Service (S3). The ASIA stack consists of a set of web services and databases to support data enrichment that are scalable independently from the DataGraft/Grafterizer stack to enable concurrent usage at different load rates (as the UI/supporting services receive less load than the ASIA stack). Finally, the Graftwerk service, which executes live the data transformation scripts provided by Grafterizer is hosted using Amazon Elastic Compute Cloud (EC2) scaling group (associated through the scaling group APIs) so that it can be scaled independently from the UI in case of large loads caused by concurrent usage of the data preparation capabilities of Grafterizer. Apart from the aforementioned
Amazon services, the deployment of the data preparation and enrichment tools uses Amazon Simple Email Service\(^2\) (SES) (for e-mails confirming user registration or resetting of passwords), Amazon Route 53\(^2\) for domain-name resolution and Amazon CloudFront\(^2\) for content delivery as well as SSL certificate application (certificate provided also by the AWS Certificate Manager\(^2\)). Finally, ASIA interacts with a remote translation service (Azure Translator Text API), vocabulary services (LOV), and with ABSTAT (an RDF summary services). Note that although ABSTAT is largely used by ASIA for the schema-level annotation functionality, it is not depicted in Figure 3 because it is a remote service running on UNIMIB’s premises.

### 2.1.2 Local Deployment of Data Preparation and Enrichment Tool

The microservice architecture of the data preparation and enrichment tool allows also for local deployments. This can be useful when there is a need to deploy the infrastructure privately, or when developing some of the open-source components of the tool. Full documentation on the deployment configuration and installation are found at [https://github.com/datagraft/datagraft-platform](https://github.com/datagraft/datagraft-platform). The repository contains a single Docker Compose\(^2\) file that allows for the simultaneous deployment of all the services necessary for the operation of the data preparation and enrichment tool. The set of microservices involved are shown in Figure 4.

![Figure 4. Minimal deployment of data preparation and enrichment tools (design time)](image)

All services that comprise the minimal deployment are the same as the ones described in Section 2.1.1, except for the absence of the Graftwerk Load Balancer, which is not necessary for the minimal deployment of the platform. Furthermore, this deployment uses a user-provided (possibly by

---

\(^2\) [https://aws.amazon.com/ses/](https://aws.amazon.com/ses/)
\(^2\) [https://aws.amazon.com/route53/](https://aws.amazon.com/route53/)
\(^2\) [https://aws.amazon.com/cloudfront/](https://aws.amazon.com/cloudfront/)
\(^2\) [https://aws.amazon.com/certificate-manager/](https://aws.amazon.com/certificate-manager/)
\(^2\) [https://docs.docker.com/compose/](https://docs.docker.com/compose/)
Docker) PostgreSQL database and file storage can be done either using a service that implements the Amazon S3 API (e.g., Minio\textsuperscript{26}), or the local file system of the container (could be set up as a mount from the host of the services stack).

### 2.1.3 Scalable Data Preparation and Enrichment Back-end

The scalable data preparation and enrichment back-end can be used when there is a need to deploy pipelines at scale - i.e., when the UI of the tool cannot support processing the entire input, or when there are multiple inputs. The back-end relies on parallelisation of the data preparation and enrichment scripts done during the design phase (see Figure 2) and can be applied when there are more than one input file. In case there is one large file, it can either be split in chunks (using a specific component), or processed using a single JAR executable (generated automatically by the data preparation and enrichment tool), in which case the use of the back end is not necessary. The back end essentially parallelizes the execution of the JAR executable produced by the front end, which include the data clean up, re-formatting and the enrichment phases. As a matter of fact, the UI of the Grafterizer tool enables users, based on setting up configuration preferences, to generate the deployment scripts (packed in an executable JAR) for the data preparation and enrichment at scale. The configuration requires users to choose the address ASIA instance that will be used for reconciliation and enrichment (if it is used at any point of the pipeline), the folder where the inputs will be provided, a folder where the scaled pipeline can store intermediate results, a folder to provide the outputs, as well as the number of parallel instances of the data preparation and enrichment image need to be deployed. The generator then creates a service description in YAML form (can be used as a Docker Compose file) as well as a YAML file that can be used in Rancher to determine the desired scaling configuration. In case ASIA is used at scale, users need to deploy the ASIA backend and services, which can be done as a single instance or a set of load-balanced instances (as described in D3.4).

Apart from the data preparation and enrichment, the use of the back-end enables users to compose more complex pipelines that include pre- and post-processing steps or deploy the pipelines as services that are triggered by the input of additional data. In order to deliver inputs, the back-end uses file system operations and coordinates data splitting between instances of the scaled transformation and between steps (if more than one). Finally, the back end can be used for other purposes than data preparation and enrichment – e.g., automating the process of fetching weather data from ECMWF/OWM, events from the Event Registry and partners’ custom events.

As part of the EW-Shopp toolkit, we have developed a set of sample functions that can be composed into multi-step data workflows\textsuperscript{27}. Furthermore, we provide extendable templates for users of the toolkit to extend with custom functionality that is not covered by the library of samples. A full description of the samples along with set up documentation is available in each sample repository on GitHub and in Appendix A.

\textsuperscript{26} https://min.io/
\textsuperscript{27} https://github.com/ew-shopp/ingestion_lib
The scalable back-end relies on Docker, Rancher and, in the case of distributed deployments with more than one host, a shared (e.g., networked or distributed) file system available for each node participating in the distribution of the data preparation and enrichment (see Figure 5). In single-node deployments it is enough to use the local file system of the host to enable the scaling of the service.

Figure 5. Back-end for running scalable data preparation pipelines (a) on a single host and (b) on multiple hosts

Once the base is in place, a scalable version of the data preparation and enrichment executable JAR file can be deployed using a Rancher stack. This is done by uploading the service description and scaling configuration YAML files in the Rancher UI for creating new stacks.
A multi-step processing workflow (e.g., with two steps, 1 - split a large file in chunks and 2 - process using the data preparation and enrichment executable JAR) can be set up by creating a set of containers that communicate through the shared file system. The file system mounts for each container are attached in a way that allows for the outputs of one step to be passed to steps that follow. Specifically following the structure in Figure 7:

- **input folder**: input data for the entire workflow; input for the first step
- Each step has a **work folder** for intermediate results (if necessary for the specific step template - see Appendix A)
- Each step has an **out folder** that holds the results of the process
- Each step (except 1st) uses the previous step’s out folder as its input folder
- The last step's out folder provides the output of the workflow (if necessary - in case the last step uploads data to a database, this is not necessary).
Once the structure is created and the Rancher stack is created, the workflow is triggered and automatically processes all of the inputted files (or other actions depending on the chosen template).

The workflow steps’ structure is depicted in Figure 8. Steps consist of pre-defined code for input and output processing as well as a workflow step action. In the case of the data preparation and enrichment, the action is a call to the executable JAR file (produced by Grafterizer/ASIA) with one of the input files as parameter. The composable step templates provided by the EW-Shopp toolkit are organised as homogeneous Docker images and stored in DockerHub\(^\text{28}\) so that they can be downloaded and instantiated on the target infrastructure.

Workflow steps rely on a set of Bash scripts for processing inputs and outputs, which implement the communication of data between workflow steps. The scripts synchronise the processing of inputs and outputs so that two parallel steps do not attempt to process the same file at one time and the

\(^{28}\)https://hub.docker.com/u/datagraft/
outputs are not obtained for processing before they are fully stored in the output folder of the step (which is also the input folder for the next step). Additionally, depending on the step action, the instantiation of a step requires a set of values for specialised parameters, which provide required configuration for the step (e.g., start/end dates for which weather data should be acquired from ECMWF - see A.4). Finally, in the case of multi-step workflows, the file system mounts need to be explicitly added as parameters to the configuration YAML file so that the input and output folders are correctly associated for consecutive steps.

### 2.2 Data Visualisation

Data Visualisation and Navigation in EW-Shopp is offered mainly through the Open Source Knowage platform, customized for the user needs and scenarios carried out in the EW-Shopp pilots as well as other similar data visualization cases, in particular ones where business data (indicators) are coupled to weather/events data, and more typically along time series of varying time granularity. As of Month 36, EW-Shopp is using version 7.0 (released October 2019) of the Community Edition, i.e. the fully Open Source version. Knowage 7.0 CE is available through various channels on the Internet including the download page on the official website: [https://www.knowagesuite.com/site/knowage-download/](https://www.knowagesuite.com/site/knowage-download/), GitLab: [https://github.com/KnowageLabs](https://github.com/KnowageLabs), and the OW2 Open Source repository: [https://projects.ow2.org/view/knowage/](https://projects.ow2.org/view/knowage/). The GitHub channel offers also a Docker image.

A more detailed description of Knowage and its integration within EW-Shopp is provided in deliverables D2.3 in the context of Business Cases and in D3.5 as part of the final release of the EW-Shopp toolkit. Here we provide the main information related to the tool and how to set-it up. Additionally, in section 3.2, we provide a more hands-on example on how to test Knowage with weather and open data.

For setting up an instance such as the one used for EW-Shopp (CE edition) the following system requirements are recommended:

- Linux machine (Windows also supported but untested in EW-Shopp)
- At least 3 GB of dedicated RAM
- 2 GB free disk space
- A Java SE Development Kit (JDK)

Root privileges are recommended to carry out the installation, in particular to administer the connected database Knowage creates to store internal data when installed. For further details we refer to the official documentation\(^\text{29}\). In EW-Shopp some branding (through customised images) was done. This can be done by starting from the default theme and modifying it. Starting from the installation root this is located in:


---

Here a set of subdirectories contain images, CSS sheets, etc. If working with the default theme it is recommended to back-up the original assets in case the final result is not the desired one. It should be noted that modifying any of the webapps assets has immediate effect (i.e. no server restart is needed).

Knowage comes with predefined administrator and user-type users. If the server is exposed on the internet it is recommended to change the default password for these users as well as creating a dedicated admin user for daily usage (in our case, we called it ‘ew-shopp_admin’).

