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/

D5.2 – WP 5 Scientific Report and Prototype Description – Y2

Work Package	WP5 – WP5 Scientific Report and Prototype Description
Lead Author (Org)	Amaryllis Raouzaiou (ATC)
Contributing Author(s) (Org)	Amaryllis Raouzaiou (ATC), Anestis Sidiropoulos (ATC), Richard McCreadie (GLA), Iadh Ounis (GLA), Graham Macdonald (GLA), Christos Doulkeridis (UPRC), Giannis Poulakis (UPRC), Jean-Didier Totow (UPRC), Maria Kanakari (UPRC), George Kousiouris (UPRC), Sophia Karagiorgou (UBI), Gina Chatzimarkaki (UBI), Marta Patiño (UPM)
Due Date	29.11.2019
Date	29.11.2019
Version	1.0

Dissemination Level

X	PU: Public (*on-line platform)
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

The work described in this document has been conducted within the project BigDataStack. This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 779747. This document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

Versioning and contribution history

Version	Date	Author	Notes
0.0	18.09.2019	Amaryllis Raouzaiou (ATC)	Initial ToC
0.1	14.10.2019	Anestis Sidiropoulos (ATC)	Sections 5 and 9 updated
0.2	09.11.2019	Richard McCreadie (GLA), Iadh Ounis (GLA), Graham Macdonald (GLA)	Section 8, GLA Contribution
0.3	15.11.2019	Christos Doulkeridis (UPRC), Giannis Poulakis (UPRC)	Section 6
0.4	18.11.2019	Amaryllis Raouzaiou (ATC)	Sections 1, 2, 3 and 4
0.5	18.11.2019	George Kousiouris (UPRC)	Section 8
0.6	21.11.2019	Sophia Karagiorgou (UBI), Gina Chatzimarkaki (UBI)	Section 7
0.7	22.11.2019	Marta Patiño (UPM)	Section 9, UPM contribution
0.8	25.11.2019	Richard Mccreadie (GLA)	Section 8
0.9	29.11.2019	Amaryllis Raouzaiou (ATC)	Revisions and edits after the internal review of the document.
1.0	29.11.2019	Amaryllis Raouzaiou (ATC)	Final version

Table of Contents

List of tables

List of figures

1 Executive Summary

BigDataStack delivers a complete high-performant stack of technologies addressing the needs of data operations and applications. The main objective of the dimensioning, modelling and interaction services building block of the BigDataStack environment, 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 offerings: the data-driven infrastructure management and the Data as a Service.

The current deliverable is the second (i.e. updated) deliverable that focuses on the aforementioned interaction services building block of BigDataStack. It contains an updated description of all the components (in terms of design specifications), along with the progress done until now (in terms of software prototypes, integration and initial evaluation outcomes), as well as the expected progress until M34.

The main updates of D5.2 are:

1. Updated design specifications of the components, including updated requirements as well as experimentation outcomes.
2. Extended description of the "Adaptable Visualisations" component – the corresponding task had not started by the time D5.1 was delivered.
3. Description of the scenario that was presented during the interim review of the project and demonstrates the integration of WP5 components.

An updated and final version of this report is planned for M34 (D5.3).

2 Introduction

2.1 Relation to other deliverables

The current deliverable, the second BigDataStack deliverable concerning **Dimensioning, Modelling and Interaction Services** (*D5.3* is scheduled for M34 and *D5.1* was delivered in M11) is related to several other BigDataStack deliverables in a direct or indirect way.

D2.1 (State of the art and Requirements analysis - I) and its updated versions *D2.2 (Requirements & State of the Art Analysis - II)* and *D2.3 (Requirements & State of the Art Analysis - III)* identify and specify the technical requirements for BigDataStack both through use case (UC) providers and technology providers, while *D2.5 (Conceptual model and Reference architecture - II),* the updated version of *D2.4,* provides information about the key functionalities of the overall architecture and the interactions between the main building blocks and their components.

We should also underline that the Requirement Tables of the corresponding components of the Dimensioning, Modelling and Interaction Services (Tables 3-19, 25-28, 30-53 and 60-73) are compiled together with the rest of requirements of BigDataStack in *D2.3 (Requirements & State of the Art Analysis - III);* they are included in this document for the reader's convenience.

Finally, D3.2 (WP3 Scientific Report and Prototype description – Y2) and D4.2 (WP4 Scientific Report and Prototype description – Y2) are the deliverables which, in combination with D5.2, 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. *Section 10* contains the conclusions of the current deliverable.

3 Solution Architecture

This section describes the technical solution for the Dimensioning, Modelling & Interaction Services of BigDataStack, based on D2.5. 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 D2.5.

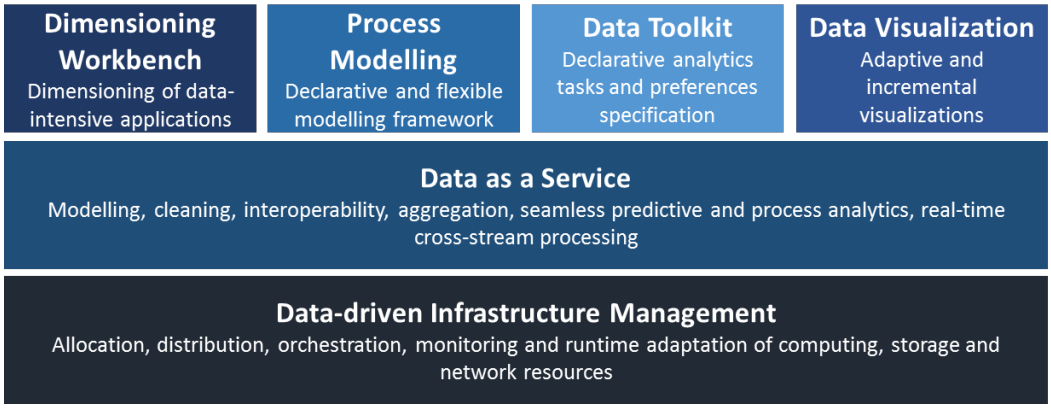


Figure 1 – BigDataStack core platform capabilities (extracted from D2.4)

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 this 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 allows for declarative and flexible modelling of process analytics, while the **Dimensioning Workbench** enables 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 D2.5).

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, **Business Analysts** design processes utilising the intuitive graphical user interface provided by the **Process Modelling** framework through the

Visualisation component, and the available list of "generic" processes. The compiled business workflow is mapped to concrete executable tasks. These mappings are performed by a mechanism incorporated in the **Process Modelling** framework, the **Process Mapping** component.

3. The graph of services is made available to **Data Scientists** through the **Data Toolkit**, where they can also specify their preferences for specific tasks, for example, what the response time of a recommendation algorithm should be.
4. **Data Scientists** are also able to ingest a new executable in case a task has not been successfully mapped by the Process Mapping mechanism.

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.).

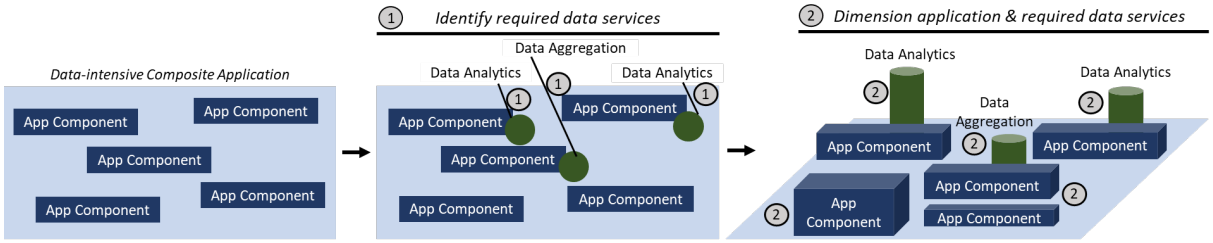


Figure 2 - Dimensioning Phase

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 Quality of Service - QoS) to needed resource parameters (such as the bandwidth, latency etc.).

3.2 Platform Roles

Table 1 lists the BigDataStack roles relevant to the 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

The UI platform has different views depending on the user role, so, apart from the Administrator, who has access to the full UI view, three more roles have been defined:

- ROL-03-Business Analyst (Process Modeller View)
- ROL-02-Data Analyst (Data Toolkit View)
- ROL-04-Application Owner/Engineer (BenchMarking, Dimensioning Workbench, Analytics View)

3.3 Example Scenario

In this section, we provide an example scenario to illustrate how the **Dimensioning, Modelling & Interaction Services** of BigDataStack are envisaged to function and how business analysts, application engineers and data scientists benefit from its functionalities.

Let us assume that we have a stock pricing application for a large European grocery retailer. The application's role is to set the prices for all goods in the consortium's online stores, including adding one-day flash sales to promote regular engagement from customers. There is an important constraint for data scientist devising the big data analytics algorithms and the application engineers deploying and executing those as compute tasks: it needs to run each night after 9pm and needs to be finished before 4am, so that the online storefronts have time to update their pricing before morning traffic.

The application itself is comprised of three main services: the price modelling service, the price application service and the store-front update service. The price modelling service needs to run first as a large batch operation, ingesting all sales from the previous twelve months and updating the internal model about product stock and popularity. This means the model update process will require access to historical big data. Once that service has finished, the price application service runs over all current stock, updating the item prices and adding sales where appropriate. As items are processed, these are sent directly to the store-front update service, which remotely updates the various consortium's store-front databases.

To make the example more concrete we consider the following assumptions concerning the Dimensioning, Modelling & Interaction Services:

a) The BigDataStack infrastructure is deployed on an opaque cloud provided by a public vendor, where compute resources can be requested on demand.
b) The cloud environment is relatively stable in terms of performance and, for the sake of simplicity, we have a unique size of allocable server.
c) The benchmarking phase gives hints (estimates) on the expected computation time depending on different variables, including the opaque cloud configuration, the application deployment configuration and even the day of the week/month, but with some non-negligible uncertainty.
d) The historical big data is stored in a secure datastore managed by BigDataStack, including the specification of what data needs to be processed by what service (e.g., the price modelling service needs the last twelve months' sales information).

Using BigDataStack User Interface (**Adaptable Visualisations**), the **Application Engineer** uploads the application to the BigDataStack platform and the **Business Analyst** specifies the main workflow of their application via the **Process Modelling**, while the **Data Scientist** uses the **Data Toolkit** to create the so-called Playbook that contains all the information related to the preferences of the application (e.g. Service Level Objective (SLO) of *end-to-end completion time* < 7 hours, agreed to be accomplished between 9pm and 4am). The **Application Engineer** can now pass the Playbook to the **Application Dimensioning Workbench**, where it is converted into multiple CDP (Candidate Deployment Pattern) Playbooks, each one describing a potential deployment configuration (what compute and memory resources to request). Each of these configurations will undergo a brief benchmarking step, where the resource usage of the application is estimated. The resultant set of CDP Playbooks with benchmarking information is passed to *Data-driven Infrastructure Management* to optimize its decision-making models.

3.4 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. It is accessed through BigDataStack User Interface (Adaptable Visualisations component) by the Business Analyst.
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, interacting with Process Modelling component. It can be accessed by both Business Analyst and Data Scientist through BigDataStack UI.
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 is the main User Interface of BigDataStack. Different roles have been defined, controlling the access to the interface. Graphs are created, saved or loaded though the UI, while reports of data and analytics outcome are presented 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/graphs are further concretized through the Data Toolkit and its output will be passed to the Dimensioning Workbench. The graphs, the analytics insights and all the relevant information feed the **Adaptable Visualisations** component.

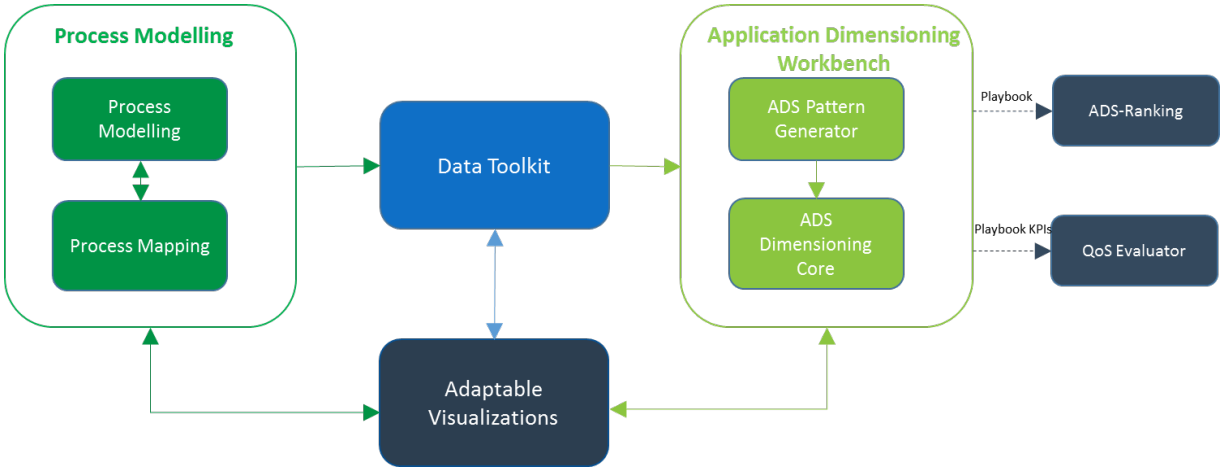


Figure 3 – Dimensioning, Modelling and Interaction Services of BigDataStack

Figure 4 illustrates the building block that provides all the interaction mechanisms.

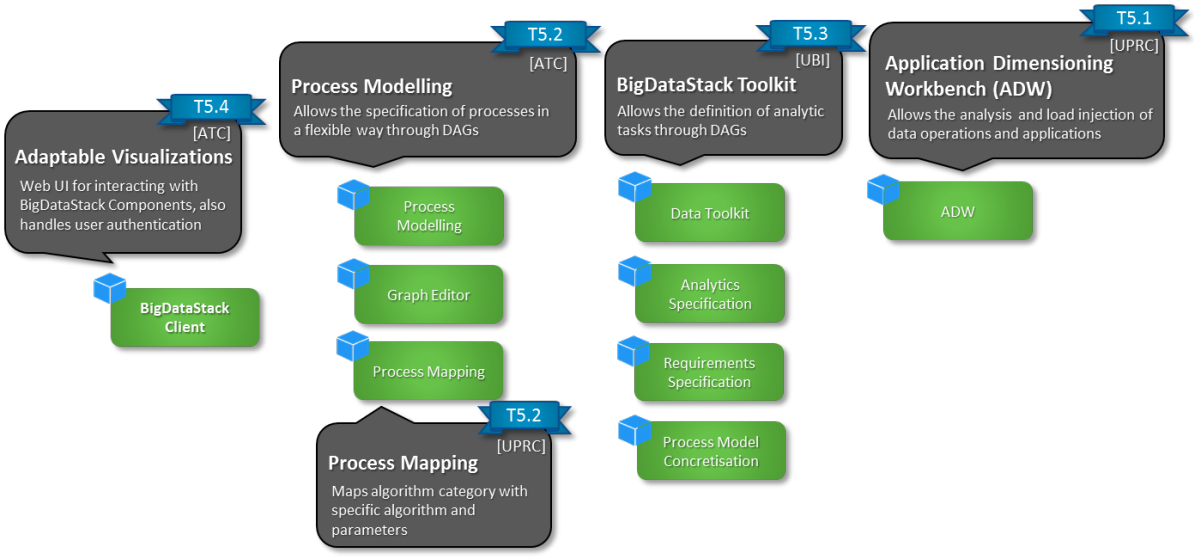


Figure 4 – Interaction Mechanisms

4 Implementation and Experimentation

This section describes how the use cases are being supported through the components of WP5, along with the implementation roadmap towards the project conclusion.

4.1 Experimental Settings

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. **Connected Consumer (CC)** and **Real-time Ship Management (RSM)** are the two use cases with which we are testing the different components presented in this deliverable.

4.1.1 Setting 1

The Connected Consumer (CC) use case, provided by ATOS, deals with a multi-sided market ecosystem (see deliverable D2.1 section 4.2). Some of the highlights of the use case are (please refer to D2.1 for the full description):

- The main challenge is to predict which consumers are the most loyal or which potential buyers are more likely to purchase a certain product or service.
- *Eroski*[1], one of the largest distribution companies in Spain with more than 35.000 workers, is collaborating with ATOS in the definition and test of a use-case related to the grocery business. It is also contributing with real data for the development of the project. The goal of this scenario is to provide data insights to EROSKI to better understand how to create and offer added-value services to their consumers.
- CC use case aims to predict both which products and which promotions are more likely to be interesting for the customers at the right time. From the analysis of different data sources provided by *Eroski*, the goal is first to predict the list of products that customers with recurrent purchases will need in the current purchase period (trend). Afterwards, add to this prediction those products that can be interesting for the user based on other similar user's behaviour (cross-selling). Finally, thanks to a deep knowledge of the customer profile, the goal is also to incorporate those promotions that can be interesting for each customer.

All the components of Dimensioning, Modelling and Interaction Services are involved in the different stages of **CC**.
The Business Analyst uses **Process modelling** framework to define the graph that represents the requested application and includes both application and data services.
This graph is loaded by the Data Scientist, who is using the **Data Toolkit** in order to concretize the analytic tasks.
The application is then analysed through the **Application Dimensioning Workbench**, deployment patterns have been generated and the deployer of BigDataStack has been utilized to perform the actual deployment on the infrastructure and obtain the analytics results through the recommendation engine that has been implemented.
These results / recommendations along with the first steps of the process have been displayed through the **Visualization** module.

[1] https://www.eroski.es/

4.1.2 Setting 2

The Real-time Ship Management (RSM), provided by DANAOS, deals with the maintenance and spare parts inventory planning & dynamic routing (see deliverable D2.1 section 4.1). 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**.
Adaptable Visualisations component offers the UI to all the users of BigDataStack, providing different functionalities according to their role. The Business Analyst logs in the BigDataStack platform and, using the **Process Modelling Framework,** creates a workflow graph or updates an available one, through the definition of the business processes and the associated objectives.
The processes included in this workflow graph will be further concretized through the **Data Toolkit**. Using the Data Toolkit, the Data Scientist can define the data ingestion and the necessary curation tasks for DANAOS dataset (weather data, tracks from vessels) and configure the runtime resources.
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.

4.2 Implementation Roadmap

Table 2 summarises the plan for Dimensioning, Modelling & Interaction Services (M26 and M34 are the tentative dates of the next planned integration meetings).

	M24	M26	M34
Process Modelling Framework	• Enrich the pallet of the available processes to model	• Define constraints between available nodes/processes	• The business analyst able to set apply constraints per node / process of the workflow

			• The business analyst able to apply constraints / parameters per edge (i.e. connections between processes of the workflow). • Enriched and complete collection of services fulfilling all Process Modelling Scenarios
Process Mapping	• Extend/complete the functionality of Process Mapping for clustering	• Integrate with Process Modelling Framework. • (Design) Extend functionality to other ML tasks, e.g., classification.	• (Development) Extend functionality to other ML tasks, e.g., classification. • Support for MLib of Apache Spark
Data Toolkit	• Data serialization in the support of diverse data objects/formats (early prototype).	• Data serialization in the support of diverse data objects/formats (early prototype).	• Data serialization in the support of diverse data objects/formats (mature prototype). This will enable data transformations within a data analytic pipeline to be realized through a message queuing system.
Application Dimensioning Workbench	• integration with the Openshift Application Simulator adapter	• Inclusion of data services in the benchmarking graph and initialization of benchmarking runs.	• Finalization of model creation and online availability for predictions at the graph level.
Adaptable Visualisations	• Integration of Data Toolkit Component. Additional login to Data Toolkit will be bypassed once the user is authenticated via JWT in the BigDS web platform	• Authentication of the user performed once upon logging in the platform. Any additional authentication for individual components should happen in the background without	• Interactive UI adapted to different devices and displays (proper operation, good user experience). • ADW: Application owner imports a playbook produced by the Data Toolkit

		further user interaction.	Component and choose Manual Mode Deployment to get Deployment Recommendations (automatically deployed and monitored). • ADW: Application owner can redeploy functionality • ADW: the user can retrieve the logs for each application • ADW: Application owner can trigger certain decision provided by the Dynamic Orchestrator • ADW: Application Simulator Capabilities (part of Data Toolkit or Adaptive Visualizations) • Dashboards: All use cases vizualized.

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

5 Process Modelling framework

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

The Process Modelling Framework is a straightforward way to provide the ability to produce high-level graphs that describe Business Processes.

5.1 Requirements

The anticipated functionalities / requirements are described in the following tables (Table 3 - Table 12), that are compiled together with the rest of requirements of BigDataStack in D2.3.

	Id[2]	Level of detail[3]	Type[4]	Actor[5]	Priority[6]
	REQ-PMF-01	System and Software	USE	ROL-03	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	N/A				
Status	Fulfilled				

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

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-02	System and Software	FUNC	ROL-02 ROL-03	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	N/A				

[2]**Identifier:** To be used in D2.2 to allow for the correct traceability of requirements.
[3]**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).
[4]**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).
[5]**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.
[6]**Priority:** Requirements can have different priorities: MAN (mandatory requirement), DES (desirable requirement), OPT (optional requirement), ENH (possible future enhancement).

Status	Fulfilled

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

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-03	System and Software	FUNC	ROL-03	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	N/A				
Status	Fulfilled				

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

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-04	System and Software	FUNC	ROL-02	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	N/A				
Status	Fulfilled				

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

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-05	System and Software	FUNC	ROL- 02	MAN
Name	Process workflow export				
Description	The data analyst should be able to export/edit/import 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.				
Status	Fulfilled				

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

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-06	System and Software	FUNC	ROL-03	MAN
Name	Support for end-to-end (in terms of process workflow) objectives				
Description	The business analyst should be able to defile end-to-end objectives. These objectives do not apply to a single process, but to the workflow as a whole.				
Additional Information	N/A				
Status	Fulfilled				

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

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-07	System and Software	FUNC	ROL-03	MAN
Name	Process constraints				
Description	The business analyst should be able to set apply constraints per node / process of the workflow				
Additional Information	N/A				
Status	Not Fulfilled				

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

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-08	System and Software	FUNC	ROL-03	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	N/A				
Status	Not Fulfilled				

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

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-09	System and Software	FUNC	ROL-02 ROL-03	MAN
Name	Save and Edit capabilities of the graph				
Description	The user should be able to export /import/edit/save the generated graph.				
Additional Information	N/A				

Status	Fulfilled

Table 11 – System Requirement (9) for Process Modelling Framework

	Id	Level of detail	Type	Actor	Priority
	REQ-PMF-10	System and Software	FUNC	ROL-02 ROL-03	MAN
Name	Arsenal of Services				
Description	The component should provide an enriched and complete collection of services which will fulfill all Process Modelling Scenarios				
Additional Information	N/A				
Status	Not Fulfilled				

Table 12 – System Requirement (10) for Process Modelling Framework

5.2 Design Specifications

The Process Modelling Component was initially implemented by utilizing as a baseline Node-RED. Subsequently taking the UI/UX into account a migration and refactoring was performed towards the VueJS framework by using ReteJS library.

Rete[7] is a modular framework for visual programming. Rete allows you to create node-based editor directly in the browser. It is possible to define nodes and workers that allow users to create instructions for processing data in your editor without a single line of code.

