

Project Title	High-performance data-centric stack for big data applications and operations
Project Acronym	BigDataStack
Grant Agreement No	779747
Instrument	Research and Innovation action
Call	Information and Communication Technologies Call (H2020-ICT-2016-2017)
Start Date of Project	01/01/2018
Duration of Project	36 months
Project Website	http://bigdatastack.eu/

D5.1 – WP 5 Scientific Report and Prototype Description - Y1

Work Package	WP5 – WP5 Scientific Report and Prototype Description
Lead Author (Org)	Amaryllis Raouzaïou (ATC)
Contributing Author(s) (Org)	Amaryllis Raouzaïou (ATC), Konstantinos Giannakakis (ATC), George Kousiouris (UPRC), Sophia Karagiorgou (UBI), Nikos Lykousas (UBI), Pavlos Kranas (LXS), Ricardo Manuel Pereira Vilaça (LXS), Jacob Roldan (LXS), Francisco Ballesteros (LXS), Patricio Martinez (LXS), Christos Doukeridis (UPRC), Peter Jason Gould (UPRC), Ismael Cuadrado-Cordero (ATOS), Orlando Avila-García (ATOS), Marta Patiño (UPM), Richard McCreadie (GLA), Stathis Plitsos (DANAOS)
Due Date	30.11.2018
Date	30.11.2018
Version	1.0

Dissemination Level

<input checked="" type="checkbox"/>	PU: Public (*on-line platform)
<input type="checkbox"/>	PP: Restricted to other programme participants (including the Commission)
<input type="checkbox"/>	RE: Restricted to a group specified by the consortium (including the Commission)
<input type="checkbox"/>	CO: Confidential, only for members of the consortium (including the Commission)

Versioning and contribution history

Version	Date	Author	Notes
0.1	01.10.2018	Amaryllis Raouzaïou (ATC)	Initial ToC
0.2	08.10.2018	Konstantinos Giannakakis (ATC), Amaryllis Raouzaïou (ATC), George Kousiouris (UPRC)	Updated ToC
0.3	25.10.2018	Konstantinos Giannakakis (ATC)	Contribution in Sections 5 and 9
0.4	26.10.2018	Sophia Karagiorgou (UBI), Nikos Lykousas (UBI)	Contribution in Section 7
0.5	30.10.2018	Amaryllis Raouzaïou (ATC) Pavlos Kranas (LXS)	Contribution in Sections 2, 3, 4, contribution in Section 8
0.6	19.11.2018	Christos Doulkeridis (UPRC), Peter Jason Gould (UPRC)	Contribution in Section 6
0.7	22.11.2018	Sophia Karagiorgou (UBI), Nikos Lykousas (UBI), George Kousiouris (UPRC), Konstantinos Giannakakis (ATC)	Revisions and edits in Section 7, contribution in Section 8, revisions and edits in Sections 5 and 9
0.8	23.11.2018	Amaryllis Raouzaïou (ATC)	Contribution in Sections 2, 3, 4, 10
0.9	29.11.2018	Amaryllis Raouzaïou (ATC), Konstantinos Giannakakis (ATC), Sophia Karagiorgou (UBI)	Revisions and edits after the internal review of the document.
1.0	04.12.2018	Amaryllis Raouzaïou (ATC)	Final version

Disclaimer

This document contains information that is proprietary to the BigDataStack Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to a third party, in whole or parts, except with the prior consent of the BigDataStack Consortium.

Table of Contents

Table of Contents	3
List of tables	4
List of figures	5
1 Executive Summary	6
2 Introduction	7
2.1 Relation to other deliverables	7
2.2 Document structure	7
3 Solution Architecture	8
3.1 Vision	8
3.2 Platform Roles	9
3.3 Design	10
4 Implementation and Experimentation	12
4.1 Experimental Setting	12
4.2 Implementation Roadmap	13
5 Process Modelling framework	14
5.1 Anticipated functionalities / requirements	14
5.2 Specification / Design	16
5.3 Early prototype	17
5.4 Use case mapping	18
5.5 Experimental Plan	18
5.6 Next steps	19
6 Process Mapping	20
6.1 Anticipated functionalities / requirements	20
6.2 Specification / Design	22
6.3 Early prototype	23
6.4 Use case mapping	24
6.5 Experimental Plan	24
6.6 Next steps	25
7 Data Toolkit	26
7.1 Anticipated functionalities / requirements	26
7.2 Specification / Design	28
7.3 Early prototype	29
7.4 Use case mapping	31
7.5 Experimental Plan	33
7.6 Next steps	33
8 Application Dimensioning Workbench	34
8.1 Anticipated functionalities / requirements	35
8.2 Specification / Design	41
8.3 Early prototype	49
8.4 Use case mapping	52
8.5 Implementation and Experimentation	58
8.6 Next steps	60
9 Adaptable Visualizations	62
9.1 Anticipated functionalities / requirements	62
9.2 Specification / Design	63

9.3	Early prototype	63
9.4	Experimental Plan	64
9.5	Next steps	64
10	Conclusions	65
	References	66

List of tables

TABLE 1 – BIGDATASTACK PLATFORM ROLES RELEVANT TO DIMENSIONING, MODELLING & INTERACTION SERVICES	10
TABLE 2 – IMPLEMENTATION ROADMAP FOR DIMENSIONING, MODELLING & INTERACTION SERVICES	13
TABLE 3 – SYSTEM REQUIREMENT (1) FOR PROCESS MODELLING FRAMEWORK	14
TABLE 4 – SYSTEM REQUIREMENT (2) FOR PROCESS MODELLING FRAMEWORK	14
TABLE 5 – SYSTEM REQUIREMENT (3) FOR PROCESS MODELLING FRAMEWORK	15
TABLE 6 – SYSTEM REQUIREMENT (4) FOR PROCESS MODELLING FRAMEWORK	15
TABLE 7 – SYSTEM REQUIREMENT (5) FOR PROCESS MODELLING FRAMEWORK	15
TABLE 8 – SYSTEM REQUIREMENT (6) FOR PROCESS MODELLING FRAMEWORK	16
TABLE 9 – SYSTEM REQUIREMENT (7) FOR PROCESS MODELLING FRAMEWORK	16
TABLE 10 – SYSTEM REQUIREMENT (8) FOR PROCESS MODELLING FRAMEWORK	16
TABLE 11 – SYSTEM REQUIREMENT (1) FOR PROCESS MAPPING	21
TABLE 12 – SYSTEM REQUIREMENT (2) FOR PROCESS MAPPING	21
TABLE 13 – SYSTEM REQUIREMENT (3) FOR PROCESS MAPPING	21
TABLE 14 – SYSTEM REQUIREMENT (4) FOR PROCESS MAPPING	21
TABLE 15 – SYSTEM REQUIREMENT (1) FOR DATA TOOLKIT	27
TABLE 16 – SYSTEM REQUIREMENT (2) FOR DATA TOOLKIT	27
TABLE 17 – SYSTEM REQUIREMENT (3) FOR DATA TOOLKIT	27
TABLE 18 – SYSTEM REQUIREMENT (4) FOR DATA TOOLKIT	27
TABLE 19 – SYSTEM REQUIREMENT (1) FOR PATTERN GENERATOR	35
TABLE 20 – SYSTEM REQUIREMENT (2) FOR PATTERN GENERATOR	36
TABLE 21 – SYSTEM REQUIREMENT (3) FOR PATTERN GENERATOR	36
TABLE 22 – SYSTEM REQUIREMENT (4) FOR PATTERN GENERATOR	36
TABLE 23 – SYSTEM REQUIREMENT (5) FOR PATTERN GENERATOR	36
TABLE 24 – SYSTEM REQUIREMENT (6) FOR PATTERN GENERATOR	37
TABLE 25 – SYSTEM REQUIREMENT (1) FOR ADW CORE	38
TABLE 26 – SYSTEM REQUIREMENT (2) FOR ADW CORE	38
TABLE 27 – SYSTEM REQUIREMENT (3) FOR ADW CORE	38
TABLE 28 – SYSTEM REQUIREMENT (4) FOR ADW CORE	39
TABLE 29 – SYSTEM REQUIREMENT (5) FOR ADW CORE	39
TABLE 30 – SYSTEM REQUIREMENT (6) FOR ADW CORE	40
TABLE 31 – SYSTEM REQUIREMENT (7) FOR ADW CORE	40
TABLE 32 – SYSTEM REQUIREMENT (8) FOR ADW CORE	40
TABLE 33 – SYSTEM REQUIREMENT (9) FOR ADW CORE	41
TABLE 34 – ADW CORE API	49
TABLE 35 – DETAILED WORKLOAD SPECIFICATION PER UC TEMPLATE AND REAL TIME SHIP MANAGEMENT INSTANTIATION	55
TABLE 36 – LXS IDENTIFICATION OF DEPLOYMENT COMBINATIONS	56
TABLE 37 – CEP IDENTIFICATION OF DEPLOYMENT COMBINATIONS	58
TABLE 38 – SYSTEM REQUIREMENT (1) FOR ADAPTABLE VISUALIZATIONS	62
TABLE 39 – SYSTEM REQUIREMENT (2) FOR ADAPTABLE VISUALIZATIONS	62
TABLE 40 – SYSTEM REQUIREMENT (3) FOR ADAPTABLE VISUALIZATIONS	63

List of figures

FIGURE 1 – BIGDATASTACK CORE PLATFORM CAPABILITIES (EXTRACTED FROM D2.4).....	8
FIGURE 2 - DIMENSIONING PHASE	9
FIGURE 3 – DIMENSIONING, MODELLING AND INTERACTION SERVICES OF BIGDATASTACK	11
FIGURE 4 - NODE-RED	17
FIGURE 5 - NODE-RED EDITING NODE	18
FIGURE 6 - RSM SCENARIO	18
FIGURE 7 – DESIGN OF PROCESS MAPPING.....	22
FIGURE 8 – UML DIAGRAM.....	29
FIGURE 9 – REGISTRATION OF DIFFERENT ANALYTIC PROCESSES	30
FIGURE 10 – COMPOSITION OF AN INDICATIVE ANALYTIC PROCESS.....	30
FIGURE 11 – STEPS PERFORMED TOWARDS THE CREATION OF AN EXECUTABLE GRAPH.....	31
FIGURE 12 – MAPPING OF DATA TOOLKIT WITH RSM UC	32
FIGURE 13 – MAPPING OF DATA TOOLKIT WITH CONNECTED CONSUMER (CC) UC.....	32
FIGURE 14 -GENERIC INFORMATION FLOW FOR ADW FROM D2.4.....	34
FIGURE 15 - ADW DESIGN BENCHMARK RUN SYSTEM.....	42
FIGURE 16 - ADW CREATE MODEL SYSTEM	43
FIGURE 17 - ADW REQUEST PREDICTION SYSTEM.....	44
FIGURE 18 - BENCHMARK DESIGN ARCHITECTURE	45
FIGURE 19 - CLASS DIAGRAM OF THE ELEMENTS IN THE BENCHMARKING COMPONENTS.....	46
FIGURE 20 - MODEL CREATION ARCHITECTURE	46
FIGURE 21 - ADS-PATTERN GENERATION ARCHITECTURE.....	47
FIGURE 22 - ANNOTATE PLAYBOOK ARCHITECTURE	48
FIGURE 23 - PATTERN GENERATION PREFERENCES SETTING UI	50
FIGURE 24 - INDICATIVE UI WITH POPULATED DIMENSIONING ESTIMATES FOR PATTERN SELECTION	50
FIGURE 25 - ADW CORE SERVICES LAYER EXAMPLE IMPLEMENTATION	51
FIGURE 26 - WORKFLOW DIAGRAM OF INTERACTION BETWEEN COMPONENTS.....	52
FIGURE 27 – BASE ARCHITECTURE FOR VISUALIZING BIG DATA.....	63

1 Executive Summary

BigDataStack delivers a complete high-performant stack of technologies addressing the needs of data operations and applications. BigDataStack holistic solution incorporates approaches for data-focused application analysis and dimensioning, and process modelling towards increased performance, agility and efficiency. A toolkit allowing the specification of analytics tasks in a declarative way, their integration in the data path, as well as an adaptive visualization environment, realize BigDataStack's vision of openness and extensibility.

The main objective of the dimensioning, modelling and interaction services building block of BigDataStack is to provide all the interaction mechanisms including the Process Modelling framework, the Data Toolkit, the Dimensioning Workbench, and the Visualization environment. These are required in order to exploit the added-value services of the “underlying” BigDataStack data-driven infrastructure management and the Data as a Service offerings.

The Process Modelling Framework will allow for declarative and flexible modelling of process analytics. Functionality-based process modelling will be concretized to technical-level process mining analytics, while a feedback loop will be implemented towards overall process optimization and adaptation.

The Process Mapping component targets the problem of identifying or recommending the best algorithm from a set of candidate algorithms, given a specific data analysis task, in an automatic way. Its role is to automatically map a step of a process to a specific algorithmic instance from a given pool of algorithms, thereby achieving “process mapping”.

The Data Toolkit facilitates Data Scientists in building operational analytic workflows by means of data pipelines through Directed Acyclic Graphs (DAGs).

The Application Dimensioning Workbench aims to provide insights regarding the required infrastructure resources for the data services and application components (micro-services), linking the used resources with load and expected QoS levels.

Finally, Adaptable Visualizations will present graphs and reports of data and analytics outcomes (as well as monitoring information from application, resources and data levels) in an adaptive and interactive way.

Thus, the current deliverable presents the vision for the Dimensioning, Modelling and Interaction Services of BigDataStack, the context, the goal and the main services realizing this vision. Moreover, the corresponding roles interacting with these services and the design of the proposed solution are discussed in the current document. The deliverable contains the use cases to be supported and the expected progress until M18, while it also describes in detail the different components, along with the corresponding requirements and the next steps. Updated versions of this report are planned for M23 and M34 respectively.

2 Introduction

2.1 Relation to other deliverables

The current deliverable, the first BigDataStack deliverable concerning **Dimensioning, Modelling and Interaction Services** (D5.2 and D5.3 are scheduled for M23 and M34 respectively) is related to several other BigDataStack deliverables in a direct or indirect way.

D2.1 (State of the art and Requirements analysis - I) identifies and specifies the technical requirements for BigDataStack both through use case (UC) providers and technology providers, while *D2.4 (Conceptual model and Reference architecture - I)* provides information about the key functionalities of the overall architecture, the interactions between the main building blocks and their components, along with a first version of the internals of these components regarding research approaches to be realised during the course of the project.

We should also state that the Requirement Tables of the corresponding components of the Dimensioning, Modelling and Interaction Services (Tables 3-33 and 38-40) are compiled together with the rest of requirements of BigDataStack in *D2.2 (Requirements & State of the Art Analysis - II)*; they are included in this document for the reader's convenience.

Finally, D3.1 (WP3 Scientific Report and Prototype description - Y1) and D4.1 (WP4 Scientific Report and Prototype description - Y1) are the deliverables which, in combination with D5.1, present the current technical status (dealing with **Data-driven Infrastructure Management** and **Data as a service** respectively) of BigDataStack project.

2.2 Document structure

Section 3 gives an overview of the various components, while *Section 4* provides information for the experimental setting and implementation roadmap. *Sections 5 to 9* follow the data flow in the dimensioning, modelling and interaction services' block of BigDataStack architecture and are dedicated to each one of the different components, namely Process Modelling framework, Process Mapping, Data Toolkit, Application Dimensioning Workbench and Adaptable Visualizations.

3 Solution Architecture

This section describes the technical solution for the Dimensioning, Modelling & Interaction Services of BigDataStack. Firstly, it gives a general overview of the BigDataStack capabilities (context, goal, main functions or services); secondly, it enumerates the platform roles interacting with these services; and finally, it describes the design of the proposed solution.

3.1 Vision

BigDataStack offerings are depicted through a full stack aiming to facilitate the needs of data operations and applications (all of which tend to be data-intensive) in an optimized way. The BigDataStack core platform capabilities are depicted in Figure 1 and further analysed in D2.4.

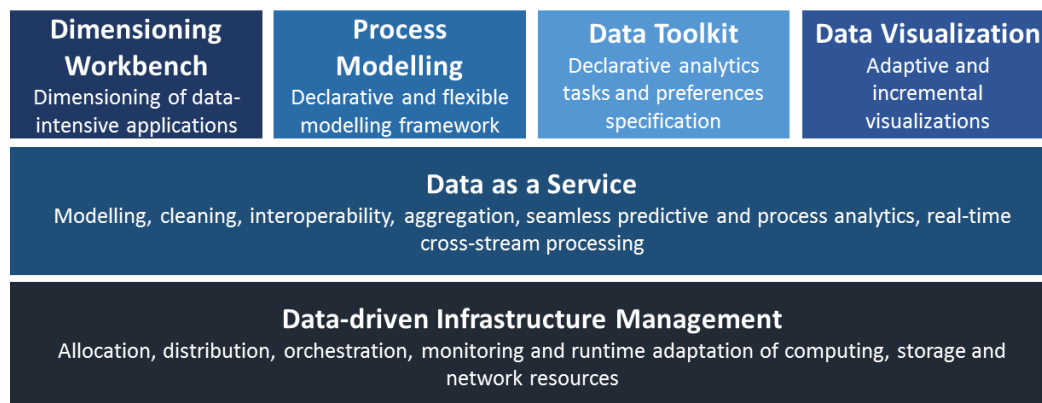


Figure 1 – BigDataStack core platform capabilities (extracted from D2.4)

These six BigDataStack core platform capabilities are envisioned to achieve the business goals or expectations from the different stakeholders. **Dimensioning Workbench**, **Process Modelling**, **Data Toolkit** and **Data Visualization** are the four core offerings of BigDataStack platform that are discussed in the present deliverable.

The goal of **Data Visualization** is to present graphs and reports of data and analytics outcome in an adaptive and interactive way, while the **Data Toolkit** facilitates BigDataStack users build operational analytic workflows by means of data pipelines through Directed Acyclic Graphs (DAGs). In the case of **Process Modelling**, the goal is to provide a framework that will allow for declarative and flexible modelling of process analytics, while the **Dimensioning Workbench** will enable the dimensioning of applications in terms of predicting the required data services, their interdependencies with the application micro-services and the necessary underlying resources.

These capabilities are mainly engaged in **Entry** and **Dimensioning** Phases of BigDataStack (see D2.4).

During the Entry Phase:

1. Data owners ingest their data in the BigDataStack-supported data stores through a unified API.
2. Given the stored data, the **Business Analysts** design and evaluate their business processes, using the user interface (UI) provided by the **Process Modelling** framework and the available list of “generic” processes. The compiled business workflow is

mapped to concrete tasks through the **Process Mapping** mechanism (incorporated in the **Process Modelling** framework).

3. The graph of services is made available to the **Data Scientists** through the **Data Toolkit**, where they can also specify their preferences and constraints.
4. The Data Scientists are also able to insert their tailor made Machine Learning (ML) algorithms facilitated by automated and managed processes, e.g. CRUD-ers will be made available on the support of this purpose.

The output of the Entry Phase is a playbook descriptor that is passed to the Application Dimensioning Phase in order to identify the resource needs for the services.

During the Dimensioning Phase (Figure 2):

1. The input from the Data Toolkit is used to define the composite application needs with relation to the required data services;
2. The identified/required data services are dimensioned (as well as all the application components, regarding their infrastructure resource needs), by exploiting a load injector generating different loads, to benchmark the services and analyse their resources and data requirements (e.g. volume, generation rate, legal constraints, etc.).

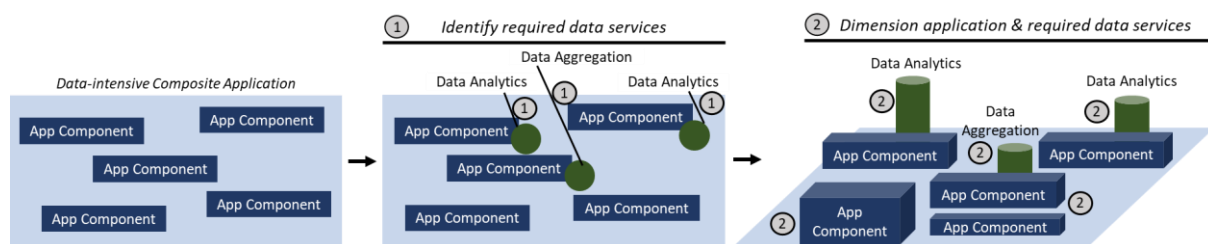


Figure 2 - Dimensioning Phase

The output of the dimensioning phase is an elasticity model, i.e. a mathematical function that describes the mapping of the input parameters (such as workload and QoS) to needed resource parameters (such as the number of VMs, bandwidth, latency etc.).