2.3 Data Analytics

The analytic tools are built on the base of QMiner, a platform for real-time analysis of large-scale streams of structured and unstructured data. The core modules of QMiner are implemented in C++ to take advantage of the low-level linear algebra libraries (OpenBLAS, Intel MKL) for efficiency. For greater user friendliness and to enable fast prototyping the core functionality is wrapped in JavaScript and exposed as a Node.js package. This also supports simple deployment as the QMiner package can be simply installed through the Node Package Manager (npm) using the command:

```
npm install qminer
```

To enable loading data directly from the ArangoDB (the EW-Shopp main datastore) and MySQL databases using the data loading code the corresponding Node.js packages must be installed as well:

```
npm install arangojs mysql
```

Finally, if the light-weight REST server is used for deployment of the model the Express web framework needs to be installed:

```
npm install express
```

The code of the analytics tools is available at the EW-Shopp GitHub repository: [https://github.com/ew-shopp/analytics](https://github.com/ew-shopp/analytics). Once the packages listed above are installed and the repository is cloned into the local system, the analytic tools are ready to use.

To run analytic tools, the following system requirements are recommended:

- Ubuntu 18.04.1 LTS or Windows 10 machine (should work on other Linux distributions as well, but it hasn’t been tested).
- At least 4 GB of RAM (preferably more). If you use keyword clustering tool described in section 4.1 in D3.5, at least 16 GB of RAM is recommended.

---

30 [https://nodejs.org/](https://nodejs.org/)
31 [https://www.npmjs.com/](https://www.npmjs.com/)
32 [https://expressjs.com/](https://expressjs.com/)
• Approximately 500MB free disk space for the repository code and all installed dependencies. This does not include disk space needed for configurations and data to be processed.

The analytic tools are not packed as Docker container. In case of the need for a Docker container, it is best to start with QMiner Docker image\(^{33}\) and clone analytics tool repository into the container. On the QMiner Docker image, all the QMiner dependencies are preinstalled. However, make sure to install Node.js version 10.x.x, Python 3.x and all the packages listed above through the Node Package Manager. Bear in mind, that QMiner is not fully supported for Node.js versions above 10.x.x.

The analytic tools are easily connected to the complete EW-Shopp pipeline. With the **Loader** functionality, a connection can be made to the same ArangoDB and/or different MySQL sources where the results of data enrichments are stored. The simplicity of connecting internal data analytic pipeline to an external data source using the loader configuration file can be seen below, where the source is set to ArangoDB and destination is set to QMiner database. In the queries list, the simple queries must be provided with the defined source’s query language (in our example in the ArangoDB Query Language (AQL)\(^{34}\)).

```json
{
    "source": {
        "type": "ArangoDB",
        "host": "http://127.0.0.1:8529",
        "database": "db_name",
        "user": "username",
        "password": "pass"
    },
    "destination": {
        "type": "QminerDB",
        "db_path": ".\dbExample\",
        "mode": "open"
    },
    "queries": [
        {
            "name": "SloveniaWeather",
            "query_src": {
                "query": "FOR forecast IN `slovenia-weather` RETURN forecast"
            },
            ...
        }
    ]
}
```

The whole process of loader script is thoroughly described in D3.2 and in the online repository\(^{35}\).

### 2.3.1 Pipeline Overview

\(^{33}\) [https://github.com/bergloman/docker-images/tree/master/qminer](https://github.com/bergloman/docker-images/tree/master/qminer)


---

EW_Shopp  
GA number: 732590  
H2020-ICT-2016-2017/H2020-ICT-2016-1
The pipeline of data analytic tools can be roughly divided as follows:

- **Loader**[^35]: It is an essential part of the internal data analytics pipeline. Before the actual modelling of the models, the raw or enriched data needs to be loaded into QMiner databases. The loader script, or rather a functionality, enables loading data to QMiner database from ArangoDB, MariaDB or other MySQL sources, and TSV/CSV files. Most supported and tested method is loading of the data from TSV/CSV files to QMiner database.

- **Pipeline**[^36]: Also called modelling pipeline is a core of data analytic pipeline. It is able to transform raw data from QMiner database and builds additional features, to train the model using a set of features and records, and finally to make predictions with the built models.
  - Weather enrichment (optional): Weather enrichment is the calculation of weather features using raw weather data that were previously loaded in QMiner database while using the Loader script.
  - Media coverage enrichment (optional): Media coverage enrichment is the process of downloading the relevant data from EventRegistry and calculating media coverage features. This process does not need raw data input, and it works as an independent entity, but needs well defined JSON configuration files.

- **Keyword clustering tool**[^37]: This tool enables to cluster large set of keywords into meaningful groups that would allow group-level modelling. The keyword clustering tool is best described in chapter 4.1 in D3.5.

To use data analytics’ pipeline, one would need to define configurations to load raw data or already enriched data from different sources to QMiner database. After loading the data to the QMiner database, we would optionally calculate weather features, using weather enrichment functionality, and/or use media coverage enrichment to enrich data with features from EventRegistry. These two steps are optional and can be entirely skipped if the features are precalculated during data enrichments and loaded into QMiner database with the Loader script or the weather and/or media coverage features are not desirable.

After all the necessary data is ready in QMiner databases, the training of the model starts. Using the JSON configuration file, we define which features we would like to use, which records, and additional model’s parameters. Similarly, we can make predictions putting the new data through the whole pipeline and use similar JSON configuration, only changing the subset of records. The output or predictions of the internal data analytic pipeline is either stored in a local TSV file or served as a server response in the JSON format.

The workflow and general structure of the data analytics’ pipeline is depicted in Figure 9. Input data is listed on the far right and all configuration files above and below. With the light orange are marked the data provided by the user, and with dark orange are marked the final outputs.

[^35]: [https://github.com/ew-shopp/analytics/tree/master/analytics/loader](https://github.com/ew-shopp/analytics/tree/master/analytics/loader)
[^36]: [https://github.com/ew-shopp/analytics/tree/master/analytics/pipeline](https://github.com/ew-shopp/analytics/tree/master/analytics/pipeline)
[^37]: [https://github.com/ew-shopp/analytics/tree/master/keyword_clustering](https://github.com/ew-shopp/analytics/tree/master/keyword_clustering)
The detailed description of the internal pipeline workflow of the data analytic tools is provided in D3.2 with complementary online documentation\textsuperscript{38}.

\textsuperscript{38} https://github.com/ew-shopp/analytics
Chapter 3  Using the EW-Shopp Toolkit

This chapter provides information about how the toolkit can be practically used in various relevant scenarios. Each chapter provides an example scenario for the use of the scalable data preparation and enrichment tool, the visualization tool and the analytics tool of the EW-Shopp toolkit and demonstrates some of the means of integration that the tools can use in order to work together.

3.1 Scalable Data Preparation and Enrichment – the Scenarios of Obtaining Weather Data from OpenWeatherMap\(^{39}\) and from EW-Shopp Custom Events API

In order to deploy the scalable data preparation workflow, our approach requires Docker and Rancher to be installed on the targeted system. Additionally, to store the data, in this example we will use a MySQL database. The installation instruction for all the prerequisite software can be found at the following URLs:

- Docker - https://docs.docker.com/install/
- MySQL database\(^{40}\) https://dev.mysql.com/doc/mysql-installation-excerpt/5.5/en/

Regarding system requirements, the following are the minimum to install the required tools and services:

- A minimum of 1GB RAM available on the host
- Accurate time synchronization on the host
- A Linux distribution compatible with Docker: (Ubuntu, CentOS, Debian, Fedora)

The weather data components are made as a set of Docker containers that are configured as a processing pipeline. The container processes are passing data through shared directories passed as Docker volumes into the process. The directories are managed by the host Operating System (OS). The host OS can provide file sharing using distributed file system. If distribution is supported, the container processes can be distributed across different processing units.

The container processes are orchestrated using Rancher for easy management. When starting a container process, it is given three volumes: \(\text{in}\), \(\text{work}\) and \(\text{out}\). A container process normally polls for input files at its \(\text{in}\) volume. When a new input file is detected it is moved to the \(\text{work}\) volume, processed and the result is passed to the \(\text{out}\) volume. Container processes will pass data to each other when \(\text{out}\) volume for one process is mapped to the same directory as \(\text{in}\) volume for another process.

\(^{39}\) https://openweathermap.org/
\(^{40}\) note: we use version 5.7.18 for the testing environment
The weather data access consists of two Docker containers:

- fetch_weather_owm (full image description in appendix section A.6) - for fetching data from the OpenWeatherMap API\(^1\) - optionally, this can be triggered on a daily schedule using CRON scheduling
- uploadtomysql (full image description in appendix section A.7) - uploading the weather data to the MySQL database management system

In a similar way, it is possible to set up a module able to use the API for sharing custom events in json-ld format developed within the project and whose specification is freely available in Swagger Hub\(^2\). Custom events have been specified through an ontology (presented in D1.4) that largely leverages pre-existing ontology elements (e.g. schema.org) to extend compatibility with other sources (e.g. Event Registry). Custom events (e.g. applying a discount to a product category) more than general events have proven to have a significant impact in ecommerce; for this reason, ASIA allows the user to enrich data sets with such information, aggregated by date, place and business category.

Similarly, the custom events access is implemented by means of two Docker containers:

- fetch_custom_events (full image description in appendix section A.16) - for fetching data from the EW-Shopp custom events API - optionally, this can be triggered on a daily schedule using CRON scheduling
- uploadtomysql (full image description in appendix section A.7) - uploading the custom events data to the MySQL database management system

Since the two scenarios presented in this section are equivalent in terms of functionality provided, in the next sections, for the sake of brevity, only the first of the two will be developed: the one that refers to weather data uploaded to MySQL database.

### 3.1.1 Creating the Weather Data Database Schema in MySQL

In order to store the data in the MySQL database, we need to create a new database and a table to store the data. The database name will be "knowage-db" and will also be used for the next part of the description of the EW-Shopp toolkit (section 3.2). In the database we need to create a table called "weather_data" with all of the necessary columns to store the data retrieved from the OpenWeatherMap API. The SQL script for creating the necessary database and table are available in the image GitHub repository\(^3\).