[7] https://github.com/retejs/rete

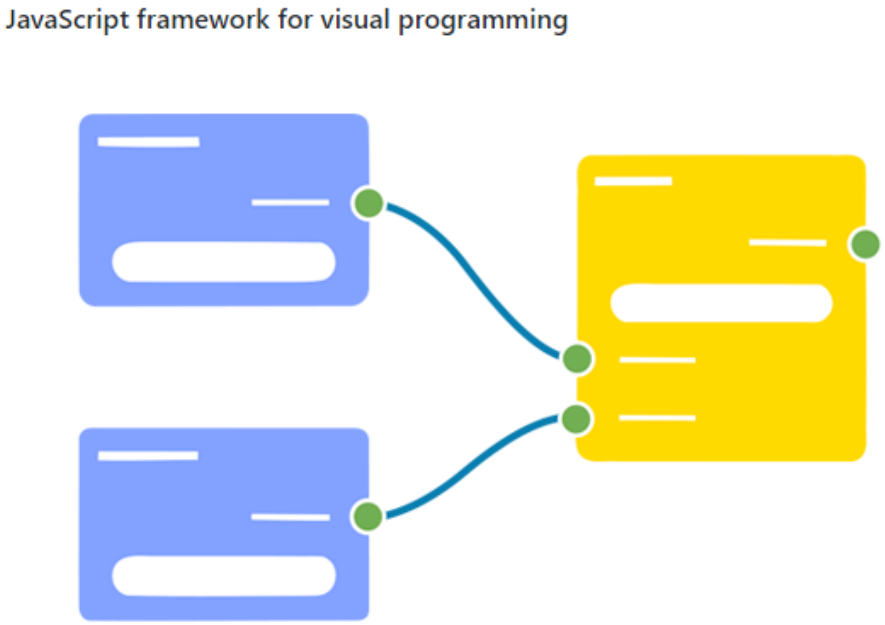


Figure 5 – RETE framework visual example

Using the capabilities provided by ReteJS library the following scenarios were implemented:

1. Provide a wide palette of available processes that can model the uses cases.
2. Create flow of processes.
3. Edit available fields depending on the type of the process. Fields can be:
    a. Process Name
    b. Process Attributes.
4. Define the overall objective of the Process Modeller graph to be generated.
5. Create a high-level graph of processes.
6. Export/edit/import/save of the generated graph in JSON format. This JSON file will subsequently be used as input for the Data Toolkit Component.

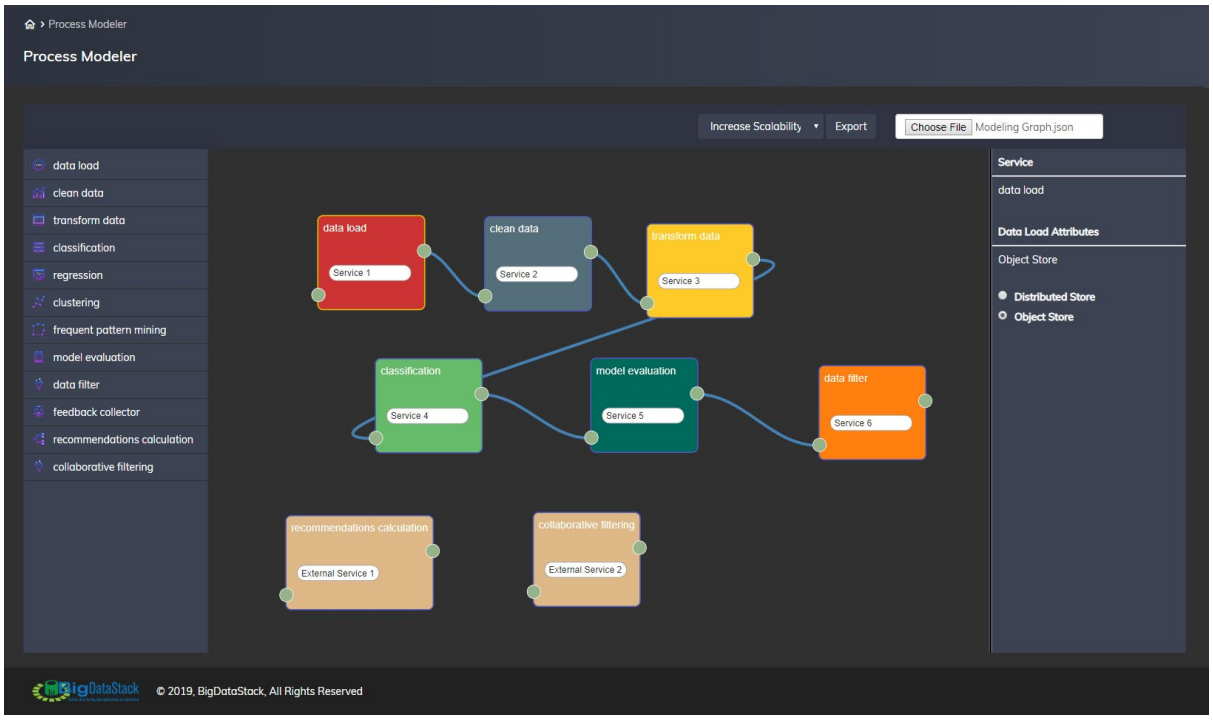


Figure 6 – Process Modeller User Interface

5.3 Experimentation Outcomes

5.3.1 Evaluation results

After integration of the Process Modeller Component in the end to end process provided by BigDataStack environment, graphs were generated to reflect scenarios from the Business Analyst perspective. For each graph, the relevant attributes were defined per process (represented by a node) and the graph description was exported in JSON format. Generated graphs were subsequently consumed by the Data Toolkit Component.
In terms of evaluation metrics and KPIs, the following objectives were fulfilled:

Good UI/UX experience

The process of creating/editing/import and export of the desired Process Modeling graph is straightforward and consistent for the User.

The Business Analyst has the ability to choose services among an arsenal of available services.

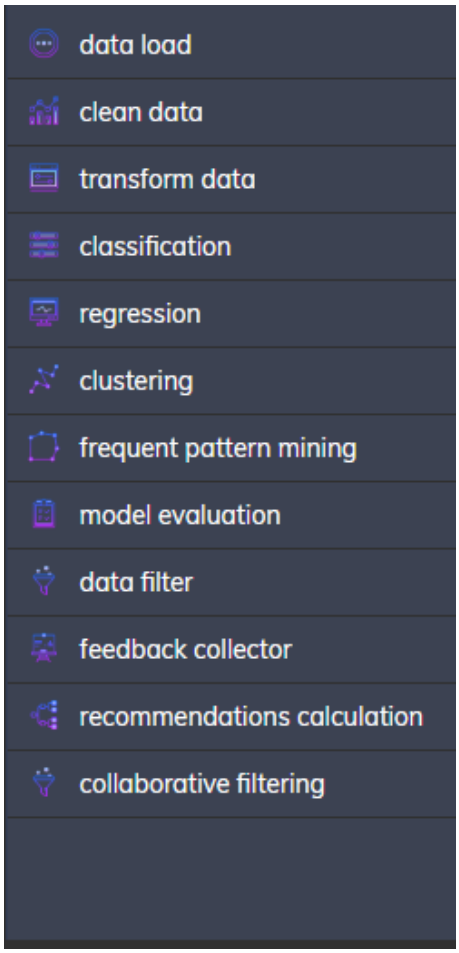


Figure 7 – Process Modeller Services

Each one of the selected services is visually represented by a node. Furthermore, for each service, the user can specify the name of the service and the attributes (attributes are dependent on the type of the service).

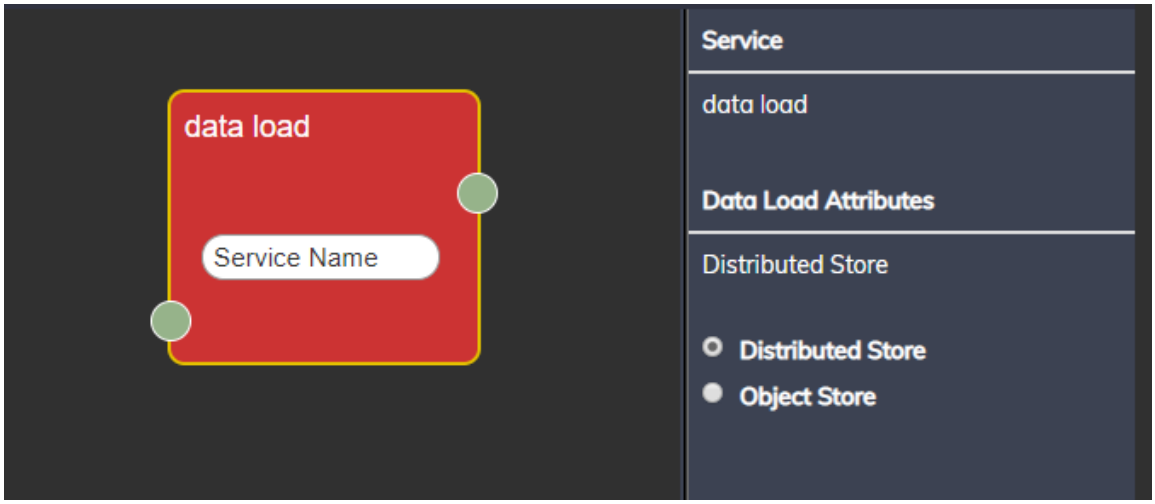


Figure 8 – Service Name and available attributes

Subsequently, the user can create relations among the services by simple drag and drop links.

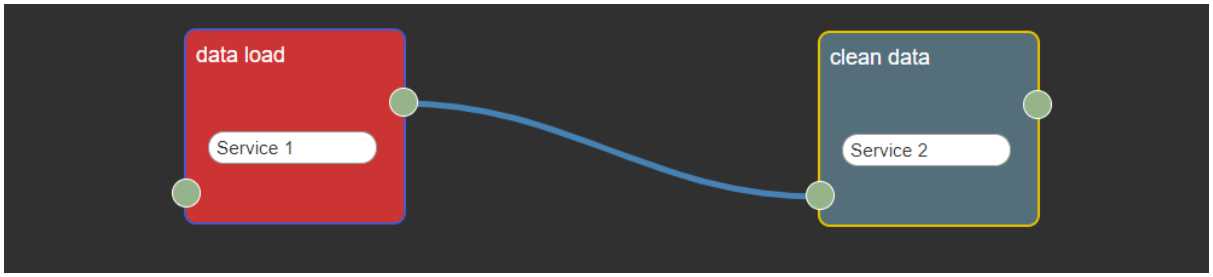


Figure 9 – Linking Services in process Modeller

Upon completion of the graph, the user can define the Overall Objective and export the graph using the horizontal menu on top of the Modeller

By pressing Export, the graph is converted in JSON format and downloaded locally. Additional capability provided by the Modeler is importing a generated graph for further Editing and Updates simply by pressing the Choose file button.

Successful modelling of the use cases

Process Modeler was used to produce graphs that correspond to the use cases. Among others, the following graphs were generated to model the respective processes:

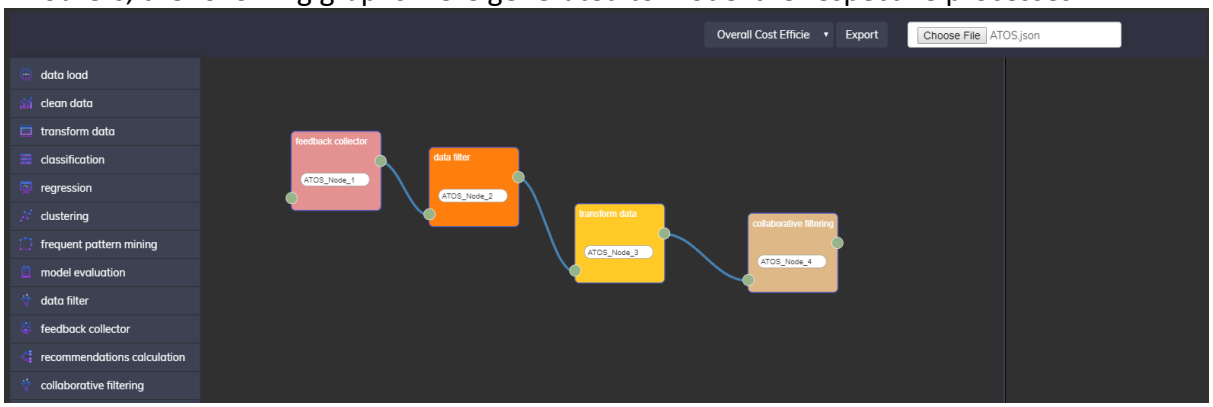


Figure 10 – Process Modelling Use Case Graph 1

Figure 11 – Process Modelling Use Case Graph 2

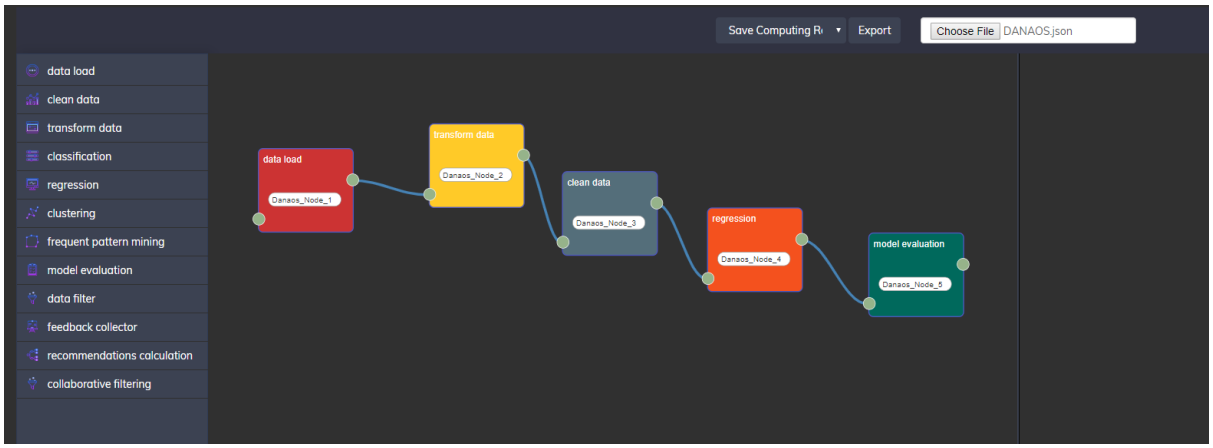


Figure 12 – Process Modelling Use Case Graph 3

Seamless integration with other components

Process Modeler Component is directly consumed and adapted to comply with the unified BigDataStack platform.

5.3.2 Comparison with other approaches

In general, there are limited options on available Visual Programming Components.

In contrast with the initial Process Modelling Framework Prototype (Node-RED), implementation of the Process Modeller using VueJS - ReteJS provided a more structured, well-defined and user friendly component. User Experience is improved and the creation of a Process Modeller Graph is more feasible.

5.4 Integration Highlights

The Process Modelling component was directly consumed by other BigDataStack components in terms of integration. JSON files can be directly imported exported locally.

5.5 Next steps

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

- Enrich the pallet of the available processes to model.
- Define constraints between available nodes/processes.
- Apply validation rules on Node attribute level and Node Connectivity level.

6 Process Mapping

The Process Mapping component targets the problem of selecting the best algorithm along with a set of values for the algorithm's input parameters, 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 so-called "process mapping".

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 (ML) 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 a challenge. Thus, our work focuses mainly on ML algorithms and the automatic selection of the most appropriate algorithm for a given input dataset. This is recently also known as "*automated machine learning*".
In more concrete terms, the key functionality targeted by the Process Mapping component is stated as follows. Given an ML task, a dataset, and a set of available ML algorithms that can handle the given task, the Process Mapping component selects the ML algorithm 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.
At the time of this writing, the focus is on unsupervised learning, namely clustering. Also, the optimization goal is the quality of the result, thus the best performing algorithm is indicated by appropriate *cluster quality indexes*, which are evaluation metrics that assess the quality of clustering.

In comparison to the deliverable D5.1 submitted on M11, the Process Mapping component has significantly evolved in terms of new functionality and features, as listed below.

1) *Pre-processing*. A variety of pre-processing steps has been implemented for data preparation, prior to the execution of task specific algorithms (*e.g., normalization of input data for clustering, handling of missing values, etc*.).
2) *Hyperparameter Tuning*. A new subcomponent has been developed and included in the overall architecture, which is responsible for Hyperparameter Tuning. Essentially, this subcomponent couples the Model Selection subcomponent and its role is to select appropriate values for the input parameters of the selected clustering algorithm in an automatic way. Its functionality is based on *Bayesian Optimization* methods.
3) *Datasets additions*. A more thorough experimental study has been conducted on a bigger collection of datasets, in order to acquire deeper insight on how the performance of clustering algorithms is affected by dataset characteristics and as a result improve the overall performance of the Process Mapping component.
4) *Larger collection of ML algorithms*. Finally, the collection of ML algorithms has been expanded (to the 7 clustering algorithms implemented in the Scikit-Learn library) to broaden the available selection choices, aiming at more realistic application scenarios.

Related Work

Regarding the Combined Algorithm Selection and Hyperparameter optimization problem *CASH,* state of the art solutions such as *AutoML, AutoWeka, TPOT, etc.,* exist and base their work on optimization techniques such as genetic and Bayesian optimization to name a few. Although the frameworks mentioned provide solutions in the context of supervised learning such as classification and regression, none of those frameworks provides solution for the *CASH* problem in unsupervised learning. This is generally due to the lack of information to be used for validation purposes, such as the true clusters of a dataset's instances leading to obscure objective functions to optimize. Although many indices based on Separation and Compactness of clustering schemas, also called internal indices, have been developed to overcome this difficulty, each comes with its own drawbacks and limitations. As a result, the definition of a universal best index for clustering evaluation remains a difficult task.

Algorithm Selection as an individual problem has been previously tackled in the literature by transferring knowledge through meta learning systems [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 most recent work of Ferrari and Castro [15] implements this procedure of Algorithm Selection for clustering problems along with some novelties, such as extracting Meta Features based on the distance distribution of instances. The Process Mapping component extends this work significantly. It adds a Hyperparameter Tuning subcomponent that tries to automatically find optimized values of input parameters, by inferring knowledge from the algorithm selection process. More concretely, in the context of algorithm selection, this allows not only the selection of an algorithm that produced the best results, but also the discovery of input parameters to warm start the optimization procedure.

6.1 Requirements

The Process Mapping component is invoked from the output of the Process Modelling framework. Recall that the Process Modelling framework is used to create process models that contain different types of tasks, including data analysis tasks. In order to map steps of an (abstract) process model to concrete implementations of corresponding algorithms, process mapping is required. In particular, for data analysis or machine learning (ML) tasks, which constitute the main target of our work, a given task can be implemented using different alternative algorithms. Quite often, it is hard for data scientists 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.

The anticipated functionalities / requirements are described in the following tables that have been compiled together with other functional requirements of BigDataStack components and have been recorded in Deliverable D2.3.

	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.				
Status	Not Fulfilled				

Table 13 – 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.				
Status	Not Fulfilled				

Table 14 – 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 *meta-learning*, 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.				
Status	Fulfilled				

Table 15 – 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.				
Status	Not Fulfilled				

Table 16 – System Requirement (4) for Process Mapping

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-05	Stakeholder	FUNC	ROL-04	MAN
Name	Hyperparameter Tuning				
Description	Given a specific machine learning algorithm (determined by algorithm selection), automatically select optimal values for the input parameters.				
Additional Information	The Hyperparameter Tuning process is performed for (a) a given machine learning task (e.g., clustering, classifications, etc.), and (b) the input parameters of each algorithm. Its objective is to automatically compute appropriate values for the input parameters, which will be used for the invocation of the specific ML algorithm.				
Status	Not Fulfilled				

Table 17 – System Requirement (5) for Process Mapping

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-06	Stakeholder	FUNC	ROL-04	MAN
Name	Data retrieval from the storage engine				
Description	The Process Mapping component is able to connect to and query the LeanXcale datastore of the storage engine, to retrieve input datasets or filtered datasets.				
Additional Information	The two sub-tasks of Process Mapping, namely ML algorithm selection and Hyperparameter Tuning, should be considered dataset-–-specific, i.e., they are executed for a given input dataset. This requirement ascertains that the Process Mapping component is able to access the datasets stored in the LeanXcale datastore in order to provide an integrated solution at system level.				

Status	Not Fulfilled

Table 18 – System Requirement (6) for Process Mapping

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-07	Stakeholder	FUNC	ROL-04	MAN
Name	Output recorded in the playbook				
Description	The output of the Process Mapping component is used to update the playbook.				
Additional Information	The output of the Process Mapping component is written in a form that is compatible with the playbook format, in order to enable the execution engine to have all the necessary information for the invocation of the selected machine learning algorithm.				
Status	Fulfilled				

Table 19 – System Requirement (7) for Process Mapping

6.2 System Architecture

The component's architecture is described in two subsections. The first one *(6.2.1)* presents the process pipeline *(Training phase)* of creating a repository of information crucial for the execution of Process Mapping. The second subsection *(6.2.2)* presents the process pipeline (*Selection phase)* for providing solutions to new dataset entries given certain user inputs and the information produced in the training phase.

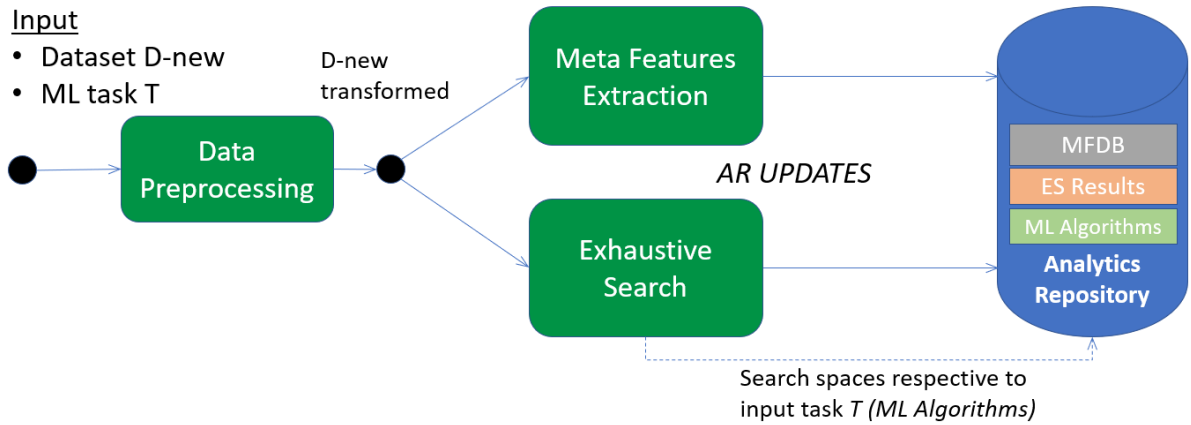


Figure 13 - Training phase of the component/ update procedure of the Analytics Repository *(AR)*

6.2.1 Training Phase – Analytics Repository Creation Process

Before moving forward to explaining the individual subcomponents used in the Training phase, it is important to clarify what the Analytics Repository consists of. The Analytics Repository *(AR)* is a collection of various types of information required for the execution of

the individual processes of the Process Mapping component. This information is collected and updated from executions of ML algorithms on previously seen datasets. Essentially, the system follows a "learning-to-learn" approach, and exploits knowledge assembled from its past usage in order to constantly improve its performance with time. As shown in Figure 13, the collection of information in Analytics Repository falls into three distinct categories:

- Metafeature database (MFDB)
- Results of Exhaustive Search (ES Results)
- Machine Learning algorithms (ML Algorithms)

which are directly linked to the subcomponents as described next.

A. Data Preprocessing

The Data Preprocessing subcomponent is responsible for handling inconsistencies and preprocessing of the input dataset (denoted D_{new}), in order to provide a transformed version that is more suitable as input for the given analysis task. As an example, in the specific case of clustering the Data Preprocessing is responsible for the following transformations on the input dataset:

- Drops columns when they consist only of unique values.
- Drops columns with 30% or more missing values.
- Scales data into [0,1] range by applying Min-Max scaling/normalization.
- Replaces missing values with KNN strategies (regressor for continuous attributes and classifier for discrete attributes).