3.2 Platform Roles

Table 1 lists the BigDataStack roles relevant to Dimensioning, Modelling & Interaction Services (see the complete list of roles in Deliverable D2.1).

Id	Name	Description
ROL-02	Data Scientist	The process model is made available to the data scientist through the Data Toolkit. BigDataStack offers the Data Toolkit to enable data scientists both to easily ingest their analytics tasks, and to specify their preferences and constraints to be exploited during the dimensioning phase regarding the data services that will be used (for example preferences for the data cleaning service).

ROL-03	Business Analysts	BigDataStack offers the Process Modelling Framework allowing business users to model their functionality-based business processes and optimize them based on the outcomes of process analytics that will be triggered by BigDataStack. The business analyst can search processes from the list of available processes, create a flow of processes and set objectives for the overall flow or per process. The visual analytical reports are made available to the business analyst through the visualization layer.
ROL-04	Application Engineers and Application Service Owners	The updated model is made available to the application owner / engineer through the Application Dimensioning Workbench. BigDataStack offers the Application Dimensioning Workbench to enable application owners and engineers to experiment with their applications and dimension it in terms of its data needs and data-related properties.

Table 1 – BigDataStack Platform roles relevant to Dimensioning, Modelling & Interaction Services

3.3 Design

The conceptual view of Dimensioning, Modelling & Interaction Services consists of four main blocks, as summarized in the following paragraphs:

1. Process Modelling

The Process Modelling Framework allows for declarative and flexible modelling of process analytics, while the Process Mapping component targets the problem of identifying or recommending the best algorithm from a set of candidate algorithms.

2. Data Toolkit

The main objective of the data toolkit is to design and support data analysis workflows. It facilitates Business Analysts and Data Scientists in building operational analytic workflows, while it also interacts with the Adaptable Visualizations component.

3. Dimensioning Workbench

The Application Dimensioning Workbench (ADW) aims to provide insights regarding the required infrastructure resources for the data services and application components (micro-services), linking the used resources with load and expected QoS levels.

4. Adaptable Visualizations

Adaptable Visualizations component will present graphs and reports of data and analytics outcome in an adaptive and interactive way.

As it is depicted in Figure 3, typical Big Data flow starts from the Process Modelling Block (Process Modelling and Process Mapping), then the defined processes are further concretized through the Data Toolkit and its output will be passed to the Dimensioning Workbench. The analytics insights from the Data Toolkit feed the **Adaptable Visualisations** component.

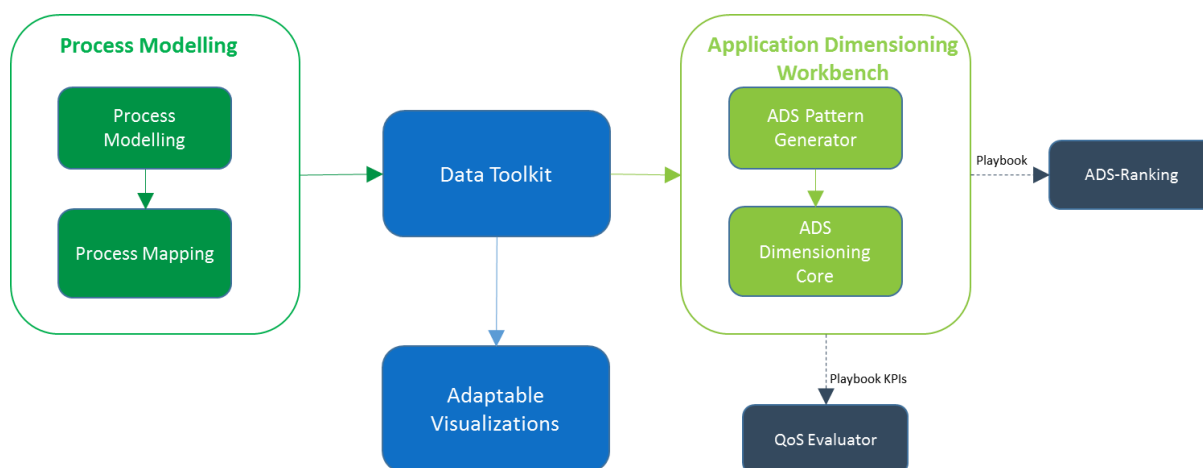


Figure 3 – Dimensioning, Modelling and Interaction Services of BigDataStack

4 Implementation and Experimentation

This section introduces the UC and scenarios to be supported in the incremental development of the solution.

4.1 Experimental Setting

This section introduces the use cases and the scenarios we are using to validate the different implementation increments (releases) of the Dimensioning, Modelling & Interaction Services. The BigDataStack use case we have chosen to test the different components presented in this deliverable is the **Real-time Ship Management (RSM)**: Maintenance and spare parts inventory planning & dynamic routing (see deliverable D2.1 section 4.1), provided by DANAOS. Some of the highlights of the use case are (please refer to D2.1 for the full description):

- Two key challenges in the ship management domain: (i) predictive maintenance combined with spare parts inventory planning, and (ii) dynamic routing;
- DANAOS, a leading international maritime player with more than 60 containerships, transporting millions of containers, sailing millions of miles to thousands of ports, and consuming millions of tons of fuel oil, which is a partner of BigDataStack, provides the consortium with real data in order to test the various components;
- Two different but complementary scenarios have been defined in the framework of RSM: (i) monitoring and predictive maintenance and (ii) requisition of a spare part and dynamic routing to the closest port where this part is available.

All the components of Dimensioning, Modelling and Interaction Services are involved in the different stages of **RSM**.

Process Modelling Framework allows an analyst to create predefined rules (rules engine component) and to actualise the required analytic tasks, through the definition of the business processes and the associated objectives, making available a high-level description of the required processes. Subsequently, the system, using **Process Mapping** component, will select from the available ML algorithms, the best performing for DANAOS dataset.

The output of this step is a workflow graph, containing the mappings of business processes to algorithms.

The processes included in this workflow graph will be further concretized through the **Data Toolkit**. Using the Data Toolkit, one can define the data ingestion and the necessary curation tasks for DANAOS dataset (weather data, tracks from vessels) and configure the runtime resources. Data Toolkit can also validate the end-to-end business objectives through the analytics insights feeding the **Adaptable Visualisations**.

The output of this step is a Playbook representing the grounded workflow for each process. It will be passed to the **Dimensioning Workbench** to identify the necessary resources for each node of the graph. The **Pattern Generator** subcomponent of the **Application Dimensioning Workbench (ADW)** is not explicitly linked to the particular UC; it forms part of the underlying application deployment backbone that supports all UCs of BigDataStack in order to identify how to deploy the user's application onto the cloud infrastructure. On the other hand, although Dimensioning core applies to the generic data services included in BigDataStack, it can be adapted to a specific UC, specifically with relation to aspects of workload, e.g. RSM contains tables that have more than 100 columns (extended description can be found at Section 8.4).

4.2 Implementation Roadmap

Table 2 summarises the plan for Dimensioning, Modelling & Interaction Services. M14 is the date of the next planned integration meeting, while M18 (June 2019) is a tentative date of the mid-project review.

	M12	M14	M18
Process Modelling Framework	Early Prototype	Use cases modelled, first round of feedback, finalization of specifications and interconnection requirements. Draft version using the implemented custom tool (see Section 5.3).	Updated working version, using the implemented custom tool.
Process Mapping	First version, containing basic functionality, serving as proof-of-concept	Improved version, including more thorough investigation of meta-features that can be exploited for meta-learning	Integrated version of Process Mapping with the Process Modelling Framework, subject to further improvements of its internal functionality in the 2 nd half of the project
Data Toolkit	Basic analytic workflows without validation	Simple end-to-end analytic workflows integrated with UIs, delivering valid Directed Acyclic Graphs (DAGs) with simple validation rules	End-to-end analytic workflows integrated with UIs, delivering valid DAGs with validation rules tailored to the UCs
Application Dimensioning Workbench	First version with containerized and configurable benchmark tools, initial version of the UIs	Integration between ADW components and external ones of BigDataStack	Benchmark runs with UC specific workloads and service configurations through the UI functionality
Adaptable Visualisations	Specifications	Interfaces with different components as data sources to be visualized	Early prototype with sample data

Table 2 – Implementation Roadmap for Dimensioning, Modelling & Interaction Services

5 Process Modelling framework

The Process Modelling Framework will allow for declarative and flexible modelling of process analytics. Functionality-based process modelling will then be concretized to technical-level process mining analytics, while a feedback loop will be implemented towards overall process optimization and adaptation.

5.1 Anticipated functionalities / requirements

The anticipated functionalities / requirements are described in the following tables (Table 3 - Table 10), that are compiled together with the rest of requirements of BigDataStack in D2.2.

	Id ¹	Level of detail ²	Type ³	Actor ⁴	Priority ⁵
	REQ-PMF-01	System and Software	USE	ROL-04	MAN
Name	UI/UX experience				
Description	The system should guide the users to complete the business diagram / flow with easy steps. It should clearly indicate what connections – interactions are possible and provide comprehensive error messages.				
Additional Information					

Table 3 – System Requirement (1) for Process Modelling Framework

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-02	System and Software	FUNC	ROL-04	MAN
Name	Multi-user support				
Description	Multiple users should be able to use the Process Modelling Framework and create diagrams at the same time. It should also support different roles: business analysts and data analysts. A business analyst will define a process in a higher level and a data analyst will provide the concrete implementations				
Additional Information					

Table 4 – System Requirement (2) for Process Modelling Framework

¹**Identifier:** To be used in D2.2 to allow for the correct traceability of requirements.

²**Level of detail:** Following the use of ISO/IEC/IEEE 29148:2011, we use the following levels: Stakeholder, System and Software (i.e., technology details).

³**Type:** Types of requirements are functional: FUNC (function), DATA (data); and non-functional: L&F (Look and Feel Requirements), USE (Usability Requirements), PERF (Performance Requirements), ENV (Operational/Environment Requirements), and SUP (Maintainability and Support Requirements).

⁴**Actor:** It needs to be either one of the BigDataStack platform roles identified in Section 3.2 or a system actor, e.g. another component or service.

⁵**Priority:** Requirements can have different priorities: MAN (mandatory requirement), DES (desirable requirement), OPT (optional requirement), ENH (possible future enhancement).

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-03	System and Software	FUNC	Business Analyst	MAN
Name	Process workflow creation				
Description	A business analyst should be able to create a process workflow in a higher level. The analyst will select nodes from a catalogue and using a drag-and-drop interface will link them together to create the flow.				
Additional Information					

Table 5 – System Requirement (3) for Process Modelling Framework

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-04	System and Software	FUNC	Data Analyst	MAN
Name	Process workflow configuration				
Description	The data analyst should be able to configure a process workflow with all the required details. The data analyst will set up the nodes parameters and define the rules for moving from one node to another.				
Additional Information					

Table 6 – System Requirement (4) for Process Modelling Framework

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-05	System and Software	FUNC	Data Analyst	MAN
Name	Process workflow export				
Description	The data analyst should be able to export the process workflow in BigDataStack format.				
Additional Information	The default format of the export will be in JSON. It will include information regarding the flows and their interconnections. Alternative export formats (YAML, Dockerfile) will be considered based on the requirements of other components. The user should be able to select the appropriate export format.				

Table 7 – System Requirement (5) for Process Modelling Framework

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-06	System and Software	FUNC	Business Analyst	MAN
Name	Support for end-to-end (in terms of process workflow) objectives				
Description	The business analyst should be able to define end-to-end objectives. These objectives do not apply to a single process, but to the workflow as a whole.				

Additional Information	
-------------------------------	--

Table 8 – System Requirement (6) for Process Modelling Framework

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-07	System and Software	FUNC	Business Analyst	MAN
Name	Process constraints				
Description	The business analyst should be able to set apply constraints per node / process of the workflow				
Additional Information					

Table 9 – System Requirement (7) for Process Modelling Framework

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-08	System and Software	FUNC	Business Analyst	MAN
Name	Edge constrains				
Description	The business analyst should be able to apply constraints / parameters per edge (i.e. connections between processes of the workflow).				
Additional Information					

Table 10 – System Requirement (8) for Process Modelling Framework

5.2 Specification / Design

The Process Modelling Framework prototype has been initially implemented by utilizing as a baseline Node-RED [1]. Node-RED is an open-source tool for creating and deploying processes with little or no code at all. It runs as a web server and provides a drag-and-drop interface for designing the process. When the process is ready, it can be deployed at the Node-RED server, by clicking Deploy. Its main use case is Internet of Things (IoT) and home automation.

The Process Modelling Framework prototype is initially built upon Node-RED functionalities and provides its own set of palettes. The available nodes in the palette should come from the Process Catalogue and be able to support all use cases. Instead of deploying a flow, it is easier to export a flow into the format required by other BigDataStack components.

The following scenario should be supported:

1. Search processes from the list of available processes in the process modelling framework
 - a. Search by functionality
 - b. Search by name
2. Create flow of processes

- a. Add processes through a drag & drop feature
 - b. Add processes following the search performed
 - c. Link processes (create flows)
3. Set objectives / constraints / parameters
 - a. Set objectives for the overall flow
 - b. Set objectives per process
 - c. Set parameters / rules that (when true) enable passing from one process to another in the flow
4. Enable parameters validation (e.g. if the business person sets an objective of data cleaning to be finished in less than 0.00001sec)
5. Export flow in a suitable format

5.3 Early prototype

An early prototype using Node-RED has been implemented. It is a fork of Node-RED GitHub project [2] and can run locally using node.js and npm. A Dockerfile is also provided so that it can be run inside a Docker container.

The prototype provides its own BigDataStack palette with some sample nodes that can be used to describe a BigDataStack workflow. Every node can be parameterized by double-clicking on it and editing its properties. Currently only exporting the flow to Node-RED's internal JSON format is supported. Node-RED palette and editing node are shown in Figure 4 and Figure 5 respectively.

It should be noted that a later version of the framework will be based on custom development and not re-use/extend an existing tool. Node-RED allows us to quickly create a prototype and engage all required members early enough, but it may not be adequate for the implementation of all required features. Towards this end, a custom tool will be implemented.

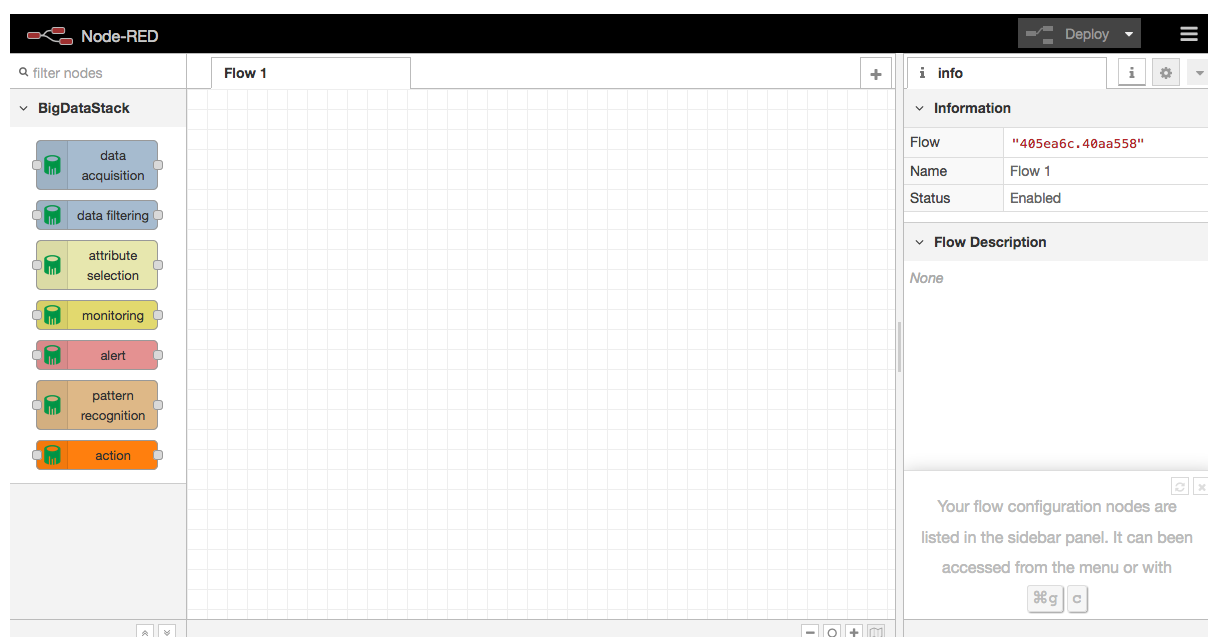


Figure 4 - Node-RED

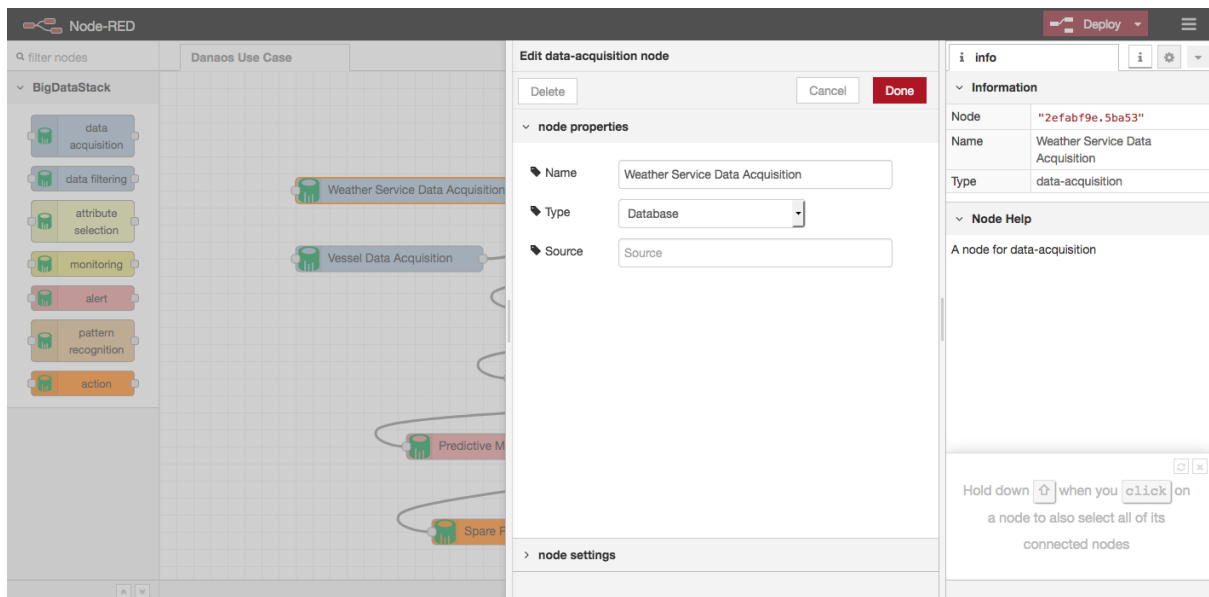


Figure 5 - Node-RED Editing node

5.4 Use case mapping

5.4.1 RSM scenario

Figure 6 depicts the RSM scenario as defined in the Process Modelling Framework Early Prototype.