### 3.1.2 Docker Container Set Up

\(^1\) [https://openweathermap.org/api](https://openweathermap.org/api)
\(^2\) [https://app.swaggerhub.com/apis/EW-Shopp/EW-Shopp_Event_API/2.2.0](https://app.swaggerhub.com/apis/EW-Shopp/EW-Shopp_Event_API/2.2.0)
\(^3\) [https://github.com/ew-shopp/ingestion_lib/blob/master/06_fetch_weather_owm/examples/weather_data_mysql_schema.sql](https://github.com/ew-shopp/ingestion_lib/blob/master/06_fetch_weather_owm/examples/weather_data_mysql_schema.sql)
Apart from the MySQL set up, we need to create the folder structure that will be used by the containers. Since we download weather data directly from OWM and then upload it to the MySQL database, we only need three folders for the 'uploadtomysql' container that we will create. It will fetch the data from the input folder and upload it to the MySQL instance. Thereby, once you can use the following command:

```bash
mkdir in work out
```

For configuring the weather fetcher (fetch_weather_owm), we would also need to provide a CSV file with the coordinates that will be used to query the OpenWeatherMap API. In this example, the CSV file will be stored in the folder "region".

```bash
mkdir region
```

An example file that contains the coordinates for the city of Milan is available on GitHub\(^\text{44}\). In order to be able to access the OpenWeatherMap API, you will also require an API key that can be obtained by registering in the OpenWeatherMap website. Finally, optionally, we can enable scheduling of the weather fetching using the Rancher CRON scheduler service that can be activated from the Rancher Catalog (see Figure 10). This will enable us to activate the weather fetcher on a regular basis - e.g., once a day at midnight (corresponding CRON expression "0 0 * * *").

\(^{44}\) [https://github.com/ew-shopp/ingestion_lib/blob/master/06_fetch_weather_owm/examples/milan-region.csv](https://github.com/ew-shopp/ingestion_lib/blob/master/06_fetch_weather_owm/examples/milan-region.csv)

![Figure 10. Rancher CRON scheduler in the Rancher service catalog](image)

### 3.1.3 Rancher Set Up
Once the full folder structure is in place, you can now deploy the weather fetching and uploading workflow by creating a new Rancher stack. This can be done by clicking the "Add Stack" button in the "Stacks" user interface. The UI of Rancher allows users to import a Docker Compose YAML file to use for the stack configuration. The current example Docker Compose can be found in GitHub.45

![Figure 11. Creating a weather ingestion stack in the Rancher UI](image)

Note that you will need to edit the configuration parameters and volumes for the following:

- Environment variables
  - WE_OWM_KEY - OpenWeatherMap API key
  - WE_SERVER_ADDR - IP address of the MySQL server
  - WE_SERVER_USR - user name (needs to have permission to upload data)
  - WE_SERVER_PWD - password for the user
- Volumes - the base folder where the input, work and output folders were created

45 [https://github.com/ew-shopp/ingestion_lib/blob/master/06_fetch_weather_owm/examples/docker-compose.yml](https://github.com/ew-shopp/ingestion_lib/blob/master/06_fetch_weather_owm/examples/docker-compose.yml)
3.2 Setting Up Visualization: an Open Data + Weather Use Case

In EW-Shopp Knowage as a visualisation and navigation tool has been demonstrated in the various business cases during the projects as presented in detail in other EW-Shopp deliverables (in particular D2.3, D3.3 and D3.4).

With the final release of the Toolkit and its release as Open Source we aim to support new potential users of the EW-Shopp toolkit with scenarios similar to the Business Cases partners in setting-up a visualization environment with Knowage. To this end a ‘Weather-based’ methodology/tutorial is described here. It should be noted that, as anticipated above, in order to facilitate potential adoption of the service, for this final release the reference Knowage version is the Community Edition one which anyone can freely download and set-up. For details on the Knowage functionalities we refer here, please refer to D3.5 as well as the official documentation.

Following a simplified schema from the EW-Shopp Dataflow methodology (see D2.3), we aim to acquire some time-series-based (daily) data and compare it and analyse in comparison to weather data from a certain location through Knowage. The idea, here, is to use a minimal example which uses Open Data and (given one or more time-based data sets, and relevant weather data), could be replicated easily. In the diagram below we summarise the high-level structure of the use case. In our example we chose a Creative Commons licensed open data set: daily public Wi-Fi usage data from the Comune di Milano. The dataset reports daily Wi-Fi usage in terms of downloads from 1 January 2019 to 21 October 2019, per day. The weather data is acquired from OpenWeatherMap (OWM) weather data provider which also offers a series of APIs for acquiring data. We show that once the data is acquired in a MySQL database the visualisation can be created through a few relatively simple SQL queries used as Knowage data sets and then used to provide various interesting visualisations and data exploration features.

---

Creative Commons Attribution: Coune di Milano, Open WiFi 2019: distribuzione per zona del traffico, dataset ID: DS499 – Last access: 07-11-2019

47 https://openweathermap.org/
We use functionality provided from the Scalable data preparation and enrichment back-end to fetch weather data continuously and deploy it in a MySQL database (see section 2.2.1 and in particular 2.2.1.3). In this case a two-step approach is followed: 1) fetch data from OWM and the Open Data provider respectively and 2) upload to MySQL. These two steps can be deployed on a scalable infrastructure (as in our case).

The MySQL database will then be used as the data source in Knowage (MySQL) to set-up data sets and visualisations. The advantage of this approach is that it leverages on a few simple SQL queries (with SQL being a very well-known data-oriented language), and is scalable to more complex use cases.

For this example, we have called the database knowage-db. The two main database tables we have after the preparation and enrichment one containing all of the weather data (‘weather_data’) and one containing the daily Wi-Fi data (‘milan_wifi_data’). The milan_wifi_data table has the following columns:

<table>
<thead>
<tr>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL_Download</td>
</tr>
<tr>
<td>ALL_Upload</td>
</tr>
<tr>
<td>zona_1_Download</td>
</tr>
<tr>
<td>zona_1_Upload</td>
</tr>
<tr>
<td>zona_2_Download</td>
</tr>
<tr>
<td>zona_2_Upload</td>
</tr>
<tr>
<td>[...]</td>
</tr>
</tbody>
</table>

With 2 columns for each of the 10 Milan zones, one for Downloads and one for Uploads. In our case we will ignore the zone information and only use the global ALL_Download information.
The weather_data table follows the OWM Weather Features also adopted in EW-Shopp (when interacting with OWM) and explained in detail on the dedicated GitHub page (https://github.com/ew-shopp/weather-data/wiki/Weather-features).

Before diving into queries and visualisations with Knowage we will create two simple MySQL views\(^{48}\) on the database to facilitate further queries. This is not a strictly necessary step but is useful to have simpler queries to the database. Here are examples of the views.

```
CREATE OR REPLACE VIEW `maxValues` AS
select
    max(`weather_data`.`2t`) AS `temp_max`,
    min(`weather_data`.`2t`) AS `temp_min`,
    (max(`weather_data`.`2t`) - min(`weather_data`.`2t`)) AS `temp_delta`,
    max(`weather_data`.`tp`) AS `precip_max`,
    min(`weather_data`.`tp`) AS `precip_min`,
    (max(`weather_data`.`tp`) - min(`weather_data`.`tp`)) AS `precip_delta`,
    max(`milan_wifi_data`.`ALL_Download`) AS `download_max`,
    min(`milan_wifi_data`.`ALL_Download`) AS `download_min`,
    (max(`milan_wifi_data`.`ALL_Download`) - min(`milan_wifi_data`.`ALL_Download`)) AS `download_delta` from
(`weather_data` join `milan_wifi_data` on
    ((`milan_wifi_data`.`date` = cast(`weather_data`.`validityDateTime` as date))))
where
    (`milan_wifi_data`.`ALL_Download` is not null)
    and (dayofmonth(`weather_data`.`validTime`) = dayofmonth(`weather_data`.`validityDateTime`))
    and (cast(`weather_data`.`validityDateTime` as time) = '12:00:00')
```

This view provides a series of maximum and minimum values as well as deltas which will help us normalise the data, which we will see, is useful for visualisation.

```
CREATE OR REPLACE VIEW `milanSingleDayData` AS
select
    cast(`weather_data`.`validityDateTime` as date) AS `date_rec`,
    `weather_data`.`2t` AS `2t`,
    (`weather_data`.`2t` - 273.15) AS `2t_temp_C`,
    `weather_data`.`tp` AS `tp`,
    (`weather_data`.`tp` * 1000) AS `tp_rain_mm`,
    `weather_data`.`tcc` AS `tcc`,
    (round(`weather_data`.`tcc`, 2) * 100) AS `tcc_cloudiness_perc`,
    `milan_wifi_data`.`ALL_Download` AS `ALL_Download`
from
```

(`weather_data`
join `milan_wifi_data` on
(`milan_wifi_data`.`date` = cast(`weather_data`.`validDateTime` as date)))
where
(`milan_wifi_data`.`ALL_Download` is not null)
and (dayofmonth(`weather_data`.`validTime`) = dayofmonth(`weather_data`.`validDateTime`))
and (cast(`weather_data`.`validDateTime` as time) = '12:00:00')

The second view serves two main objectives: 1) it extracts relevant weather data (i.e., temperature, precipitation and cloudiness), filters to have only the latest (same day) forecast and at 12 o’clock, i.e. the middle of the day. We could of course average 3-hourly readings but, in our case, this is not needed; additionally, in the case of temperature, night readings seem to lower the average too much if we take into account that the Wi-Fi data we are analysing is only aggregated at the day level. 2) some conversions are made to facilitate user reading, for example temperature is converted from Kelvin to Celsius and precipitation (i.e., rain) from metres to millimetres.

### 3.2.1 Creation of the Data Source and Data Sets in Knowage

Once the database is in place and accessible from the Knowage machine, we can create a Data Source which uses the database (in our case MySQL). Here is a screenshot of the data source created:

![MySQL data source](image)

**Figure 13. Creation of the MySQL data source connecting to the Scalable data preparation and enrichment back-end database**

Note that usually the use of a root user to access a database is highly discouraged for security – especially in production.

Once the Data Source for the database is created, we can use it as a source for our Data Sets. In our case data sets will be some simple SQL queries with parameters which will fetch the data we will...
then use for the visualisations. In the screenshot below we see the data set creation interface, and below the queries used are reported.