It should be mentioned that no restrictive assumptions are made with respect to the original input data. It is expected to consist of n rows (records) and m columns (attributes) of numeric values. Obviously, both n and m may vary, depending on the dataset at hand.

B. Meta Features Extraction

This subcomponent is responsible for the creation of data descriptors in the form of numerical vectors that describe various features of the dataset. The Meta Features Extraction subcomponent computes several characteristics of a dataset, also known as meta-features, that capture information about the objects in the dataset, based on Statistics, Information Theory and the distribution of pairwise distances of the objects (Table 20-Table 21). This subcomponent is responsible for producing the information that is stored in the MFDB (Meta Features Database) in the Analytics Repository (AR), which is subsequently used for finding similar datasets to the dataset at hand.

C. Exhaustive Search

The Exhaustive Search subcomponent produces instances of a setting explored (ML Algorithm and a set of values for input parameters) and the proper evaluation metric that was observed for this setting with respect to the ML Task. This is achieved using two of the most well-known exploration methods of Hyperparameter Tuning. The first method that is used is a "brute-force" approach called *Grid Search* that searches every possible combination of values of finite parametric spaces and is computationally intensive. The second one, known as *Random Search*, randomly selects a handful of

combinations from those available for evaluation, is less time-consuming and has proven to outperform Grid Search in certain cases *(see 6.4.3)*.

Both of these methods need a defined search space which is drawn from *ML Algorithms* in the Analytics Repository**.** This information, formatted in standard JSON, is responsible for recording the Machine Learning algorithms that are currently supported by the Process Mapping component, their input parameters and the search spaces for the execution of Grid and Random Search. After the search space is set, every combination of ML algorithm and input parameters is executed for a given dataset in an Analytics Engine (practically a runtime for ML algorithms). The collection of information derived from Exhaustive Search *(ES Results)* is later used for the selection of ML models for new unseen datasets and provides a baseline for evaluating results.

	Meta-Features Based on Information Theory
MA-1	Log2 of the No of Objects
MA-2	Log2 of the No of Attributes
MA-3	Percentage of Discrete Attributes
MA-4	Percentage of Outliers
MA-5	Mean Entropy of Discrete Attributes
MA-6	Mean Concentration between Discrete Attributes
MA-7	Mean absolute Correlation between continuous attributes.
MA8	Mean skewness of continuous attributes
MA9	Mean kurtosis of continuous attributes

Table 20 - Meta Features based on Statistics and Information Theory used for Algorithm Selection

	Meta-Features based on the distance distribution of Instances
MD-1	Mean of distances vector
MD-2	Variance of distances vector
MD-3	Standard Deviation of distances vector
MD-4	Skewness of distances vector
MD-5	Kurtosis of the distances vector
MD-6-14	Percentage of distance values in each of ten intervals that equally comprise range [0,1]

MD-15-19	Percentage of distance values with absolute z-score in four intervals of range [0, inf)

Table 21 - Meta-Features used, based on the characteristics of the distance distribution of instances

6.2.2 Selection Phase – Process Mapping "In Action"

Figure 5 presents the architecture design of the Process Mapping component. The input is provided as a dataset (D_{new}) and a specific analysis Task (*T*), e.g. Clustering, Classification, etc. A Machine Learning algorithm is then selected (or recommended) along with specified values for the input parameters. Based on the ordering presented in Figure 14, a comprehensive explanation of each process of the overall architecture follows.

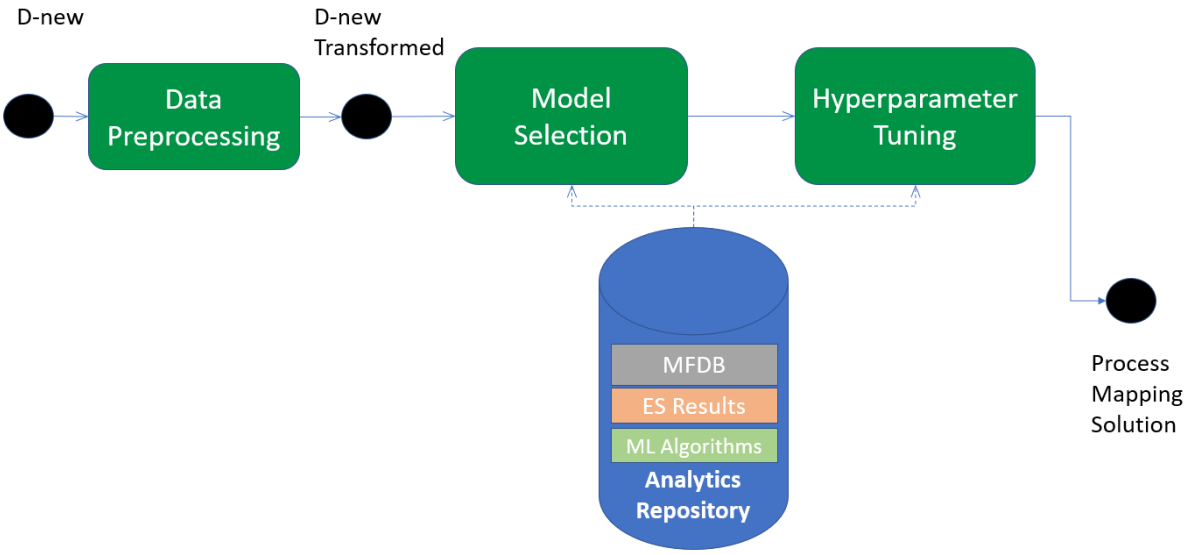


Figure 14 - Process Mapping System Architecture

A. Model Selection

The main purpose of this process is to select or recommend a set of ML algorithms for the input dataset D_{new} and analysis task *T*. The idea behind this process is that similar datasets should have similar solutions for a given task *T*. In order to define similarity among datasets, several characteristics, known as meta-features, are extracted from each dataset and create a feature vector which allows for the calculation of similarity measures between datasets. The feature vector consists of measurements based on Statistics and Information-Theory *(*Table 20*)* and the distribution of pairwise distances of the instances *(*Table 21*).*

When this process is initiated, the meta features of the dataset are extracted and its similarity is computed against the meta features of instances of the Metafeature DB (MFDB), corresponding to previously seen datasets, located in the analytics repository. Then the most similar dataset is found along with the algorithm that

produced the best result, based on the Exhaustive Search that occurred during the training phase.

B. Parameter Tuning

After the Model Selection has concluded its operation, the Process Mapping component is able to return a specific ML algorithm that is considered suitable for the dataset at hand. However, it is yet unclear what values should be used for its input parameters. For instance, in the case of K-means the number of returned clusters should be provided as input. As another example, in the case of DBSCAN, two input parameters must be set: MinPts and Eps.

To address this problem, the Hyperparameter Tuning process is responsible for the optimized selection of the input parameter values. It is based on *Bayesian Optimization*, a state-of-the-art method that searches the global optima of an objective function f(x) through a surrogate model of the f(x) due to cheaper evaluations.

Depending on the hypothesis for the surrogate model, two approaches can be distinguished: (a) *Gaussian Process Bayesian Optimization* and (b) *Bayesian optimization with Tree Parzen estimator*. The former employs Gaussian processes to model the target function because of their expressiveness, smooth and well calibrated uncertainty estimates and closed-form computability of the predictive distribution. Typical downsides of this approach include cubical scaling due to the increase in the number of data points and poor scalability to high dimensions. The latter, instead of modeling the probability $p(y|\lambda)$ of observations y given the configurations λ, models density functions $p(\lambda|y < a)$ and $p(\lambda|y \geq a)$. Given a percentile α the observations are divided in good and bad observations and simple 1-d Parzen Windows are used to model the two distributions. The ratio $\frac{p(\lambda|y<a)}{p(\lambda|y\geq a)}$ is related to the expected improvement acquisition function and is used to propose new hyperparameter configurations. Bayesian optimization with TPE is conceptually simple and can be naturally parallelized [16]. In the implementation of Process Mapping, the Tree Parzen Estimator was selected due to its inherent ability to handle both categorical and discrete values.

For Bayesian Optimization to build the surrogate model, a number of evaluations of the objective function are needed to initiate the procedure. With respect to the previous subcomponent, we select a parametric space close to the parameters already suggested, in order to speed up and improve the overall performance for scenarios with limited resource budget. For float parameters the new space is defined as [x-0.15, x+0.15] and for integers x+-2. For the specific case where the number of clusters is required as input parameter, the search space is defined as [2, $int(\sqrt{\frac{Nsamples}{2}})$] where *Nsamples* is the number of instances of the input dataset.

6.3 Implementation and Integration Highlights

Implementation Changes

In order to accommodate the requirements of the Process Mapping component, the programming language used for implementation of the proof-of-concept prototype presented in M11 was Java. At the current stage of the project, our prototype is based on Python as programming language. This choice complements many improvements of the component, as it supports a wide variety of tools for Data Science. More specifically, Python libraries such as Scikit-Learn offer a large pool of implemented ML algorithms that can be used in the process of algorithm selection. Furthermore, state of the art optimization techniques are widely supported in many open source Python projects, thus enabling the addition of Hyperparameter Tuning in the Process Mapping component. Last, but not least, in the context of Big Data, Python is supported by all modern frameworks for distributed storage and parallel processing (e.g., MongoDB, Spark, etc.). This feature enables handling and processing of big volumes of data, which in turn broadens the applicability of our prototype in use cases related to Big Data.

System Integration with Process Modelling Framework

The output of Process Modeller, a JSON formatted file that contains information about the processes the Business Analyst chose for her analytics Task via the graphical user interface, can be ingested by the Process Mapping component before it is (in turn) recorded in the playbook of the Data Toolkit component. After its execution, the Process Mapping is able to include the selected algorithm and a set of values for the input parameters in this JSON formatted file, and later record it in the playbook. This procedure is instantiated if the Process Mapping is activated as part of the process design procedure by the Business Analyst.

Proof-of-concept

To visually demonstrate the functionality of the Process Mapping component, the procedure has been tested on synthetic data of two dimensions. The data was produced with the use of scikit-learn, a well-known Python library in the Data Science domain. A set of two hundred instances was created pseudo-randomly, centered around four centers with standard deviation of 1. This configuration is well suited for showcasing the performance of the component in Clustering scenarios as the data belong by definition in visually distinct clusters. Process Mapping was able to select the optimal setting (ML algorithm and a set of values for input parameters) for clustering the synthetic data according to their structure in under 40 seconds. The original dataset and the clustering result are depicted in Figure 15.

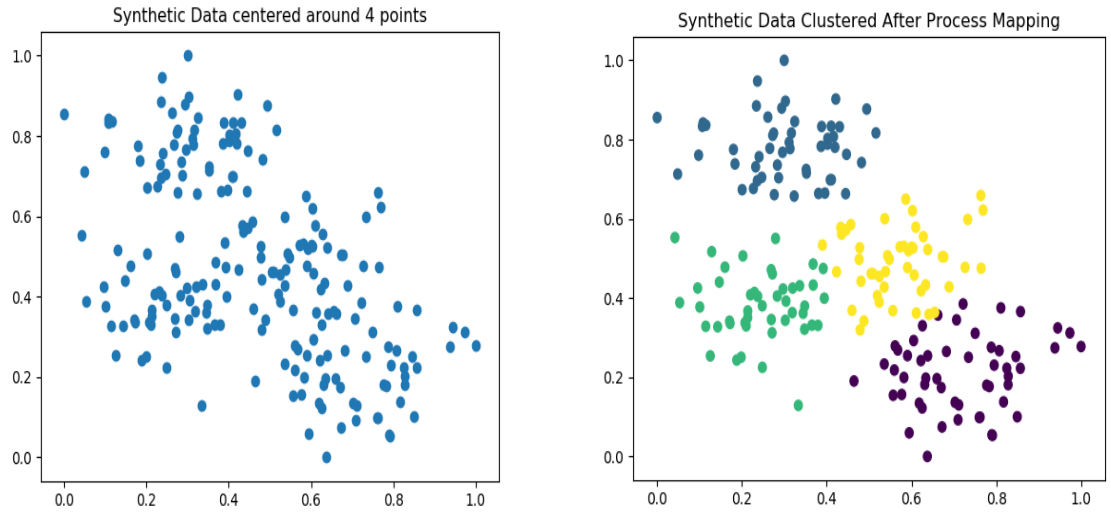


Figure 15 - Testing of Process Mapping for simple Synthetic Data

6.4 Experimental Evaluation

We conducted an experimental study to demonstrate the advantages of Process Mapping using real-life datasets. As already mentioned, our prototype is implemented in Python.

Datasets. A collection of 45 datasets in total was used for the evaluation of clustering: 40 real-life datasets were downloaded from the UCI Machine Learning Repository, while the other 5 from an open to the public collaborative dataset repository namely Data World. All of them were pre-processed in order to form the dataset used in our evaluation. The datasets used are the following:

UCI: Absenteism_at_work.txt, ae_train, buddymove_holidayiq, c1r4r_01, c1r4r_02, c1r5r_01, c1r5r_02, c1r6r_01, c1r6r_02, c1r7r_01, c1r7r_02, Frogs_MFCCs, gesture_phase_a2_raw, gesture_phase_b3_raw, gesture_phase_c3_raw, HTRU_2, l1n_01, l1n_02, l1n_03, l1_04, l1_05, l1r_01, l1r_02, l1r_03, l1r_04, l1r_05, LG_G-Watch_1, movement_libras_1, movement_libras_5, movement_libras_8, movement_libras_10, mturk_cluster_data, mturk_data_feature, perfume_dataset, Sales_Transactions_Dataset_Weekly, SCADI, seeds_dataset, turkiye-student-evaluation-generic, Wholesale_20customers_20data.

Data World: Indian_Premier_League, Customer_Segmentation, Historical_Public_Debt, Asia_Economic_Outlook, Pokemon_Stats.

Algorithms. As already mentioned, we focus on a specific Machine Learning task (clustering) and we use seven clustering algorithms in our evaluation, those provided by the scikit-learn library, which is a library used quite extensively by many data scientists. In particular, the clustering algorithms used in our evaluation are: AffinityPropagation, K-means, SpectralClustering, Agglomerative, DBSCAN, Optics and Birch. Regarding Hyperparameter Tuning, we evaluate Grid search, Random search and Bayesian Optimization.

Metrics. For the evaluation of model selection, we use accuracy as main metric, which is defined as the number of times model selection decided to suggest the algorithm that performed best on the dataset at hand. The best algorithm is obtained offline by brute-force evaluation of all clustering algorithms. Unfortunately, the evaluation of clustering is a long-researched topic, and various cluster quality (validity) indexes have been proposed and are used in practice, without a clear winner. Therefore, we employ three different cluster quality indexes.

Silhouette Coefficient (SL): Validates the clustering performance based on the pairwise difference of between and within-cluster distances. Higher Values indicate better Clustering.

$$\frac{1}{NC}\sum_i\{\frac{1}{ni}\sum_{x\in c_i}\frac{b(x)-a(x)}{\max[b(x),a(x)]}\}$$

Calinski - Harabasz Index (CH): Evaluates the cluster validity based on the average between and within cluster sum of squares.

$$\frac{\sum_i n_i d^2(c_i,c)/(NC-1)}{\sum_i\sum_{x\in C_i} d^2(x,c_i)(n-NC)}$$

Davies – Bouldin Index (DB): For each cluster C, the similarities between C and all other clusters are computed, and the highest value is assigned to C as its cluster similarity. Then the DB index can be obtained by averaging all the cluster similarities. Smaller values of this index indicate better clustering.

$$\frac{1}{NC}\sum_i max_{j\neq i}\{[\frac{1}{n_i}\sum_{x\in C_i} d(x,c_i)+\frac{1}{n_j}\sum_{x\in C_i} d(x,C_j)]/d(c_i,c_j)\}$$

Composite Score (CS): A linear combination of the 3 indexes presented above, after scaling of them in [0,1] range. Higher values indicate better clustering.

$$CompScore = (SL.Scaled + CH.Scaled - DB.Scaled)/3$$

Evaluation Methodology. Our experimental methodology is structured as follows:

- First, we conduct an experiment in order to demonstrate the accuracy of Process Mapping with respect to selection of appropriate clustering algorithm.
- Then, we demonstrate the advantage of Hyperparameter Tuning by evaluating both the accuracy of the parameter values as well as the performance (in terms of running time) of Hyperparameter Tuning.
- Lastly, we present evidence for including Random Search in the Exhaustive Search section of the Analytics Repository by implementing it as a Hyperparameter Tuning method for three test datasets and comparing it with Grid Search Results.

In the following, we present the results of the empirical evaluation using real-life data sets.

In order to obtain a better understanding on performance, from the results of the Process Mapping component, the evaluation process is distinguished in two main parts: Section 6.4.1 and Section 6.4.2. The former aims to evaluate the accuracy of model, while the latter aims to clarify the performance and subsequently justify the need for Hyperparameter Tuning. Finally, Section 6.4.3 justifies the inclusion of Random Search in our experimental evaluation.

6.4.1 Evaluation of Model Selection

First, the 45 datasets were split into training (75%) and testing (25%). The training dataset was used to extract meta-features, which were stored in the MetaFeaturesDB. Then, we executed all clustering algorithms over these datasets, and we evaluate the generated clustering using different cluster quality indexes (SL, CH, DB), in order to obtain the best performing clustering algorithm for each dataset. Thus, we can select the best performing algorithm that serves as ground truth for the model selection problem based on different cluster quality indexes. In addition, we use Composite Score (CS) which is the linear combination of the three indices already mentioned.

Then, for each dataset of the test set, we extract the respective metafeatures and use a k-nearest neighbor (kNN) classifier in order to find the k most similar datasets in the training set. Similarity between metafeatures is computed in two different ways: using the Euclidean distance or using the Cosine similarity. We varied the value of k from 1 to 10, and obtained the algorithms that performed best for these k nearest neighbors (datasets) to the given dataset. Then, we used *majority voting* in order to select the algorithm that is selected by the Process Mapping component, i.e., we select the algorithm that performed best in most of the k datasets.

Figure 16 presents the accuracy of model selection for different cluster validity indices: SL, CH, DB and CS. Also, the charts depict the obtained accuracy when using the Euclidean distance and the Cosine similarity. We observe a general trend, namely that accuracy increases for higher values of k, although in some cases the accuracy drops. This is due to the fact that higher values of k return more datasets that are deemed similar to the one at hand, and (consequently) more algorithms are returned as candidates for selection. When the same algorithm is returned many times, this is strong evidence that it is suitable for the dataset at hand, and this is reflected in the increased accuracy values. However, in some cases (e.g., from k=1 to k=2), two different candidate algorithms are returned, and then the selection is practically random, which may cause a decrease in accuracy values.

When comparing the absolute accuracy values obtained, we observe that the use of CH (accuracy 88%) and DB (accuracy 87%) result in the highest values. This means that when we perform model selection based on these cluster quality indexes, we obtain higher accuracy. The SL metric performs worse than all others do, and the CS method is in between these two extremes.

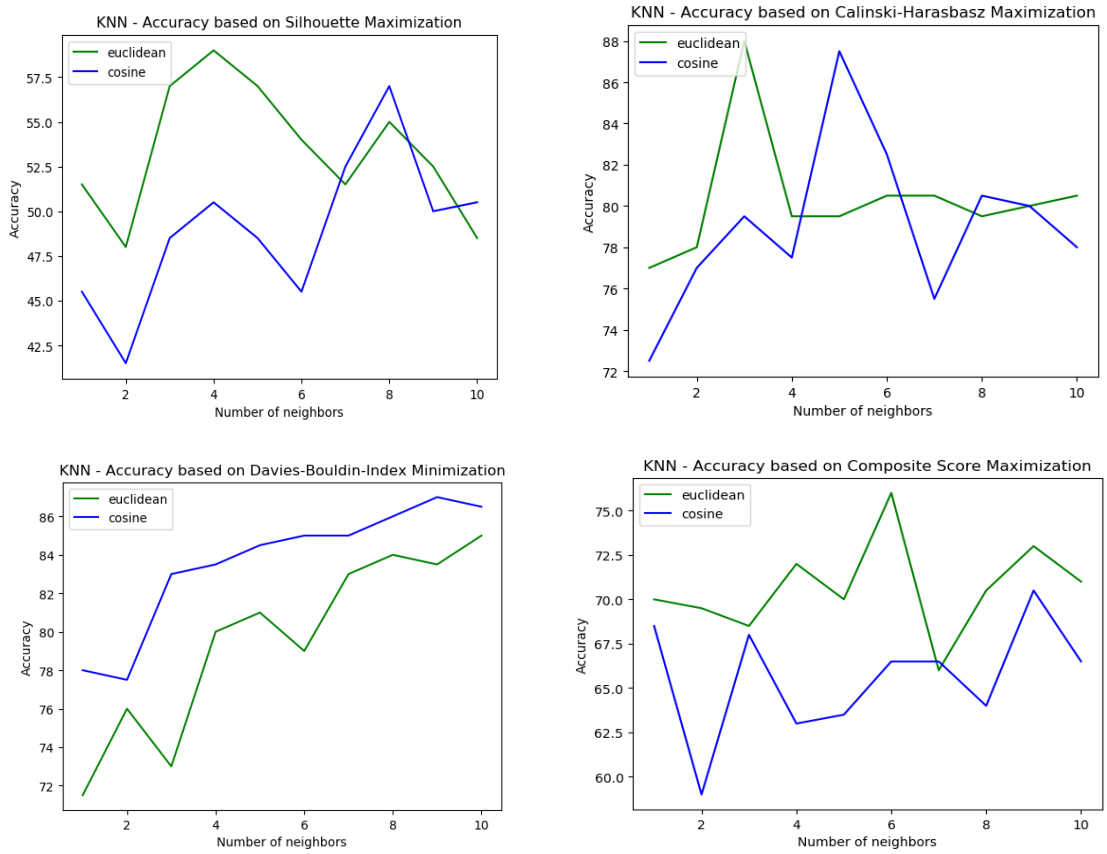


Figure 16 - Algorithm Selection Evaluation

When considering the two alternative similarity measures (Euclidean and Cosine), we observe that only in the case of DB, there is a clear winner, namely Cosine. In the other cases, the results are mixed, although in most cases the use of the Euclidean distance returns higher accuracy values.

Figure 17 shows how accuracy is affected when Process Mapping returns the top-N algorithms (1, 2 or 3 algorithms), instead of a single algorithm. Recall that the pool of available algorithms contains 7 algorithms, therefore we are interested to explore the accuracy values when returning 2 or 3 algorithms, since this is always much better (in terms of saving time of the data analyst) than following a brute-force approach that would execute all 7 algorithms and pick the best performing one.

As can be seen in Figure 17**,** when using CH as cluster quality index and the top-2 or top-3 algorithms, we obtain very high accuracy values (95.5% and 98% respectively). This is strong evidence about the merits of our approach, since it can be interpreted as follows: when using CH and suggesting 2 or 3 algorithms, we manage to find the best performing algorithm in at least 95% of the cases. DB and CS also achieve high accuracy values. SL manages to achieve 91.5% accuracy when selecting 3 algorithms.