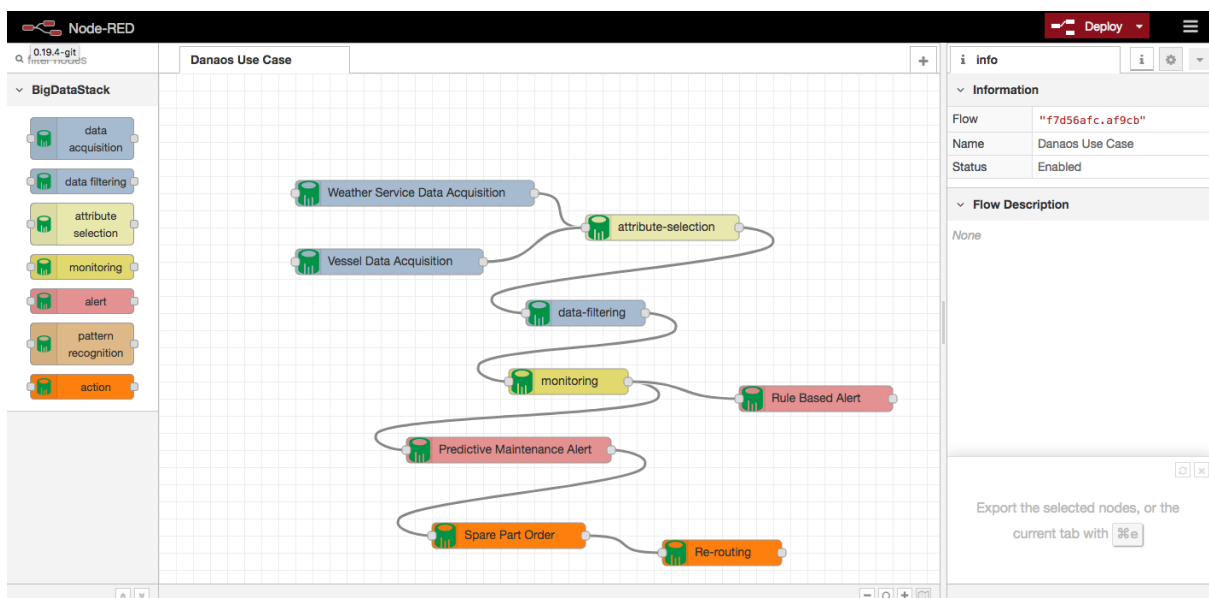


Figure 6 - RSM Scenario

5.5 Experimental Plan

In order to evaluate the Process Modelling Framework an experimental plan has been created. The goal of the first prototype is to initially model the scenarios of the use cases.

Both the Business Analyst and the Data Analyst should have all the required tools in their hands in order to model the scenarios. It should be validated that it is possible to use a drag-and-drop interface to model BigDataStack use cases and then export the model to an appropriate output format. The next step will be to implement a custom tool that will be able to load processes/nodes dynamically, integrate with recommendation engines and support advanced validation.

In terms of evaluation metrics and KPIs, the main objectives are:

- Successful modelling of all use cases
- Good UI/UX experience with emphasis on validation
- Seamless integration with other components

5.6 Next steps

Towards a complete Process Model Framework implementation, the following steps need to be completed:

- Complete palette for RSM scenario. All nodes should support the appropriate configuration and event parameters;
- Complete palette for Connected Consumer (CC) - ATOS WORDLINE UC scenario. Similar to RSM scenario full configuration should be possible;
- Export workflow to BigDataStack process format.

6 Process Mapping

The Process Mapping component targets the problem of identifying or recommending the best algorithm from a set of candidate algorithms, given a specific data analysis task, in an automatic way. Its role is to automatically map a step of a process to a specific algorithmic instance from a given pool of algorithms, thereby achieving “process mapping”.

6.1 Anticipated functionalities / requirements

The Process Modelling framework is used to create process models that contain different types of tasks, including data analysis tasks. In order to obtain an executable program from the process model, process mapping is required, which maps steps of the process model to concrete implementations. In some cases, this mapping is straightforward and can be easily derived. However, in other cases, most notably in machine learning (ML) tasks, a given task can be implemented using different alternative algorithms. Quite often, it is hard for ML experts to select the best performing algorithm, and even more so for the non-expert user. Consequently, there is a need for a system that identifies the most promising ML algorithm for the given task.

Hence, the key functionality targeted by the Process Mapping component is stated as follows. Given a machine learning task, a dataset, and a set of available ML algorithms that can handle the given task, the component selects (or recommends) the subset of ML algorithms with best performance. Essentially, the problem can be cast as a search problem, where the search space consists of the available ML algorithms, and the objective is to identify the best performing algorithms.

Obviously, covering all possible types of processes is a tedious task that goes beyond this project. In fact, previous EU projects, most notably METAL [11] and MiningMart [12], have focused on algorithm selection for specific problems. Instead, in the context of the Process Mapping component, the focus will be on Machine Learning tasks, since this is very important for the successful analysis of big data. Moreover, ML algorithm selection is challenging, because the connection between an ML algorithm and the characteristics of the data under analysis is still not well-understood. Towards this goal, the Process Mapping component follows a meta-learning approach [10]. We refer to [13] for a survey of the problem of meta-learning for algorithm selection, and also to recent notable works for classification [14] and clustering [15] (the former having been the object of much more extensive studies).

The anticipated functionalities / requirements are described in the following tables (Table 11-Table 14), that are compiled together with the rest of requirements of BigDataStack in D2.2.

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-01	Stakeholder	FUNC	ROL-04	MAN
Name	Compatibility with output of Process Modelling				
Description	The Process Mapping component is able to process the output of Process Modelling, in order to select appropriate ML algorithm(s) for specific Process steps.				

Additional Information	This requirement practically ascertains that the two components (Process Modelling and Process Mapping) are compatible and that the output of the first can be consumed by the second.
-------------------------------	--

Table 11 – System Requirement (1) for Process Mapping

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-02	Stakeholder	FUNC	ROL-04	MAN
Name	Extraction of metadata				
Description	Given a dataset, extract a set of metadata that is sufficient in order to discover similarities between datasets, in particular regarding the underlying data distributions and other statistical properties.				
Additional Information	The metadata should cover at least statistical and information-theoretic characterization of a given dataset.				

Table 12 – System Requirement (2) for Process Mapping

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-03	Stakeholder	FUNC	ROL-04	MAN
Name	Build and maintain a meta-knowledge repository				
Description	Collect and store information about datasets, metadata, and the performance of ML algorithms that have been executed on the datasets. This information is referred to as meta-knowledge, because it is essentially knowledge about the learning process. This meta-knowledge repository is going to be used for <i>meta-learning</i> , which is defined as the study of methods that exploit meta-knowledge to obtain efficient models and solutions by adapting machine learning processes.				
Additional Information	The meta-knowledge repository is augmented with information about the execution of ML algorithms on new datasets.				

Table 13 – System Requirement (3) for Process Mapping

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-04	Stakeholder	FUNC	ROL-04	MAN
Name	ML algorithm selection				
Description	Given a machine learning task, a dataset, and a set of available ML algorithms that can handle the given task, select (or recommend) the subset of ML algorithms with best performance.				
Additional Information	It assumes the availability of a pool of ML algorithms (e.g., a ML library) and an execution environment for running ML algorithms on different datasets and evaluating their result quality.				

Table 14 – System Requirement (4) for Process Mapping

6.2 Specification / Design

The inputs of the Process Mapping component consist of:

- The analysis task T (e.g., Regression, Classification, Clustering, etc.) that the user wishes to perform. This task is typically an individual step of a process model;
- Additional information that is dependent on the analysis task T (e.g., the response – predictor variables in the case of Supervised Learning, the desired number of clusters in the case of Clustering, etc.);
- A dataset D that is subject to the analysis task T .

The output of the Process Mapping component is a selected algorithm $A_i(T)$ from a set of available algorithms $\{A_1, A_2, \dots, A_n\}$ that are applicable to task T , which is predicted to be the most suitable for executing the data analysis task T at hand. In practice, the component achieves mapping of steps of a Process to concrete algorithms.

Figure 7 presents the design of the Process Mapping component, which is a refined version of the one reported in the global architecture (cf. deliverable D2.4). As already mentioned, the input is provided by a user that has a dataset D and needs to perform an analysis task T . In the first step, a descriptive model $M(D)$ of the input dataset D is generated. This is also referred to as metadata. This model can be conceived as a feature vector that contains various data characteristics, capturing different aspects of the dataset D , and aiming at providing sufficient information to be able to compute similarities between models at a later step. At this phase in the project, the focus is on features, such as: dimensionality, intrinsic dimensionality, cardinality, correlation between dimensions, entropy, mutual information, and sparsity-related statistics. However, it is foreseen that the set of features is extensible, and other features can be added in later phases of the project, based on further research and empirical evaluation.

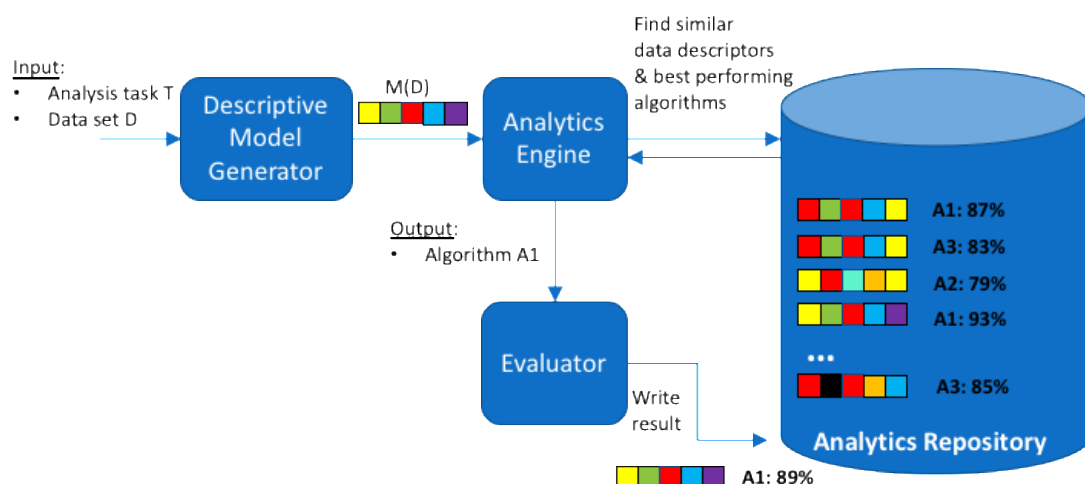


Figure 7 – Design of Process Mapping

In the second step, the Analytics Engine receives the model $M(D)$ and attempts to discover similar models $M(D')$ of any other dataset D' that has been processed in the past. Information about past models, algorithms that were executed on models, and evaluation results, is stored in the Analytics Repository. Discovering the most similar model $M^*(D')$ to the given

model $M(D)$ from a set of models $M(D')$ can be performed in different ways. One option is to use a properly defined similarity function that operates on two models and computes the similarity between them. In the case of feature vectors, potential similarity functions could be based on the cosine similarity, or on a weighted Euclidean distance. Another alternative is to learn such a similarity function, based on the data stored at the Analytics Repository, which can be used for training. In the example depicted in Figure 7, algorithm A1 is selected because it has been executed on a similar dataset to the one at hand and has produced the best result quality.

At the final step, the Evaluator executes the selected algorithm A1 on the dataset D, evaluates the result using a quality metric appropriate for task T, and records this result in the Analytics Repository (the meta-knowledge repository), in order for this information to be available for future analysis tasks.

6.3 Early prototype

The overall objective is to have a first version of the prototype by M18, which will enable the assessment of the result quality of Process Mapping, in order to identify potential improvements that need to be performed in the second half of the project. In addition, even though big data aspects are considered in the final implementation, the current focus is on rapid prototyping and quick evaluation of results, to have early feedback on the underlying methods employed. Therefore, parts of the prototype on M12 are built using standard technologies (e.g., python libraries, tools such as WEKA [5], etc.). However, it is foreseen that the final prototype will be based on big data technologies, and already parts of the prototype have become big data ready.

By M12, the early prototype of Process Mapping has focused on the following functionalities: (a) design and implementation of the Descriptive Model Generator, (b) designing the Analytics Engine, (c) rapid prototyping of the Evaluator module, and (d) building the Analytics Repository.

The features extracted from the Descriptive Model Generator are critical to achieve high result quality. To this end, a literature survey has been conducted in order to identify which features and metadata can be exploited, in order to model datasets adequately and enable discovering similarities in a subsequent phase. The current implementation uses (i) basic metadata such as dimensionality and cardinality, (ii) descriptive statistics per column (average, mean, st.deviation), (iii) statistical tests between columns (correlation coefficient), and (iv) information-theoretic measures (entropy, mutual information). A partial objective of the early prototype is to identify the limitations of using these measures, in order to identify additional measures that can be exploited for data representations in the future.

The Analytics Engine is the second critical module, as it exploits the generated descriptive models, in order to discover similarities between datasets. In the early prototype, the focus is on vector-based similarity functions that are suitable for high-dimensional representations, such as the cosine similarity. However, in the future, it is expected that alternative approaches will be considered, such as learning a similarity function by training a machine learning algorithm using historical data.

The Evaluator module aims at executing a selected algorithm on a given dataset and evaluating the result quality. For M12, we use off-the-shelf tools and libraries (e.g., python libraries, WEKA), which are easy to use. However, an instantiation of the prototype has been implemented in Spark's MLlib [6], which is going to be the final platform for parallel execution of scalable machine learning algorithms.

Finally, the Analytics Repository is currently implemented as a storage repository, maintaining data descriptors, results of algorithms execution, and quality indices for each executed algorithm. Later in the project, the plan is to implement this by using NoSQL technologies, in order to achieve scalability also at this level of the prototype, as well as to support flexibility in the schema and types of data and metadata that need to be maintained.

6.4 Use case mapping

Continuing on the example of Section 5, the Process Mapping component can be applied to the RSM UC, by selecting the mapping of a step to a ML task in an automatic way. For example, when the depicted pattern recognition node is used in a diagram, then its mapping to a ML algorithm is going to be performed automatically by the Process Mapping component.

Furthermore, it should be clarified that the underlying mechanism of the Process Mapping component is independent of the actual UC, and can be applied on other UCs in BigDataStack.

6.5 Experimental Plan

For the evaluation of Process Mapping, an experimental plan has been designed aiming at a thorough investigation of the quality of the mapping. To this end, the initial focus will be on a specific sub-category of ML tasks, which contains a small set of algorithms, in order to check that the Process Mapping component is indeed able to provide promising results in terms of algorithm selection. During this phase, the extraction and exploitation of various meta-features will be evaluated as well. In the next phase, the intention is to generalize the Process Mapping component to a wider category of ML tasks. This evaluation methodology is typical for computer science research, starting from simple and specific versions of the problem and moving gradually to more generalized scenarios of use.

In terms of evaluation metrics and KPIs, the main objective of Process Mapping is to provide a relative ranking of the set of available algorithms, so that the most promising/suitable algorithm(s) for a given task can be selected. For the evaluation, our first intention is to compare against a baseline solution that randomly selects an algorithm from the set, in order to show that our method is much better than a random selection algorithm. Second, we intend to measure the accuracy of algorithm selection, in comparison with an oracle that always selects the best algorithm(s). In addition, the gain in performance will be quantified, in terms of the time saved by our approach compared to the brute-force approach that runs all algorithms and selects the best one based on post-execution evaluation. Last but not least, we are going to investigate how the system improves the quality of Process Mapping, when the Analytics Repository is augmented with more meta-knowledge, based on more results of algorithm execution on new datasets.

6.6 Next steps

The next steps and planned activities of Process Mapping component towards M18 are the following:

- Have a prototype implementation that (a) comprises all designed modules, (b) works end-to-end, and (c) provides first promising results with respect to the quality of algorithm selection;
- Identify potential weaknesses of the Descriptive Model Generator and the Analytics Engine as of M18, since these are the two most challenging modules of the architecture, thereby providing valuable feedback that can be exploited in the second half of the project for tuning the methods, optimizing performance, or trying alternative approaches;
- Perform empirical evaluation using both synthetic and real-world datasets, in order to verify the quality and accuracy of produced results.

7 Data Toolkit

7.1 Anticipated functionalities / requirements

The Data Toolkit facilitates Business Analysts and Data Scientists build operational analytic workflows by means of data pipelines through Directed Acyclic Graphs (DAGs). These graphs consist of nodes and edges with properties where the end-user can define the starting and ending stage and the intermediate processing stages she wants to perform towards the realization of her analytic task. The pipelines enable to define the set and the sequence of the stages required to be executed in order to set up end-to-end Big Data analytics based on a framework agnostic manner. These pipelines comprise the entire data orchestration lifecycle coupled with the corresponding executables. This means that the end user will be only aware and will take care of the conceptualisation of her analytics functionality and the desired objectives to be achieved in an agnostic way (i.e. REST APIs for data curation, transformation, analytic task such as classification, clustering, etc.). For instance, a Business Analyst has access to a higher abstraction level (BPMN like), services and the respective UIs of her Big Data analytics and end-to-end application objectives. At the same time, a Data Scientist, having the experience and knowledge to specify more details in the workflow set up, she has also the ability to define connection details to the services, specific algorithm selection from a set of relative algorithms (through an algorithms taxonomy), parameters configuration for the analytics algorithms and/or performance metrics. The Data Toolkit enables end-users to design point-to-point Big Data pipelines through drag-and-drop tools and intuitive UIs with the capability to define nodes, edges and properties in both nodes and edges in order to realize the operation, iteration and execution of the required pipelines in an ordered way. The expected outcomes are to:

- Create and handle valid data workflows by means of a managed graph creation process, which combine stream and batch data with the capabilities to define the required parameters, transformations and configuration settings per node.
- Facilitate end-users to reduce the time that is required to design, develop and produce executable analytic pipelines.
- Continuously monitor and manage pipelines performance, which is important especially in configuration of multiple analytic tasks with diverse requirements.

The tables that are following (Table 15 - Table 18) describe the requirements engineering method specified in D2.1 and are compiled together with the rest of requirements of BigDataStack in D2.2.

	Id	Level of detail	Type	Actor	Priority
	REQ-SY-DT-01	Software	FUNC	ROL-02, ROL-03	MAN
Name	Describe data mining and analysis processes through data workflows				
Description	Support for the description of data mining and analysis processes, interconnected to each other in terms of input/output data streams/objects. The corresponding metadata and an algorithms taxonomy for the categorisation of the analytic processes, type of data and connection details will be used to facilitate the description of individual nodes.				

Additional Information	The playbook must be represented in the form of a descriptor (e.g. through a yaml file) that can be incorporated into the Dimensioning Workbench as well as the Dynamic Orchestrator.
-------------------------------	---

Table 15 – System Requirement (1) for Data Toolkit

	Id	Level of detail	Type	Actor	Priority
	REQ-SY-DT-02	Software	FUNC	ROL-02, ROL-03	MAN
Name	Express data workflows through graphs using nodes and edges				
Description	Data workflows are represented in the form of an analysis application graph that includes the set of individual processes as nodes of the graph along with their binding/dependencies in the form of virtual links (i.e. edges). The links may include properties representing constraints, KPIs or objectives which are desirable at specific analytic stage.				
Additional Information					

Table 16 – System Requirement (2) for Data Toolkit

	Id	Level of detail	Type	Actor	Priority
	REQ-SY-DT-03	Software	FUNC	ROL-03	MAN
Name	Validate graph through chain-ability constraints				
Description	This requirement resolves chain-ability constraints through different nodes in the data workflows. The target is to produce a valid graph. This is the reason why a set of checks will be performed to meet these prerequisites. If these prerequisites are not met, the graph is not considered valid.				
Additional Information					

Table 17 - System Requirement (3) for Data Toolkit

	Id	Level of detail	Type	Actor	Priority
	REQ-SY-DT-04	Software	FUNC	ROL-03	MAN
Name	Link valid graphs with viable executables for Big Data analytic processes				
Description	This step links the graph with the actual executable image. In order to cope with the problem of vendor lock-in format of the executable the container format has been chosen. To this end, the actual container pulling will be performed.				
Additional Information					

Table 18 - System Requirement (4) for Data Toolkit

7.2 Specification / Design

The Data Toolkit fulfils all the system requirements that are needed to deliver the execution engine for analytic pipelines including Spark MLlib, other Distributed Machine Learning frameworks and functionalities and machine learning algorithms defined by the end-user to support the BigDataStack UCs. This component also interacts with Application Dimensioning Workbench which produces metrics on the workflows' performance and Process Mapping which finds the best setting between analytic tasks and the corresponding algorithm selection.