![Figure 14. Creation of the data set made of a parametrised SQL query to fetch the weather + download data](image)

Before saving the data set we can preview the query result in a table by clicking on the PREVIEW button:

![Figure 15. Preview of the data set columns](image)

We essentially create two datasets: one with a query which returns daily weather + downloads data ‘as is’, just with some conversions for the weather data (e.g. temperature to °C):

```sql
SELECT DATE(weather_data.validityDateTime) as date_rec, weather_data.2t - 273.15 as temp_C, weather_data.2t - 273.15 as temp_sort, weather_data.2t / (SELECT
```
MAX(weather_data.2t)
FROM weather_data
JOIN milan_wifi_data on milan_wifi_data.‘date‘ = DATE(weather_data.validityDateTime)
WHERE milan_wifi_data.ALL_Download IS NOT NULL
AND DATE(weather_data.validTime) BETWEEN $P{start_date} AND $P{end_date}
AND DAY(weather_data.validTime) = DAY(weather_data.validityDateTime)
AND TIME(weather_data.validityDateTime) = ‘12:00:00‘ ) as temp_norm,
ROUND(weather_data.tcc,2) as cloudiness,
ROUND(weather_data.tcc,2) as cloud_sort,
weather_data.tp * 1000 as rain_mm,
weather_data.tp * 1000 as rain_sort,
weather_data.tp / ( SELECT MAX(weather_data.tp) FROM weather_data
JOIN milan_wifi_data on milan_wifi_data.‘date‘ = DATE(weather_data.validityDateTime)
WHERE milan_wifi_data.ALL_Download IS NOT NULL
AND DATE(weather_data.validTime) BETWEEN $P{start_date} AND $P{end_date}
AND DAY(weather_data.validTime) = DAY(weather_data.validityDateTime)
AND TIME(weather_data.validityDateTime) = ‘12:00:00‘ ) as rain_norm,
milan_wifi_data.ALL_Download as downloads_daily,
milan_wifi_data.ALL_Download as downloads_sort,
(milan_wifi_data.ALL_Download) / ( SELECT MAX(milan_wifi_data.ALL_Download) FROM milan_wifi_data
) as downloads_norm,
(milan_wifi_data.ALL_Download) / ( SELECT MAX(milan_wifi_data.ALL_Download) FROM milan_wifi_data
) as dl_max
FROM weather_data
JOIN milan_wifi_data on milan_wifi_data.‘date‘ = DATE(weather_data.validityDateTime)
WHERE milan_wifi_data.ALL_Download IS NOT NULL
AND DATE(weather_data.validTime) BETWEEN $P{start_date} AND $P{end_date}
AND DAY(weather_data.validTime) = DAY(weather_data.validityDateTime)
AND TIME(weather_data.validityDateTime) = ‘12:00:00‘
ORDER BY date_rec DESC
Notice that we could omit or remove the final ORDER, as all visualisation widgets are able to sort data based on user preference. We keep it more for the purpose of debugging the query. If you look at the query closely you will note that we are repeating some values, for instance returning columns such as rain_sort. As the name suggests, this has to do with how Knowage consumes data sets: columns can be either ‘attributes’ or ‘measures’. Attributes would typically be on the X axis of a chart while measures on the Y. In some chart widgets if we want to order by a measure, we also need to have the relative attribute. This will become clearer further below when we provide the various visualisations.

The second query returns data normalised. In fact, for visualisation on charts comparing different scales (even on if using distinct series) can be less effective (for instance the temperature has a relatively low delta from minimum to maximum). We apply a rather simple normalization here where the minimum value of a series will be 0 and the maximum will be 1: in this way we can put all the series on the same graph and appreciate the trends.

```
SELECT date_rec,
2t / (SELECT temp_max from maxValues) as temp_norm_1,
(2t - (SELECT temp_min from maxValues)) * (SELECT temp_delta from maxValues) as temp_n,
(SELECT temp_max - temp_min from maxValues) * (SELECT temp_delta from maxValues) as temp_delta_max,

(SELECT precip_max from maxValues) as precip_norm_1,
(tp - (SELECT precip_min from maxValues)) * (SELECT precip_delta from maxValues) as precip_n,
(SELECT precip_max - precip_min from maxValues) * (SELECT precip_delta from maxValues) as precip_delta_max,

(SELECT download_max from maxValues) as download_norm_1,
(ALL_Download - (SELECT download_min from maxValues)) * (SELECT download_delta from maxValues) as download_n,
(SELECT download_max - download_min from maxValues) * (SELECT download_delta from maxValues) as download_delta_max,

(SELECT temp_n) / (SELECT temp_delta_max) as temp_norm_2,
(SELECT precip_n) / (SELECT precip_delta_max) as precip_norm_2,
(SELECT download_n) / (SELECT download_delta_max) as download_norm_2,

from `knowage-db`.milanSingleDayData
WHERE DATE(date_rec) BETWEEN $P{start_date} AND $P{end_date}
```
Note that for this query to work we need to have created the milanSingleDayData view on the database as explained above.

### 3.2.2 Creation of the Visualization Cockpit and Widgets

Once we have these queries saved as data sets, we can create our visualisation cockpit. This is done from the Document Browser (indicated by the folder shape icon highlighted in red in the screenshot) and by selecting the pink + sign and ‘Cockpit’, as shown in the screenshot below:

![Figure 16. Creation of a Knowage cockpit for the data visualisation](image)

Once we create the cockpit we will be presented with an empty canvas. The first step is to add our datasets to the canvas. This is done by selecting the database icon at the top of the + menu.

![Figure 17. Adding the needed data sets to the cockpit](image)

We add our two previous datasets keeping all the default values.
Additionally, we create an association between the two date (‘date_rec’) fields of the data sets. This means that the dates will always be in sync in all widgets (e.g. if a user selects a single date to view in detail). We must remember to click on the ‘save’ icon (diskette icon) before saving the whole dataset configuration.

Now we have our datasets ready we start to fill our empty canvas, first let’s create a few widgets for title, an image and some text crediting the Creative commons attribution of the Open Data set. These are an image widget, a text one and an HTML widget.
Then we create 3 additional ‘sheets’ (i.e. similar to tabs) in the bottom.

![Figure 21. Sheets added to the cockpit](image)

As the names indicate the sheets will contain: a single chart with all of the ‘real’ data showing download data, temperature and rain; a sheet containing a chart with normalized data including download data, cloudiness, rain and temperature; a sheet with two charts displaying the data by sorting it by downloads (one chart) and temperature (other chart); finally a sheet containing an interactive table (called ‘Cross Table’ in Knowage), where one can drill into the data, sort it as well as select single dates to analyse.

**Single Chart**

Like with almost all visualisation widgets, we select the desired data set to use. In this case it is the MILAN_WiFi_Weather one.

![Figure 22. Addition of the data set to the single chart widget](image)
We then select a line chart from the Chart Engine designer to start with.

![Figure 23. Char type selection](image)

Still from the Chart engine designer we will further go into the structure to select the data to visualise and customise the visualisation of data.

In this case, we create an independent series for each of the download, temperature and rain data. This is to make sure that each data will use its own scale (Knowage can calculate min-max automatically, and we will leave it like this here). The final configuration is shown in the following screenshot:

![Figure 24. Set-up of the chart structure](image)
For all of the series we deselect the display value checkbox, given the density of the data. In fact, the generated chart will be interactive so that if we hover a data point we get a detailed tooltip with value etc.

As we can have different types chart types in the same widget, we explicitly set the Series item type as follows:

- Downloads_daily: Line
- Temp_C: bar
- Rain_mm: bar

We also set-up names and suffixes which will show respectively in the legend and tooltips. We can also decide the precision to be shown (e.g. we have set it to 1 for temperature).

From the Configuration tab we select the Show Legend checkbox from the Title item and in the Legend Items menu, but we deselect “Enable hide/display values on chart by legend checkboxes”, as shown in the following two screenshots.
Finally, before saving our chart, we select some custom colours for the three series through the Colour Palette selection (feel free to choose your favourite colours) as shown below:

Figure 26. Chart configuration: legend
Figure 27. Chart configuration: custom colours

After saving the chart will be created and updated with the data. The result (depending also on screen resolution) should look as follows:

Figure 28. Final cockpit screenshot: single chart with all data with the different units
The chart is completely interactive. In particular a user can show/hide series by clicking on the related legend item, zoom in/out to a certain range (in this case a data range), hover any data point and see a tooltip with detailed information.

If you are happy with the result remember to save the updated cockpit with the specific menu button in the top right corner as shown here below:

![Figure 29. Saving the cockpit](image)

Now we will add the other charts and the table. In this case, we are providing screenshots to guide you through the creation.

**Normalized**

Here we present a chart with all of the normalized values. We chose an area chart with a light grey colour for cloudiness (positioned at the end of the visualisation stack), line charts for temperature (red) and downloads (black) and a light blue bar chart for rain (precipitation). The normalized graph alters the actual proportions of data but helps better grasp the trends and also compare trends of the different data which otherwise is on completely different scales (downloads are in the order of the hundreds of thousands, temperature in °C goes from below zero to maximum 35, etc.).

![Figure 30. Adding the data set with the normalised data](image)
Let’s keep in mind we can drill down into the graph, for example to see in more detail a date interval as shown here below:
Sorted

In this sheet, we add two slightly different charts that allow to concentrate on the variation of other data as the temperature decreases and, in complementary manner, as the download increase. It is important to understand that, although the charts are stacked vertically, in this case the data are not sorted by date but according to the sorted order. These charts actually allow us to appreciate that there is a slight increase trend in downloads with the increase of temperature. Of course, this could be due to higher temperature increasing in summer when there’s also an increase in the number of tourists. It would be interesting to further cross this data with daily visitor presence in Milan: unfortunately, at the time of writing this data is not available.

As anticipated above, in order to achieve the desired sorting, we need to have some ‘sorting’ columns as ‘attributes’ within the dataset, reason why we included these in our query. To set-up the sorting we select the Ordering column menu from the Categories dropdown menu.
Here we are presented with a set of attributes (independently from which are displayed on the X axis), to choose as well as the sorting order. For instance, ‘temp_sort’ in Descending order for the first chart.

The two resulting charts:
Table

Finally, we want to provide the user a table with all of the data (real, not normalized), so that they can analyse it and drill into it. To this end we use the Cross Table widget which also offers some nice formatting and visual features. Additionally, clicking on any date in the table will zoom the charts in all other sheets to that particular date.