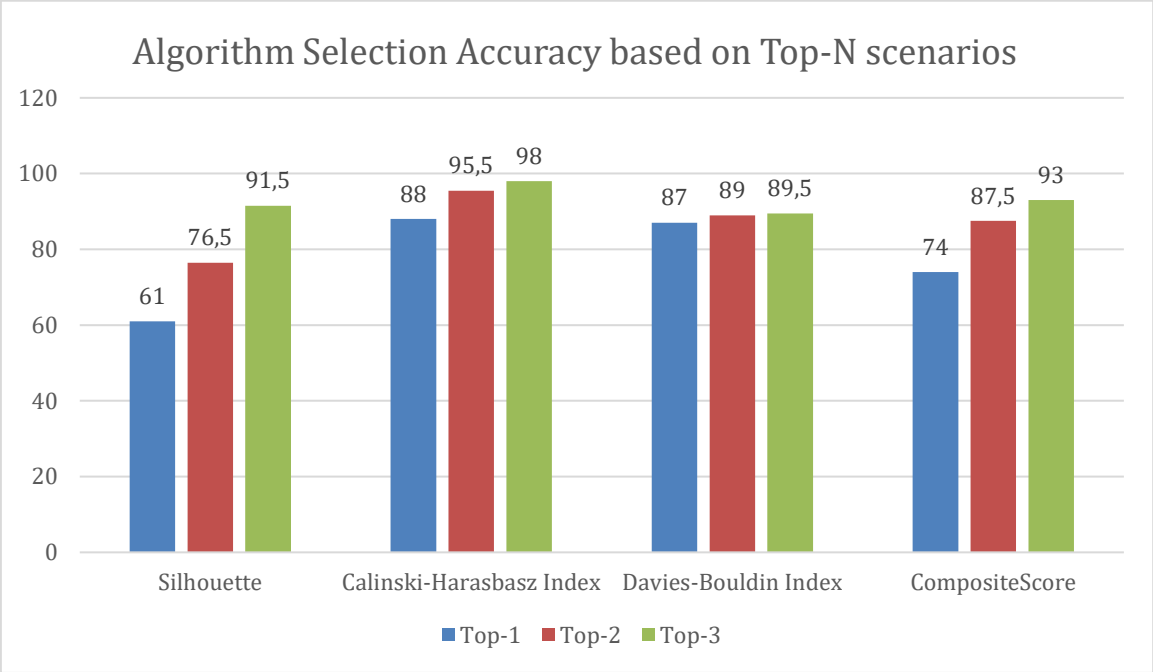


Figure 17 - Top-N accuracy of Algorithm Selection, neighbours and distance metric chosen from the best setting of Figure 13

In summary, our Process Mapping approach achieves high accuracy values (88%) when using the CH cluster quality index and selecting only 1 out of 7 available algorithms. Another important finding is that if we can return 2 or 3 algorithms, instead of a single one, the use of the remaining cluster indexes produces results of sufficiently high accuracy too. These results demonstrate that model selection for clustering is indeed a well-performing solution, with as few as 30 training datasets.

6.4.2 Evaluation of Hyperparameter Tuning

In this experiment, we evaluate the performance of the Hyperparameter Tuning subcomponent, which relies on Bayesian Optimization. We compare the result obtained by our approach against the result of Exhaustive Search (ES), using: (a) the number of times that our selection led to better results compared to the best achieved in ES Results and (b) the deviation of the rest from the best achieved in ES Results. Deviation is measured using the Mean Absolute Error (MAE), which is the absolute difference of the scoring of the model suggested after Bayesian Optimization from the best score obtained by Exhaustive Search.

$$MAE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}} |y_i - \hat{y}_i|$$

Low values of the MAE metric indicate that the expected performance of the parameters selected from Hyperparameter Tuning will be approximately close to the best performing parameters of Exhaustive Search. For experimentation purposes, the number of evaluations is set to fifty (50) of the surrogate models, for every one of the 45 datasets.

Index	MAE (When Bayesian OPT Results are lower than ES)	No Instances Bayesian OPT outperformed ES
Silhouette Coefficient	0.1134	14/45

Table 22 - Hyperparameter Tuning Evaluation

Table 22 demonstrates the ability of Bayesian Optimization to select parameters, that (a) outperform traditional Hyperparameter Tuning methods, or (b) underperform only by a negligible amount compared to the best solutions of ES.

The MAE value 0.1134 can be described as approximately how much lower will be the scoring of the cluster schema selected by our Hyperparameter Tuning compared to the best scoring clustering schema of an offline Exhaustive Search. Taken into consideration the range of Silhouette Coefficient [-1, 1], 0.11 is reasonable difference in evaluation metric, for results produced in a much more timely manner than ES Results.

In addition, for 14 out of 45 datasets the Bayesian Optimization method was also able to achieve higher Scorings than ES, showing its capability to outperform Grid Search and Random Search.

6.4.3 Random Search in Analytics Repository

The inclusion of Random Search as a method for optimizing input parameters of a given ML algorithm stems from the inherent ability of the method to achieve better results compared to Grid Search in certain cases. When the number of input parameters that account for the optimization of a function is relatively small, either because the respective algorithm requires the definition only of a few or because only a handful of them are responsible for the variance of an objective function, Random Search is able to achieve better results most of the times in a less computational intensive manner [17].

In Table 23 the results of the two methods are compared, to provide a better understanding of the advantages of Random Search and validate its inclusion in our approach.

MODEL	GS_Parameters Suggested	SC (Grid Search)	RS_ParametersSuggested	SC (Random Search)	RS>Gs
Affinity Propagation	damping: 0.5	0.3032	damping: 0.9222286448558172	0.3550	True
Kmeans	N_clusters: 3	0.5857	N_clusters: 2	0.5056	False
Spectral Clustering	N_clusters: 2, Gamma: 0.5	0.5741	N_clusters:2 Gamma: 1.407633	0.5056	False
Agglomerative	N_clusters:3, Affinity: Euclidean, Linkage: Ward	0.58524	N_clusters:2, Affinity: Cosine, Linkage: linkage	0.50555	False
DBSCAN	Eps: 0.2,	0.36372	Eps: 0.41511246,	0.58526	True

	Min_samples: 5		Min_samples: 7		
Optics	Cluster_method: "xi", min_samples: 6	0.11201	Cluster_method: dbscan, min_samples: 2	0	False
Birch	N_clusters: 3, Threshold: 0.3	0.58524	N_clusters:210, Threshold: 0.361893	0.58636	True

Table 23 - Comparison of the two methods for a single dataset (In terms of Silhouette Coefficient) that comprise Exhaustive Search in the Analytics Repository.

The results of Table 23 are produced by running Grid Search and Random Search to optimize the input parameters of seven different algorithms for the ML task of clustering for a given dataset. In 3 out of 7 cases Random Search is producing better results in terms of the Silhouette Coefficient. Those cases include a few input parameters to be specified, some of them being real-valued. This indicates that in order to produce results that will later be referenced for comparison both of these methods should be used. It is worth mentioning at this point that Random Search is much faster (its execution time is more than 7 times smaller than Grid Search), as Random Search evaluates much fewer models than Grid Search (Table 24).

METHOD	Execution Time *(in minutes)*	Sets of Input Parameters Explored
Random Search	0.67	50
Grid Search	4.66	1639

Table 24 - Execution Specifications of the two Methods

6.5 Next steps

Next steps and planned activities of Process Mapping component are the following:

- Extend the functionality of the component for supporting the algorithm selection and Hyperparameter Tuning for other ML tasks (e.g., classification).
- Support of the MLib of the Apache Spark framework, for parallel execution of the ML Algorithms.
- System integration in the BigDataStack architecture, using an appropriate demonstrator.

7 Data Toolkit

Today's current processing infrastructure capacity and price and the availability of large amount of data have enabled the development of new and more complex applications. However, in order to fully exploit such opportunity, a team should deal with different expertise, coming from the business domain and coupled with diverse programming skills and infrastructure maintenance capabilities. Sometimes, one wants just to test a hypothesis about the data having the role of a Business Analyst. Other times, one knows the technical details, having the role of a Data Scientist, and needs to concretize the parameters of an analysis task. To both engage Business Analysts and Data Scientists and let them collaboratively join their forces, we introduce the Data Toolkit in the BigDataStack project.

The Data Toolkit enables the end-users design, implement, experiment, test and deploy data processing tasks coupled with machine learning capabilities in order to set up more complex and data-intensive applications. The Data Toolkit also provides the means to visually design and define the analytic workflows through a higher level of abstraction and graphical user interfaces which are used in order to set the required parameters, objectives and dependencies for these machine learning applications.

7.1 Requirements

The Data Toolkit facilitates Business Analysts and Data Scientists build operational analytic workflows by means of data pipelines through Directed Acyclic Graphs (DAGs). The data pipelines consist of a group of processing tasks, which can be instantiated through the respective microservices. The graphs consist of nodes and edges with properties where the end-user can define the starting and ending stage and the intermediate processing stages that 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 level of abstraction (i.e. 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, the technical information and knowledge to specify more details in the workflow set up, she has also the ability to define connection details and interfaces 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 ability to define, configure and parameterize nodes, edges and properties 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 and per edge.
- 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 and in different business domains.

The tables that are following (Table 25 - Table 28) describe the requirements engineering method and have been compiled together with the rest of requirements of BigDataStack in D2.3.

	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	This is a support regarding 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.				
Status	Fulfilled				

Table 25 – 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	N/A				
Status	Fulfilled				

Table 26 – 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 ensuring that the services interdependencies have been correctly specified. 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, and therefore the application descriptor via the Yaml file cannot be produced.
Additional Information	N/A
Status	Fulfilled

Table 27 - 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	N/A				
Status	Not Fulfilled				

Table 28 - System Requirement (4) for Data Toolkit

7.2 Design Specifications

The UI of the Data toolkit is provided as a web-based application and the created application graph can be exported as a yaml file and sent to the OpenShift container application platform of the BigDataStack project for execution. This yaml file can be also used by the Dimensioning Workbench and the Dynamic Orchestrator. From the UI of the Data Toolkit, the user can access the main Dashboard which contains an overview of the resources used, historical data and services logs, as presented in Figure 18. She can also access the Components, the Applications and the Instances.

The Application Definition and Management UI provides a dedicated page with forms that can be used to define the Components. The definition of the Components through the UI allows the configuration of the services, the requirements deployment, the definition of the type of the component (if it is a component or a function), the container image it uses, its interfaces, and other helpful parameters, as depicted in Figure 19, Figure 20 and Figure 21.

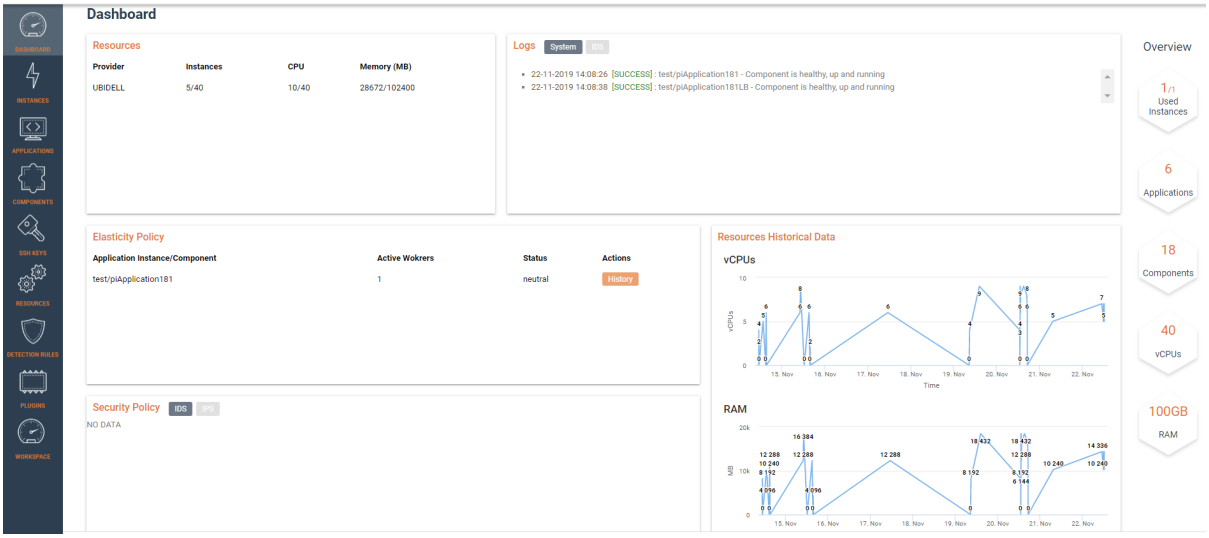


Figure 18 - Data toolkit Main Dashboard

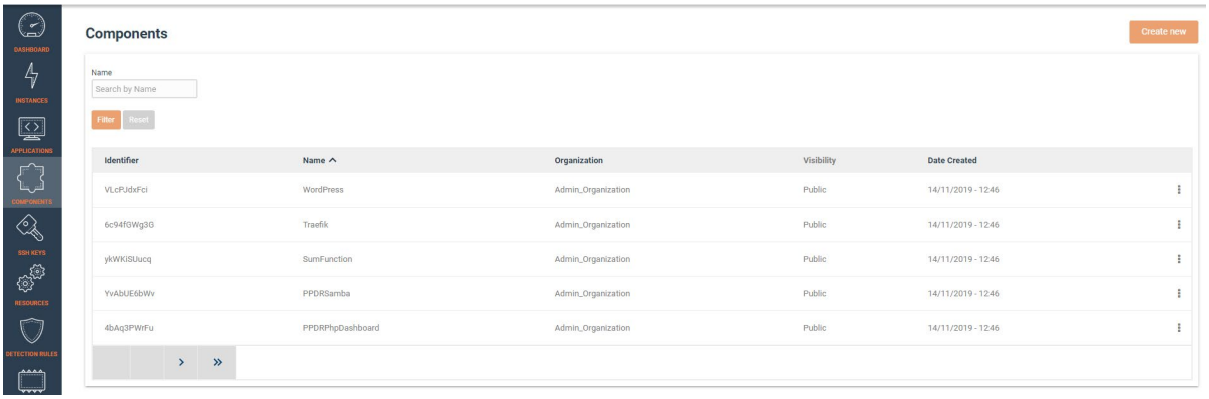


Figure 19 - Creation of new Components

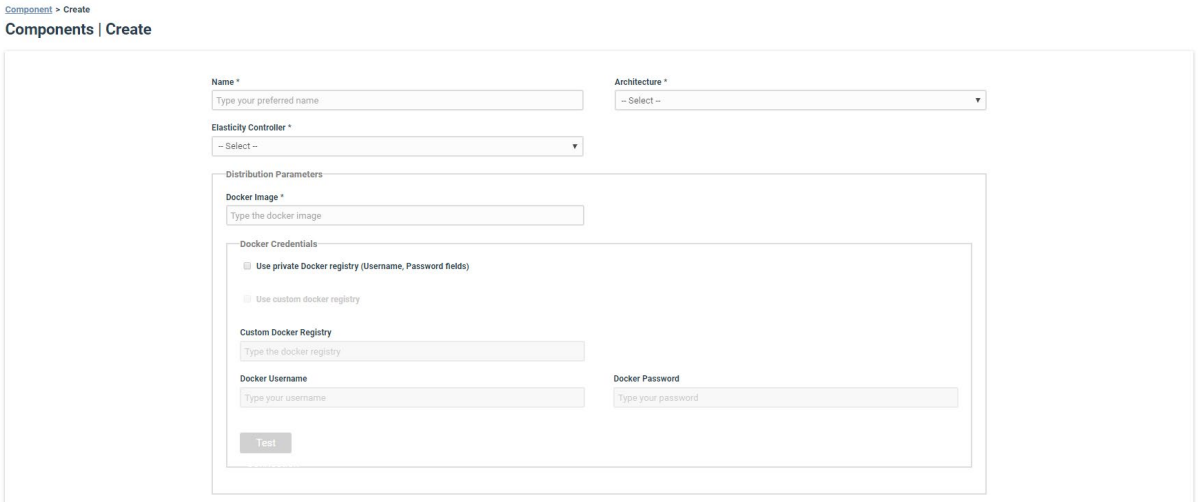


Figure 20 - New Component configuration 1

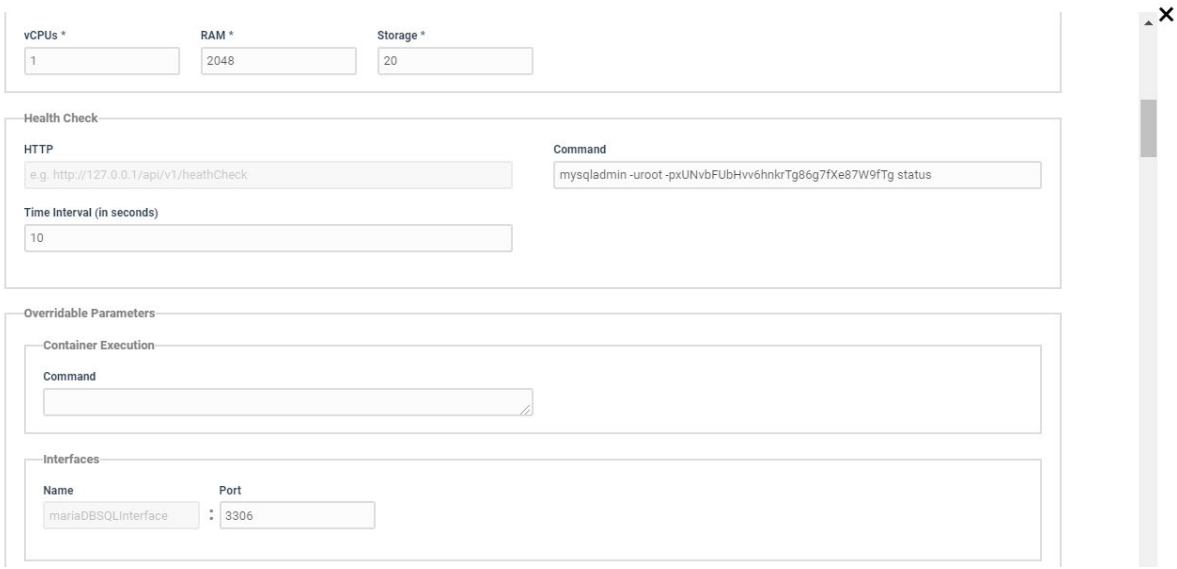


Figure 21 - New Component configuration 2

After the user creates all the needed Components of her data pipeline, the configuration of the application can be performed from the graph editor. As seen in Figure 22, the graph editor can be used to configure the Application and the application Components. Through the graph editor, the user defines the connections between the Components, by connecting the appropriate, matching interfaces of the components. Execution requirements can be also defined, and custom health checks can be added to ensure that the service is deployed properly. She can also specify the minimum and maximum requirements on how two services are interacting (i.e. Hard/Soft constraints), as presented in Figure 23. For example, a new Application Instance has been created namely “App Inst” having a front-end interface coupled with a back-end database, as presented in Figure 24.

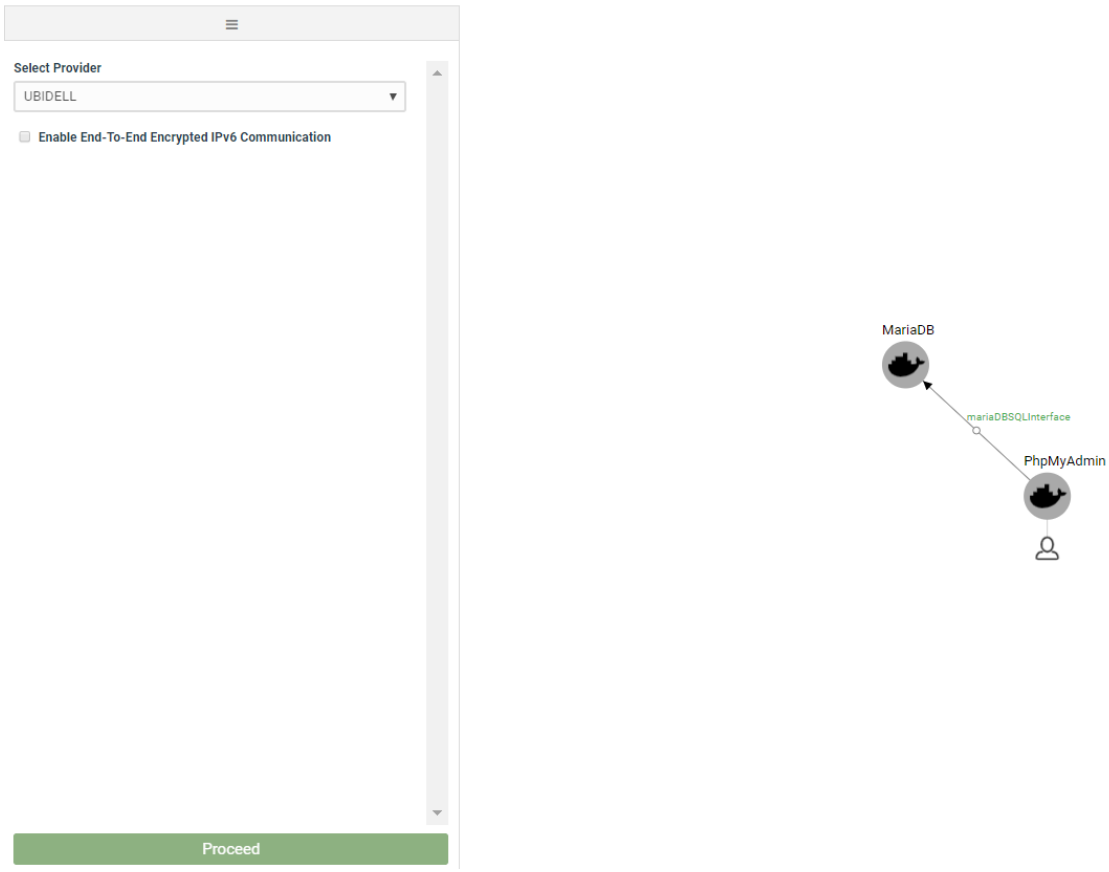


Figure 22 - New Application Instance configuration

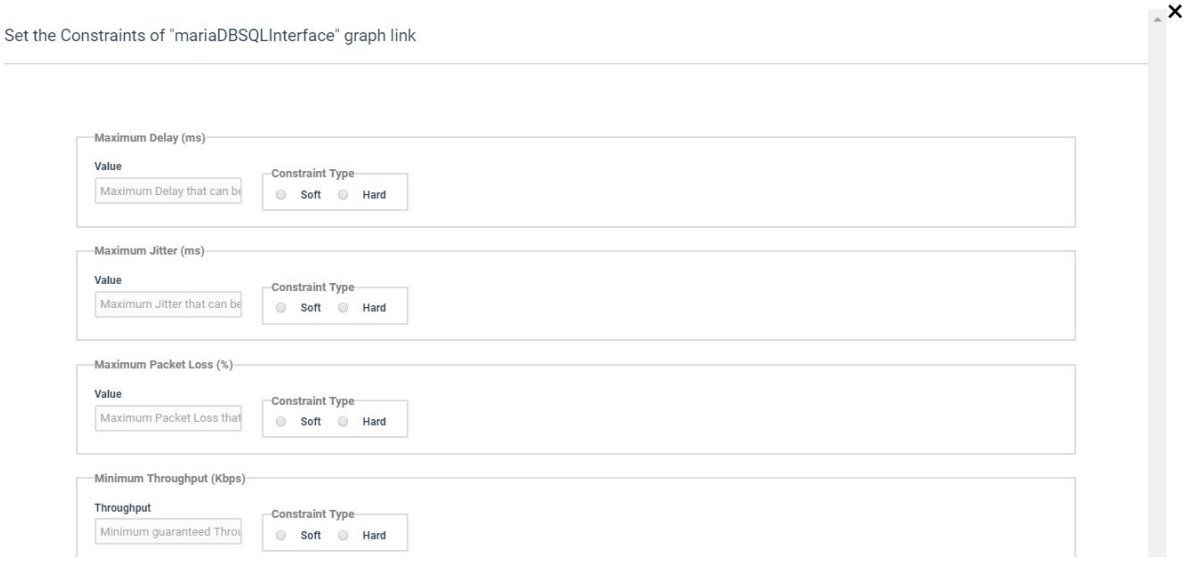


Figure 23 - Constraints over interacting services

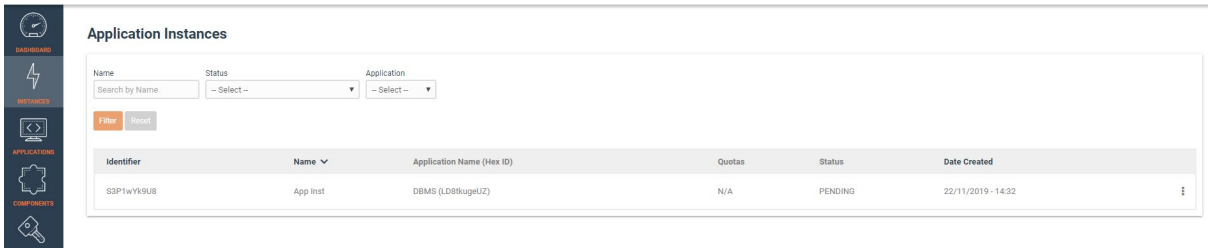


Figure 24 - New Application Instance creation

7.3 Implementation and Integration Highlights

The UI of the Data Toolkit has been developed using React, while the back-end services have been developed in JAVA. React is a powerful JavaScript library for building user interfaces and is used to couple the back-end services of the Data Toolkit with the front-end functionalities delivered to the end user. The several back-end functionalities developed in JAVA are exposed as microservices through REST APIs. Therefore, the front-end UIs interact with the back-end services through REST APIs.