The pipeline describes the flow of data from the origin system to the destination systems and defines how to transform the data along the way. The pipeline includes interfaces (high-level APIs) to execute basic data handling operations such as filtering, sampling, etc., feature selection and data transformations, basic statistics (e.g. mathematical transformations) and machine learning algorithms including classification, clustering, regression, collaborative filtering and frequent pattern mining. The data workflow should be in a serializable format adopting Data Frames (e.g. through Resilient Distributed Datasets (RDDs) structures) in order to support different data types (structured and unstructured data, text, vectors, etc.). Each node expresses a distinct processing stage and its output should be expressed in JSON format (or other formats if needed). To facilitate Dimensioning Workbench and Dynamic Orchestrator, the result of Data Toolkit is provided in the form of a descriptor such as a yaml file.

Data travel through the pipeline in batches. Each stage is directly mapped into a node of the Directed Acyclic Graph. Origin nodes read data from the BigDataStack system or as data arrive in the case of real-time data streams. A proceeding node may be either a data curation task or a data analysis task. In the case of data curation, the task refers to filtering, sampling, dimensionality reduction and feature selection. Each of these tasks is correlated with a set of runtime parameters which can be specified at this stage or refined upon experimentation. In the case of an analytic task, either the type of analysis can be specified (e.g. classification) or the set of the algorithms (e.g. logistic regression, decision tree, random forest, etc.) which supports the respective analytic task. The data move from node to node until they reach the ending node. The ending node should feed Adaptable Visualizations with insights derived by this analysis pipeline.

The Data Toolkit facilitates a **workflow enactment** meaning that it **grounds the pipeline into an executable workflow**. It requires a set of runtime values for its optimal configuration. These runtime values include end-to-end analytic task objectives, runtime parameters, runtime properties and runtime resources. A Business Analyst may be able to define a high level abstraction of the DAG along with her end-to-end business objectives in a BPMN like manner. A Data Scientist can define and specify, in case she knows in advance, more details along the analytic workflow. The analytic task objectives are related with the KPIs that are defined by the user coupled with each specific scenario of her UC and may include time constraints, requirements regarding algorithm accuracy, scalability and performance. The runtime parameters are parameters that the end-user defines in the functions triggered in the pipeline in the form of arguments in order to invoke specific algorithms with specific parameters. In the case of a Data Scientist, when she starts the pipeline, she may know in advance the parameter values to use or let to be determined while experimentation of iterative analytic tasks with different settings through the Process Mapping. The runtime parameters define values for the algorithms of the pipeline

or a stage of the pipeline. For example, while invoking kNN (k-Nearest Neighbors) algorithm for classification, either the end user sets in advance the value of k or lets to be determined after some cycles of experimentation based on the collected results. The runtime properties may include different sets of values / volumes / veracity for different datasets. The runtime resources include configuration regarding the number of CPUs, RAM, the number of processing nodes and can be set in an end-to-end workflow execution scenario by the user and/or updated by the Dimensioning Workbench.

The end-user can also inject her/his own library, analytic function or script triggered at specific stages of the pipeline. This is realized by defining at each node / stage all the configuration that is required (e.g. connection details, APIs, etc.) in order to invoke a specific image, procedure or task along with the respective details if needed.

The pipelines can be saved and loaded by the Catalogue of Predictive Analytics to facilitate end-users use and re-use already existing analytic workflows, jobs and topologies. Also, the support of Kubernetes enables to execute scalable workload with time elasticity constraints.

In Figure 8, we present the high-level functionalities of the Data Toolkit in a UML Component diagram including both end-users, i.e. Business Analyst and Data Scientist.

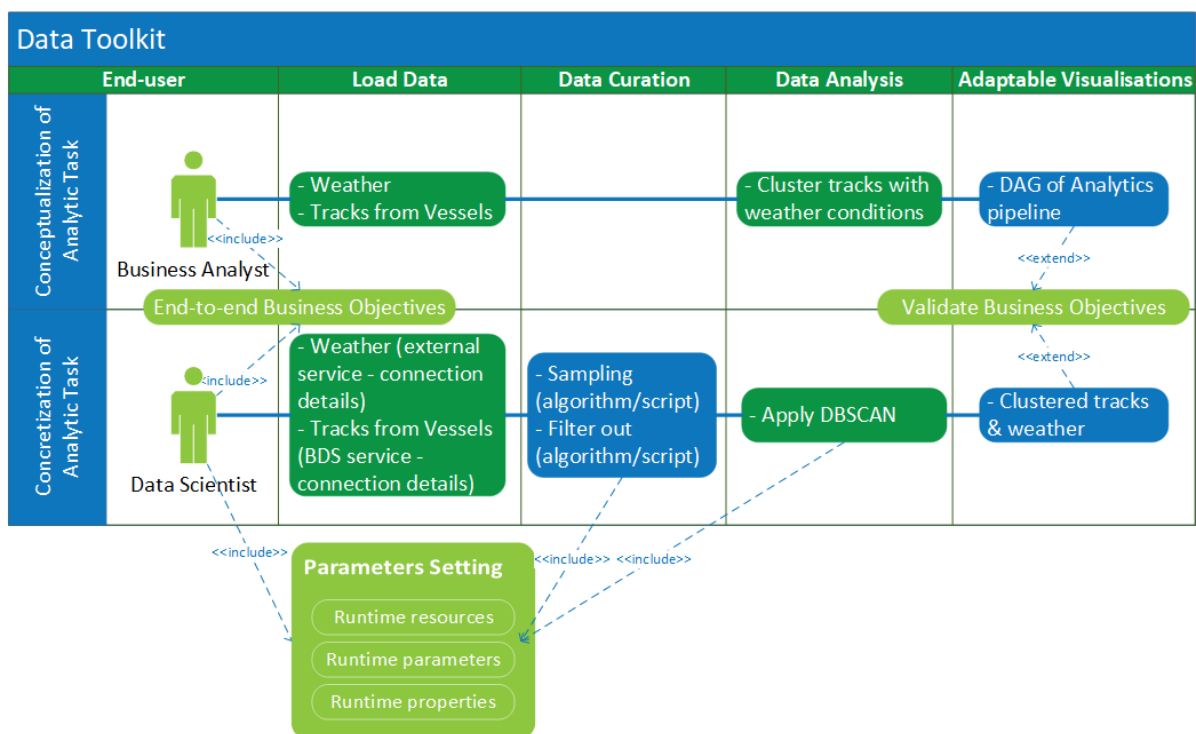


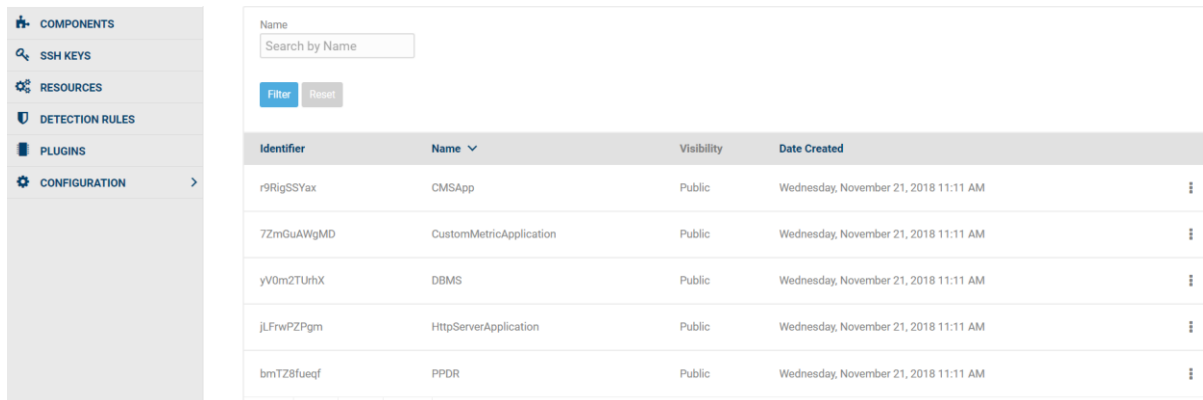
Figure 8 – UML Diagram

7.3 Early prototype

The early prototype of the Data Toolkit includes the different services, wrappers, APIs and tools that consist the BigDataStack solution towards the workflows enactment through the automation of Big Data analytics. The development and deployment of the services is currently a running task with an early version of the Data Toolkit to be expected in M18. In

the following, we present some screenshots exposing the current functionalities of the early prototype of Data Toolkit.

Figure 9 presents an indicative UI where the end-users can register their own analytic processes, provide a short description and create their own analytics palette.



Identifier	Name	Visibility	Date Created
r9RigSSYax	CMSApp	Public	Wednesday, November 21, 2018 11:11 AM
7ZmGuAWgMD	CustomMetricApplication	Public	Wednesday, November 21, 2018 11:11 AM
yV0m2TUrhX	DBMS	Public	Wednesday, November 21, 2018 11:11 AM
jLFrwPZPgM	HttpServerApplication	Public	Wednesday, November 21, 2018 11:11 AM
bmTZ8fueqf	PPDR	Public	Wednesday, November 21, 2018 11:11 AM

Figure 9 – Registration of different Analytic Processes

Figure 10 demonstrates an indicative example of composing an analytic process by fulfilling some prerequisites (i.e. constraints), where a php component requires a sql interface and has the constraint to be connected with a MariaDB instance.

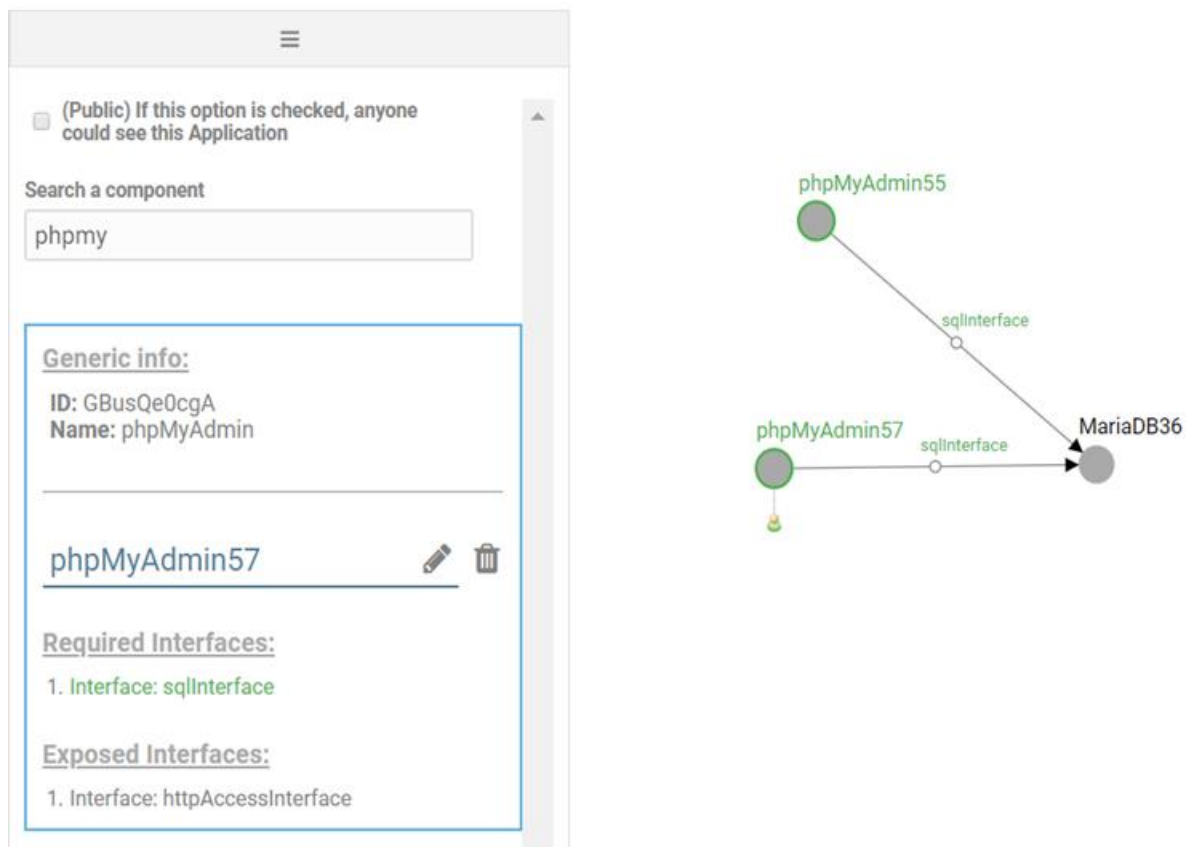


Figure 10 – Composition of an indicative Analytic Process

To produce an executable graph a set of validation steps and dependencies resolution will be executed in the background to facilitate end users correctly set up their own analytic processes including: a) check executable prerequisites; b) fetch corresponding images; c) wait until all dependencies are resolved and d) register data workflow upon health-check passes. Figure 11 presents an indicative deployment of an executable graph through the Data Toolkit along with its log.

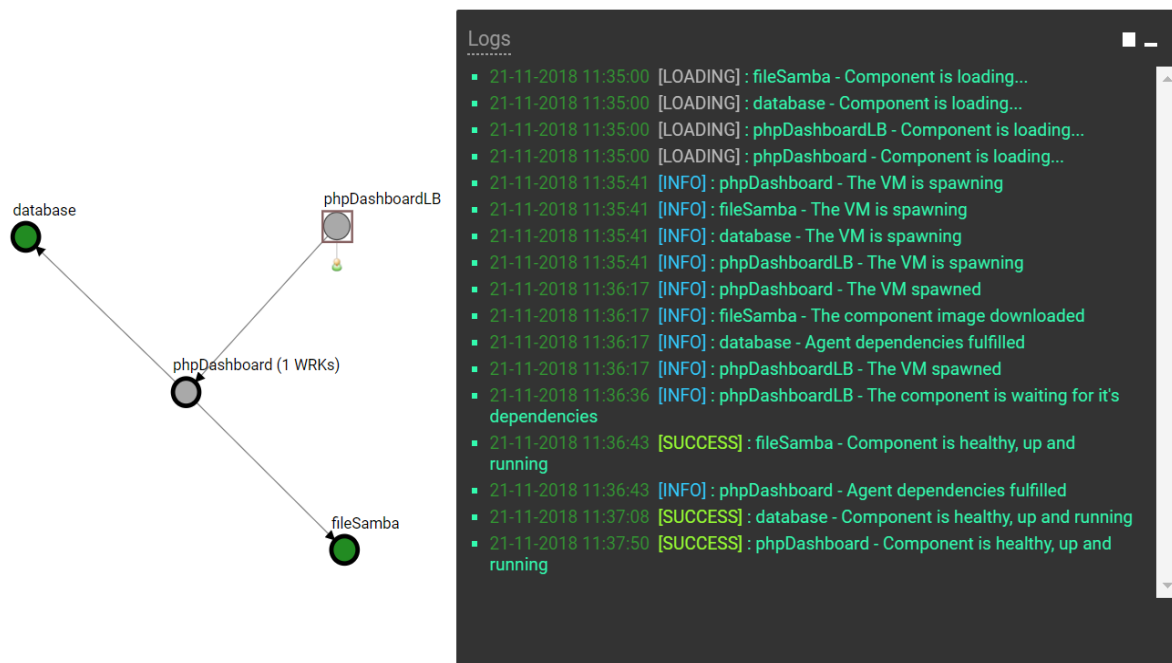


Figure 11 – Steps performed towards the creation of an executable graph

7.4 Use case mapping

The **Business Analyst** and/or the **Data Scientist** use the **Data Toolkit**, to perform a series of tasks related to the **concretization of the stages** incorporated into the nodes and the edges of the Directed Acyclic Graph such as:

- Identifying the end-to-end business objectives in terms of specifying KPIs and criteria for the evaluation of the UC scenarios;
- Defining the data source bindings from where the datasets related to the task will be ingested;
- Defining any data curation tasks (i.e. data cleaning, feature extraction, data enrichment, data sampling, data aggregation, Extract-Transform-Load (ETL) operations) necessary for the algorithms and the related steps;
- Configuring and parametrizing the runtime resources, parameters and properties related with the analytics tasks and the respective algorithms;
- Validating the end-to-end business objectives through the analytics insights feeding the Adaptable Visualisations.

In Figure 12, we map the functionalities of the Data Toolkit with an indicative analytics scenario of RSM UC.

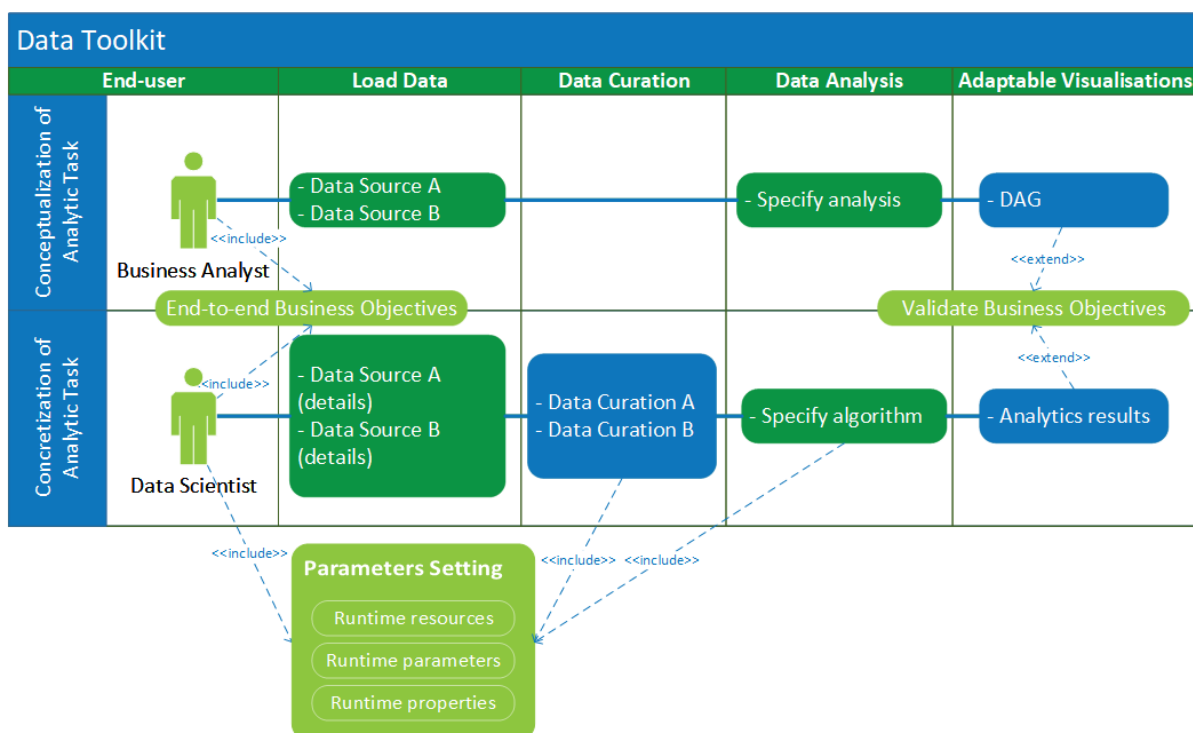


Figure 12 – Mapping of Data Toolkit with RSM UC

In Figure 13, we map the functionalities of the Data Toolkit with an indicative analytics scenario of Connected Consumer (CC) UC.