The cross table is selected from the widget selection menu as follows:

![Selection of the cross table widget](image)

As with other widgets we need to select the data set. Additionally, here we have to select (via drag and drop) the data attributes to show as rows and the measures as shown below:
Clicking on the cog wheel icon for each data item enables to set-up formatting and visual styles for each of the rows/columns. In this case in particular we add some visual icons for high temperatures.

The cross table widget has many interesting styling features which users can explore by editing each column. Our end result is as follows. The user can sort by columns by simply clicking the related header.
3.3 Setting Up Analytics

The purpose of the analytics is to process the curated and enriched data to produce a model that captures the internal workings of the data. The model is a mathematical description of the data’s intrinsic nature – an abstraction that can be used to predict future dynamics of the system it was
built to describe. The analytics modelling workflow can be roughly split into two main steps: *data ingestion* and *model fitting*. In the EW-Shopp analytics toolkit, the first step is performed by the **data loader** component and the latter by the **modelling pipeline** component. The workflow is illustrated by the chart on **Figure 41**.

![Figure 41: The analytics workflow from data ingestion to model building.](image)

For demonstration purposes in this section we will use a simplified version of the JOT business case. Let's assume we would like to model the level of daily impressions of an AdWords keyword. A keyword in this sense is a short phrase (e.g. “FIFA Football league”, “burger delivery”) that is used as a query in a web search engine such as Google. An impression happens when a user searches for that keyword in the search engine. Since advertisements are shown besides the search results, knowing which keywords are going to be popular in advance represents an important advantage to a marketing professional. For our case, let’s assume we want to make the prediction using three weather features: the average daily temperature, average daily cloud coverage and total daily precipitation. Of course, this is a great simplification – for information on the full model see the full business case description in D4.5. The input data we are provided is in the form of a table stored in a csv file:

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Impressions</th>
<th>Avg_temp</th>
<th>Avg_cloud_cover</th>
<th>Total_prec</th>
</tr>
</thead>
<tbody>
<tr>
<td>02/06/2017</td>
<td>46</td>
<td>26</td>
<td>0.31</td>
<td>1.5</td>
</tr>
<tr>
<td>03/06/2017</td>
<td>161</td>
<td>21</td>
<td>0.87</td>
<td>7.9</td>
</tr>
<tr>
<td>04/06/2017</td>
<td>74</td>
<td>25</td>
<td>0.22</td>
<td>0.0</td>
</tr>
<tr>
<td>05/06/2017</td>
<td>53</td>
<td>27</td>
<td>0.09</td>
<td>0.0</td>
</tr>
<tr>
<td>06/06/2017</td>
<td>17</td>
<td>28</td>
<td>0.04</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Data Loader**

The purpose of the data loader component is to move data from a source to a destination. The most common use is to take the input data from a source and loading it into the QMiner internal store which is optimised for running machine learning algorithms. Another option would be to upload predictions from the internal store to a database if we’d choose to run predictions directly on the
data in the store (for example for model debugging purposes). The supported formats for source and destination are the QMiner store, MySQL database, ArangoDB database and a csv file.

The entire loading operation is specified using a single configuration JSON structure. Which specifies the source, destination and which data to transfer. An example configuration for the simplified JOT case data presented above is:

```json
{
  "source": {
    "type": "Csv",
    "dir": "<path_to_dir_with_data_files>",
    "custom_fn_path": "path_to_data_loading_code",
    "delimiter": "",
  },
  "destination": {
    "type": "QminerDB",
    "db_path": "<path_to_QMiner_DB_dir>",
    "mode": "createClean"
  },
  "queries": [
    {
      "name": "JOT Schema",
      "use_query": false,
      "use_schema": true,
      "schema": [
        {
          "name": "Keywords",
          "fields": [
            {
              "name": "Timestamp",
              "type": "datetime"
            },
            {
              "name": "Impressions",
              "type": "float"
            },
            {
              "name": "Avg_temp",
              "type": "float"
            },
            {
              "name": "Avg_cloud_cover",
              "type": "float"
            },
            {
              "name": "Total_prec",
              "type": "float"
            }
          ],
          "joins": [],
          "keys": [
            {
              "field": "Timestamp",
              "type": "linear"
            }
          ]
        }
      ]
    }
  ]
}
```
The configuration JSON is passed to the data loader command line tool which performs the data loading operation. The full configuration API is explained in detail in deliverable D3.2. Here it suffices to note that the source and destination are specified as well as the structure of the QMiner internal store as it is created from scratch (as opposed to loading the data into an existing store). In the case of MySQL or ArangoDB store this configuration would also map which source columns are transferred into which destination columns. In the case of a csv file as the source, we need to provide the code that parses the lines of the csv, which are essentially strings of text, into their proper data types. Here this is the onLineKeyword function defined in a separate file:

```javascript
"use strict";

function parseDate(dateStr) {
    let year = parseInt(dateStr.substring(0,4));
    let month = parseInt(dateStr.substring(4,6));
    let day = parseInt(dateStr.substring(6,8));
    return new Date(year, month, day, 0, 0, 0, 0);
}

function maybeNull(valStr) {
    if (valStr === "") {
        return 0;
    }
    return valStr;
}

function onLineKeyword(base, lineVals) {
    // create keyword record
    let rec = {
        "Timestamp": parseDate(maybeNull(lineVals[0])),
        "Impressions": parseFloat(maybeNull(lineVals[1])),
        "Avg_temp": parseFloat(maybeNull(lineVals[2])),
        ...
```
Technically, it is possible to perform other minor transformations as part of this data parsing code, but since the input data has already been processed by the data transformation and enrichment tool, there is typically no need for that.

### Modelling pipeline

The modelling pipeline passes the data from the QMiner internal storage to a machine learning algorithm, which processes it and builds a model. The entire process is again specified using a JSON structure. The specification determines which data columns are used as features and which is used as the target variable for the model. The parameters of the machine learning algorithm used are also set. The configuration JSON is passed to the modelling pipeline command line tool, which performs the modelling operation. An example configuration JSON for the simplified JOT case is:

```json
{
    "name": "JotKeywordImpr",
    "version": "0.1",
    "dir": "/",
    "description": "Regression of keyword impressions.",
    "pipeline": {
        "input": {
            "primary_key": ["Timestamp"]
        },
        "extraction": [
            {
                "module": "generic_feature_selector",
                "params": {
                    "input_db": "<path_to_QMiner_db>",
                    "input_store": "Keywords",
                    "features": [
                        "Av_temp",
                        "Avg_cloud_cover",
                        "Total_Prec"
                    ],
                    "normalize": "scale"
                }
            }
        ],
        "model": {
            "module": "SVR",
            "params": {
                "SVM_param": {
                    "algorithm": "LIBSVM",
                    "kernel": "linear",
                    "epsilon": 0.1,
                    "gamma": "scale"
                }
            }
        }
    }
}
```
"c": 100,
"j": 1,
"batchSize": 10000,
"maxIteratios": 100000,
"cacheSize": 500,
"degree": 3,
"kernel": "RBF",
"type": "EPSILON_SVR",
"maxTime": 12000
},
"score": "cv",
"model_filename": "<path_to_model_file>",
"show_all_weights": true,
"verbose": true
}

"input_extraction": {
  "module": "generic_input_extractor",
  "params": {
    "input_db": "<path_to_QMiner_DB>",
    "search_query": {
      "$from": "Keywords"
    },
    "target_var": "Impressions"
  }
}

Deployment

The model produced by the modelling pipeline is in essence a mathematical formula producing a predicted value from the input feature values. Once fit to data, it is stored into a file and can be deployed into the target production system. Deployment specifics can differ a lot depending on different business needs (e.g. in an internal application or on a website). A simple REST server is provided; it is built using JavaScript, which can load the model and produce predictions. It is specified by a configuration JSON structure specifying the hostname and port where the server listens. Security credentials can also be provided if the server is secured using a certificate. The configuration JSON is passed to the server command line tool which sets everything up and runs the server. An example setup is:

```json
{
  "model_filename": "<path_to_model_file>",
  "host": "localhost",
  "port": "1234",
  "security": {
    "key_filename": "<path_to_key_filename>",
    "certificate_filename": "<path_to_certificate_filename>"
  }
}
```
Appendix A  EW-Shopp Workflow
Composable Steps Library

A.1. Introduction

The components in this library are made as a set of Docker containers that are can be configured as to implement a processing pipeline. The container processes are fed with data available in shared directories passed as Docker volumes. The directories are managed by the host OS. The host OS can provide file sharing using distributed file system. If distribution is supported, the container processes can be distributed at different processing units.

The container processes are orchestrated using Rancher for easy management. The Docker containers don’t have direct usage of Rancher services and should be possible to start using any orchestration system.

When starting a container process, it is given three volumes in, work and out. A container process normally polls for input files at its in volume. When a new input file is detected it is moved to the work volume, processed and the result is passed to the out volume. Container processes will pass data to each other when out volume for one process is mapped to the same directory as in volume for another process.

It is important that the OS user for the processes have access to the volumes they are given.

Currently the consists of the following Docker containers:

- 00_fetch_weather_ecmwf
- 01_uploadtoarango
- 04_extract_weather_ecmwf
- 05_uploadtoarango_dual
- 06_fetch_weather_owm
- 07_uploadtomysql
- 10_unzip
- 11_tsv2csv
- 12_split
- 13_transform
- 14_toarango
- 15_jot_categorize
- 16_jot_aggregate_kwds
- 17_jot_closest_kwds
- 18_fetch_custom_events
**A.1.1. Common**

The implementation of the different containers are based on a common set of scripts that are included in and executed by all containers. Docker does not support files used by several containers. The only way to handle this is to copy the scripts in the container during the built process. This is done by the 'make_docker.sh' script.

Other scripts that are used to provide a common input/output interface are:

- 'main.sh' is the main entry point for the container. It sets up a worker process 'main_worker.sh' running in the background. The main supports graceful shutdown by capturing SIGTERM and SIGINT. It signals to the worker process to stop by deleting the run-file. The main waits until worker process terminates. This arrangement allows the worker to finalize processing of a file before terminating. If the container specific script 'single_shot.sh' is present this file will be called, and the main loop will not be started.