Also, the Data Toolkit gets as input a JSON file produced by the Process Modelling framework which represents the high-level application graph defined by the Business Analyst. The Data Scientist uses the Data Toolkit to configure and set the several parameters, requirements, constraints and objectives regarding her analytics application. The output of the Data Toolkit is produced in a yaml file which contains the necessary information to execute the analytics application in the OpenShift container application platform over the Dynamic Orchestrator.

7.4 Experimentation Outcomes

The Data Toolkit as it does not introduce or advance any methodology or algorithm, it has not been experimented over existing approaches. The comparative analysis performed includes the literature review, deployment of proof-of-concept of similar frameworks and workflow engines in order to perform a gap analysis and identify the features which are good to incorporate in the Data Toolkit.

The efficient execution of analytics workflows and the management of resources are recently realised through automated workflow engines providing a collection of functionalities and abilities to configure and extend their settings. In the context of the BigDataStack project, optimization and automation of workflows on different frameworks as presented in the following have be examined for performance, along with tasks scheduling on resources to meet resource constraints/performance constraints/time constraints. Depending on the workflow and the frameworks in use, we also explored the abilities of the framework's scheduler to optimize the workflow and guarantee analytics efficiency and time performance.

The StreamSets DataOps Platform[8] operationalizes data flows and enables continuous data delivery by addressing the entire design-deploy-operate lifecycle of data pipelines. It supports functionalities which facilitate to design simple pipelines or complex data flows comprising dozens of pipelines in a single topology using a cloud-based visual UI with pre-built origins,

[8] https://streamsets.com/

destinations and transformations. Pipelines are executed in memory on standalone systems or scalable distributed systems using YARN, Mesos or Kubernetes mechanisms.

Apache Airflow[9] is a platform for programmatically author, schedule and monitor workflows. It's one of the mostly maintained project in the GitHub[10] community. Apache Airflow makes the workflow a little bit simple and organised by allowing to divide it into small independent (not always) task units that are easy to organise and easy to schedule. The entire workflow can be converted into a DAG with Airflow. Once the workflows are defined by the corresponding code, they become more maintainable. With the feature rich user interface, the workflow pipelines can be easily visualised, monitored and troubleshooted. Airflow also provides a rich set of command line utilities, which can be used to perform complex operations on DAG.

Spring Cloud Data Flow[11] provides tools to create complex topologies for streaming and batch data pipelines. The data pipelines consist of Spring Boot apps, built using the Spring Cloud Stream or Spring Cloud Task microservice frameworks. Spring Cloud Data Flow supports a range of data processing use cases, from ETL to import/export, event streaming, and predictive analytics.

The Data Toolkit differentiates to the current offerings in the direction of simplicity and reusability. It consists of a web-based application coupled with a set of microservices developed in JAVA to support the required functionalities for the BigDataStack project. It can be easily set up and fulfils the entire life cycle of designing, configuring and deploying complex data-intensive analytics applications. Its ease of use lies within the simplicity where it does not require over killing definitions and configurations to efficiently realize data pipelines. It only requires designing, defining and configuring the connection details, the sequence and constraints of the interacting microservices. The presented frameworks are either coupled with programming languages specificities (i.e. Python, JAVA, etc.), or cannot be easily parameterized and configured due to licensing restrictions. As a result, we focused on the fulfilment of the BigDataStack requirements, including the loading of a high-level application graph in JSON, easy to use UIs to further concretize the interacting services and the production of a yaml file which can be easily deployed over any cloud-native environment.

7.5 Next steps

Towards a more complete Data Toolkit, the following steps need to be completed:

- Addition of more functionalities supporting and streamlining ML as a service capabilities.
- Usage of an intermediate queueing system to address data transformation and serialisation issues among different format in end-to-end analytic pipelines.

[9] https://airflow.apache.org/
[10] https://github.com/
[11] https://spring.io/projects/spring-cloud-dataflow

8 Application Dimensioning Workbench

The Application Dimensioning Workbench (ADW) component aims to provide information regarding the necessary resources needed and respective QoS levels that would aid in making informed decisions during the deployment and runtime management process of an application. To this end, it needs to obtain baseline configurations, data samples and performance models at the level of the base components (application level components and data service components) and use these in order to populate potential combinations of resources that would satisfy the end user. Overall, the subcomponents included or used as part of the ADW are presented in Table 29. The ADW performs two main roles within BigDataStack using these sub-components. Firstly, during initial installation of the BigDataStack platform, the Openshift Application Simulator Adapter is used to collect initial data points on the performance of the underlying infrastructure for different applications and workloads. Second, when a user uploads a new application into BigDataStack, the Pattern Generator and ADW Core can be used to benchmark that application to aid in subsequent selection of resources for deployment for that application.

It is worth highlighting key pieces of terminology used in this section which are critical to understand the functioning of the Application Dimensioning Workbench:

- **BigDataStack Playbook**: This is a description of a user's application as provided from the Data Toolkit component of BigDataStack. It specifies services to be deployed and desired quality of service (QoS) for the application, but not how that application should be deployed.
- **Candidate Deployment Pattern Playbook**: This is a description of how to deploy a user's application on the available cluster infrastructure, derived from a BigDataStack Playbook. One BigDataStack Playbook can have multiple Candidate Deployment Pattern Playbooks, representing the different ways that application might be deployed.
- **Dimensioned Deployment Playbook**: This is a Candidate Deployment Pattern Playbook with additional injected information about predicted quality of service and resource usage of the application. Dimensioned Deployment Playbooks are used by downstream components such as ADS-Ranking for deployment configuration selection.

Component Name	Purpose
Pattern Generator	Create various candidate deployment combinations for inspection, represented as Candidate Deployment Pattern Playbooks.
ADW Bench (included in ADW Core)	General load/benchmark management and execution framework for simplifying and coordinating data acquisition process and load injection
ADW Runtime (included in ADW Core)	Used to understand the deployment descriptor structure of the Pattern Generator suggestions, retrieve performance data from the benchmarks and populate the anticipated QoS fields. In effect, the ADW Runtime takes Candidate Deployment Pattern Playbooks as input and outputs Dimensioned Deployment Playbooks.

Openshift Application Simulator Adapter (OASA)	This is a stand-alone component designed to collect initial performance information from various user application types on the cluster infrastructure, when BigDataStack is first deployed.
Application Type Experiment Plugin	These are plugins for the OASA that enable particular pre-configured standard applications to be deployed.

Table 29 - List of ADW related parts and their functionality

8.1 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 30-Table 35), that are compiled together with the rest of the requirements of BigDataStack in D2.3.

	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	N/A				
Status	Requirement met in latest version.				

Table 30 – 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	N/A				
Status	Requirement met in latest version.				

Table 31 – 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	N/A				
Status	Scheduled for implementation in Tier 2.				

Table 32 – 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	N/A				
Status	Requirement met in latest version.				

Table 33 – 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	N/A				
Status	Scheduled for implementation in Tier 2.				

Table 34 – 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	N/A				
Status	Scheduled for implementation in Tier 2.				

Table 35 – System Requirement (6) for Pattern Generator

8.1.2 ADW Core

The ADW Core functionality extends across two areas:

a) 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.
b) Provide predicted QoS and resource usage predictions for individual candidate deployment patterns (produced by the Pattern Generation component)

Requirements gathered and refined from D2.1 as well as the technical process in BigDataStack are presented in the following tables (Table 36-Table 49) with relation to the ADW Core. These tables are compiled together with the rest of the requirements of BigDataStack in D2.3.

	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 able to be specified in order for the system to estimate the anticipated QoS levels for the desired range of application level workload. 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).				

Table 36 – 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).				
Status	Completed				

Table 37 – 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 Candidate Deployment Pattern 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.				
Status	Partially completed at the level of understanding and populating the service graph, pending part to integrate with performance measurements retrieval.				

Table 38 – 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.				
Status	Completed (load injection clients report on such values).				

Table 39 – 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	N/A
Status	Completed for Jmeter, Partially completed for YCSB.

Table 40 – 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	N/A				
Status	Completed				

Table 41 – 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	N/A				
Status	Completed through the automation of the benchmarking configuration.				

Table 42 – 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.				
Status	Stress testing and benchmarking is performed prior to the deployment process, therefore it is not included in the delays for that phase, only the enquiries towards the results. Stress test deployment has been met through the dockerization process of the benchmarks.				

Table 43 – System Requirement (8) for ADW Core

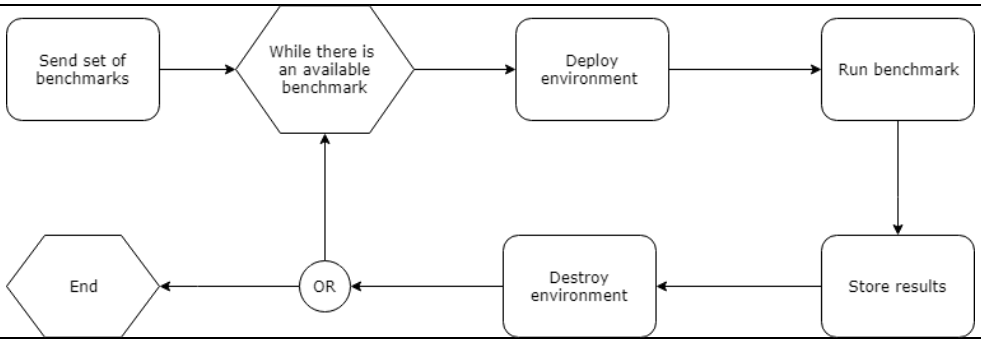
	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-05	Software	FUNC	ROL-04	ENH
Name	Benchmarking Workflow implementation				
Description	During the benchmarking phase, there should be a controlled manner in which the various combinations described in REQ-SY-DW-02 and REQ-SO-ADW-03 are enforced during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

Table 44 – System Requirement (9) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-06	Software	Non FUNC/USE	ROL-04	MAN
Name	Trace driven simulation				
Description	Except for the parameter sweep definition of a stress test that is used in the context of REQ-SO- ADW-03, the ADW user should also be able to define a trace file with a time series of load aspects for the framework to undertake				

	its sequential implementation, simulating a historical fluctuation of demand for the service.
Additional Information	Trace definition should be made as simple as possible, for example through uploading a trace file on a gitlab account. Trace file structure should also be defined e.g. one row per setup.
Status	Completed

Table 45 – System Requirement (10) for ADW Core

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-SO- ADW-07	Software	Non FUNC/SUP	ROL-04	DES
Name	Subflow grouping of relevant implementation parts				
Description	The implemented flows created in the context of the ADW Core can become quite complex and thus difficult to maintain or extend. Thus suitable grouping and formulation into more clear and well defined reusable subflows should be performed.				
Additional Information	This feature should be implemented primarily for the cases of platform descriptor/coordination logic and/or baseline benchmark invocation and incorporation. These are the two main extension points for the ADW load injection.				
Status	Partially completed for configuration nodes, ongoing for cases of larger subflows.				

Table 46 – System Requirement (11) for ADW Core

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-SO- ADW-08	Software	FUNC	ROL-04	MAN
Name	Enablement of parallel and isolated modes of execution				
Description	ADW Load injection can be configured to launch a variety of configurations under the same test setup (e.g. in the parameter sweep definition of a test). The test instances stemming from the parameter combinations should be able to be performed in either a sequential, isolated mode, that guarantees repetitiveness of an experiment under the same conditions or in a parallel mode, in order to check aspects of multitenant applications.				
Additional Information	The tool logic should prevent different modes from being applied at the same time, thus accession to the stress test cluster should be considered as a race condition and handled appropriately.				
Status	Completed				

Table 47 – System Requirement (12) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-09	Software	FUNC	ROL-04	DES
Name	Enhanced benchmark results filtering				
Description	Results acquired through the benchmarking may be queried in multiple, less or more advanced forms. In the case of more advanced ones, setup of the experiment and how closely it relates to a specific desired deployment option should be considered, while an additional level of complexity may be included if the query includes a given metric goal and relevant tolerance for the returned results (in terms for example of a percentage coverage of the goal).				
Additional Information	Definition of the aforementioned parameters should be included in the available interfaces towards the platform so that they can be set by the user (i.e. through a REST API, Data Toolkit annotations, the pattern generator candidate patterns or the end user UI of the ADW tool).				
Status	Partially Completed				

Table 48 – System Requirement (13) for ADW Core

	Id	Level of detail	Type	Actor	Priority
	REQ-SO- ADW-10	Software	FUNC	ROL-04	MAN
Name	Functionality offered through REST APIs				
Description	In order to further automate the various processes such as result querying, test setup submission etc., the main functionalities that relate to these processes should be offered via a RESTful API, so that they can be included in relevant software implementations.				
Additional Information	This functionality is in addition to the user interface built for ADW.				
Status	Completed				

Table 49 – System Requirement (14) for ADW Core

8.1.3 Openshift Application Simulator Adapter and Application Type Plugins

The aforementioned ADW Core component enables the management and coordination of benchmarking experiments on cloud infrastructures for different real user applications. Indeed, this data is critical to provide effective selection of candidate deployment patterns for the user's application (see D3.1 and D3.2). However, early in the lifetime of an installation of the BigDataStack platform, little data on how different application types will perform on the underlying hardware will be available (since the users will not yet have tried deploying their applications). This is an issue, since down-stream components such as ADS-Ranking need to be initially trained/tuned for the particular hardware that BigDataStack has been deployed upon, otherwise such component's performance will be poor. Hence, it is important to provide an alternative solution to tackle this 'cold-start' problem. This is a new addition to

WP5 T5.1, based on further analysis of the BigDataStack Realization components that was carried out in year 2.

The Openshift Application Simulator Adapter (or OASA) is a stand-alone piece of software that is developed within WP5 to deal with the cold-start issue by providing a turn-key solution for collecting basic application performance information. Its role is to deploy a pre-defined set of standard applications onto the cluster infrastructure and collect their performance information under different resource constraints. For the purposes of BigDataStack, we aim to deploy applications that resemble each of the application use cases of the project. In this way, the underlying infrastructure can be interrogated to determine how suitable it is for different applications and workloads. Subsequently, from this data, down-stream components such as ADS-Ranking which need information about how different application types are affected by the underlying infrastructure can undergo initial training. It is envisaged that when the BigDataStack platform is first deployed, the OASA will be run once to collect these initial data points.

The requirements for OASA are listed in the following tables (Table 50-Table 53):

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-T5.1-AS-01	System	FUNC	ROL-04	DES
Name	OpenShift Application Simulator Adapter				
Description	The application simulator needs to support a central control server that can be deployed onto OpenShift. This control server may receive requests from ADW Bench and maintain a queue of application deployment experiments (loaded from a configuration file or provided through the request) that need to be run. Then it will launch experiments and monitor their performance (quality of service and resource usage metrics). It will support both storage of the resultant metrics, as well as metric export in a format suitable for down-stream training of ADS-Ranking.				
Additional Information	N/A				
Status	Requirement met				

Table 50 – System Requirement (1) for OASA

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-T5.1-AS-02	System	FUNC	ROL-04	DES
Name	Simulation Type: Flink-Based Streaming Applications				
Description	The application simulator needs to be able to deploy different types of user applications. This requirement represents the class of real-time streaming applications, comprised of a series of transformers. These types of systems				

	are often used to support alerting use-cases. The Danaos use-case is an example of this application type.
Additional Information	N/A
Status	Requirement met

Table 51 – System Requirement (2) for OASA

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-T5.1-AS-03	System	FUNC	ROL-04	DES
Name	Simulation Type: Supervised Training				
Description	The application simulator needs to be able to deploy different types of user applications. This requirement represents the class of applications that perform batch training of a machine learned model, e.g. using deep learning. The EROSKI use-case contains an application of this type.				
Additional Information	N/A				
Status	Scheduled for implementation in Tier 1				

Table 52 – System Requirement (3) for OASA

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-T5.1-AS-04	System	FUNC	ROL-04	DES
Name	Simulation Type: API Service				
Description	The application simulator needs to be able to deploy different types of user applications. This requirement represents the class of applications that return a data packet on request to a RESTful API. Examples of these types of applications are web-sites and item classifiers. The GFT use-case is an example of this application type.				
Additional Information	N/A				
Status	Scheduled for implementation in Tier 2				

Table 53 – System Requirement (4) for OASA

8.2 Design Specifications

8.2.1 ADW Core System Use Cases

Following the analysis of the requirements in the previous section, we have created the set of system use cases 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 ADW Core user needs to design a range of stress tests/benchmarks (Figure 25) that are needed in order to cater for the data set 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 use cases of the services and applications and/or tailored to the specific use-cases of BigDataStack. To support application benchmarking, base load clients need to be determined and dockerized in order to be used as load injectors. QoS metrics per benchmarked element need also to be defined a priori and the service owner 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 test combinations.

Once the service is deployed, then the stress test (launch of the distributed clients) can be performed. Therefore, there is an asynchronous step for benchmarking to wait until the setup of the stress system is complete.

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

1) Predefined workloads (per data service and/or BigDataStack UC) and ways to feed them as input during the stress test.
2) Configuration options that affect data service/algorithm performance and associated BigDataStack Playbooks.
3) Dockerized base load clients for each tool needed by the BigDataStack data services to emulate load.
4) Main QoS metrics per service and way of acquisition/storage in a given run.

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 use case. 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 during the actual deployment process.

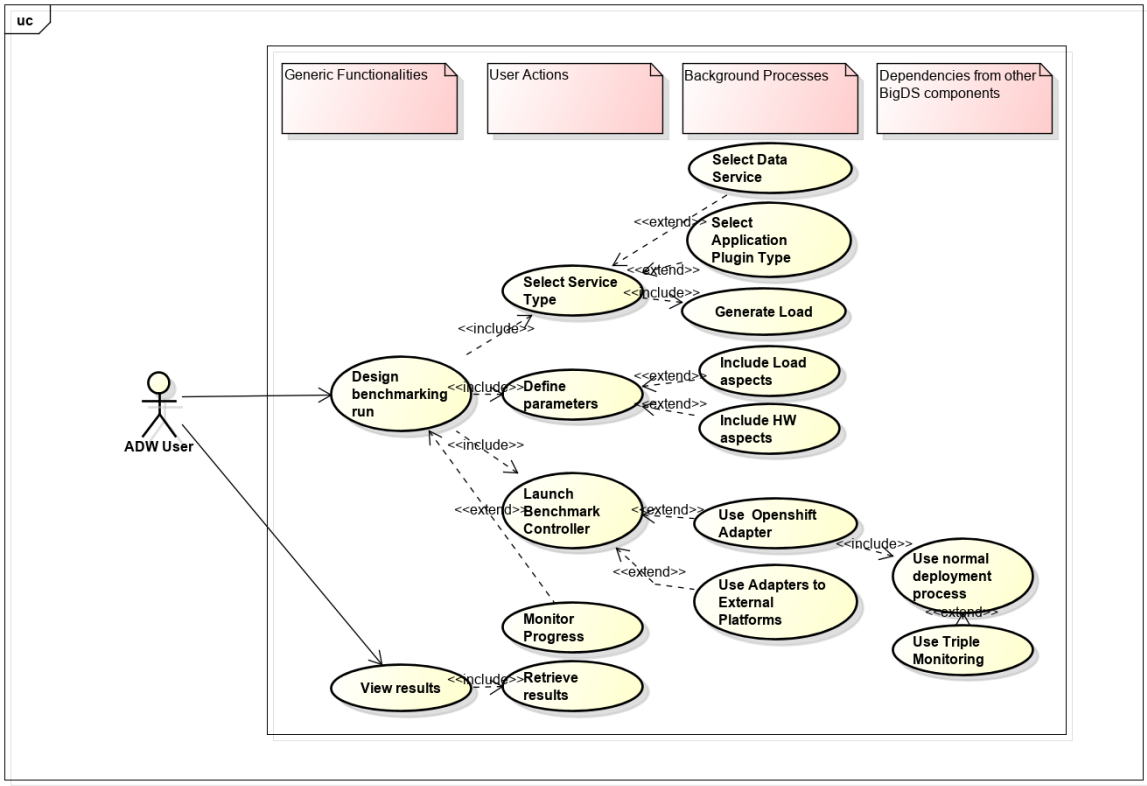


Figure 25 - ADW Design Benchmark Run System Use Case

Following the creation and acquisition of the relevant data set, the service owner may initialize the process of predictive model creation (Figure 26) 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.

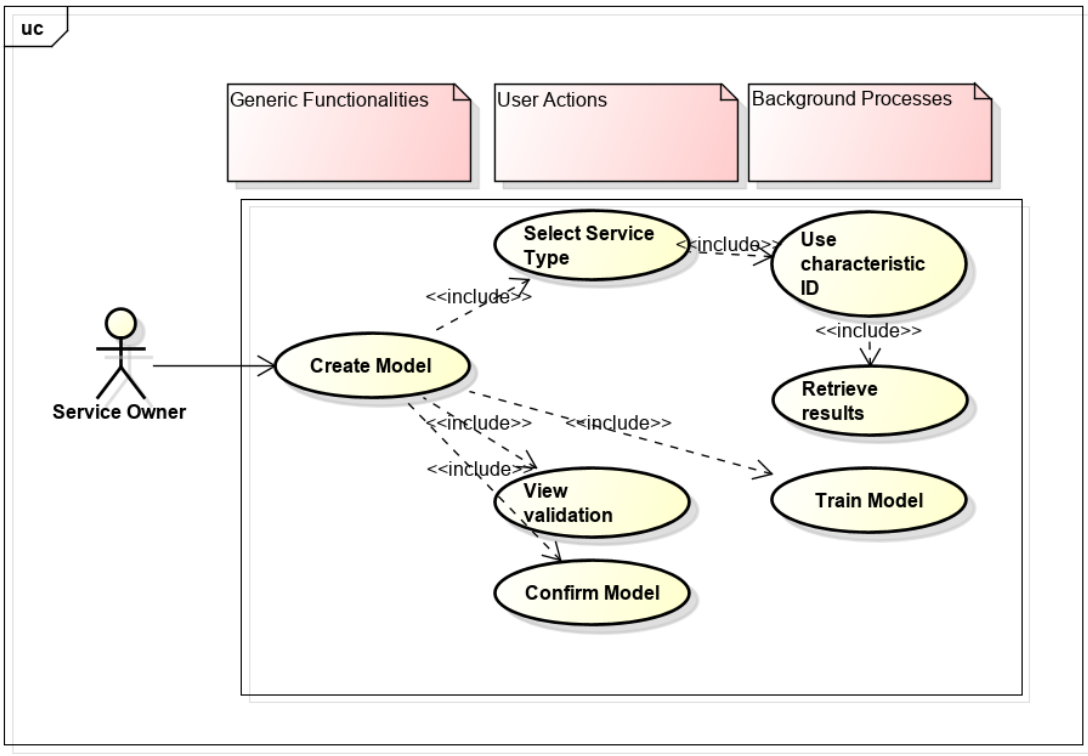


Figure 26 - ADW Create Model System Use Case

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, in the Data Toolkit and/or in the ADS Ranking UI, will insert the needed data services instances and indicate anticipated input workloads and needed QoS levels (Figure 27). The annotated Candidate Deployment Pattern Playbook, enriched by the Pattern Generator with the HW deployment options, will be fed into the ADW Core, that 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:

1) The metrics made available to the Data Scientist need to be in accordance with the ones supported by the benchmarking and monitoring process.
2) 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.