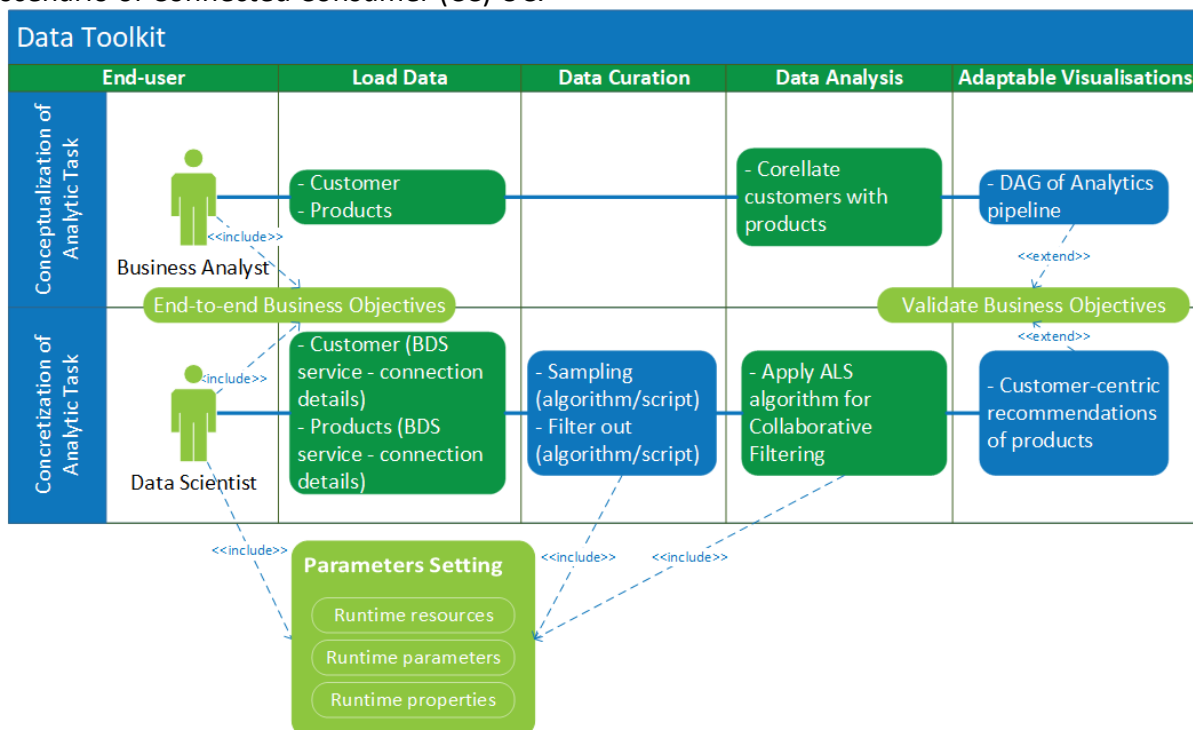


Figure 13 – Mapping of Data Toolkit with Connected Consumer (CC) UC

7.5 Experimental Plan

The Experimental Plan regarding the Data Toolkit includes to firstly setting up an end-to-end analytic case by firstly exploiting the data provided by the UC partners. This involves some indicative and simple analytic end-to-end application scenarios including basic algorithms and functionalities either for RSM UC or CC UC. The task involves providing the API services and the data orchestration to deploy valid Big Data and Machine Learning pipelines.

This plan also includes diverse configuration, mixing and matching ML technologies and algorithms ingested by the Data Scientists to validate Data Toolkit efficiency, diversity and applicability in an application agnostic manner which is independent from specialised frameworks.

7.6 Next steps

Particular attention will be drawn around the following topics: specification of valid DAGs and tools for online tractability of misaligned or mis-defined pipelines, which data format should be used to manage different pipelines with diverse analytic requirements, how to put in interaction and communication of BigDataStack different components and external tools.

As a next step of the work performed, it will be the conceptualization of the Data Toolkit initiated by the Architecture to be reflected and deployed in the BigDataStack environment as a tool of workflows enactment which defines and deploys valid, orchestrated and executable Big Data analytic tasks. The UCs also facilitate to address the main functionalities and deployment considerations in respect to the requirements expressed.

8 Application Dimensioning Workbench

As indicated in D2.4, the Application Dimensioning Workbench aims to provide insights regarding the required infrastructure resources for the data services and application components (micro-services), linking the used resources with load and expected QoS levels. To this end, it needs to cater for both cases of resources needed, creating prediction/correlation models between the application/service related information (such as KPIs and workload, parameters of the data service etc.) and the used resources to be able to provide recommendations towards the deployment mechanisms. Benchmarking against these services is an option that may help in concentrating the original dataset that is needed for the creation of such supervised models, as well as historical data from previous runs. The general architecture of ADW appears in the following figure from D2.4, broken down to two main subcomponents, the Pattern Generator and the Dimensioning Core. Following, details on the two subcomponents are given ([3], [4]).

Furthermore, in order to gather sufficient data for model training, a benchmarking phase is designed, where several configurations are tested under extreme, regulated circumstances. From the information obtained in the execution of this component, the system obtains knowledge on the most suitable configurations for the dimensioning phase. While bibliography already contemplates the existence of different suites for Big Data benchmarking, these cannot be used for the BigDataStack project, given that they are designed for very specific workflows or cannot be adapted for this project ([3], [4]). Thus, in BigDataStack project it will be necessary to run different benchmarks adapted to the UCs, both in terms of workloads and needed QoS metrics.

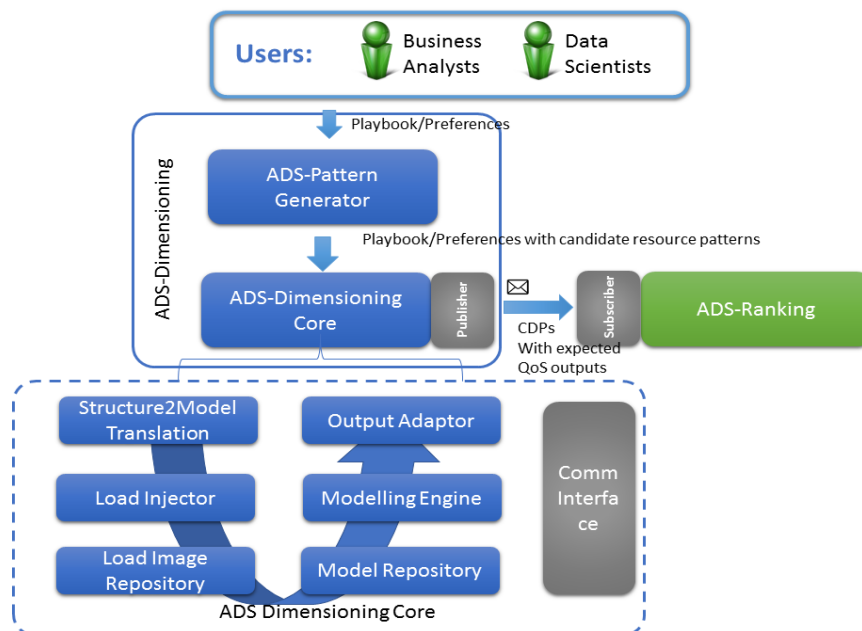


Figure 14 -Generic Information flow for ADW from D2.4

8.1 Anticipated functionalities / requirements

8.1.1 Pattern Generator

The aim of pattern generation is to define the different ways that a user's application might be deployed on available cloud infrastructure. Prior to pattern generation, the user has defined in a conceptual manner what their application is comprised of and how the different components of that application interact. It is the job of pattern generation to map this conceptual view of the application into concrete specifications for how the application components can be physically deployed.

Given the wide variety of hardware available on most cloud platforms, there are potentially a very large number of deployment configurations for a user's application. Each deployment configuration may place application components on different machine types for instance. We refer to a specific deployment configuration for a user application as a *candidate deployment pattern*. In effect, pattern generation aims to produce a set of candidate deployment patterns for a user's application that span the range from low-cost/single machine deployments up-to high-cost/high-performance computing deployments.

Later components within the Application Dimensioning Workbench and subsequently the Realization system within BigDataStack will automatically analyse these candidate deployment patterns, as well as examine their suitability given the user requirements and preferences, with the end-goal of selecting the best one that will fit the user's needs.

The anticipated functionalities / requirements are described in the following tables (Table 19-Table 24), that are compiled together with the rest of requirements of BigDataStack in D2.2.

	Id	Level of detail	Type	Actor	Priority
	REQ-T5.1-PG-01	System and Software	FUNC	ROL-04	MAN
Name	Ingest Playbook				
Description	The Data Toolkit sends to the Pattern Generation a Playbook containing the graph of the user's application. The Pattern Generation receives the playbook and initiates creation of candidate deployment patterns.				
Additional Information					

Table 19 – System Requirement (1) for Pattern Generator

	Id	Level of detail	Type	Actor	Priority
	REQ-T5.1-PG-02	System and Software	FUNC	ROL-04	MAN
Name	Load Hardware Directory (File)				
Description	To produce candidate deployment patterns, Pattern Generation needs to know what hardware is available to deploy the components of the user's application upon. Initial versions will load this information from a static file.				

Additional Information	
-------------------------------	--

Table 20 – System Requirement (2) for Pattern Generator

	Id	Level of detail	Type	Actor	Priority
	REQ-T5.1-PG-03	System and Software	FUNC	ROL-04	MAN
Name	Load Hardware Directory				
Description	To produce candidate deployment patterns, Pattern Generation needs to know what hardware is available to deploy the components of the user's application upon.				
Additional Information					

Table 21 – System Requirement (3) for Pattern Generator

	Id	Level of detail	Type	Actor	Priority
	REQ-T5.1-PG-04	System and Software	FUNC	ROL-04	MAN
Name	Service-Hardware Mapping (1-1)				
Description	The main process in Pattern Generation is mapping the different components (services) to potentially suitable hardware. The first version of this functionality produces only 1-1 mappings, i.e. one service is mapped to one piece of hardware (e.g. machine).				
Additional Information					

Table 22 – System Requirement (4) for Pattern Generator

	Id	Level of detail	Type	Actor	Priority
	REQ-T5.1-PG-05	System and Software	FUNC	ROL-04	MAN
Name	Service-Hardware Mapping (1-M)				
Description	The main process in Pattern Generation is mapping the different components (services) to potentially suitable hardware. The second version of this functionality produces only one to many mappings, i.e. one service can be mapped to multiple piece of hardware (e.g. spread over multiple machines). This may be advantageous in cases such as were a single 'big' machine is more expensive than multiple smaller machines.				
Additional Information					

Table 23 – System Requirement (5) for Pattern Generator

	Id	Level of detail	Type	Actor	Priority
	REQ-T5.1-PG-06	System and Software	FUNC	ROL-04	DES
Name	Service-Hardware Mapping (M-1/Pods)				
Description	The main process in Pattern Generation is mapping the different components (services) to potentially suitable hardware. The third version of this functionality produces only many to one mappings, i.e. multiple services can be co-located on a single piece of hardware. This may be advantageous when services perform high-volume data transfers that would be expensive over a network.				
Additional Information					

Table 24 – System Requirement (6) for Pattern Generator

8.1.2 ADW Core

The ADW Core functionality extends across two areas:

- Initially gather a dataset that includes executions at least at the data service level, with indicative differentiations related to deployment options and input workloads and their measured influence on the observed QoS outputs of the service. This may be later on used in order to further generalize based on a set of identified attributes
- Reply to the Pattern Generator for the anticipated QoS levels on investigated service deployments

Requirements gathered and refined from D2.1 as well as the technical process in BigDataStack are presented in the following tables (Table 25-Table 33) with relation to the ADW Core. These tables are compiled together with the rest of requirements of BigDataStack in D2.2.

	Id	Level of detail	Type	Actor	Priority
	REQ-SY-DW-01	System	PERF/ NONFUNC	ROL-02	MAN
Name	Response Time and Workload				
Description	The service provided by the data applications (e.g. recommender system) must have enough speed so consumers will not notice the time taken by the request. This implies that the Data Scientist should be able to dictate what are the required levels of QoS, selecting them from available metrics and appropriate levels for them.				
Additional Information	This requirement poses initially the feature of metric selection and insertion at the Data Toolkit layer, for the Data Scientist to express their desires. Then the annotated Playbook gets passed to the following components (primarily ADW). Inside the Application Dimensioning Workbench, an initial candidate solution set is created, its estimated QoS level is enriched and the solution set is returned to the Data Scientist for final selection. Workload features (e.g. maximum/average etc. number of concurrent users) should also be				

	<p>able to be specified in order for the system to estimate the anticipated QoS levels for the desired range of application level workload.</p> <p>This indicates that per category of data service or data service+analytics function a suitable selection of workload and QoS metrics should be performed and supported across the system (including also other components like monitoring)</p>
--	---

Table 25 – System Requirement (1) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SY-DW-02	System	NONFUNC / PERF	ROL-04	MAN
Name	Scalability and configurability of stress tests for load injection				
Description	The system should have knowledge of a mapping between workload and QoS levels of the data services and algorithms (in order also to support REQ-SY-DW-02). Therefore, it should be able to launch stress tests against the data services that can easily scale to support the client sizes needed. Furthermore, different parameters of workload should be able to be determined				
Additional Information	Given that different data services exist in the project ecosystem, different baseline benchmarking tools should be identified per case. Following their selection, they need to be configured based on the respective workload parameters and scaled based on an abstracted generic approach (e.g. Docker containerization and Docker swarm approach)				

Table 26 – System Requirement (2) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SY-DW-03	System	FUNC	ROL-04	MAN
Name	Dimensioning output				
Description	The Dimensioning workbench should provide a list of candidate dimensioning suggestions along with the expected QoS levels towards the ADS Deploy component (and eventually the Application Engineer role), for the former to filter them based on an extra set of criteria and the latter to perform the final selection.				
Additional Information	Upon reception of the playbook with the service graph, ADW needs to estimate QoS level based on the results obtained through REQ-SYS-DW-02 and populate the respective fields. The operation should be offered through a REST service interface for automating the process and hiding complexities.				

Table 27 – System Requirement (3) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SY-DW-04	System	FUNC	ROL-04	MAN
Name	Monitoring requirements for dimensioning				
Description	The Dimensioning workbench should have a means to obtain monitoring information from the deployed data services and application components for a given deployment to extract training data for the performance models. The rationale of the requirement is that for every needed metric (workload oriented e.g. number of current users, requests etc. or QoS oriented e.g. response time, throughput) in the model the respective endpoint should exist from which the monitoring component would extract metrics values. This applies to both actual runtime and benchmarking phase				
Additional Information	Relevant Tools affected: Data services, application components, triple monitoring engine.				

Table 28 – System Requirement (4) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-01	Software	FUNC	ROL-04	MAN
Name	Load injector dockerization				
Description	To support a generic load injection process as indicated by REQ-SY-DW-02, “dockerization” of the respective load generators per type of service needs to be performed. Thus, a specific Docker container image per needed load generator tool needs to be provided, along with a unified process for feeding the per case load description file based on the Docker API and configuration process.				
Additional Information					

Table 29 – System Requirement (5) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-02	Software	FUNC	ROL-04	MAN
Name	Service structure specification				
Description	The service graph specification coming as input from the Process Modelling and Data Toolkit should follow the Docker Compose specification, to be understandable by the Dimensioning workbench. Following, the Dimensioning phase should add the respective candidate resource deployment options as additional custom metadata in the file to be used by the Deployment selection. The same applies for the benchmarking runs, which should be based on the same format (even without the inclusion of the PM and Data Toolkits). All requirements needed for deploying the				

	benchmarking environment should be described using this common agreed standard.
Additional Information	

Table 30 – System Requirement (6) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-03	Software	FUNC	ROL-04	MAN
Name	Representative nature of gathered data samples				
Description	In order to create representative and accurate performance models, dataset creation from benchmarking should take into account different conditions such as applied workloads, configuration aspects of the service, deployment options etc. In this way different bottlenecks may be examined and the final decision making can be adapted per case of service usage.				
Additional Information					

Table 31 – System Requirement (7) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-04	Software	FUNC	ROL-04	ENH
Name	Deployment time for stress tests				
Description	The overhead added by the benchmarking setup should be negligible and not included in the measurement process.				
Additional Information	Since the deployment phase is done in a containerized manner, the time used in instructions different than launching the benchmark or storing data should not be significant.				

Table 32 – System Requirement (8) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-05	Software	FUNC	ROL-04	ENH
Name	Benchmarking Workflow implementation				
Description	During the benchmarking phase, there should be a controlled manner in which the various combinations described in REQ-SY-DW-02 and REQ-SO-ADW-03 are enforced during an automated process in order to ease data collection.				

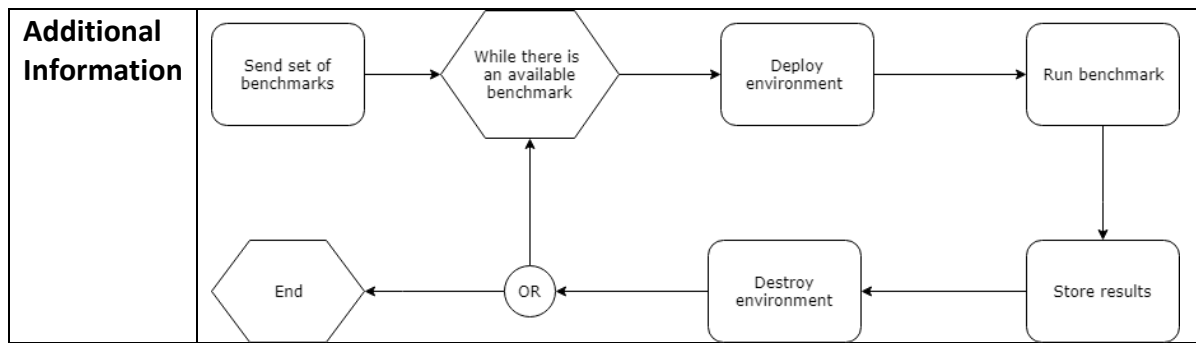


Table 33 – System Requirement (9) for ADW Core

8.2 Specification / Design

Following the analysis of the requirements in the previous section, we have created the set of system UCs for the ADW subsystem. For each case, the vertical separation refers to aspects such as Generic Functionalities (high level actions that the component needs to perform), specific sets of User Actions (i.e. selection from a relevant UI etc.), the set of Background Processes that need to be enacted following user preferences and any Dependencies from external components (or internal subcomponents of ADW) that are needed in order to complete the process.

Initially the service owner needs to design a range of stress tests/benchmarks that are needed in order to cater for the dataset collection, including the UI based insertion of a set of needed information such as target service, examined workload etc. In order to aid them in this direction, a set of predefined workloads may be created from which the users may select the subset that they are mostly interested in. These predefined workloads may be mapped to common UCs of the services and/or tailored to the specific UCs of BigDataStack. Furthermore, in order to include the various hardware (HW) deployment features, it is evident that the ADW Core needs also to contact the Pattern Generator, feeding predefined elementary playbooks for the given services and acquiring the various deployment options for the stress test. Base load clients per data service need also to be determined and dockerized in order to be used as load injectors per case. QoS metrics per data service need also to be defined a priori, while the service owner needs to define which ones are of interest to maintain and correlate. Another aspect is the various configuration options for the data services, e.g. modes of operation, deployment etc., that might change a service's performance profile. This needs to be investigated on a service level and should be included in the elementary playbooks included as available for the stress tests.

Once the data service is deployed, then the stress test (launch of the distributed clients) can be performed. Therefore, there is an asynchronous step for benchmarking, it should wait for the elementary playbook's deployment, before launching the test.

In a nutshell, the issues that need to be handled offline and/or in agreement with respective parties include:

- Enumeration of data services and/or data service+analytic algorithm options
- Predefined workloads (per data service and/or BigDataStack UC) and way to feed them as input during the stress test
- HW deployment features from Pattern Generator

- Configuration options that affect data service/algorithm performance and according elementary playbooks
- Dockerized base load clients for each tool needed by the BigDataStack data services to emulate load
- Main QoS metrics per data service and way of acquisition/storage in a given run

Most of the aforementioned features are discussed in the context of this document (even if for not all of the BigDataStack services) while more concrete points such as the elementary playbook creation will be considered during the experimentation phase.

We include as the main actor the role of the Data service owner, since this is the most generic approach. Having a wide set of data for a given data service enables the more generic and abstract mapping to individual deployment instances of a specific scenario. Otherwise, benchmarking needs to be performed for every single service graph, a process that is expected to be both complicated and time consuming for the Data scientist/application owner.