- 'main_worker.sh' is the main loop polling for an input file each 60 seconds. If no file is found a wait_count is incremented. When it reached retry_max_count the main loop will terminate. Retry_max_count can be set to 0 to disable this functionality. The actual polling is done by 'fetch_and_process.sh'.

- 'fetch_and_process.sh' checks for input file based on the file pattern given by the container specific script 'get_input_file_spec.sh'. If file(s) are found one of them is chosen and moved to work. If the container specific script 'check_input_file.sh' is present the, file is tested before its moved. During the whole operation the in folder is locked using a lock file. The lock file avoids race conditions caused by other containers accessing the same file. After the file is moved to work the container specific script 'process_job.sh' is called for processing and finally provide an output file.

- 'process_job.sh' can call the script 'move_to_output.sh' to safely move one or more files to the out folder using a lock file. The lock file avoids race conditions caused by other containers accessing the same file.

**A.1.2. Common_py**

Common python files can be in this folder. Docker does not support files used by several containers. The only way to handle this is to copy the scripts before building. This is done by the 'make_docker.sh' script that can be found on the GitHub repository.49

**A.1.3. Building a Container**

---

49 https://github.com/ew-shopp/ingestion_lib/tree/master/common_py
Change directory to the 'Dockerfile' for the container to build. Call the build script for copying common files, building and pushing the container.

Example:

```bash
../common/make_docker.sh "datagraft/lib_17_jot_closest_kwds"
```

### A.2. 00_fetch_weather_ecmwf

This script fetch GRIB weather data from ECMWF service. Based on the GRIB file it extracts weather data for a set of regions listed in a csv. The extracted data are moved to the out directory.

The time range to fetch data for is given in a job-description-file in json format. This makes it possible to have this container waiting for jobs and fetch weather data on a periodic basis.

Example job file:

```json
{
    "start_date": "2018-05-31",
    "end_date": "2018-05-31"
}
```

### A.2.1. Volumes

Volume setup. Must match arguments:

- `/code` - internal location for scripts in the container (created internally)
- `/in` - not used by this container (mandatory)
- `/work` - used for intermediate files (mandatory)
- `/out..` - location for process results for next step in the pipeline (mandatory)
- `/root/.ecmwfapirc` - ECMWF credentials must be present in this file

### A.2.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)
- Full path to the regions csv-file
Example params:  0 /code /in /work /out /coordinates/bbox-csv

To enable logging, set the environment variable LOG_JOBS to 1.

### A.2.3. Source

Docker image: datagraft/lib_00_fetch_weather_ecmwf

Source code: [https://github.com/ew-shopp/ingestion_lib/tree/master/00_fetch_weather_ecmwf](https://github.com/ew-shopp/ingestion_lib/tree/master/00_fetch_weather_ecmwf)
A.3. 01_uploadtoarango

This script uploads a json file to an Arango database using HTTP API. Credentials are given in arguments.

A.3.1. Volumes

Volume setup. Must match arguments:

- /code - internal location for scripts in the container (created internally)
- /in - not used by this container (mandatory)
- /work - used for intermediate files (mandatory)
- /out...location for process results for next step in the pipeline (mandatory)

A.3.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)
- Server_url
- Username and password
- Db_name
- Collection_name

Example params: 0 /code /in /work /out 192.168.4.17:1234 root:xxx weather germany

To enable logging set the environment variable LOG_JOBS to 1

A.3.3. Source

Docker image: datagraft/lib_01_uploadtoarango

Source code: https://github.com/ew-shopp/ingestion_lib/tree/master/01_uploadtoarango

A.4. 04_extract_weather_ecmwf
This script fetches the region weather forecast stored in the GRIB files (obtained from ECMWF) and transforms that in a CSV (default) or JSON forecast for the cities inputted in an input CSV file.

### A.4.1. Example of Parameter Settings:

```bash
WE_INPUT_TYPE=env
WE_START_DATE=2018-05-14
WE_END_DATE=2018-05-14
WE_GRIB_FILE=path-to/weather.grib
WE_FORECAST_DAYS=16
WE_KEY_SEQUENCE=2d,2t,10u,10v,ptype,sd,sf,sund,ssr,tcc,tp,vis,ws,rh,cityName,region,strRegion,geonameId,position,validTime,validityDateTime
WE_NEW_KEY_SEQUENCE=,,,,,,,,,,,,,,,,,,,,,,
```

### A.4.2. Volumes

Volume setup. Must match arguments:

- `/code` - internal location for scripts in the container (created internally)
- `/in` - not used by this container (mandatory)
- `/work` - used for intermediate files (mandatory)
- `/out` - location for process results for next step in the pipeline (mandatory)

### A.4.3. Arguments

Arguments are passed to the container entrypoint and as environment variables:

**Container entrypoint params:**

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)

Example params: 0 /code /in /work /out

To enable logging set the environment variable LOG_JOBS to 1

The takes as arguments:

- the coordinates of the cities to which one wants the forecast
- the name of the outputfile (without extension)
- (optionally) the name of the input json file with the forecast parameters
The forecast parameters can be passed via a json file or via environment variables. The default is via environment variables. If you want the parameters to be passed via json, you must set the environment variable WE_INPUT_TYPE to "json". If the variable is not set or set to "env", the script will take the parameters from environment variables.

Below is a description of the parameters to be passed via environment variables:

- **WE_INPUT_TYPE**: to be set to "env" or "json". "env" means forecast parameters inputed via environment variables and "json" via json file.
- **WE_START_DATE**: start date in format YYYY-MM-DD
- **WE_END_DATE**: end date in format YYYY-MM-DD
- **WE_GRIB_FILE**: grib file with the forecast
- **WE_FORECAST_DAYS**: number of forecast days
- **WE_KEY_SEQUENCE**: specify which weather forecast fields of the grib will be shown on the csv and their order. The fields must be separated by commas (no spaces)
- **WE_NEW_KEY_SEQUENCE**: this is used to possibly rename the forecast fields when writing the csv. It must contain the same number of elements as the WE_KEY_SEQUENCE. And be empty for the fields not to be renamed. The position field will be expanded and renamed to "Latitude" and "Longitude" regardless of the WE_NEW_KEY_SEQUENCE.
- **WE_JSON_OUTPUT**: if set to true, the output forecast will be written to a json file instead of a cvs. The default is false.

### A.4.4. Source

Docker image: datagraft/lib_04_extract_weather_ecmwf

Source code: [https://github.com/ew-shopp/ingestion_lib/tree/master/04_extract_weather_ecmwf](https://github.com/ew-shopp/ingestion_lib/tree/master/04_extract_weather_ecmwf)

### A.5. 05_uploadtoarango_dual

This script uploads a json file to an Arango database using HTTP API. It detects whether the file is destined for a document collection or an edge collection and uploads to the correct collection. Credentials are given in arguments.

### A.5.1. Volumes

Volume setup. Must match arguments:

- `/code` - internal location for scripts in the container (created internally)
- `/in` - not used by this container (mandatory)
- `/work` - used for intermediate files (mandatory)
- `/out..` - location for process results for next step in the pipeline (mandatory)
A.5.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)
- Server_url
- Username and password
- Db_name
- Doc_collection_name
- Edge_collection_name
- From_prefix_name
- To_prefix_name

Example params: 0 /code /in /work /out 192.168.4.17:1234 root:xxx weather Germany Germany_edge xx yy

To enable logging set the environment variable LOG_JOBS to 1

A.5.3. Source

Docker image: datagraft/lib_05_uploadtoarango_dual

Source code: https://github.com/ew-shopp/ingestion_lib/tree/master/05_uploadtoarango_dual

A.6. 06_fetch_weather_owm

This script fetches weather data for city locations specified in CSV file. Each city is a separate request to OWM (OpenWeatherMap) and results in todays forecast is stored in a JSON file. The forecast for all cities is collected and transformed that in one CSV (default) or json file. The process does run as a one-shot, fetching weather data and terminates. It will be started by a CRON job at Monday once a week '0 1 * * 1'. The process input location is not polled. More information about CRON setup can be found at https://crontab.guru/#0_1_*_*_1

A.6.1. Volumes

Volume setup. Must match arguments:

- /code - internal location for scripts in the container (created internally)
• /in - not used by this container (mandatory)
• /work - used for intermediate files (mandatory)
• /out -...location for process results for next step in the pipeline (mandatory)
• /coordinates - location for location CSV file (match path in environment)

A.6.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

• Max wait for file count. Set to '0' (not used by this container)
• Name of code location. Set to '/code'
• Name of input location. Set to '/in' (must match given volume name)
• Name of work location. Set to '/work' (must match given volume name)
• Name of out location. Set to '/out' (must match given volume name)

Example params: 0 /code /in /work /out

The forecast parameters are passed via environment variables:

• WE_INPUT_TYPE: to be set to "env"
• WE_OWM_KEY: OWM API key used when accessing the weather service
• WE_REGION_CSV_FILE: csv file listing all the regions and their coordinates
• WE_KEY_SEQUENCE: specify which weather forecast fields of the grib will be shown on the
csv and their order. The fields must be separated by commas (no spaces)
• WE_NEW_KEY_SEQUENCE: this is used to possibly rename the forecast fields when writing
the csv. It must contain the same number of elements as the WE_KEY_SEQUENCE. And be
empty for the fields not to be renamed. The position field will be expanded and renamed to
"Latitude" and "Longitude" regardless of the WE_NEW_KEY_SEQUENCE.
• WE_JSON_OUTPUT: if set to true, the output forecast will be written to a json file instead of
csv. The default is false.