The outputs of ADW Core are then the annotated playbooks, which are referred to as Dimensioned Deployment Playbooks.

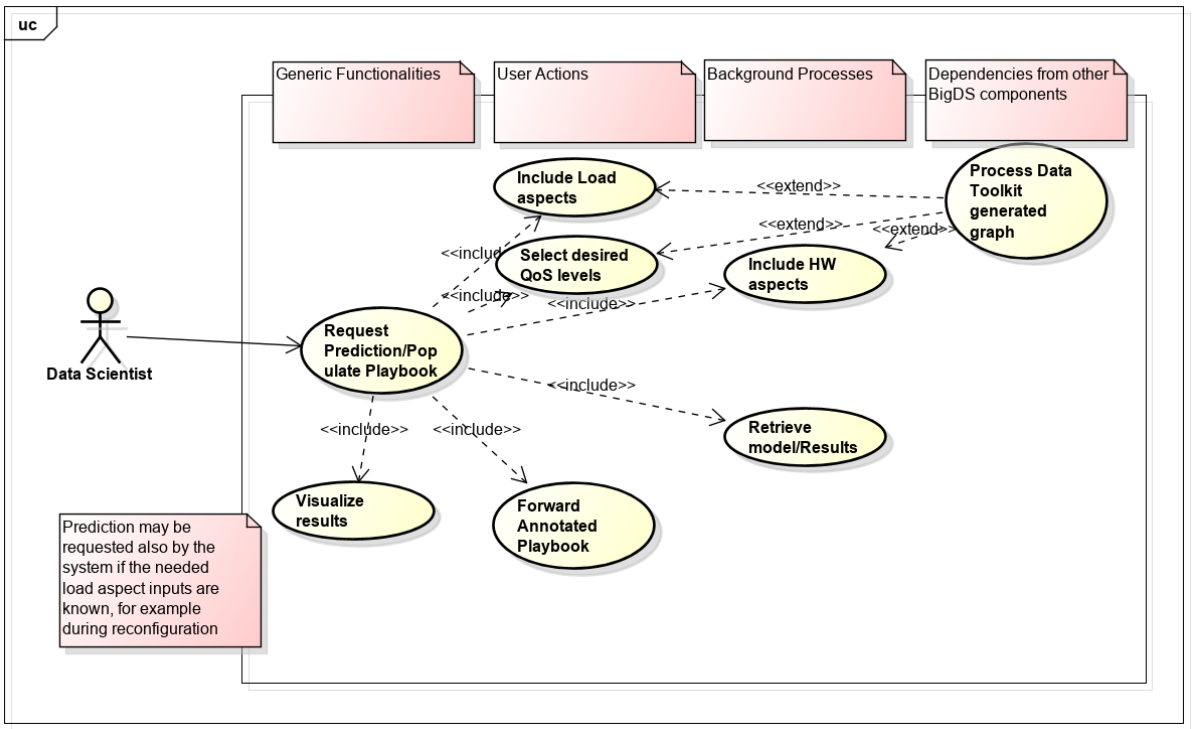


Figure 27 - ADW Request Prediction System Use Case

8.2.2 System Design

The design of the ADS-Dimensioning appears in the following figure, as updated during the second year of the project. We summarize the design of the main sub-components shown in this diagram in the remainder of this section.

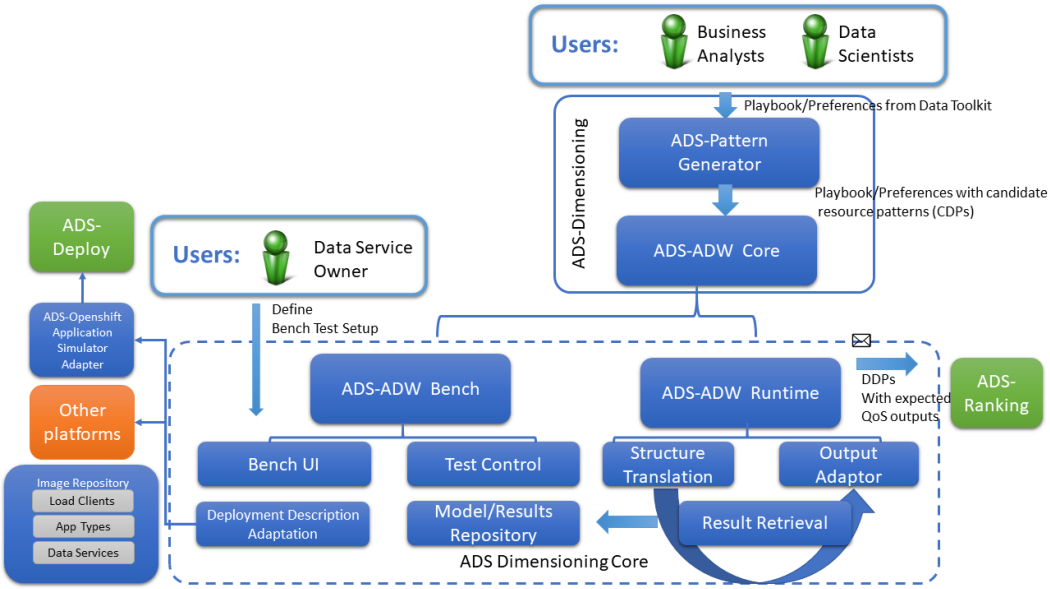


Figure 28 - Overall ADW Design Diagram

8.2.3 Pattern Generator

Pattern Generation is designed as an independent Apache Spark streaming service. The Data Toolkit component of BigDataStack passes Pattern Generation a BigDataStack 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 29 below. Within Figure 29, Spark transformers are shown in orange while non-spark components are shown in blue. As we can see from Figure 29, 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.

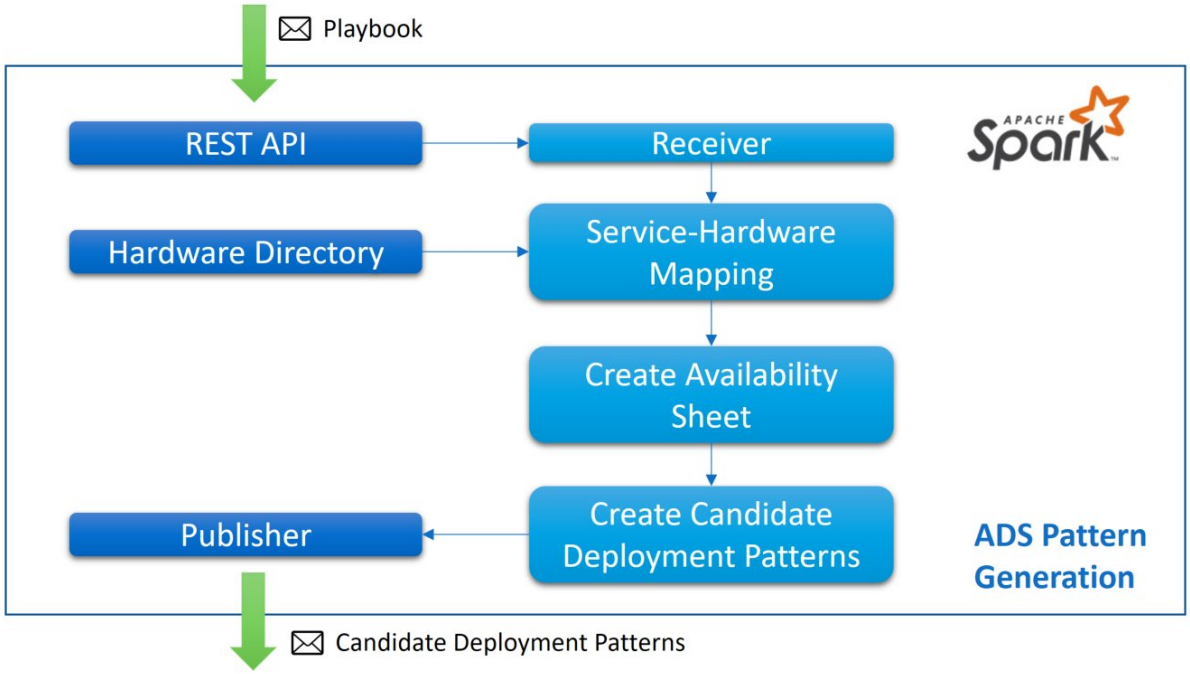


Figure 29 - ADS-Pattern Generation Architecture

8.2.4 Benchmarking and Model Creation of ADW Core

Initially, the ADW Core needs to create performance models for the elementary components in a service graph 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 data set 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 tailored 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 component (e.g. data service), along with capable configuration of the docker image to initialize parameters per execution.
- Implementation of interfaces towards the execution platforms (e.g. ADS Deploy, Openshift, Docker Swarm) in order to submit the request to deploy the respective service and load clients.

Ability to check the state and progress of a running test is also needed. The architecture needed for this phase appears in Figure 30.

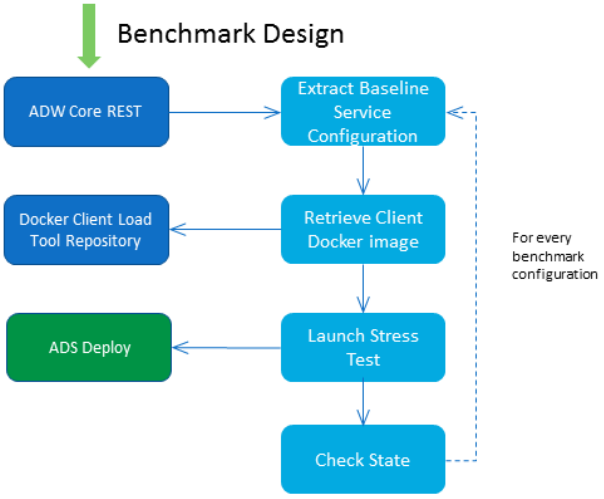


Figure 30 - Benchmark Design Architecture

Following the creation of a representative dataset, model creation needs to be triggered based on the same REST interface layer of ADW Core. Acquisition of relevant data is based on the component naming used. Once the models for each component (e.g. data service) are created, they are ready to be used during the online phase for populating the various 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 benchmarked element.

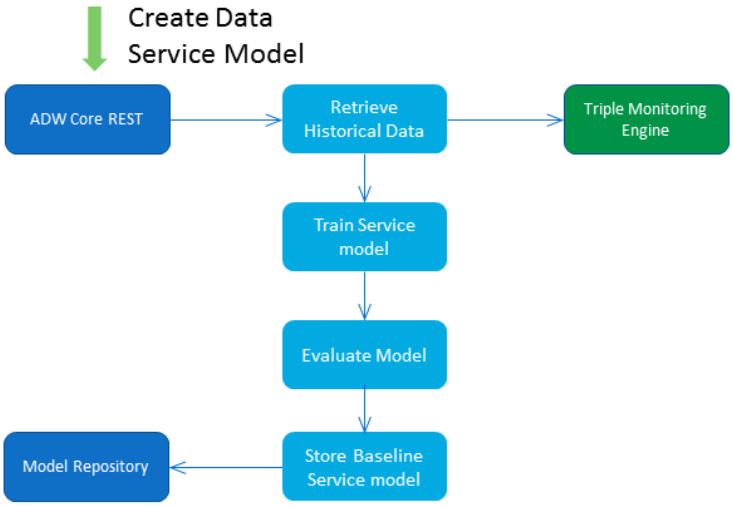


Figure 31 - Model Creation Architecture

8.2.5 Openshift Application Simulator Adapter and Application Type Plugins

The goal of the Openshift Application Simulator Adapter is to provide a turn-key solution for sequentially deploying a series of standard applications and collect performance statistics from them (which we refer to as running experiments), on top of Openshift. This is aimed at dealing with the cold-start issue that is discussed in Section 8.1.3. To enable the adapter to be generic, it is designed to be agnostic to the application types that it runs. In effect, the tests can be considered to act as 'plugins' to the application simulator adapter, where each experiment is a self-contained piece of software that performs the experiment. This allows the application simulator to be lightweight, needing to hold only the list of experiments to run and the logic for operationalizing deployment on OpenShift (represented as pre-configured BigDataStack playbooks). Thus it has a dual nature, initially of adapting to the Openshift and BigDataStack deployment process and finally to deploy specific application level standard component categories (a kind of application level benchmark) that resemble the ones found in BigDataStack. We discuss the different components of the application simulator below.

Openshift Adapter Server: The adapter server is the primary component of the application simulator. This is a containerized Java application that launches and monitors the application level experiments. Upon launching of the adapter server, an OpenShift config map is first mounted as a volume. This config map contains the configuration for each of the experiments that are to be deployed. The adapter server will also mount a separate writable volume for holding the outcome of each experiment. Once the volumes are mounted, the control server will sequentially run each experiment. This involves the submission of the BigDataStack playbook to the ADS-Deploy component, which operationalises the deployment of the needed containers on the cluster infrastructure for the current experiment. Once all experiments are complete, the adapter server exits and the volumes released.

Prometheus[12]: OpenShift itself maintains a Prometheus monitoring and time-series database. The application simulator adaptor uses this database to store the performance information for each experiment that is run. Metrics stored here can be considered to be of two types:

- Resource Usage: These are standard resource usage metrics that OpenShift monitors by default for each running container on the cluster. These are: CPU Shares, Memory and Disk Usage.
- Quality of Service: These are application-particular metrics that describe how successful the deployment was. These may include factors such as Response Latency, Completion Time and Throughput. It is the responsibility of each experiment to define the quality of service metrics for that application.

Experiment Application Type Plugins (one or more): The application simulator needs to deploy a series of experiments targeted at similar components with the applications in BigDataStack. To that end it needs to have a packaged generic and standardized version of such an application type. Each experiment is described by a BigDataStack Playbook, listing the containers and metadata about the application type plugin. Experiments are typically launched by OpenShift 'Job' objects, as they are finite containers. We describe the first of the implemented experiments in Section 8.3.1.

[12] https://prometheus.io/

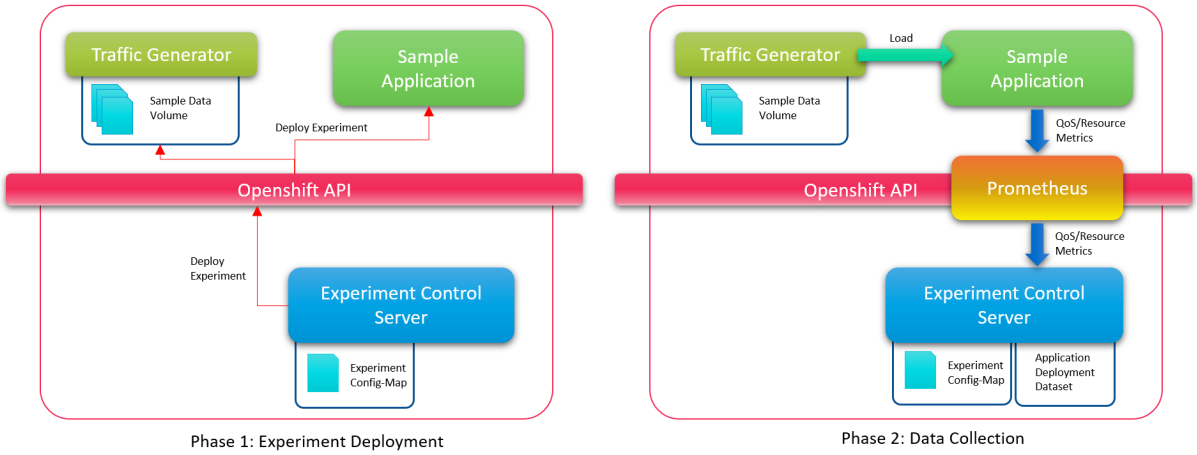


Figure 32 - Openshift Application Simulator Adapter Diagram

8.2.6 Load Clients Plugins

When needing to create the necessary load for the various benchmarks and experiments, relevant external tools such as Jmeter and YCSB are used. Even though some of them (e.g. Jmeter) are also designed to work on a distributed manner in order to reach the necessary stress levels, there is a significant amount of manual intervention for deployment, configuration and load injection execution. To automate this process ADW Bench includes a number of coordination actions (more details are provided in Section 8.3.2.5). A prerequisite for such a process includes the creation or extension of dockerized versions of such tools in order to be able to accept in a parametric manner the various needed configuration details. An example of such a case can be found in our current version of the implementation of Jmeter[13]. Additions are required in baseline dockerfile scripts in order to include relevant startup scripts[14] and other dependencies. Also startup scripts need to cover for changes in container behaviour, specific requirements for collecting and forwarding the results as well as passing and utilizing configuration parameters in the baseline tools such as Jmeter.

8.2.7 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 baseline model, apply the inputs and get the result, propagating it as input to the next element of the graph. On

[13] http://bigdatastack-tasks.ds.unipi.gr/gkousiou/adw/blob/master/adwdocker/jmeter_workloads/Dockerfile
[14] http://bigdatastack-tasks.ds.unipi.gr/gkousiou/adw/blob/master/adwdocker/jmeter_workloads/docker-entrypoint-new-cli.sh

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 tradeoff.

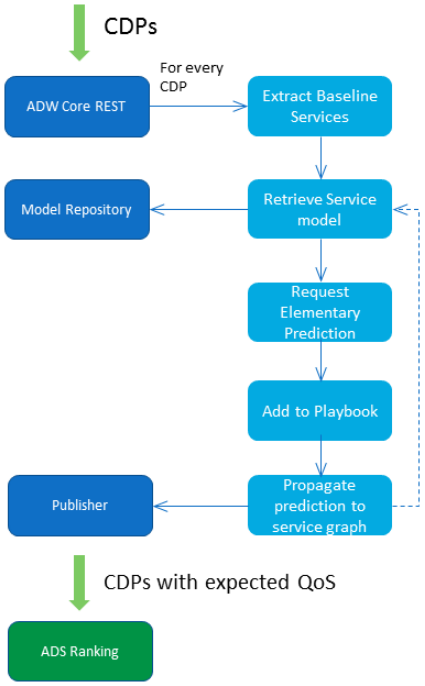


Figure 33 - Annotate Playbook Architecture

8.3 Implementation and Integration Highlights

In the following sections, the main implementation and integration highlights of Y2 are presented with relation to the various parts of ADW.

8.3.1 Application Type Experiment Plugin: Real-time Stream Processing

Experiment plugins are applications that can be deployed by the Openshift Application Simulator Adapter component, allowing a particular type of application to be tested. The first implemented experiment plugin is the real-time stream processing plugin. This plugin is designed to simulate a stream processing application, i.e. an app that takes in a continuous stream of data items, processes those items sequentially through a series of transformers, and then publishes the outcome. Within the BigDataStack project, the Danaos Shipping use-case, is an example of this type of application. There, sensor data from a series of ship-board IoT sensors produce logging data about ship engine status and efficiency. These sensor feeds are processed in real-time, first by transforming and aggregating them, and second by using the merged information to predict whether a pattern as emerged that would indicate component failure in the near future. The real-time stream processing experiment plugin aims to simulate applications of this form.

The real-time stream processing plugin is implemented as an Apache Flink application. Apache Flink is a framework designed to enable the development of streaming applications for JVM-based languages, such as Java and Scala. Flink deployment consists of a JobManager that manages the work, and one or more TaskManagers that execute tasks. The real-time

stream processing plugin first deploys a JobManager and one or more TaskManagers onto the cluster infrastructure as containers. Once these containers have reached a running state and have initialized, Flink applications can be submitted to the JobManager. To simulate real-time streaming applications, like the Danaos shipping use-case, we deploy a configurable Flink application with a defined set of properties. The Flink application is comprised of a sequence of data transformers, where the number of transformers, which take an input data record, simulates some computation on that record and then emits a new record to the next transformer can be defined. The properties of these transformers can be configured for each experiment. For each transformer, the following properties can be customised:

- The amount of CPU time needed to process each record
- The memory usage of the transformer
- The size of the output record
- Processing delay added for retrieving data from an external data store

Additionally, as streaming applications need a data source, a separate container is also launched, which provides data load onto the Flink application. This load-generator can also be configured in terms of:

- How many records to send to the application at one time (batch size)
- The duration between sending records (delay between batches)

By altering these configurables, we can simulate a range of different real-time stream processing applications with different properties, and hence generate a range of data-points on how well they perform on whatever cluster infrastructure BigDataStack is deployed upon. Indeed, the Openshift Application Simulator Adapter can be configured to launch a range of experiments using the real-time stream processing plugin, with the aim of collecting data-points about real-time streaming applications of varying types without needing real user applications.

8.3.2 ADW Core

ADW Bench aims at supporting different testing cases for a variety of business models and to enable the acquisition of sufficiently large datasets that can be afterwards be used online during the CDP population or during performance model training. As an example, Figure 34 includes the following indicative cases, from which the variety of implemented features especially for the ADW Bench part is determined given that they are considered as requirements for implementing these scenarios.

a) Generic Load Injection

In this case, ADW Bench may be used as the main environment for a Stress Testing as a Service offering, testing at an existing and external application endpoint. Through the use of a dedicated load generation cluster (or through the use of available public cloud resources if the former is not available), relevant loads can be scaled and injected towards externally deployed applications. In this mode ADW Bench does not actually control the applications but only generates the necessary load needed for reaching anticipated scales through its dockerized and coordinated client execution.

b) Application Baseline Performance Model

In this case, an interested entity needs to create a baseline performance model for a given application that captures dependencies of the application from different workloads, parameters of execution, deployment etc. For this case, a number of different parameters may be defined in a parameter sweep fashion and guarantees on the isolated execution of the application should exist to avoid any interference effects that could tamper with the results. Given that this requirement cannot be set in public Clouds (except for cases of dedicated hosts in public Cloud offerings), this scenario is primarily targeted at private cloud cases.

c) Application Multitenant Performance Model (Offering of the application as SaaS)

In this case, an interested entity needs to create a multitenancy performance model for a given application that captures performance interference between this instance of the application and other concurrently running applications or instances. Performance interference in multitenant environments has been proven to cause significant QoS degradation for the same amount of resources used when compared to a dedicated deployment mode [21], hence this aspect is specifically important when one needs to determine the pricing terms with which they will offer different QoS flavors of the SaaS application, taking under consideration that they will need to cater for this underprovisioning due to the multitenancy aspects.

d) Public Cloud Benchmarking

In this case an interested entity needs to measure/benchmark the performance of public (or even private) Cloud platforms, in which case they need to utilize the dockerized version of an application or benchmark in a repetitive manner and potentially through the usage of various combinations. This is the mode that mostly resembles the mode available by other state of the art tools, however it was decided to be included for completeness of ADW Bench and the ability to act also as a generic benchmark framework.