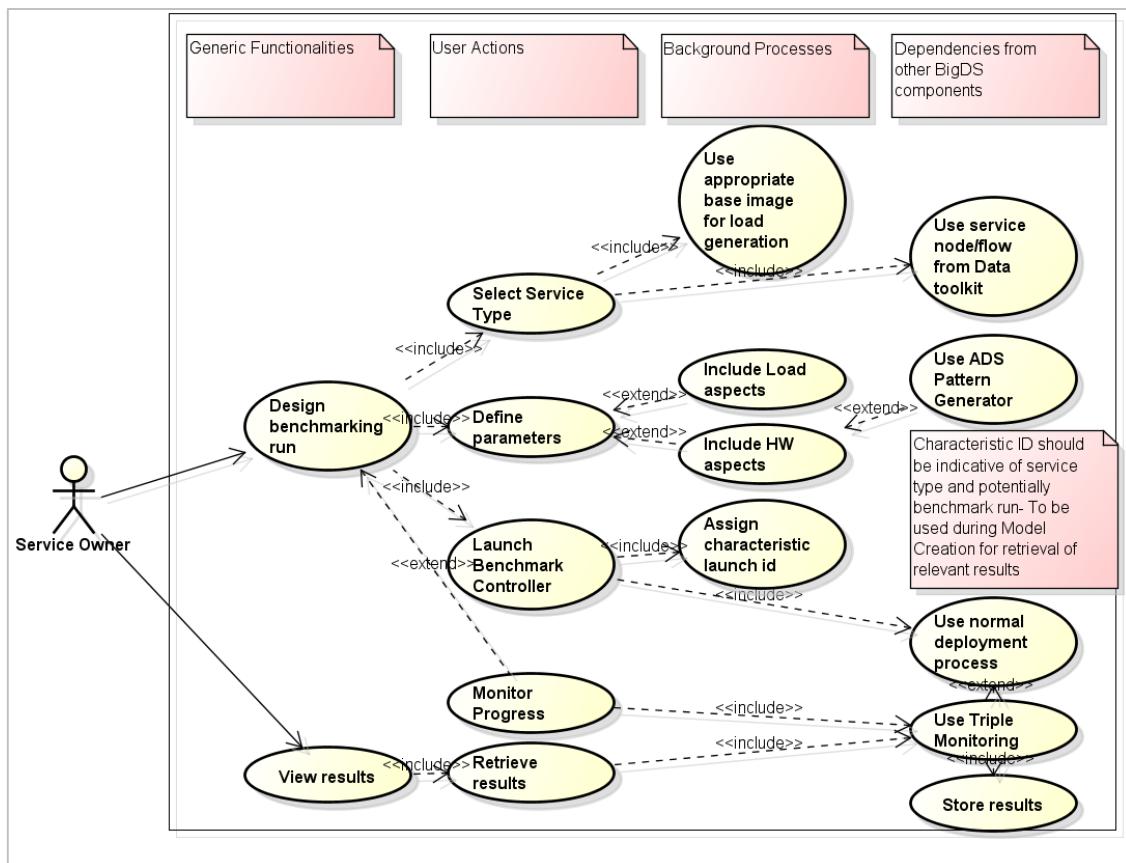


Figure 15 - ADW Design Benchmark Run System

Following the creation and acquisition of the relevant dataset, the service owner may initialize the process of predictive model creation in order to create the generalized predictive model per case. Based on a given name during the benchmarking phase, they may collect all relevant data and feed them to the model creation process.

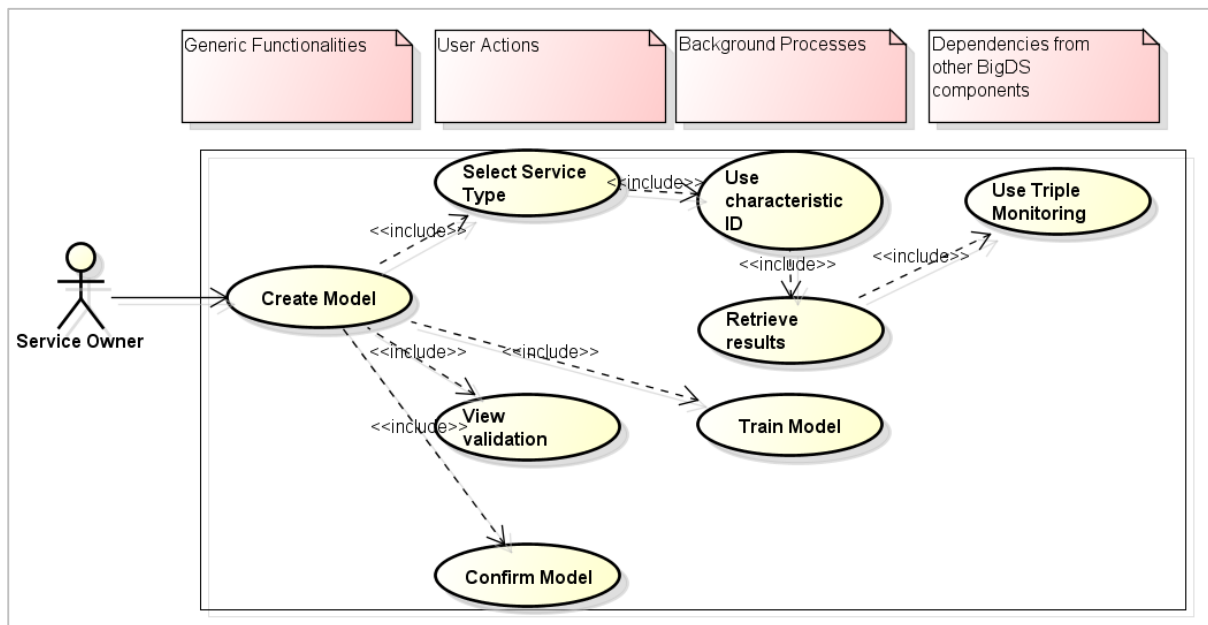


Figure 16 - ADW Create Model System

Once the previous phase has been completed, the acquired data and/or models may be exploited in the context of a given service instance to be deployed with given QoS needs and workload aspects. In this case, the Data Scientist, either in the Data Toolkit and/or in the ADS Ranking UI, will insert the needed data services instances and indicate anticipated workloads and needed QoS levels. The annotated playbook, enriched by the Pattern Generator with the HW deployment options, will be fed into the ADW Core, which will analyse the individual elements and provide the estimates (from the benchmark history and/or models) that more closely resemble the given deployment instance. Points of attention here include:

- The metrics made available to the Data Scientist need to be in accordance with the ones supported by the benchmarking and monitoring process
- The ADW Core needs also to annotate the initial input playbook with the anticipated QoS levels per service element and forward it to the ADS Deploy component for final selection and deployment.

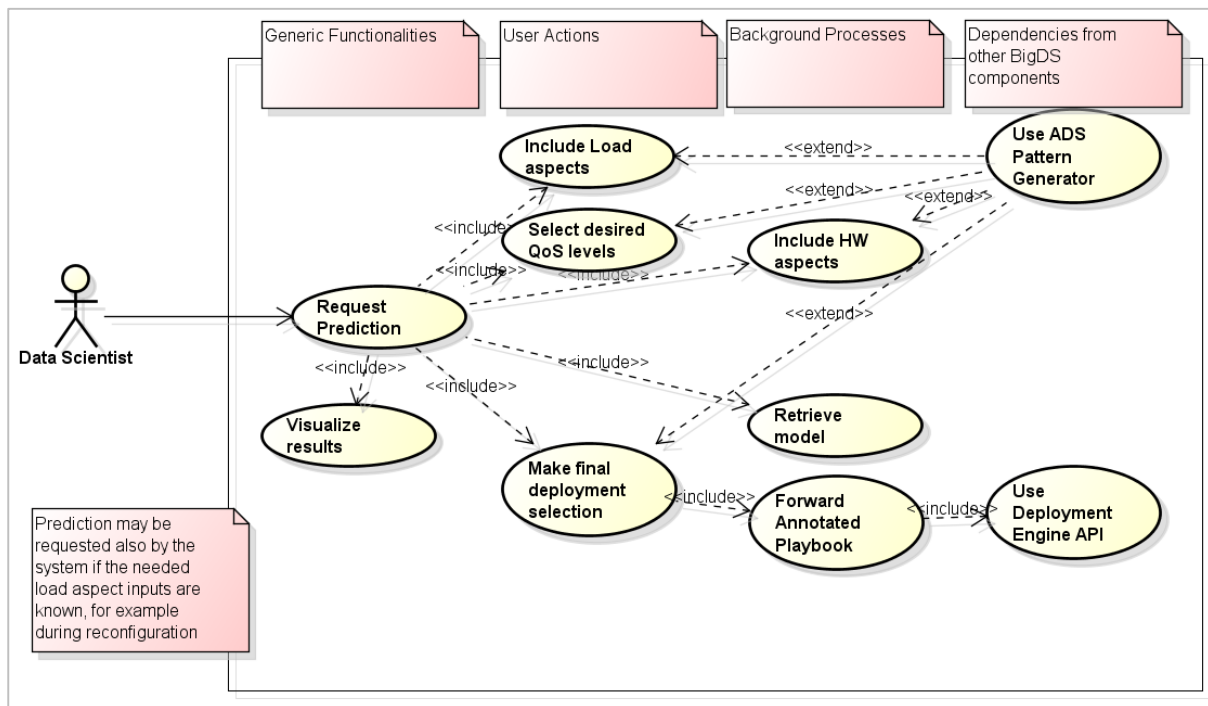


Figure 17 - ADW Request Prediction System

Following the identified system scenarios, we present the generic design architecture for the two main subcomponents, the Pattern Generator and the ADW Core.

8.2.1 Benchmarking and Model Creation of ADW Core

Initially, the ADW Core needs to create performance models for the elementary data services of BigDataStack (or combinations of services and analytics algorithms). This is needed in order to be able to reason on necessary resources needed per deployed instance of the service. However, in order not to need tests prior to each and every deployment request, an initial benchmarking phase is anticipated in order to gather a representative dataset with which a performance model can be created (thus abiding to requirements REQ-SO-ADW-03, REQ-SY-DW-03 and REQ-SY-DW-01), but for every type of data service and for a variety of workloads and service configurations.

Based on the envisioned system UCs presented, the service owner needs to design the benchmark phase in order to cater for representative load cases. To this end, a relevant UI is needed to enter the various parameters, implemented in Node-RED. The purpose of this is to gather the parameters and wrap them to the necessary JSON format that is the input to the ADW Core relevant RESTful endpoint. In order to minimize the inserted information, relevant fields need to be included in a parameter range type of format (e.g. min/max value and step), meaning that the back end wrapper needs to unwrap the various combinations and launch the according configurations. This launch could be performed in either a sequential or parallel mode, for reducing sampling time, if the available testbed resources are adequate. For launching the stress test for the given configuration, two features are needed:

- Dockerization of relevant tools that can generate base load towards the data service, along with capable configuration of the Docker image to initialize parameters per execution.
- Implementation of the ADS Deploy client interface in order to submit the request to deploy the respective data service or existence of a specialized Docker Swarm cluster to launch the data service (implying also the existence of a dockerized version of the data service itself).

Ability to check the state and progress of a running test is also needed. The Benchmarking Controller (BC) subcomponent automates the process of sequentially running a set of benchmarks. For each benchmark, the BC deploys an environment (that is, a set of data service and load injection containers working together), and retrieves the data to run a particular model. This component is agnostic to the data, and limits itself to creating the environment, running a configuration and storing the results.

For each benchmark, the input to the BC is:

- **List of containers:** Location of all containers to be deployed;
- **Data & configuration:** Location of data to be fed to each container and list of specific instructions to be run inside the container;
- **Storage:** Location in which the output is stored.

The architecture needed for this phase appears in Figure 18.

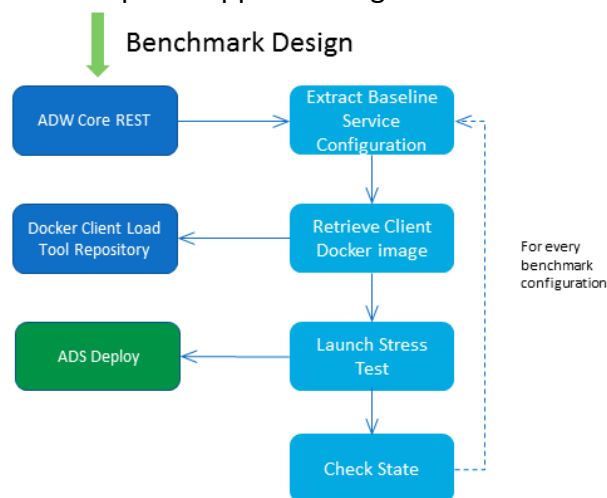


Figure 18 - Benchmark Design Architecture

Every benchmark is run sequentially, varying the configuration to be used for the tests. Figure 19 describes a UML description of the component architecture. As shown, the benchmarking component receives a set of benchmarks, in the form of multiple JSON documents following the standardized playbook file, each one of which is independently run. For the BC to individually deploy and run each benchmark, it needs to have access to a description of the containers (for each container, it needs the container image, a set of instructions to run inside and location of the data). Once deployed, the BC component retrieves the resulting data and stores it in a specific location.

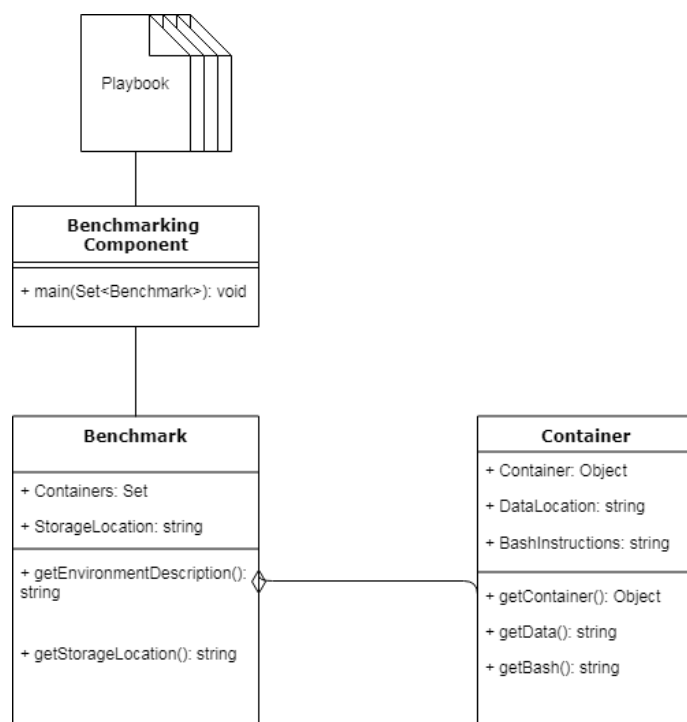


Figure 19 - Class diagram of the elements in the benchmarking Components.

Following the creation of a representative dataset, model creation needs to be triggered based on the same REST interface layer of ADW Core. Model creation is performed in the GNU Octave [7] environment through relevant service wrappers to offer it via REST. Acquisition of relevant data is based on the data service naming used. Once the models for each data service are created, they are ready to be used during the online phase for populating the various candidate deployment patterns (CDPs). It is necessary to stress that model structure is based on the various configuration options and workload aspects, so that they act as predictors, while the predicted output is the relevant QoS metrics for each data service.

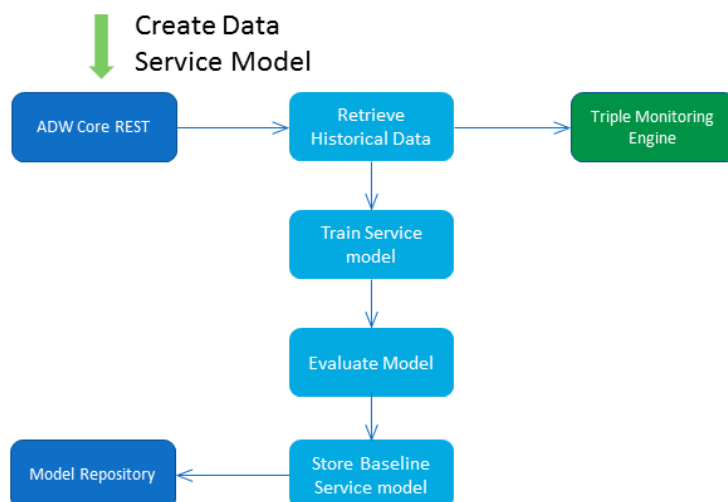


Figure 20 - Model Creation Architecture

8.2.2 Pattern Generator

Pattern Generation is designed as an independent Apache Spark streaming service. The Data Toolkit component of BigDataStack passes to the Pattern Generation service a Playbook, containing the conceptual view of the user's application. This Playbook is passed through a series of Spark transformation functions that perform the core service mapping functionality. The final function within the Spark topology posts the created candidate deployment patterns to a mailbox which can be read by the next component in the BigDataStack application deployment pipeline.

The architecture of the Pattern Generation component is shown in Figure 21 below. Within Figure 21, Spark transformers are shown in orange while non-spark components are shown in blue. As we can see in Figure 21, Pattern Generation ingests Playbook objects via a RESTful API, which directly passes that playbook into the main Spark processing pipeline via a Spark receiver. Once a Playbook is ingested, it is first split into services, and each service is mapped to different types of available hardware, where that hardware is specified in an external directory. This directory may be loaded from file or directly populated from the cluster infrastructure management system (OpenStack in our case). Once individual or groups of services have been mapped to hardware, these service mappings are then re-combined into what we refer to as an availability sheet, which contains all valid service to hardware mappings. Finally, this availability sheet is used to produce a large number of unique candidate deployment patterns, where one candidate deployment pattern contains a service to hardware mapping for each service in the user's application. These candidate deployment patterns are then published for consumption by the next step in the BigDataStack application deployment pipeline, the ADW Core.

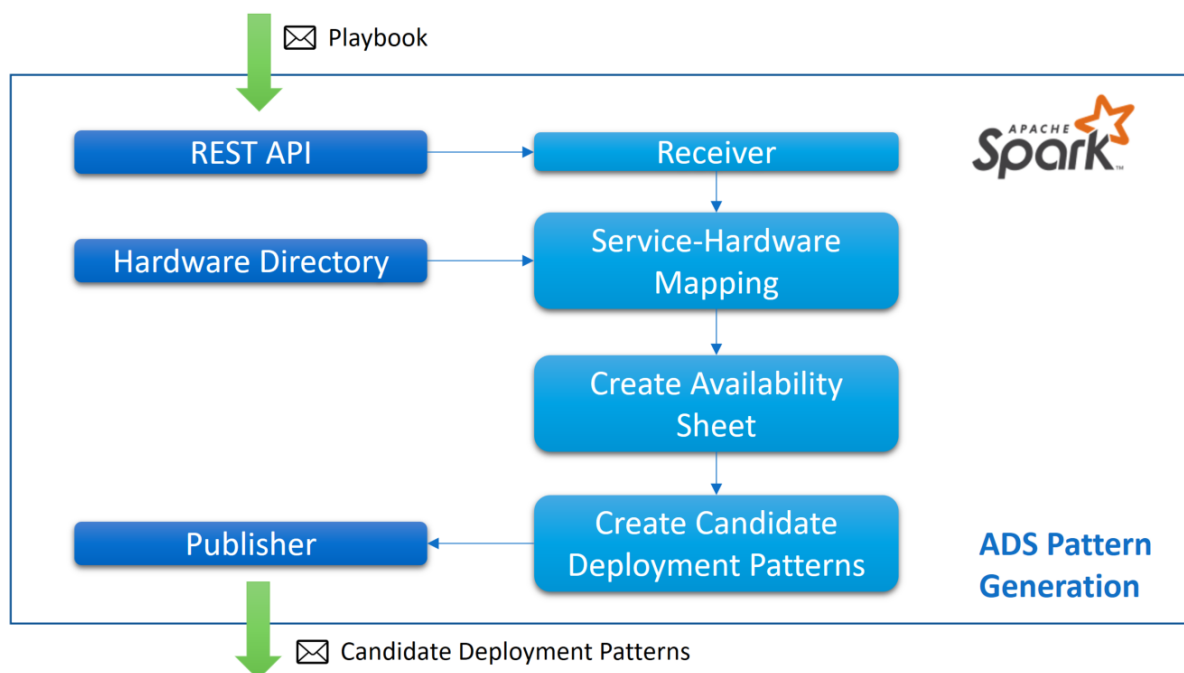


Figure 21 - ADS-Pattern Generation Architecture

8.2.3 ADW Core Online Request prediction phase

Following the population of the playbook with the various CDPs, it gets published to the relevant REST API offered by ADW Core. For each CDP, the ADW Core needs to populate it with the respective expected QoS levels. Thus it needs to break down the input per CDP, extract the service graph and start predicting the QoS level per service element. Given that the service elements are interconnected, one element's input will be the previous element's output. Thus, the predicted output of the first stage will act as input to the following and so on. For each prediction the component needs to retrieve the relevant data service baseline model, apply the inputs and get the result, propagating it as input to the next element of the graph. On completion, the various CDPs, annotated with the QoS levels, are then forwarded to the ADS Ranking component to investigate and decide on the finally selected trade-off.