Example of environment vars:

WE_INPUT_TYPE: env
WE_OWM_KEY: <YOUR_KEY_GOES_HERE>
WE_REGION_CSV_FILE: /coordinates/regions_measurence.csv
WE_KEY_SEQUENCE:
2d,2t,10u,10v,ptype,sd,sf,sund,ssr,sp,tcc,tp,vis,ws,rh,cityName,region,strRegion,geoNameId,position,validTime,validityDateTime
WE_NEW_KEY_SEQUENCE: ',,,,,,,,,,,,,,,,,,,,,,,,,,,,,,'

A.6.3. Source

Docker image: datagraft/lib_06_fetch_weather_owm
A.6.4. Rancher Setup Example:

test-owm:
    image: datagraft/test_06_fetch_weather_owm
    environment:
        WE_INPUT_TYPE: env
        WE_OWM_KEY: lkljrjwerlkj134u9024
        WE_REGION_CSV_FILE: /coordinates/regions_measurence.csv
        WE_KEY_SEQUENCE: 2d,2t,10u,10v,pctype,sd,sp,sund,ssr,tcc,tp,vis,ws,rh,cityName,region,strRegion,geonameId,
                        position,validTime,validityDateTime
        WE_NEW_KEY_SEQUENCE: ',,,,,,,,,,,,,,,,,,,,,'
        stdin_open: true
        volumes:
            - /bigdata/steffen//run_daily_weather_owm/in:/in
            - /bigdata/steffen//run_daily_weather_owm/_work00:/work
            - /bigdata/steffen//run_daily_weather_owm/_out00:/out
            - /bigdata/steffen//run_daily_weather_owm/coordinates:/coordinates
        tty: true
        user: 1000:1000
        command:
            - '/code'
            - '/in'
            - '/work'
            - '/out'
        labels:
            io.rancher.container.start_once: 'true'
            io.rancher.container.pull_image: always
        cron.schedule: 0 1 * * 1
A.7. 07_uploaddomysql

This script fetches a CSV file with one header row and upload the data to a MySQL database table. The script monitors the input directory for csv files to upload. If not file present it will sleep for 60 sec before trying again.

A.7.1. Volumes

Volume setup. Must match arguments:

- /code - internal location for scripts in the container (created internally)
- /in - used for passing input files to this container (mandatory)
- /work - used for intermediate files (mandatory)
- /out...location for process results for next step in the pipeline (mandatory)

A.7.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (how many idle 60 sleep periods before terminating)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)

Example params: 0 /code /in /work /out

A set of environment variables are needed to access the database

- WE_SERVER_ADDR: IP address of name to the MySQL server
- WE_SERVER_USR: Username
- WE_SERVER_PWD: Password
- WE_DB_NAME: Database name.
- WE_TABLE_NAME: Table name in the database to upload to.
- WE_TABLE_COLUMNS: Column names to use. Has to be a comma separated list.

Example of environment vars:

WE_SERVER_ADDR=192.168.1.21
WE_SERVER_USR=my_user
WE_SERVER_PWD=xyzz
WE_DB_NAME=test_import
WE_TABLE_NAME=weather_data
WE_TABLE_COLUMNS=(storeId, date, temp_avg, temp_max, temp_min, precip_tot, cloud_cov_avg)
A.7.3. Source

Docker image: datagraft/lib_07_uploadtomysql

Source code: https://github.com/ew-shopp/ingestion_lib/tree/master/07_uploadtomysql
https://github.com/ew-shopp/weather_import/tree/master/docker_daily_import/06_fetch_weather_owm

A.7.4. Rancher Setup Example:

test-mysql-upload:
  image: datagraft/test_07_uploadtomysql
  environment:
    WE_SERVER_ADDR: 192.168.1.21
    WE_SERVER_USR: root
    WE_SERVER_PWD: xyzz
    WE_DB_NAME: test_import
    WE_TABLE_NAME: weather_data
    WE_TABLE_COLUMNS: (storeId, date, temp_avg, temp_max, temp_min, precip_tot, cloud_cov_avg)
    LOG_JOBS: '1'
  stdin_open: true
  volumes:
    - /bigdata/steffen//run_daily_weather_owm/_out01:/in
    - /bigdata/steffen//run_daily_weather_owm/_work02:/work
    - /bigdata/steffen//run_daily_weather_owm/_out02:/out
  tty: true
  user: 1000:1000
  command:
    - '0'
    - /code
    - /in
    - /work
    - /out
  labels:
    io.rancher.container.pull_image: always
    io.rancher.container.start_once: 'true'
A.8. 10_unzip

This script unzip a zip-archive without folder information. The files in the archive are moved to the out directory.

A.8.1. Volumes

Volume setup. Must match arguments:

- /code - internal location for scripts in the container (created internally)
- /in - not used by this container (mandatory)
- /work - used for intermediate files (mandatory)
- /out..--location for process results for next step in the pipeline (mandatory)

A.8.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)

Example params: 0 /code /in /work /out

To enable logging set the environment variable LOG_JOBS to 1

A.8.3. Source

Docker image: datagraft/test_06_fetch_weather_owm

Source code: https://github.com/ew-shopp/weather_import/tree/master/docker_daily_import/06_fetch_weather_owm

A.8.4. Rancher Setup Example:

```sh
00-Unzip:
  image: datagraft/lib_10_unzip
  environment:
    LOG_JOBS: '1'
    stdin_open: true
    volumes:
```

EW_Shopp GA number: 732590 H2020-ICT-2016-2017/H2020-ICT-2016-1
- /bigdata/steffen/run_bc4_cat_germany/in:/in
- /bigdata/steffen/run_bc4_cat_germany/_work00:/work
- /bigdata/steffen/run_bc4_cat_germany/_out00:/out

tty: true
user: 1000:1000
command:
  - '0'
  - /code
  - /in
  - /work
  - /out
labels:
  io.rancher.container.pull_image: always
  io.rancher.container.start_once: 'true'
A.9. **11_tsv2csv**

This script converts a tab-separated datafile to a comma-separated datafile. The converted files are moved to the out directory.

A.9.1. **Volumes**

Volume setup. Must match arguments:

- `/code` - internal location for scripts in the container (created internally)
- `/in` - not used by this container (mandatory)
- `/work` - used for intermediate files (mandatory)
- `/out` - location for process results for next step in the pipeline (mandatory)

A.9.2. **Arguments**

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)

Example params: `0 /code /in /work /out`

To enable logging set the environment variable LOG_JOBS to 1

A.9.3. **Source**

Docker image: `datagraft/lib_11_tsv2csv`


A.9.4. **Rancher Setup Example:**

```
01-tsv2csv:
  image: datagraft/lib_11_tsv2csv
  environment:
    LOG_JOBS: '1'
  stdin_open: true
  volumes:
    - /bigdata/steffen/run_bc4_cat_germany/_out00:/in
```
- /bigdata/steffen/run_bc4_cat_germany/_work01:/work
  - /bigdata/steffen/run_bc4_cat_germany/in_all_kwds:/out
  tty: true
  user: 1000:1000
  command:
    - '0'
    - /code
    - /in
    - /work
    - /out
  labels:
    io.rancher.container.pull_image: always
    io.rancher.container.start_once: 'true'
A.10. **12_split**

This script splits a csv file if its above 800000 rows. The header row is not duplicated. The splitted files are moved to the out directory.

A.10.1. **Volumes**

Volume setup. Must match arguments:

- `/code` - internal location for scripts in the container (created internally)
- `/in` - not used by this container (mandatory)
- `/work` - used for intermediate files (mandatory)
- `/out` - location for process results for next step in the pipeline (mandatory)

A.10.2. **Arguments**

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to `/code`
- Name of input location. Set to `/in` (must match given volume name)
- Name of work location. Set to `/work` (must match given volume name)
- Name of out location. Set to `/out` (must match given volume name)

Example params: 0 /code /in /work /out

To enable logging set the environment variable LOG_JOBS to 1

A.10.3. **Source**

Docker image: [datagraft/lib_11_tsv2csv](https://github.com/ew-shopp/ingestion_lib/tree/master/12_split)

Source code: [https://github.com/ew-shopp/ingestion_lib/tree/master/12_split](https://github.com/ew-shopp/ingestion_lib/tree/master/12_split)
A.11. **13_transform**

This script fetches a csv file and applies a transformation to it. The transformation is given as a .jar file generated by the Grafterizer system.
The script monitors the input directory for csv files to upload. If not file present it will sleep for 60sec before trying again.
Depending on the size of the transformation input the container may require memory reservation at start-up.

A.11.1. **Volumes**

Volume setup. Must match arguments:

- /code - internal location for scripts in the container (created internally)
- /in - used for passing input files to this container (mandatory)
- /work - used for intermediate files (mandatory)
- /out -... location for process results for next step in the pipeline (mandatory)
- /transformation... - location for transformation JAR file (match path in argument)

A.11.2. **Arguments**

Arguments are passed to the container entrypoint:

Container entrypoint params:

- Max wait for file count. Set to '0' (how many idle 60 sleep periods before terminating)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)
- Path to transformation. Set to '/transformation/xxx.jar' (must match given volume name)

Example params: 0 /code /in /work /out /transformation/transformation.jar

A.11.3. **Source**

Docker image: datagraft/lib_13_transform


[https://github.com/ew-shopp/weather_import/tree/master/docker_daily_import/06_fetch_weather_owm](https://github.com/ew-shopp/weather_import/tree/master/docker_daily_import/06_fetch_weather_owm)
A.11.4. **Rancher Setup Example:**

```
transform-own-data:
  image: datagraft/grafterizer-jar-executor
  stdin_open: true
  volumes:
  - /bigdata/steffen/run_daily_weather_owm/_out00:/in
  - /bigdata/steffen/run_daily_weather_owm/_work01:/work
  - /bigdata/steffen/run_daily_weather_owm/_out01:/out
  - /bigdata/steffen/run_daily_weather_owm/transformation:/transformation
  tty: true
  mem_reservation: 4194304000
  user: 1000:1000
  command:
    - '0'
    - /code
    - /in
    - /work
    - /out
    - /transformation/transformation.jar
  labels:
    io.rancher.container.pull_image: always
```
A.12. **14_toarango**

Transforms a CSV file to document and edge collections based on an external JS-script. See [https://github.com/datagraft/datagraft-csv-to-arangodb](https://github.com/datagraft/datagraft-csv-to-arangodb) for details.