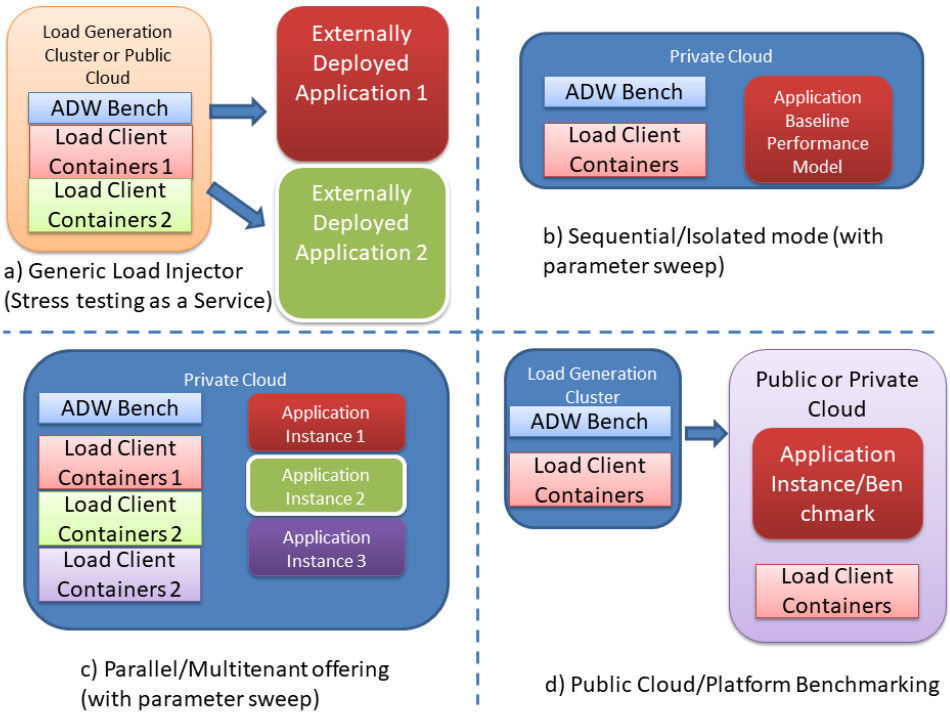


Figure 34 - Potential testing/load injection models

However, the process of acquiring sufficiently large datasets in order to come to useful conclusions is always a tiresome, error-prone and background needing task, that requires experience around the measurement process, the used benchmark/load injector (and the specific needs for action sequence coordination) as well as execution in a distributed manner so that the produced artificial input reaches the scale of a realistic stress test without worrying about client side bottlenecks. Even with this knowledge, one would have to iterate a considerable number of manual steps (creation and configuration of distributed agents, monitoring of test status etc.), thus limiting in effect the scale of the experiment in terms of gathered test cases or "dataset lines".

The aim of ADW Bench is to act as a benchmark/stress test management and execution framework that targets at:

- abstracting this tedious process, thus reducing the knowledge barrier needed for the execution of the test, through intuitive web UI driven setup and monitoring, hiding the complexity of test setup, coordination needs, test execution, result gathering and cleanup.
- Incorporating dockerized versions of benchmark executables of commonly used baseline benchmarks (such as Apache Jmeter and YCSB) that can easily scale on a target execution platform such as Docker Swarm or Openshift, thus reaching the necessary stress test load generation, while respecting their requirements in terms of test setup and launch.
- Enabling on-demand spawning of these stress test clusters and the incorporation or not of the benchmarked service as part of the benchmark setup where applicable.
- Enabling automation aspects in the form of parameter sweep definition experiments to be incorporated either through the UI or through a REST based API that offers the same functionalities and can be used to further automate the process. Parameters may refer to the injected load, the type and size of resources used by the bundled application.
- Enabling the simple definition of a trace driven scenario that may follow a specific variation of the load based on historical data and automation of the results acquisition process for the defined sequence. Through this feature, various scenarios such as scalability testing, endurance testing, stress testing and spike testing [25] can be easily applied.
- Enabling the execution of the various combinations in either parallel or sequential mode, in order to support the different business/technical cases requirements that were described previously.
- Offering increased test reliability through the monitoring and report on the actually executed tests, anticipated samples and acquired ones as well as mutual blocking of combinations that may need either parallel or sequential mode concurrently.
- Applying a modular architecture and implementation that can lead to the extension towards new baseline tests and target platforms incorporation.

Therefore, ADW Bench aims not to present yet another low or mid-level benchmark tool, but to handle and coordinate such available ones (like Jmeter or YCSB) in order to automate their launching against a target application or software stack and act either as a benchmarking tool or as a load injection tool. Furthermore, it aims to decouple the test management from the

underlying baseline benchmark used, so that it is easier to reuse the higher layer management framework with other baseline benchmarks or generic load injectors.

8.3.2.1 ADW Bench Setup

Setup of a test series is performed either through the UI or through a relevant REST API interface. Necessary details include the name of the test (which implies if it is a simple load service or a bundled data service), the type of the node used (with included naming conventions to indicate the platform type, used afterwards for selecting the correct launch adapter) and the type of the workload, which is the list of the uploaded available files in gitlab. Furthermore, other parameters can be inserted in a parameter sweep fashion, for example minimum, maximum and step nodes for the data service and client setups, operations per second of the clients as well as switches to indicate if the execution is a tracedriven one, a parallel or a sequential one. Finally details on the test setup name, endpoint of the target platform and endpoint of the results database are included. The respective Node-RED implementation on which the Setup UI is based is presented in Figure 36. The user can also exploit historical data and retrieve previous test setups for the same type of service. Given that in many cases the information in most fields may be repeated, this is a feature that is expected to help users speed up the process by avoiding to repopulate identical fields.

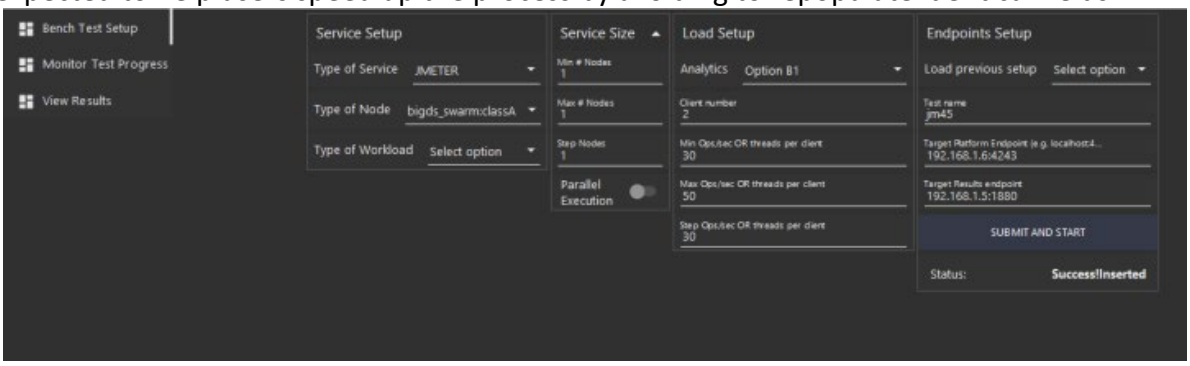


Figure 35 - ADW Bench Setup UI

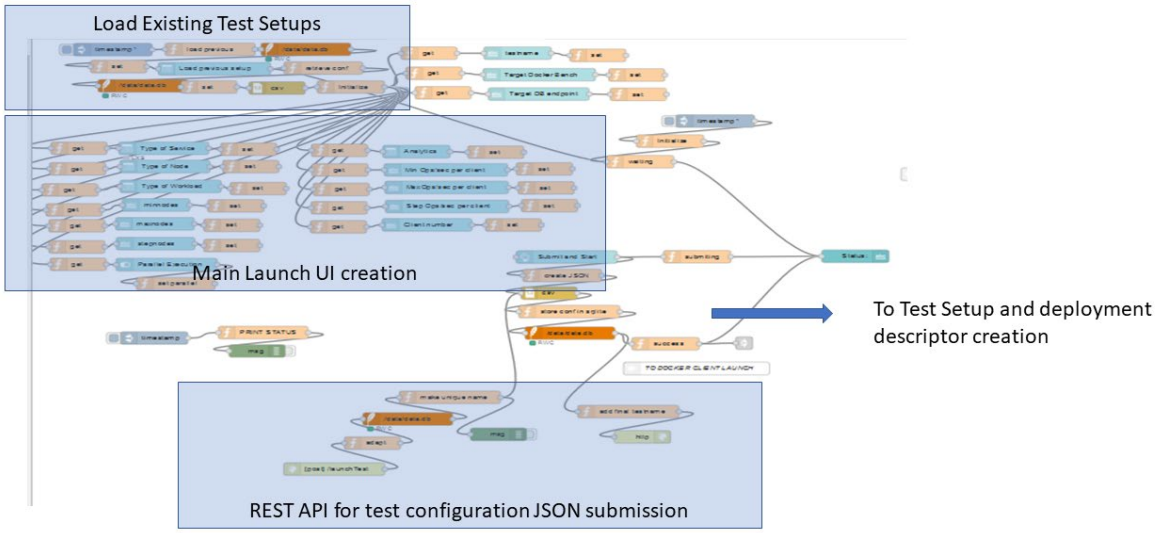


Figure 36 - Node-RED implementation flow of the Setup Test UI

The aforementioned process can also be applied programmatically for further automation through the submission of the relevant JSON configuration file through a REST POST endpoint. The JSON fields and structure appear in Figure 37.


```
payload: object
    testName: "lxs_launch1111111"
    parallel: "false"
    service: "LXS"
    nodeType: "Medium"
    workloadType: "Workload A"
    minNodes: 1
    maxNodes: 1
    stepNodes: 1
    analytics: "Option B1"
    minOps: 100
    maxOps: 200
    stepOps: 50
    clients: 2
    dockerEndpoint: "'192.168.56.103:4243'"
    dbEndpoint: "'192.168.56.104'"
    startTime: 1555002118710
```


Figure 37 - JSON specification of the REST API test submission

For the trace driven experiment case, the user is anticipated to have uploaded a relevant file that includes the various trace steps as a sequence of the main workload aspect e.g. number of users. Then the execution is performed in a stepwise manner and for each line of that datafile. Result ingestion is expected to be performed on a tool specific case, through utilizing a generic REST POST method. Thus the load injectors of each tool should be able to perform such a call in order to push the acquired results in the backend results database. For each client type used, there is a relevant table with the necessary fields, however this is transparent to the end user since redirection to the respective table is performed by the tool and based on the test setup configuration. The overall API calls related to the setup stage appear in Table 54.

Context	Method	Path	Input	Output
ADW Bench Setup	POST	/launchTest	JSON configuration file for parameters (tool selection, workload features)	Return message for test id
ADW Bench Data Input	Post	/pushResults	JSON object with results from the load clients of each type	

Table 54 - ADW Core API calls for Test Setup and Results Ingestion

8.3.2.2 Test Lifecycle Management

In this section it is necessary to clarify the various terms and concepts used in the remainder of the document, for clarification purposes:

- A test setup is a complete experiment series that is defined by the application owner/performance engineer in a single configuration/parameter sweep fashion. In that sense, a single test setup may contain variations or ranges of the parameters used, that will break eventually into multiple individual test executions or iterations in a stepwise manner. Each test setup is assigned a unique ID.
- A test iteration is a single execution instance of the test, with a concretely specified and unique combination of the input parameters, stemming from the ranges defined in the test setup. Single in this case does not refer to the samples gathered (multiple samples are gathered during the test execution), but to the existence of a single set of test parameters (such as number of clients, target throughput etc.). Each test iteration is also assigned a unique ID.
- Reporting of the test results is always performed at the lowest level of execution granularity, thus the test iteration is stored in the tool's database. This result includes the benchmark input (type of load etc.) as well as the test iteration ID and the test setup ID.
- In the case of the parallel execution of all iterations of a given test setup, one should retrieve and accumulate results as well as load by grouping relevant result rows at the test setup ID field, so that all parallel running instances are considered. This way the multitenancy effect scenario can consider all concurrently running test instances at the final stage of the dataset extraction (to train a multitenancy prediction model for example).
- In the case of the sequential execution, only the specific iteration is running, so the relevant test results should be grouped by the test iteration ID field in order to extract the overall dataset (e.g. to train a prediction model that associates necessary resources and expected QoS output based on given load inputs). Whether there will be one or multiple result rows per test iteration ID depends on the benchmark used in each case and whether multiple client nodes are used to generate the traffic load. There are benchmarks like Jmeter that handle result acquisition from the distributed slave nodes used, thus resulting in one row per test iteration ID. Others (like YCSB) do not include this step, hence multiple result rows would appear in the database (one for each reporting client node).

The rationale behind the needed semaphore-like behaviour has already been identified for purposes of experiment isolation and similar conditions guarantee (sequential tests) or for the need to investigate concurrent execution overheads in multitenant environments (parallel tests). The implemented semaphore structure requires two elements:

- A global Boolean flag (for sequential purposes), that indicates if a sequential test is already running.
- A global counter variable that indicates how many tests are being executed at the moment.

The blocking logic is as follows:

- Parallel tasks check the sequential flag before launching:
  - If false they launch, increasing the counter by 1
  - If true they sleep and check again after an interval
  - On finish they reduce the counter by 1
- Sequential tasks check the counter:
  - If 0 they launch, set the Boolean flag to true and increase the counter
  - Otherwise they sleep and check again after an interval
  - On finish they reset the Boolean flag to false and reduce the counter by 1
- When a sequential task finishes, all the sleeping (sequential or parallel) tasks race for the resource, meaning the permission to launch. In this case:
  - Either one of the sequential iterations gets it and starts executing while blocking all remaining ones (sequential or parallel)
  - Or one of the parallel gets it and all the pending parallel ones can be executed
  - Selection of the winner is more or less random, depending on which task exits the sleeping period first after the lock has been lifted. A discussion on the tradeoff of this approach follows.

The behaviour of the system appears in the following figure for an indicative arrival and execution scenario.

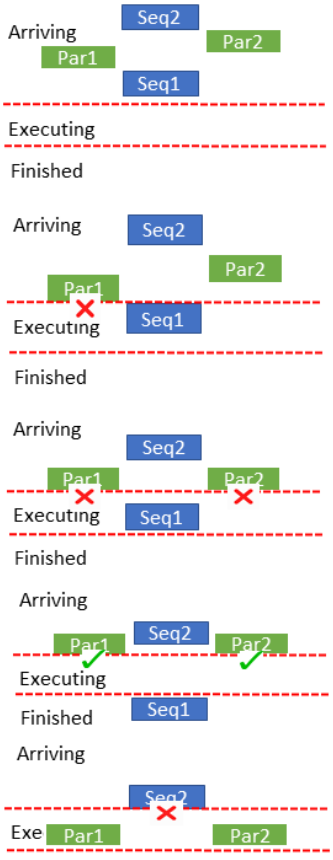


Figure 38 - Semaphore-like behaviour for test combinations blocking

Guarantee on atomicity of operations

For any semaphore like behavior to be successful, it is well known that the atomicity of the test-and-set-lock operation (reading of a shared variable and changing its value without any intermediate execution interrupt) needs to be guaranteed. Due to the fact that the nodejs framework (in which our implementation is based) is single-threaded and a nodejs function does not get interrupted by the framework until it completes or waits for a callback, the atomicity of the global variables check and/or manipulation is guaranteed, if this is performed inside the same function.

Tradeoff with relation to next task selection

As noted above, whenever the conditions for a new launch are met and we have a number of parallel or sequential executions waiting in line, there is no way with the current implementation (unless a relevant queue is created) to dictate which execution will follow. If a parallel one happens to wake up, poll and get the token of execution then all waiting parallel ones may start. If a sequential one is successful, then all parallel ones again have to wait. This approach was followed for two reasons. Initially there is no specific requirement that a test should finish before another test or within a given time constraint. Also it would be unfair for the parallel tests (in one or more test setups) to wait for the finalization of potentially many sequential iterations in another test setup just because that setup arrived a few moments earlier (and given that the isolation requirement is only needed by this setup). Thus the followed approach achieves a fairer trade-off, given that multiple parallel tests are more probable to acquire the lock (and thus enable all parallel ones to be launched without further wait) and is expected to reduce the overall waiting time in the system.

Tradeoff with relation to extension of isolation between different parallel setups

Another case of design decision relates to whether the system should enable isolation between different parallel setups. Thus if a specific series test instances is already running in parallel, whether another test setup (and all its children instances) could be launched before the first parallel setup is complete or not. The decision in this case was to enable different parallel setups to be executed, primarily due to the fact that one of the main business cases of the tool is to enable a stress test creation framework against external targets. Thus blocking concurrent parallel setups against different targets would significantly reduce the utilization of the client creation cluster and significantly deteriorate the prospects of a Stress Testing as a Service model, since even if one parallel setup was running even at a small part of the cluster resources, all of the latter would be blocked.

In case one needs to launch parallel setups that are somehow isolated from one another with the application bundled and running within the client cluster, then other forms of isolation may be applied to achieve that goal (e.g. dictating to the container orchestrator that these containers should not be collocated with other ones).

ADW Data Model

With relation to the data model used for storing state and test results, this appears in Figure 39. It consists of 6 tables with the following purposes:

- Mappings table holds the names of the container images to be used for each service

or type of load.

- Nodetypes table holds the resource types of each platform.
- Tests table holds the main test setup information along with all the configuration details.
- *_results tables include one specific table for each type of measurement that holds the case specific metrics and details.
- Workloads table holds details of workloads. Although given that this information is retrieved from a relevant gitlab folder implies that the usage of this table is deprecated to informative purposes only.

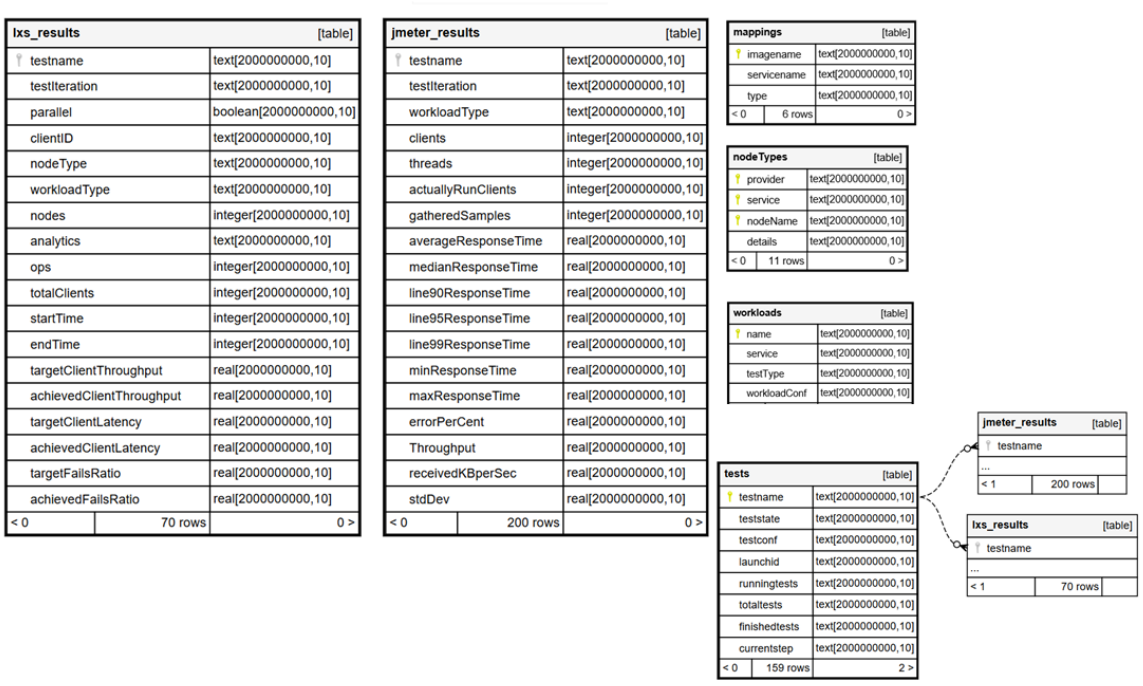


Figure 39 - ADW Bench Data Model

The database layer of ADW is based on sqlite, given the fact that only high level summary results are maintained inside the tool and due to the fact that this type of database is very portable and directly integrated into Node-RED. This enhances the portability aspects of the tool and alleviates from the need to have an externally deployed DB.

8.3.2.3 Test Monitoring

After launching a test, one can monitor the progress of the various combinations included in the setup. To do so, they can navigate to the respective UI tab and select the specific test setup name. A text field can be used in order to filter from the available tests based on a given partial naming pattern. Following, the status of the test(s) is presented, including information on total, started and finished combinations (Figure 40). The main operation is also offered as a REST API call (/testState). The relevant Node-RED flow appears in Figure 41 while the available API calls are included in Table 55.

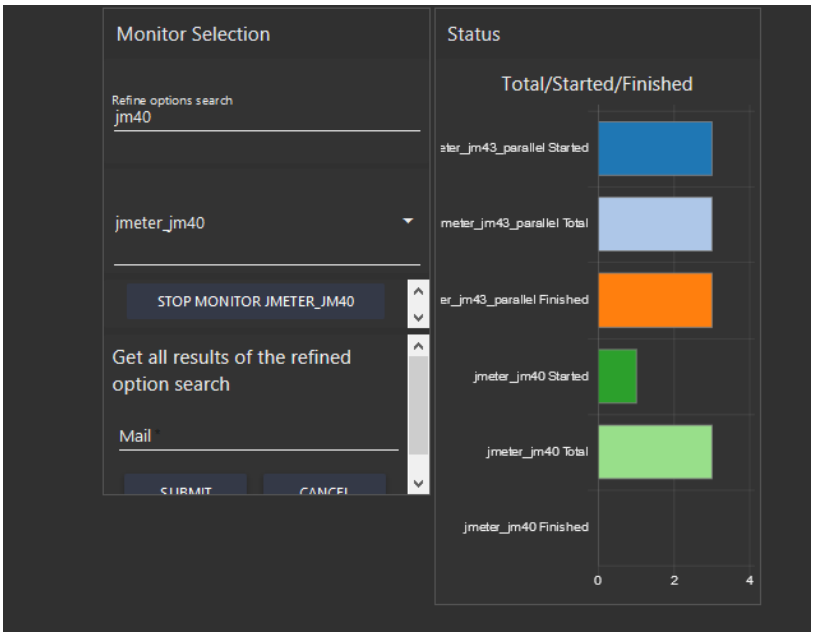


Figure 40 - Test Monitoring

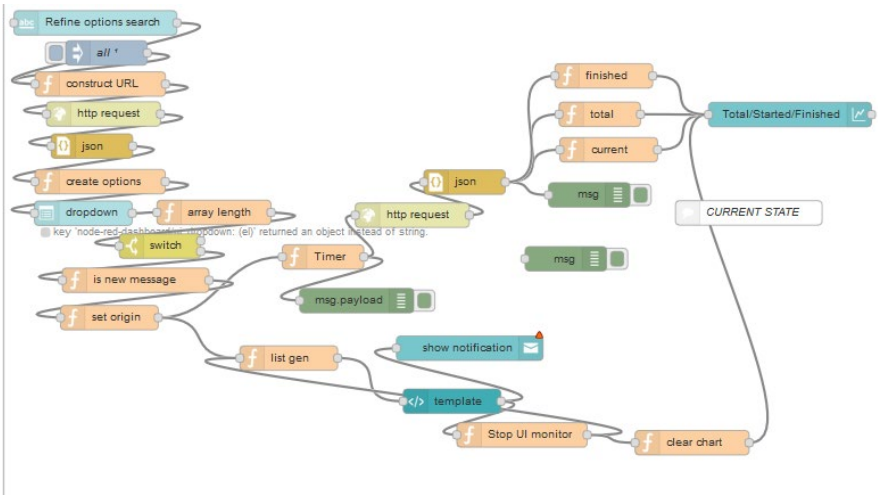


Figure 41 - Node-RED flow for Test Monitoring

Context	Method	Path	Input	Output
ADW Bench Test Monitoring	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 for that service type
ADW Bench Test Monitoring	GET	/testStatePartial/:testname	Partial name of the testname for filtering related tests	Test state of all tests whose testname

				matches the input argument
ADW Bench Test Monitoring	GET	/testState/testname	testname	Return all info. If all is used as the testname, all test info is returned
ADW Bench Test Monitoring Trace	GET	/traceTests/	-	Return all tracedriven tests
ADW Bench Test Monitoring Trace	GET	/serviceConf/:servicename	Name of the test setup	Return of configuration object for the test that includes all setup details

Table 55 - ADW Core API for Test Monitoring

8.3.2.4 Data Input and Filtering Layer

The Data Filtering Layer aims at initially creating an API in front of the results database as well as implementing a set of post processing queries that aim to facilitate and enhance the result filtering and querying process. Furthermore, it provides the necessary calls for the UI layer (Figure 42) for presentation to the user, in parallel with the available REST calls. Filtering options include the type of the service, the needed metric as well as needed QoS values, in the sense of a target value and a percentage tolerance around it. For example, if a user enters that she is interested in an average response time of 1000 msec for a given service, with a +-50% percentage, all relevant results that have response times between 500 and 1500 msec will be retrieved. The top ranked one will be presented in detail in the main UI panel, in which key metrics of the experiment will appear (such as throughput, latency etc.), as a percentage of the goal value, taking also under consideration of the metric is of ascending or descending order.