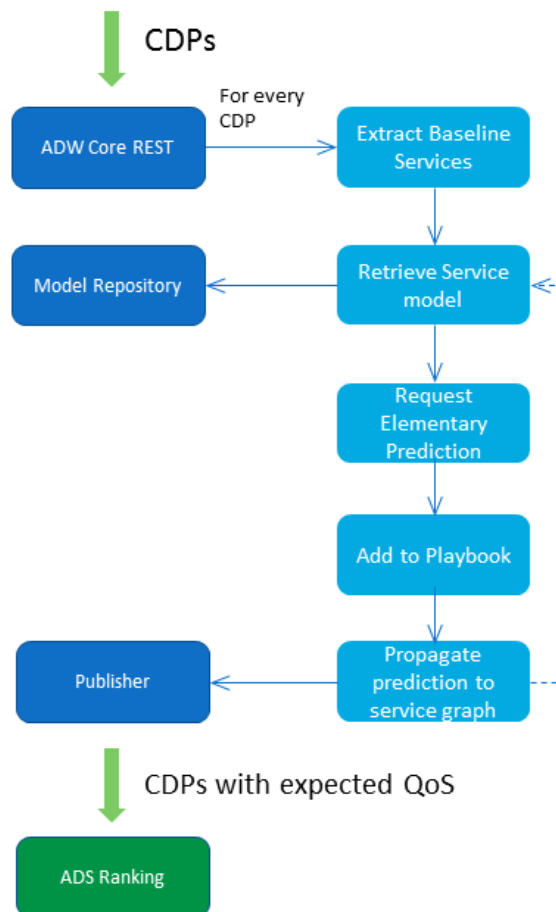


Figure 22 - Annotate Playbook Architecture

The overall API for the ADW Core, described in the previous sections, appears in Table 34.

Method	Path	Input	Output
POST	/postPlaybook	Playbook YAML String	Annotated Playbook YAML string with QoS tags. This primarily implements the request prediction operation
POST	/launchTest	JSON configuration file for parameters (tool selection, workload features)	Return message for test id
GET	/testState/id	Test id	State of the test (Complete/Ongoing)
GET	/testState/id/conf	Test id	Configuration of the test (workload features, deployed service options etc.)
GET	/ServiceTestIDs/service_name	Service name (from available enumeration of available services (aims to return all tests for that service	JSON array with test ids

Table 34 - ADW Core API

8.3 Early prototype

8.3.1 Pattern Generator

A Tier-0 version of the Pattern Generation component has been developed, tested and deployed. This component provides functionality for ingesting playbooks (REQ-T5.1-PG-R1), loading hardware directories from file (REQ-T5.1-PG-R2) and one-to-one service to hardware mapping (REQ-T5.1-PG-R4).

Indicative generation of patterns for service elements UI, passed to dimensioning can be seen in Figure 23.

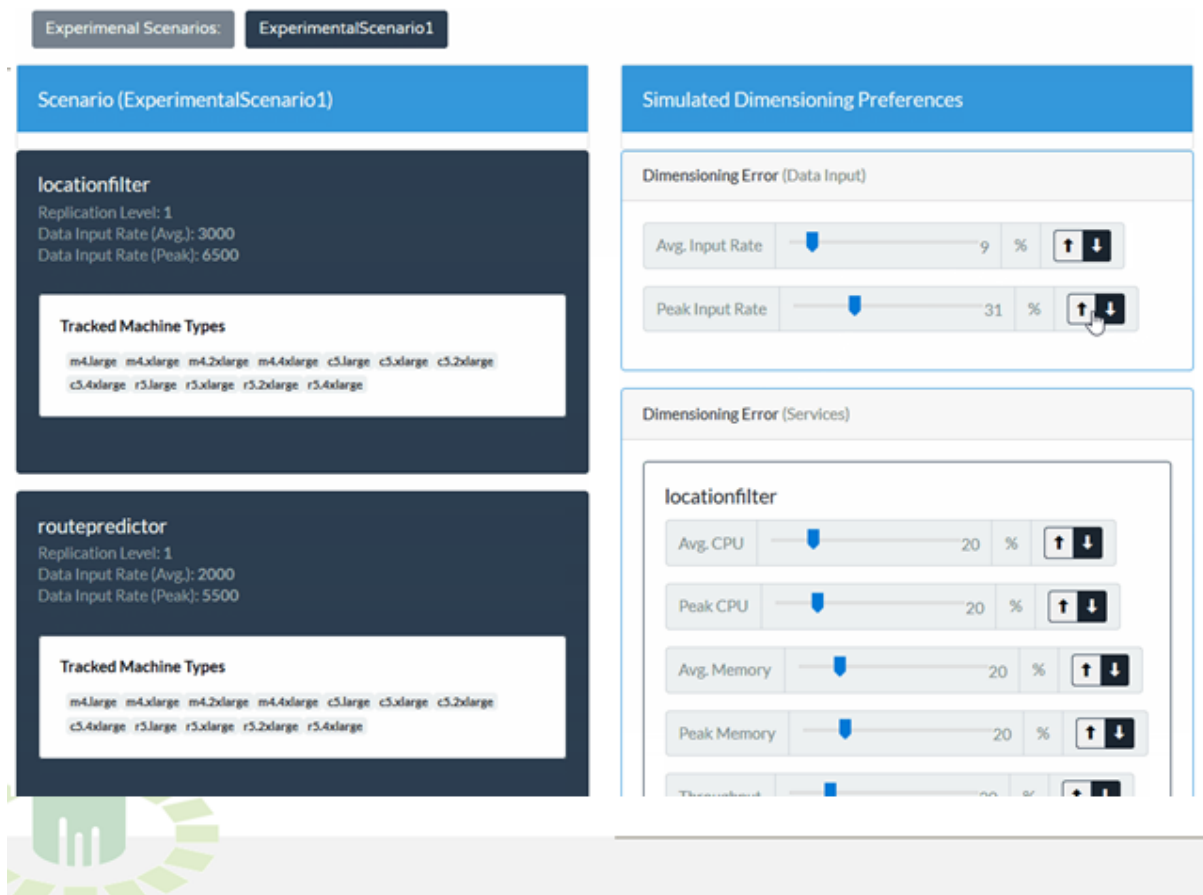


Figure 23 - Pattern Generation preferences setting UI

Indicative interfaces linking with dimensioning responses (simulated at this stage) appear in Figure 24.

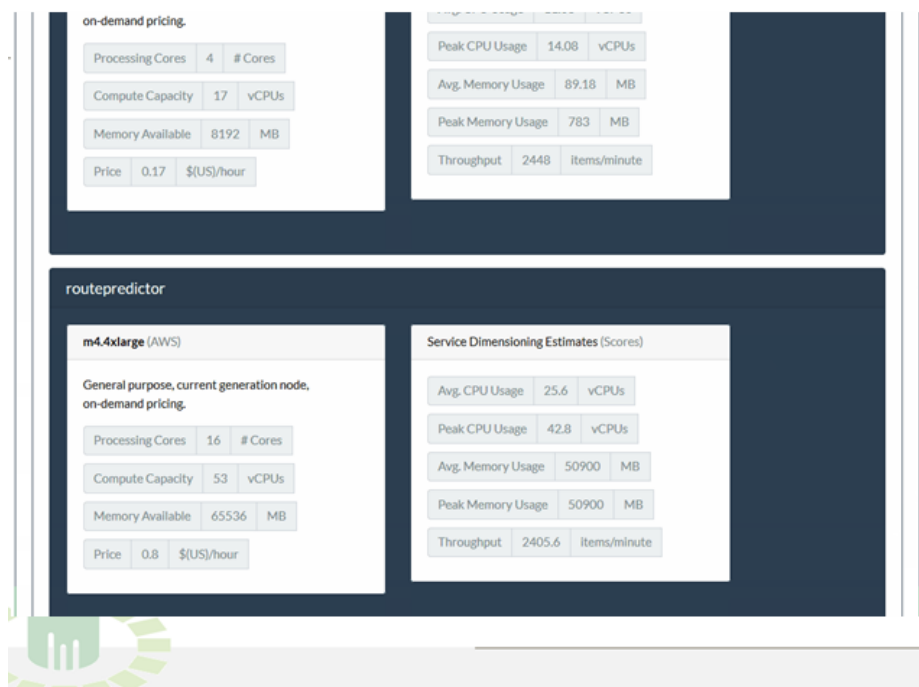


Figure 24 - Indicative UI with populated dimensioning estimates for pattern selection

8.3.2 ADW Core

A Tier-0 version of the ADW Core is under development, testing and deployment, to provide functionalities for requirements REQ-SY-DW-02, REQ-SO-ADW-01, REQ-SO-ADW-02. These refer primarily to stress test implementation and service graph understanding inside the ADW.

The REST and UI interface of the ADW Core is built in Node-RED, given the latter's ability for easy information manipulation (such as the one needed for receiving and processing JSON and other diverse formats) as well as adaptation to various technology layers and protocols. Furthermore it can easily integrate between different components and create asynchronous flows for management of information. An example of a REST service of the ADW Core (/postPlaybook functionality) appears in Figure 25.

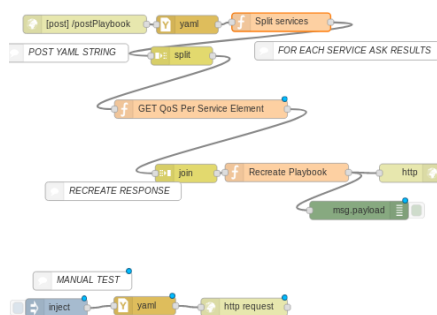


Figure 25 - ADW Core Services Layer example implementation

The deployment of the environment is managed by the ADS-Deploy component, described on deliverable D2.4 [4]. Figure 26 shows the technical workflow of the Benchmarking Component. The process is run between three components: the pattern generator, which generates the HW types and numbers combinations (more info on the potential combinations per data service is included in Section 8.4.2) and launches the service; the Benchmarking Component; and the ADS-Deploy component, which takes care of the deployment of the environment. The communication between the Benchmarking Component (BC) and the Dynamic Orchestrator (DO) is done using standardized JSON files for creation and destruction of environments.

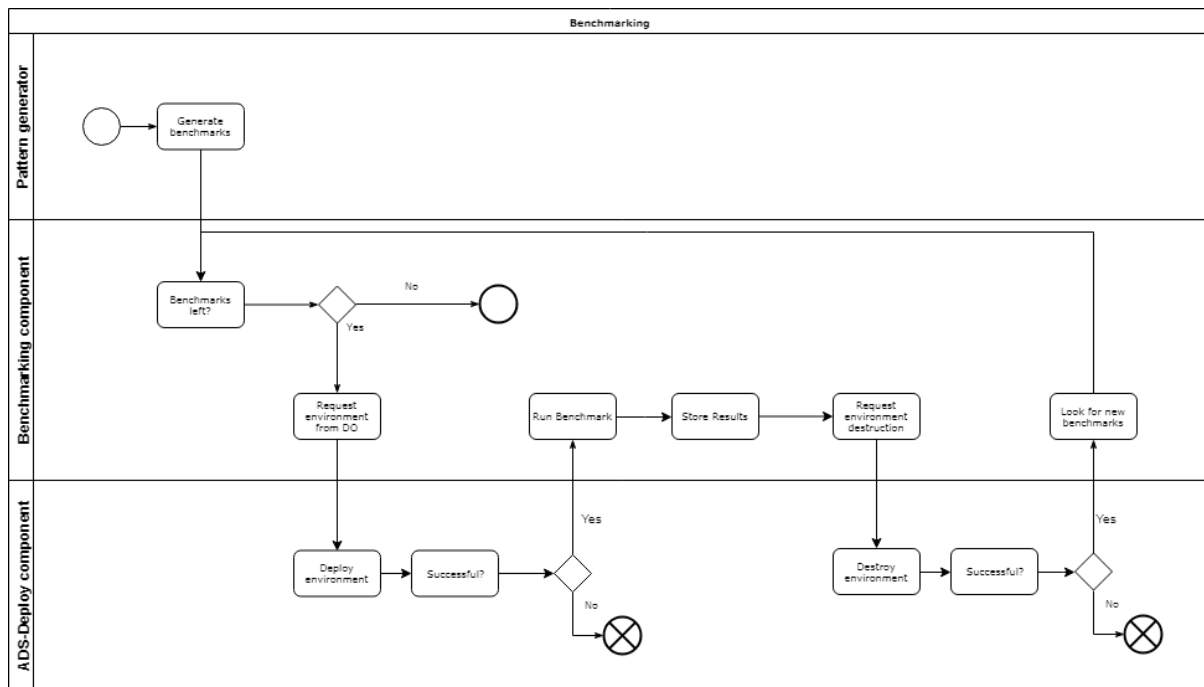


Figure 26 - Workflow diagram of interaction between components

An early pseudo-code for the BC should be as follows:

```
func Benchmarking_Component (set benchmarks)
```

```
  for (benchmark in benchmarks):
```

```
    ed = benchmark.getEnvironmentDescription()
```

```
    storageLocation = benchmark.getStorageLocation()
```

```
    dcCreationFile = createDeploymentFile(ed, storageLocation)
```

```
    dcDestructionFile = createDestructionFile(ed)
```

```
    DynamicOrchestrator.requestNewEnvironment(dcFile)
```

```
    DynamicOrchestrator.requestEnvironmentDestruction(dcFile)
```

8.4 Use case mapping

The Pattern Generation component of the application dimensioning workbench is not explicitly linked to any particular UC, as it forms part of the underlying application deployment backbone that supports all UCs in the project. As such, an effective pattern generation component can be considered an implicit requirement for all user scenarios (SCE-RSM-01, SCE-RSM-01, SCE-CC-01, SCE-CC-02, SCE-IMB-01 and SCE-IMB-02).

Effective pattern generation is an important step in the process of automatically identifying how to deploy the user's application onto the cloud infrastructure while meeting the user's needs and quality of service goals. This process aims to tackle the concerns of the Data Generators and Providers (STA-02) by enabling lower-cost application deployments, as well as the concerns of Technology Providers (STA-03) by more accurately identifying the hardware needed for application scalability and performance.

Furthermore, Dimensioning applies primarily to the data services included in BigDataStack and as such can be viewed as generic. However, better adaptation can be targeted towards the specific BigDataStack UCs, specifically with relation to aspects of workload that are more

specific for the UC scenarios encountered in the project. This may aid us in the incorporation of such relevant aspects during the experimentation (e.g. benchmarking process) in order to include cases that are expected to be encountered during the course of the project.

As an example, the RSM UC contains tables that have >100 columns. This is a clear indicator that during YCSB (Yahoo! Cloud Serving Benchmark) benchmarking of LXS, such figures in column numbers should be included in the investigated dataset, in order also to address REQ-SO- ADW-03, while other examples include needed throughput levels. In the following paragraphs such details are displayed more concretely per UC.

Mapping to individual features of the data services is also considered important and is listed in the context of this section.

8.4.1 Real-time Ship Management: Workload aspects example

Following, an initial analysis is performed on the DANAOS Real Time Ship Management scenario aspects. Similar analysis will be performed for the remaining scenarios in the following months. This analysis will help us to identify aspects that should be considered as inputs (predictors) to the models, as well as configure the generic benchmark tools used (e.g. YCSB) with specific configuration parameters for each case. Thus it may be considered that either we merge the acquired information in one overall configuration file or create a separate YCSB configuration file for each type of workload (thus one client container per row of workload type in Table 35) and launch them in parallel. Probably the second case is preferable since we will be able to distinguish (from a benchmark results acquisition point of view) QoS metrics per workload row/type, given that different such aspects/limits are defined per case. The majority of the specific parameters can also be considered as inputs to the relevant UI through which a data service owner may design a benchmark run.

In general, the RSM UC has as a major step data ingestion coming from the vessels. This at the moment happens via a batch file that is sent from each ship every 3 hours and contains vessel and engine data on a key value row structure (timestamp, vessel code, sensor name, sensor value), with a granularity of 1 minute measurements. Each sensor includes in this overall zipped file its relevant measurements on a separate file and each sensor file is handled separately (thus committing at the end of each sensor file). Ingestion is performed through upsert SQL queries per sensor file, since the combination of timestamp and vessel code is the primary key. Thus this functionality has an insert/update type of workload, with the majority of operations referring to updates (first occurrence of timestamp+vesselcode will be an insert, all other columns will be updates). For example, in the vessel data table with 23 columns we will have for each minute sample 1 as insert and $23-3=20$ updates (the columns removed are the first inserted metric column and the timestamp and vessel code columns), resulting in an approximate 5-95% ratio of inserts vs updates. In the context of the project and WP4, a variation of this operation may be performed, including the merging of all sensor values in one row on the ship side, through a relevant deployed CEP instance. For this reason, we have also included this case of streaming data from the vessels (DANAOS-OP-2B) that may be investigated.

Other secondary operations include telegram sending from ships to the main offices, but these take place much less frequently (once per day and at the start and end of each journey). The other frequent operation is a select on the averages of a given interval, which is also the

main current usage of the data and is used to monitor the ship's situation on a day-to-day basis. The overall grouping of operations along with other necessary (from a workload definition perspective) information appears in Table 35. In some cases, if historical data are available we can proceed with an analysis to discover aspects such as distribution of requests, scan sizes (for the requested intervals) etc.

Use Case Name	Type of Operation	Frequency	Number of users	Targeted Data Service	Schema details (# of tables, table size, # constraint, rule size etc.)	Other info (e.g. distribution of requests)	Indicative needed QoS level
DANAOS-OP-1	Read (select a time frame and average)	On request, typically 80 requests per working day per user	7 (Data Scientist@DANAOS premises)	LXS	On tables with 23 and 102 fields (vessel data, main engine)	Configurable granularity on time intervals (implied scan size on the table), log file of intervals could aid in identifying scan sizes distribution, log file of requests for requests interarrival times distribution	3 seconds response time for vessel data, 6 seconds for engine data (per request), 7 max concurrent transactions per second
DANAOS-OP-2A	Insert/Update new data, 180 inserts-> (180-2)*#columns updates ratio)	Every 3 hours batch data from each ship, 1 row per minute of these 3 hours (#columns*180 key value raw data rows, 360 output tuple rows)	60 (max number of ships, currently 35 with sensors)	LXS	On tables with 23 and 102 fields (vessel data, engine data)	Constant rate, batch mode implying bursts of operations, first column of time+vessel_code is inserted (PK), next metric columns are updated, 1 file per metric column for all 3 hours, commit at the end of each column update	10 seconds for ingestion from the time the data are available following their satellite transfer
DANAOS-OP-2B (Alternative version of 2A)	Insert new data (vessel and engine)	Streaming mode, 1 overall row per ship per minute, with merged tuple data from CEP instance on vessel	60 (max number of ships, currently 35 with sensors)	LXS	On tables with 23 and 102 fields (vessel data, engine data)	Constant rate, no need for updates compared to DANAOS-OP-2A	1 second per incoming message to be ingested

DANAOS-OP-3	Update (with weather data from external weather service)	Every 3 hours per ship	60	LXS	On tables with 23 fields (vessel data)	Constant rate (every three hours) per ship but need log files of batch arrivals to check distribution between ships	10 seconds from external data acquisition to storage
DANAOS-OP-4	Insert (telegram) (very rare updates)	1 per day per ship and at start and end of voyage	60	LXS	On tables with 14 fields (telegrams)	Constant and bursty (all ships at 12:00 UTC) for daily, log file for voyages to discover average time between telegrams for start and end of journey	Response time for operation < 5 seconds, max 60 ships/5 seconds = 12 transactions per second
DANAOS-OP-5	Insert (damages)	very rare, about 24 per year overall	60	LXS	On tables with 5 fields (damages)	Arbitrary and of no specific concern	Of no specific concern
DANAOS-OP-6	Check data	Depending on inputs from DANAOS- OP-2 & 3		CEP	5 rules, 1 with for depending on # of SLAs and 4 comparisons, 4 other with an average of 11 fields per rule (10,6,16 and 12) and input stream	Depending on inputs from DANAOS- OP-2	<1 sec of response time from data availability to alert raising

Table 35 - Detailed Workload Specification per UC template and Real Time Ship Management instantiation

In the following months, similar tables will be created for all the UCs following the concretization of the respective scenarios.