A.12.1. **Volumes**

Volume setup. Must match arguments:

- `/code` - internal location for scripts in the container (created internally)
- `/in` - not used by this container (mandatory)
- `/work` - used for intermediate files (mandatory)
- `/out` - location for process results for next step in the pipeline (mandatory)
- `/transformation` - location for transformation scripts (match path in argument)

A.12.2. **Arguments**

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)
- Path to transformation. Set to '/transformation/xxx.js' (must match given volume name)

Example params: 0 /code /in /work /out /transformation/transformscript.js

To enable logging set the environment variable LOG_JOBS to 1

A.12.3. **Source**

Docker image: [datagraft/lib_14_toarango](https://github.com/ew-shopp/ingestion_lib/tree/master/14_toarango)

Source code: [https://github.com/ew-shopp/ingestion_lib/tree/master/14_toarango](https://github.com/ew-shopp/ingestion_lib/tree/master/14_toarango)

A.13. **15_jot_categorize**
This script correlates keywords in the input file to categories given as start-up information. The categoriser is documented in https://github.com/JozefStefanInstitute/ew-shopp-public/tree/master/keyword_clustering

The keyword to category mapping are normalized and joined with the original input file. The result is a new file with category columns added.

### A.13.1. Volumes

Volume setup. Must match arguments:

- `/code` - internal location for scripts in the container (created internally)
- `/in` - not used by this container (mandatory)
- `/work` - used for intermediate files (mandatory)
- `/out` - location for process results for next step in the pipeline (mandatory)

### A.13.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

**Container entrypoint params:**

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)

**Example params:** 0 /code /in /work /out

To enable logging set the environment variable LOG_JOBS to 1

The following parameters are passed via environment variables:

- **JOB_CAT_FASTTEXT_FULL_PATH**: Path to the FastText model binary file.
- **JOB_CAT_EMBEDDER_FULL_PATH**: Path to the embedder parameters json file.
- **JOB_CAT_CATEGORIES_FULL_PATH**: Path to the categories file.
- **JOB_CAT_CATEGORIES_COLUMN**: Name of column containing categories in the categories csv file. (default: Category)
- **JOB_CAT_CATEGORIES_ID_COLUMN**: Name of column containing category ids in the categories csv file. (default: CategoryID).
- **JOB_CAT_KEYWORDS_COUNTRY_CODE**: Country code to be used from the keywords file.
- **JOB_CAT_KEYWORDS_CHUNK_SIZE**: (Optional param) Number of keywords to categorize in one operation. Default 100000, reduce if memory issues.
- **JOB_NOR_JAR_FULL_PATH**: Full path to normalized jar file from grafterizer.
A.13.3. Source

Docker image: datagraft/lib_15_jot_categorize

Source code: https://github.com/ew-shopp/ingestion_lib/tree/master/15_jot_categorize

A.13.4. Rancher Setup Example:

```
02-categoriser:
  image: datagraft/lib_15_jot_categorize
  environment:
    LOG_JOBS: '1'
    JOB_CAT_FASTEXT_FULL_PATH: /categoriser/cc.de.300.bin
    JOB_CAT_EMBEDDER_FULL_PATH: /categoriser/de_embedder.json
    JOB_CAT_CATEGORIES_FULL_PATH: /categoriser/productsservices.csv
    JOB_CAT_CATEGORIES_COLUMN: Category
    JOB_CAT_CATEGORIES_ID_COLUMN: Criterion ID
    JOB_CAT_KEYWORDS_COUNTRY_CODE: DE
    JOB_NOR_JAR_FULL_PATH: /categoriser/normalize-clusters.jar
    stdin_open: true
  volumes:
    - /bigdata/steffen/run_bc4_cat_germany/_out01:/in
    - /bigdata/steffen/run_bc4_cat_germany/_work02:/work
    - /bigdata/steffen/run_bc4_cat_germany/_out02:/out
    - /bigdata/steffen/run_bc4_cat_germany/categoriser:/categoriser
  tty: true
  mem_reservation: 15728640000
  user: 1000:1000
  command:
    - '0'
    - /code
    - /in
    - /work
    - /out
  labels:
    io.rancher.container.pull_image: always
    io.rancher.container.start_once: 'true'
```
A.14. 16_jot_aggregate_kwds

This script aggregate keywords from all input files into a common list of unique keywords. The output file is used for finding the most popular keywords for a set of categories.

A.14.1. Volumes

Volume setup. Must match arguments:

- /code - internal location for scripts in the container (created internally)
- /in - not used by this container (mandatory)
- /work - used for intermediate files (mandatory)
- /out...location for process results for next step in the pipeline (mandatory)

A.14.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

**Container entrypoint params:**
- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)

Example params: 0 /code /in /work /out

To enable logging set the environment variable LOG_JOBS to 1

The following parameters are passed via environment variables:

- JOB_CAT_KEYWORDS_COUNTRY_CODE: Country code to be used from the keywords file.

A.14.3. Source

Docker image: datagraft/lib_16_jot_aggregate_kwds

Source code: https://github.com/ew-shopp/ingestion_lib/tree/master/16_jot_aggregate_kwds

A.14.4. Rancher Setup Example:

```
10-all-kwds:
  image: datagraft/lib_16_jot_aggregate_kwds
  environment:
```

EW_Shopp  GA number: 732590  H2020-ICT-2016-2017/H2020-ICT-2016-1
A.15. 17_jot_closest_kwds

This script finds the n closest keywords in the input file for all categories given as startup information. The categoriser is documented in https://github.com/JozefStefanInstitute/ew-shopp-public/tree/master/keyword_clustering

The keyword to category mapping are normalized.

A.15.1. Volumes

Volume setup. Must match arguments:

- /code - internal location for scripts in the container (created internally)
- /in - not used by this container (mandatory)
- /work - used for intermediate files (mandatory)
- /out...location for process results for next step in the pipeline (mandatory)

A.15.2. Arguments

Arguments are passed to the container entrypoint and as environment variables:

Container entrypoint params:

- Max wait for file count. Set to '0' (not used by this container)
- Name of code location. Set to '/code'
- Name of input location. Set to '/in' (must match given volume name)
- Name of work location. Set to '/work' (must match given volume name)
- Name of out location. Set to '/out' (must match given volume name)

Example params: 0 /code /in /work /out
To enable logging set the environment variable LOG_JOBS to 1

The following parameters are passed via environment variables:

- **JOB_CAT_ALL_KEYWORDS_FULL_PATH**: Path to file with all keywords.
- **JOB_CAT_FASTTEXT_FULL_PATH**: Path to the FastText model binary file.
- **JOB_CAT_EMBEDDER_FULL_PATH**: Path to the embedder parameters json file.
- **JOB_CAT_CATEGORIES_FULL_PATH**: Path to the categories file.
- **JOB_CAT_CATEGORIES_COLUMN**: Name of column containing categories in the categories csv file. (default: Category)
- **JOB_CAT_CATEGORIES_ID_COLUMN**: Name of column containing category ids in the categories csv file. (default: CategoryID).
- **JOB_CAT_N_KEYWORDS**: Number of closest keywords to match each category.

### A.15.3. Source

**Docker image**: datagraft/lib_17_jot_closest_kwds

**Source code**: [https://github.com/ew-shopp/ingestion_lib/tree/master/17_jot_closest_kwds](https://github.com/ew-shopp/ingestion_lib/tree/master/17_jot_closest_kwds)

### A.15.4. Rancher Setup Example:

```yaml
11-closest-kwds:
  image: datagraft/lib_17_jot_closest_kwds
  environment:
    LOG_JOBS: '1'
    JOB_CAT_FASTTEXT_FULL_PATH: /categoriser/cc.de.300.bin
    JOB_CAT_EMBEDDER_FULL_PATH: /categoriser/de_embedder.json
    JOB_CAT_CATEGORIES_FULL_PATH: /categoriser/productsservices.csv
    JOB_CAT_CATEGORIES_COLUMN: Category
    JOB_CAT_CATEGORIES_ID_COLUMN: Criterion ID
    JOB_CAT_N_KEYWORDS: '1000'
    JOB_CAT_ALL_KEYWORDS_FULL_PATH: /in/unique_keywords.csv
  stdin_open: true
  volumes:
  - /bigdata/steffen/run_bc4_cat_germany/out_all_kwds:/in
  - /bigdata/steffen/run_bc4_cat_germany/work_closest_kwds:/work
  - /bigdata/steffen/run_bc4_cat_germany/out_closest_kwds:/out
  - /bigdata/steffen/run_bc4_cat_germany/categoriser:/categoriser
  tty: true
  mem_reservation: 15728640000
  user: 1000:1000
  command:
  - '/code'
  - '/in'
  - '/work'
  - '/out'
  labels:
    io.rancher.container.pull_image: always
    io.rancher.container.start_once: 'true'
    02-categoriser:
      image: datagraft/lib_15_jot_categorize
      environment:
```

EW_Shopp  GA number: 732590  H2020-ICT-2016-2017/H2020-ICT-2016-1
A.16. 18_fetch_custom_events

This module fetches custom events data from an EW-Shopp enabled custom events API. It connects to a custom events endpoint and downloads events (in json-ld format) for starting from a specific date and for a certain number of days. The downloaded custom events are eventually moved to the out directory.

The time range to fetch data for is given in a job-description-file in key-value format (.properties). This makes it possible to have this container waiting for jobs and fetch custom events data on a periodic basis.

Example of the configuration file:

```
API=https://virtserver.swaggerhub.com/EW-Shopp/EW-Shopp_Event_API/2.2.0/event/
num_days=5
starting_date=2018-01-01
results_dir=events
outputOnFile=enabled
fileName=output.json
```

A.16.1. Volumes

Volume setup. Must match arguments:

- /code - internal location for scripts in the container (created internally)
• /in - used by this container to provide the configuration file (mandatory)
• /work - used for intermediate files
• /out...location for process results for next step in the pipeline (mandatory)

### A.16.2. Arguments

Arguments (optional) can be passed to the container entrypoint and as environment variables. They are the same of the properties file, that is:

- API, the EW-Shopp enabled custom event endpoint
- num_days, number of days to be fetched
- starting_date, the date from which the module has to download the events
- results_dir, directory where to save the events
- outputOnFile, enable or disable the process of direct storing the events in ArangoDB
- fileName, the name of the file generated if the storing the results in ArangoDB is disabled

### A.16.3. Source

Docker image: datagraft/lib_18_fetch_custom_events

Source code: [https://github.com/ew-shopp/ingestion_lib/tree/master/18_fetch_custom_events](https://github.com/ew-shopp/ingestion_lib/tree/master/18_fetch_custom_events)