8.4.2 Data Services Specific Configuration, Deployment and Monitoring options

In order to have a basis for determining the performance modelling needs of the main BigDataStack data services, an analysis is performed in these sections with relation to each service. In order to drive the investigation, a table is populated for each case indicating the following information:

- What are the main components of the service;
- On a per component basis, if the specific component has been designed in order to support horizontal scaling, i.e. if it can run in distributed mode and thus be able to utilize more nodes. In this case, we do not examine if it makes sense to actually scale it, this is the scope of the next point. Furthermore, we do not include analysis of vertical scaling, since the latter is the capability of the infrastructure to provide a larger

resource and applies in all cases (again we do not examine in this case if it is actually beneficial to scale);

- On a per component basis, whether it would be worth examining the relationship between horizontal scaling and actual improvement of performance and up to which point. If based on partners' expertise and anticipated workloads, a specific component is not expected to need this testing in any realistic scenario, we can reduce the number of experimentation needed. The need is indicated by a Low/Medium/High state;
- Similarly to the previous point, but for the case of vertical scaling effect.

At the moment, the analysis has been performed for LXS (Section 56) and CEP (Section 8.4.2.2), while the respective one for the Object Store will be complete in the following months.

8.4.2.1 LeanXscale data store

8.4.2.1.1 LXS Configuration/Deployment options

LeanXscale data store is a fully ACID and SQL compliant distributed database that is consisted of three main pillars: the Key-Value Store, the SQL Query Engine and the Transactional Manager. All these components can either co-exist being deployed in the same node, or be deployed separately, while they can scale independently. However, for improved performance, it is suggested that an instance of a query engine should co-exist with a data node in order for the former to exploit the locality of the data stored in the latter and avoid transmitting them over the network, resulting in significant overhead caused by the network transmission and the wasted CPU cycles. The transactional manager on the other hand, can scale linearly up to 100s of nodes, and in typical scenarios it is deployed separately. As a result, a LeanXscale distribution consists of the data engine nodes, where the data are stored and accessed via the query engine of the data store, and metadata nodes, which holds metadata and other information required for ensuring the transactional semantics. The metadata nodes contain the services needed for metadata structures: Zookeeper, Configuration Manager, the Transactional Manager services and the metadata for the distributed key-value store. It is worth to mention that the components of the metadata nodes are not CPU intensive, thus they can be typically replicated for tolerance but are not usually required to scale out, which is usually a requirement driven by the increased needs for throughput or data size. The data engine nodes on the other hand consist of the query engine, the local transactional manager and logger, and the data store nodes. The number of the data engine nodes required to be deployed depends on the workload in terms of throughput of the queries and transactions issued by the applications and the volume of data.

Component	Ability to Scale Horizontally (i.e. run in distributed mode)	Expected to need testing in horizontal scaling (No (if no ability to scale horizontally) /Low/Medium/High)	Expected to need testing in vertical scaling (Low/Medium/High)
Metadata Manager (including Transactional, Configuration Manager etc.)	No	No	Medium
Query Engine+Datanode	Yes	High	High

Table 36 - LXS Identification of Deployment Combinations

LeanXcale uses Ansible [8] to be deployed. The database administrator has to define which elements of the data store would be part of the data node services and which one would constitute the metadata services. As already mentioned, usually the KiVi dataserver (LeanXcale's distributed key value store), the query engine along with an instance of the logger of the local transactional manager would consist a data node, while the zookeeper with the commit sequencer, the conflict and configuration manager, along with the KiVi metaserver would formulate a metadata node. Having a set of machines available, the database administrator has to define which one of those will be dedicated for the metadata nodes, and which are available for the data nodes. (S)he can also define the size of memory of the machine and the number of the available CPUs. When all configurations are finished, then during the execution of the playbooks all components are automatically moved to the target nodes, along with all necessary configuration information. Upon initialization, LeanXcale advises this configuration in order to establish proper connectivity among all its components, and starts them with the appropriate order.

8.4.2.1.2 LXS Monitoring metrics

During the run-time, LeanXcale provides a wide set of monitoring information that could be used by a system administrator or the platform itself to ensure normal behaviour according to what has been specified. The monitoring information that is of interest is the information that is being produced by the data nodes, as these are the components that should be scaled in/out in order to improve performance under high workload. The produced metrics can be categorized in two groups: the ones provided by the query engine and the ones provided by the storage data node. The query engine is written in Java and provides monitoring info using the Dropwizard framework [9]. The advantage of the latter is that it can additionally provide statistical information on a monitoring metric, like mean time, mean time between a period of time, the histogram of the metric etc. Dropwizard can be used with a jmx plugin which publishes the metrics as managed beans via the jmx. Other metrics are also published and are available directly via the jmx, while the use of the latter allows to take advantage of Java's built-in monitoring information which is available for every java virtual machine (i.e. number of threads, memory usage, garbage collection statistics etc.). Additionally, the usage of jmx to publish monitoring information makes the integration with any monitoring tool straightforward. Query engine's monitoring information can be grouped by specific categories (version, network, logger performance, query executions, general information etc.). On the other hand, the information that can be obtained from the data nodes provides valuable insights regarding the distribution of the data, statistics of the usage per data table basis which can be used mostly by the query engine optimization component that is responsible to select the most efficient query plan among all candidates in order to improve the overall performance. Due to this, the statistical information provided by the data nodes is frequently relevant only to the query engine itself. The list of all available metrics that are provided by the key-value store can be grouped in four main categories: I/O, memory usage, memory management and data table specific information.

8.4.2.2 CEP

8.4.2.2.1 CEP Configuration/Deployment options

The Complex Event Processing (CEP) engine is a distributed streaming engine made by several components: CEP Orchestrator, Instance Manager, JCECP client driver, Reliable Registry and

Metric Exporter. The CEP can be either deployed in a single node or in a cluster. The Instance Manager is the worker component that does the actual processing and allows the CEP to scale. There may be as many Instance Managers as needed. In a single node deployment Instance Managers that process the same query should be started in the same NUMA node to minimize the communication latencies and maximize the performance.

The CEP Orchestrator is a standalone process that is in charge of managing the CEP cluster. It is used to register and deploy queries and it is not involved in the actual data processing.

The Metric Server is a standalone process used to collect metrics from the rest of components and expose those metrics to the BigDataStack monitoring system.

The Reliable Registry is based on Zookeeper and it stores information related to the query deployments and components status.

The JCEPC driver is the interface between the CEP and other applications and it runs in the client applications.

Component	Ability to Scale Horizontally (i.e. run in distributed mode)	Expected to need testing in horizontal scaling (No (if no ability to scale horizontally) /Low/Medium/High)	Expected to need testing in vertical scaling (Low/Medium/High)
CEP Orchestrator	No	No	No
Instance Manager	Yes	High	High
Metric Exporter	No	No	No
Reliable Registry (Zookeeper)	Yes	Medium or Low	Medium or Low

Table 37 - CEP Identification of Deployment Combinations

At start-up, the administrator decides the number of Instance Managers to launch and new Instance Managers can be added to the CEP cluster at run-time as needed.

8.4.2.2.2 CEP Monitoring Metrics

The CEP provides information about the **throughput** and **latency** of the queries being executed at run-time. All the metrics are collected by the Metric Server component which exposes them the rest of BigDataStack platform. Each Instance Manager sends metrics regarding the CPU consumption, throughput and latency of each operator deployed on it to the Metric Server. The user can set the rate at which these metrics are sent to the Metric Server.

8.5 Implementation and Experimentation

The Implementation and Experimentation plan for the different components of ADW is presented in the following sections.

8.5.1 Pattern Generation

The Pattern Generation component is planned to be initially tested as part of the evaluation scenarios planned at M12 (Inference without Data Access) and M15 (Inference with Data

Access). Under these evaluation scenarios a single user application will be deployed by the BigDataStack platform and instrumented under variable load conditions.

Evaluation Setting: For each evaluation scenario, an application Playbook will have been created by an up-stream process that describes the application services that are to be deployed. This Playbook will be ingested by the Pattern Generation component, which will produce a series of candidate deployment patterns, which can be sent for benchmarking. The true suitability of each pattern will be evaluated based on actual deployment of the user application using the configurations defined in those patterns.

Metrics: When evaluating the deployment patterns created by the pattern generation component, we are primarily interested in two main metrics.

- **Best Pattern Suitability:** First metric we target is *best pattern suitability* for a user's application, i.e. once we have evaluated the different patterns for an application, how good was the best pattern we produced? We want best pattern suitability to be high across a range of application deployments, as if we cannot produce at least one suitable pattern, we will likely cause quality of service violations or at least waste resources during application run-time. A pattern is considered suitable if it meets all of the user's quality of service requirements while not wasting significant computational/memory resources (that are related to cost);
- **Number of Patterns Produced:** One way to get around the problem of missing suitable patterns would be to brute-force generate all possible patterns. However, this would place significant load on the benchmarking functionality discussed later in this section, leading to delays in application deployment as benchmarking evaluates each in turn. Hence, the second metric we consider is the *number of patterns produced*. We want to minimise the number of patterns produced to reduce benchmarking load. We expect that there is a trade-off between best pattern suitability and the number of patterns produced.

8.5.2 Benchmarking

Evaluation Setting: For each data service, an application Playbook will have been created that describes the elementary data services that are to be deployed. This Playbook will be ingested by the Pattern Generation component, which will produce a series of candidate deployment patterns, which can be sent for benchmarking. Following, the various workload aspects (as described for example in Table 35) will be applied in the used benchmarking tools in order to generate relevant load.

Metrics: When evaluating the ability to benchmark, we are primarily interested in two main metrics.

- **Flexible Stress Test size:** The stress test creation should be straightforward and be easy to be configured to scale. Actual scalability limits may depend on available testbed size but the launch process should be agnostic to that size and adaptable based on user input.
- **Parameter sweep definition:** The user should be able to define a number of parameters from which relevant combinations should be found and applied. This ensures coverage of the search space as well as improved model performance in the end.

8.5.3 Model Creation

Evaluation Setting: For each data service, and following dataset acquisition from the benchmarking phase, a relevant performance model will be built, based on the necessary HW, workload and QoS features.

Metrics: When evaluating the ability to predict the behaviour of the service, we are primarily interested in two main metrics.

- **Mean Absolute Percentage Error (MAPE):** MAPE is the mainstream metric for performance prediction of the anticipated QoS levels against the actually achieved ones based on the gathered dataset. A MAPE of less than 20% is in general considered as operational.
- **Response time of the estimation process:** The estimation process should be responsive and not delay the actual deployment stage for too long. A baseline time of about 5 seconds per estimation should be acceptable during the investigation of the various CDPs.

In terms of experimentation, we anticipate in the following months to initialize the benchmarking phase for the various data services in BigDataStack, in order to gather the necessary results for the model creation in Tier-1 of the implementation.

8.6 Next steps

8.6.1 Pattern Generator

It is currently envisaged that there will be two further releases of the Pattern Generation component during BigDataStack, integrating more advanced functionality:

- **Tier 1:** This second version of the Pattern Generation component will replace hardware directory loading from file (T5.1-PG-R2) direct population from the OpenStack cluster infrastructure management system (T5.1-PG-R3). This version will also extend the current service mapping system to incorporate one to many mappings service-hardware mapping (T5.1-PG-R5);
- **Tier 2:** the third and final version of the Pattern Generation component will include automatic construction of service pods, where multiple services can be co-located on particular pieces of hardware (T5.1-PG-R6), in addition to the more traditional mapping functionality provided by the earlier Tiers.

8.6.2 ADW Core

For ADW Core similarly, the initial version targets at meeting requirements with relation to initializing and executing benchmarks in order to collect the data for the next phases. To this end it will cater for load creation, dockerization and setup of the environment and analysis of the needed types of workloads, as well as performing the experiments for the various data services and relevant loads. Furthermore, UI driven benchmark design will be included in this tier.

Two more versions will follow, integrating advanced functionality:

- **Tier 1:** The second version will aim at addressing requirements, aiming at having initial models for the included data services in BigDataStack and the main functionality for adding QoS estimates in the service graph for a given data service deployment;
- **Tier 2:** the third and final version of the ADW Core will handle networked estimated at the entire service graph level.

9 Adaptable Visualizations

Adaptable Visualizations will present graphs and reports of data and analytics outcome in an adaptive and interactive way. Based on the form and the size of the data, different visualizations will be dynamically presented. Performance aspects such as computing, storage and networking infrastructure data, data sources information, and data operations outcomes will be visualized.

9.1 Anticipated functionalities / requirements

The anticipated functionalities / requirements are described in the following tables (Table 38 - Table 40), that are compiled together with the rest of requirements of BigDataStack in D2.2.

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-01	System and Software	USE	ROL-04	MAN
Name	Interactive and Responsive UI				
Description	The system should provide an interactive UI that should adapt to different devices and displays in order to provide a proper operation of the solution and a good user experience.				
Additional Information					

Table 38 – System Requirement (1) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-02	System and Software	FUNC	ROL-04	MAN
Name	Automatic graph selection				
Description	Appropriate graphs and reports should automatically be selected for different data sets.				
Additional Information					

Table 39 – System Requirement (2) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-03	System and Software	FUNC	ROL-04	MAN
Name	Live data for different data sources				
Description	The system should be able to display live data obtained from application probes, resource probes and data operations probes.				
Additional Information	Adaptable selection of sources should be possible both in terms of application, resources or data operations, as well as in terms of the datasets				

	selected and visualized per each one of these cases. Combinations should also be enabled.
--	---

Table 40 – System Requirement (3) for Adaptable Visualizations

9.2 Specification / Design

Figure 27 depicts the most commonly used architecture for visualizing big data.

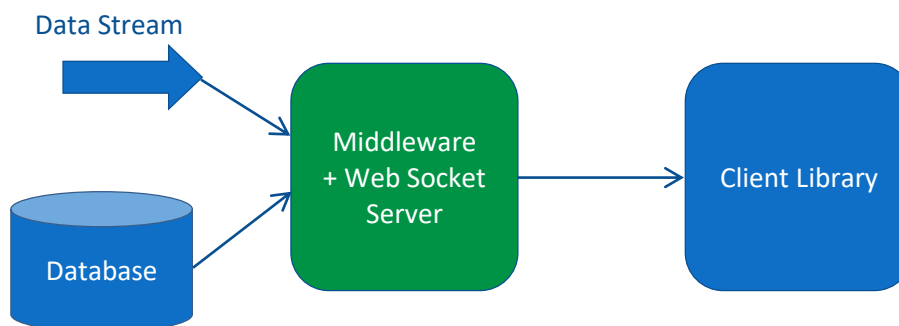


Figure 27 – Base architecture for visualizing big data

The data originate either from a Data Stream or from a Database (NoSQL or Relational). A middleware server component consumes the data and converts them to a format suitable for the visualization client-side library. Live update of the data is achieved through a web socket interface between the server and the client.

Numerous alternatives are available for the data streams, with Apache Spark and Apache Flink being the most prominent ones. Similar many options are available for the Middleware (Node.js, Spring Boot). The client library must provide graph implementations of many types, interactivity, responsiveness and integrations with many Javascript frameworks. State of the art alternatives are:

- D3js [16]: Javascript library for manipulating documents based on Data. It is Open Source software that provides great flexibility and power with the cost of requiring a lot of effort for the implementation of every graph type. For this reason, many wrapper libraries around it are provided;
- Highcharts [17]: Royalty-free, commercial, javascript library. Provides the implementations of hundreds of interactive graph types that can be easily integrated to any Javascript Application;
- Chart.js [18]: Open source javascript library that provides simple yet flexible charting for developers and designers.

9.3 Early prototype

The first prototype of Adaptable Visualizations will be a Single Page Application utilizing Reactjs framework and Highcharts library. Diagrams with dummy data will be initially deployed. This will allow getting feedback about the implementation and going through an iterative development process, before data from other components are ready.

9.4 Experimental Plan

The experimental plan for validation Adaptable Visualization has as first step the implementation of the early prototype. As stated above the early prototype will display diagrams with dummy data. The next step will be to integrate with other components and start receiving real data.

In terms of evaluation metrics and KPIs, the main objective will be to provide all necessary reporting tools for acquiring a complete picture of BigDataStack's runtime operation.

9.5 Next steps

As the project matures, the visualization scenarios will become more concrete. The implementation of the Adaptable Visualizations Components will proceed as follows:

- Define Visualization Scenarios
 - Define connections with other components
- Connection with stream of data
- Create mock-ups with dummy data

10 Conclusions

This document presents the components of one of the main building blocks of BigDataStack, the *Dimensioning, Modelling & Interaction Services*, along with their current design specifications and their initial implementation and status. For every component, the anticipated functionalities along with its architecture are presented. Information is also provided, on component level, regarding the next steps and the experimental plan. *Real-time ship management* UC is used to validate the different releases of the components for the initial prototypes, while all project use cases will be exploited for the next iterations of the designs and prototypes of the dimensioning, modelling and interacting services of BigDataStack.

References

- [1] <https://nodered.org/>
- [2] <https://github.com/node-red/node-red>
- [3] X. Tian *et al.*, "BigDataBench-S: An Open-Source Scientific Big Data Benchmark Suite," *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Lake Buena Vista, FL, 2017, pp. 1068-1077.
- [4] Ivanov *et al.*, "Big Data Benchmark Compendium", *Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things*, Springer International Publishing, 2016, pp. 135-155.
- [5] <https://www.cs.waikato.ac.nz/ml/weka/>
- [6] <https://spark.apache.org/mllib/>
- [7] <https://www.gnu.org/software/octave/>
- [8] <https://www.ansible.com/>
- [9] <https://www.dropwizard.io/>
- [10] Pavel Brazdil, Christophe G. Giraud-Carrier, Carlos Soares, Ricardo Vilalta: *Metalearning - Applications to Data Mining*. Cognitive Technologies, Springer 2009, ISBN 978-3-540-73262-4, pp. I-X, 1-176
- [11] METAL: A meta-learning assistant for providing user support in machine learning and data mining. ESPRIT Framework IV LTR Reactive Project Nr. 26.357, 1998-2001. <http://www.metal-kdd.org>.
- [12] K. Morik and M. Scholz. The MiningMart approach to knowledge discovery in databases. In N. Zhong and J. Liu, editors, *Intelligent Technologies for Information Analysis*, chapter 3, pages 47–65. Springer, 2004. Available from <http://www-ai.cs.uni-dortmund.de/MMWEB>.
- [13] Kate Smith-Miles: Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.* 41(1): 6:1-6:25 (2008).
- [14] Mustafa V. Nural, Hao Peng, John A. Miller: Using meta-learning for model type selection in predictive big data analytics. *BigData 2017*: 2027-2036.
- [15] Daniel Gomes Ferrari, Leandro Nunes de Castro: Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods. *Inf. Sci.* 301: 181-194 (2015).
- [16] <https://d3js.org/>
- [17] <https://www.highcharts.com/>
- [18] <https://www.chartjs.org/>