Figure 42 - Result filtering UI

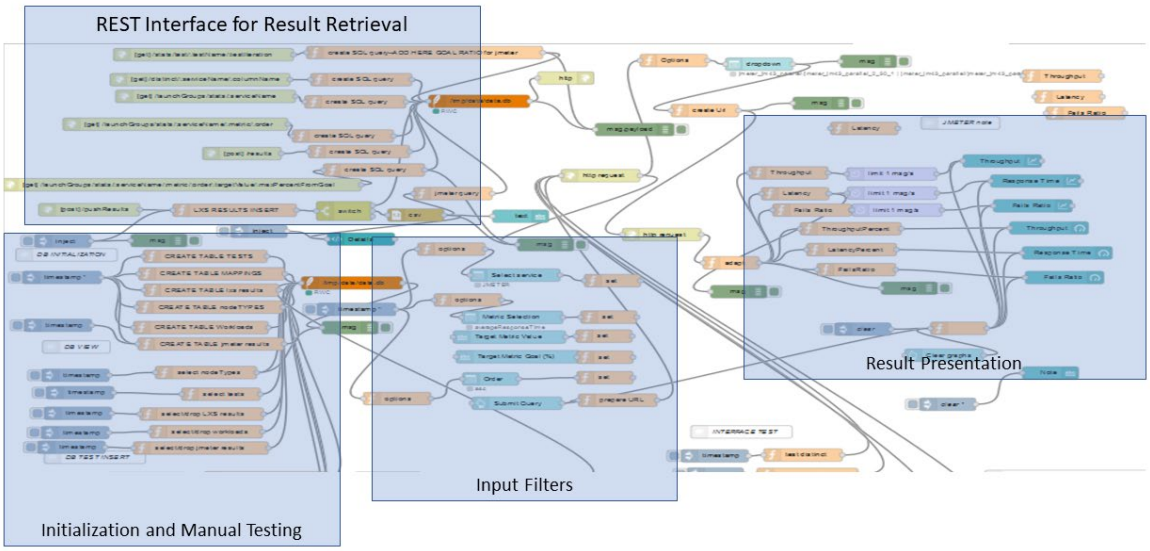


Figure 43 - Node-RED implementation flow for Data Input and Filtering

Specifically for the tracedriven experimentation case, and given that in this case one needs to retrieve results in a complete trace setup and directly linked to the main workload parameter specified, a relevant specific tab has been implemented (Figure 44). In this case, the user can partially filter the available trace driven setups with a textual input, while the relevant trace file is plotted (top graph) in conjunction with the QoS reported results (bottom graph) for direct comparison purposes on the effect of a sudden spike in the load for example. Furthermore, they can select another tracedriven setup (potentially with different resources used) that is also plotted in the same bottom graph with the previously selected setup, in order to enable quick comparisons between the two setups. The user can also select the metric of the results to be portrayed.

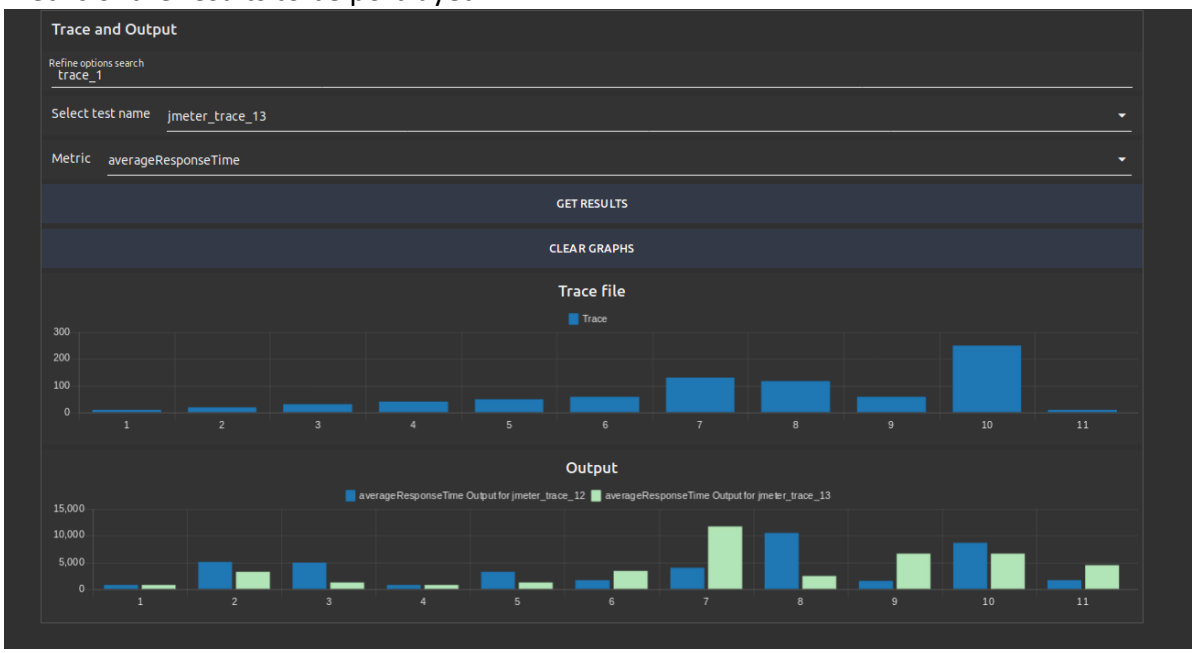


Figure 44 - Trace driven report and status tab

Context	Method	Path	Input	Output
ADW Bench Data Filtering	GET	/stats/test/:testName/:testIteration	Specific test iteration from a specific test setup	Various statistics based on the reported metrics of each tool (including goal ratios etc.)
ADW Bench Data Filtering Trace	GET	/traceSteps/:testName/:metric	Tracedriven testname and relevant metric	Ordered list by trace steps with the result of the metric
ADW Bench Data Filtering Trace	GET	/traceStatePartial/:testname	Partial naming to be used for trace driven tests filtering	Array of objects for trace driven tests that match the naming convention
ADW Bench Data Filtering	GET	/distinct/:serviceName/:columnName	Name of a supported service and metric	Returned list of supported metrics for results
ADW Bench Data Filtering	GET	/launchGroups/stats/:serviceName	Name of a service type	Accumulated results for individual clients of a service type, grouped by test setup id and test iteration id
ADW Bench Data Filtering	GET	/launchGroups/stats/:serviceName/:metric/:order	Name of a service type, metric on which to sort and order (asc/desc)	Accumulated results for individual clients of a service type, grouped by test setup id and test iteration id, sorted by a relevant metric
ADW Bench Data Filtering	GET	/launchGroups/stats/:serviceName/:metric/:order/:targetValue/:maxPercentFromGoal	Name of a service type, metric on which to sort and order (asc/desc)	Accumulated results for individual clients of a service type, grouped by test setup id and test iteration id, sorted by a relevant metric and based on a target value and a percent deviation
ADW Bench Data Filtering	GET	createDataset/service_name	Service name (from available enumeration [lxs,ibmos,cep] of available services (aims to return dataset lines for that service)	JSON array with [conf, metrics] objects

Table 56 - ADW Core API for Data Filtering

8.3.2.5 Execution and Platform Coordination and Layer

One of the key features of ADW Bench is the ability to follow a benchmarking process's necessary steps. Variations between available baseline tools include the existence or not (and with what setup) of distributed versions of the tests, reporting performed overall for all clients or not etc. As an example, the case of Apache Jmeter is presented for the case of Docker Swarm as an execution platform. When using the specific benchmark (following the acquisition of a case specific workload file), one needs to ensure that a correct sequence of actions is enforced in order to maintain the test's validity or to setup a coordinated environment. Thus the following set of actions needs to be performed in the specific strict sequence:

1) The user uploads on a gitlab account the Jmeter workload file and inputs in the UI all the relevant test setup information. Upon launching, and provided that the semaphore logic does not block the test setup's execution, the automated process may start.
2) Initially, the virtual helper resources needed for the execution of the test are created, i.e. a shared virtual network so that the distributed client nodes can discover and communicate with each other as well as a shared storage volume in which data for the experiment may be stored or parameters shared (in this case properties files and raw data for the measurements which need to be maintained after the test end for archival purposes).
3) Afterwards, the main client nodes (server slaves) that are used to generate the actual traffic towards the target endpoint are configured and launched. Configuration includes aspects such as mounting the shared folder, joining the virtual network etc. The Jmeter server slaves need to be started before the coordinating Jmeter Master node, given that the latter is configured with the returned IP addresses of the slaves, among other information (such as test id, test iteration, target number of clients etc.). Hence the sequence is: start server slave containers->obtain their IP-> launch master Jmeter container with the slave IPs as arguments. At this point it needs to be stressed that it is not enough that the server slave containers are started, the initialization phase needs also to have completed (i.e. the servers are up and running and accepting requests). In this case there is a dilemma on whether to wait until all slaves are up or accept the fact that in some cases slaves may fail occasionally so the master should go ahead with the ones available. Given that actual client numbers are reported in the end, it was decided to proceed with the second option, since result reporting validity is guaranteed and in order to be more resilient for cases of occasional failures without the need to rollback a long running experiment.
4) The Jmeter master node now sends the load directives to the slaves and waits until all of them have finished. In this case, suitable post processing logic needs to be triggered in the master node in order to concentrate and create aggregate reports (enriched with the test setup id, target clients, workload name etc.) that are to be inserted in the tool's main result database, along with the specific test measurements. These are the primary results that are used in the filtering processes of the tool, while the detailed raw data are kept in the shared volume in

case they are needed in the future. This approach is applied in order to maintain a lightweight version of the results database in the main tool that will enable faster retrievals and queries, as well as to create data rows that include all the necessary information (test setup details and respective measurements acquired). Upon finish, all the created resources (containers and virtual networks except for the shared volume) are deleted in order to clean up the stress test cluster.

Calls to the Docker Swarm environment are performed through the relevant REST API of the latter that enables all the various functionalities (storage and network creation, container startup and deletion etc.). The respective flow that implements this functionality appears in Figure 46.

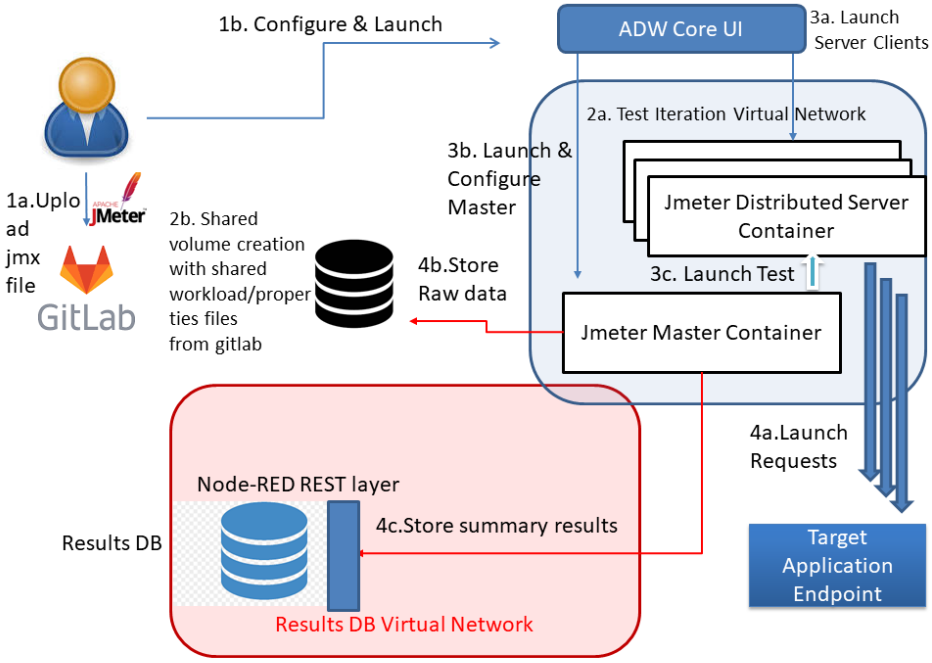


Figure 45 - Example of Jmeter execution coordination in Docker Swarm

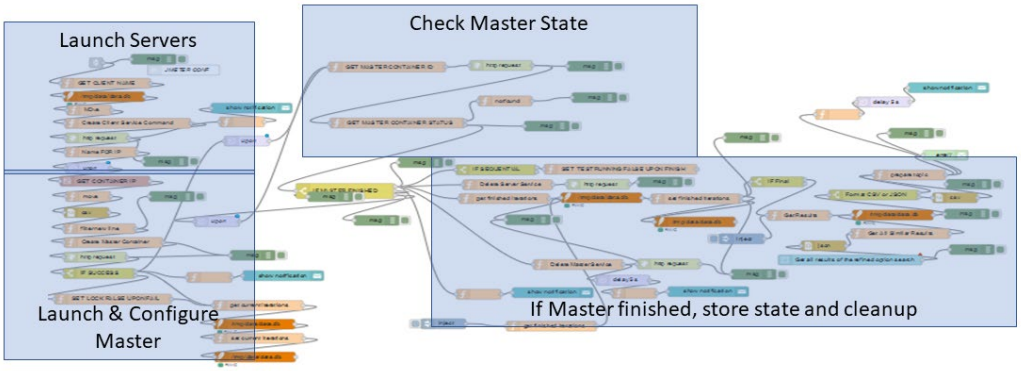


Figure 46 - Jmeter Adapter for Docker Swarm Node-RED implementation

Similar adapters are currently being developed for AWS Elastic container service APIs for the Jmeter case as well as the YCSB tool case. An example of interconnection with the Openshift adapter appears in Figure 47.

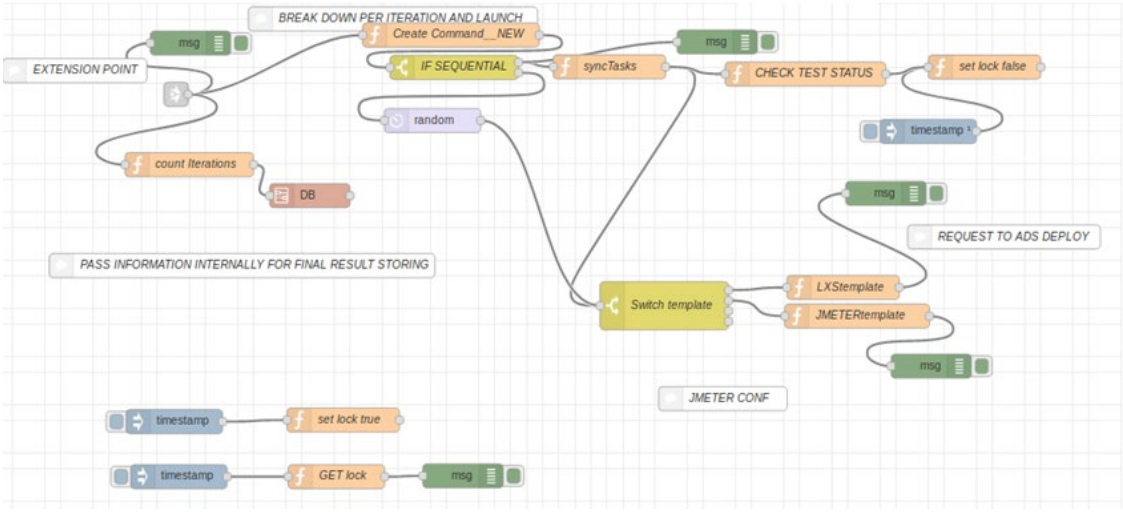


Figure 47 - Link to Openshift Adapter Node-RED flow

8.3.2.6 ADW Runtime Playbook Population

The main runtime usage of the ADW Bench results is performed during the call by the Pattern Generator component in order to enrich the described patterns with anticipated levels of QoS based on the available performance data. To do so, ADW Runtime includes a relevant POST method, in which the PG component submits the service manifest along with the variations of the patterns in terms of resources used (Figure 48). The backend Node-RED flow (Figure 49) then brakes down the description per type of pattern suggestion and queries the results database in order to find relevant results (by also utilizing the API of Table 56). These results are then used to populate the respective QoS fields.

```
"services":{
   "normalizerService":{
      "image":"alpine:latest",
      "labels":{ },
      "deploy":{ },
      "userPreferences":{
         "resourcePreferences":{
            "numberOfReplicas":1,
            "minimumMemory":-1,
            "minimumComputeCapacity":-1,
            "minimumThreads":2,
            "maximumCost":-1
         },
         "qosPreferences":{ },
         "qosRequirements":{ },
         "inputWorkloadFeatures":{ }
      },
      "recommendedResources":{ },
      "dimensioning":{
         "metrics":[
            {
               "Name":"Throughput",
               "Type":"average",
               "typeLimit":null,
               "Value":400,
               "higherIsBetter":true,
               "Unit":"ops/sec"
            },
            {
               "Name":"Throughput",
               "Type":"max",
               "typeLimit":null,
               "Value":500,
               "higherIsBetter":true,
               "Unit":"ops/sec"
            },
            {
               "Name":"latency",
               "Type":"average",
               "typeLimit":null,
               "Value":5,
               "higherIsBetter":false,
```


Figure 48 - Indicative Playbook JSON Structure and Population

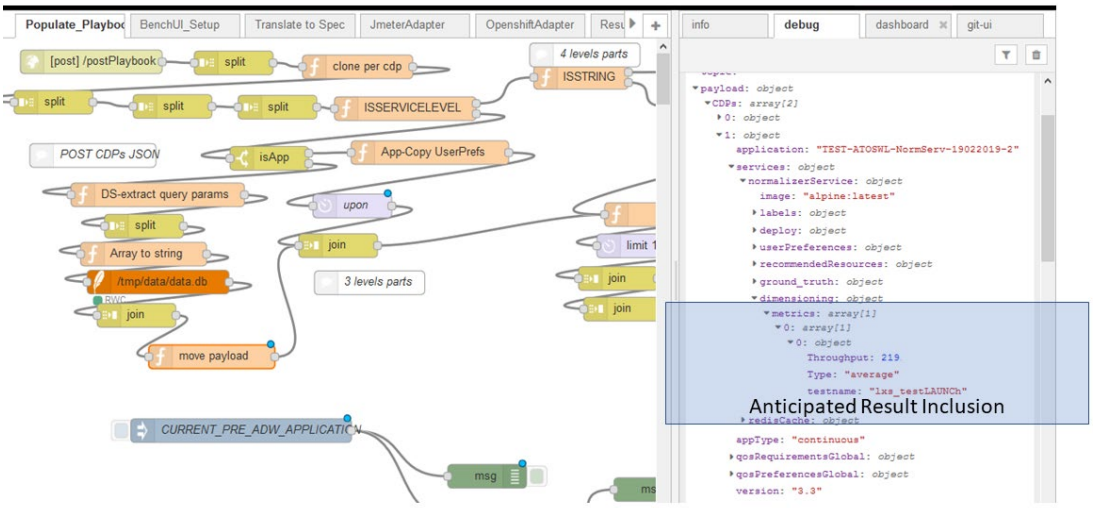


Figure 49 - Post Playbook Node-RED flow and Result

Context	Method	Path	Input	Output
ADW Runtime	POST	/postPlaybook	Playbook YAML String	Annotated Playbook YAML string with QoS tags. This primarily implements the request prediction operation

Table 57 - ADW Core API

8.4 Experimentation Outcomes

Besides the functional evaluation presented in the previous paragraphs, a set of experiments have also been performed in order to check out specific aspects of ADW.

8.4.1 Evaluation results

8.4.1.1 Scalability of framework for test setup submission

The testing scenario refers to examination of the ADW Bench API based test submission. In order to stress that aspect, a number of sequential test setups are submitted to the system via the API call. Sequential tests were selected since these linger in the system for a long time due to the blocking logic and therefore we can check the system's responsiveness under a large number of pending combinations. The responsiveness is measured by obtaining the response time needed for submitting a test setup through the relevant REST call. A stressed system would produce larger delays in all relevant calls including the /launchTest method used to submit a test setup. During such a call, the system receives the input, checks the test setup name selected and if not unique it adds the current timestamp to make it unique. It then stores the test setup configuration in the tool DB, retrieves the associated image names for the test, counts and stores the iterations and launches the individual messages that are responsible for triggering each test combination. These messages are then blocked due to the sequential nature of the test and when the flag is raised, one of them will proceed with execution. Given that the client runs consecutively and tests are blocked, overall test combinations in the system increase (based on either the defined request frequency and/or the size of the setup).

Various testing cases have been examined such as:

- Submitting a test setup with 10 combinations every 10 seconds (Figure 50). This is the most anticipated area in which the tool would be used under normal circumstances and it gives a view of the baseline response times of the service. From the graph, it can be seen that the response times are in the range of 20-50 msec, with some occasional spikes around 200 msec.
- Submitting a test setup with 100 combinations every 1 second (Figure 51). This indicates an average response time of 466 msec while being very stable for cases up to approximately 100,000 pending combinations (with averages in the range of 60msec). After that there is a gradual increase in the response times up to the point of around 170,000 combinations, after which the system reaches its limits.

- Submitting a test setup with 100 combinations every 10 seconds (Figure 51). This indicates an average response time of 700 msec while again being very stable for cases up to approximately 100,000 pending combinations. Compared to the previous case, it presents a slightly higher level of test combination endurance (around 190,000 combinations). It might indicate a higher average response time from the previous case which might seem strange given the fact that it has a smaller frequency, but this is due to the fact that it is still responsive in higher numbers of submitted jobs in which the response times are significantly higher.
- Submitting a test setup with 100 combinations every 0.1 seconds (Figure 51). This indicates an average response time of 2200 msec while again being very stable for cases up to approximately 100,000 pending combinations (average of 60 msec up to that point). However, it starts to deteriorate at a lower level that the previous cases, in the area of 150,000 pending combinations.
- Submitting a test setup with 10000 combinations every 10 seconds (Figure 51). This setup was included in order to check the effect on large submissions due to the internal breakdowns to combinations in the system. It indicates an average response time of 3200 msec while again being very stable for cases up to approximately 100,000 pending combinations (average of 68 msec up to that point). Also in this case, the breaking point seems to be around 180,000 pending combinations, however in this case the system was able to reach slightly over 200,000.
- Submitting a test setup with 10 combinations every 0.01 seconds (Figure 52). This setup was included in order to check the effect on higher request frequencies and smaller job sizes (thus more fragmented tests than in the previous case). With relation to the previous cases, it indicates a very high average response time of 20000 msec from the early stages of load, due to the increased overhead posed by the fragmentation and request handling process. The breaking point in this case is very early, around 10000 combinations.

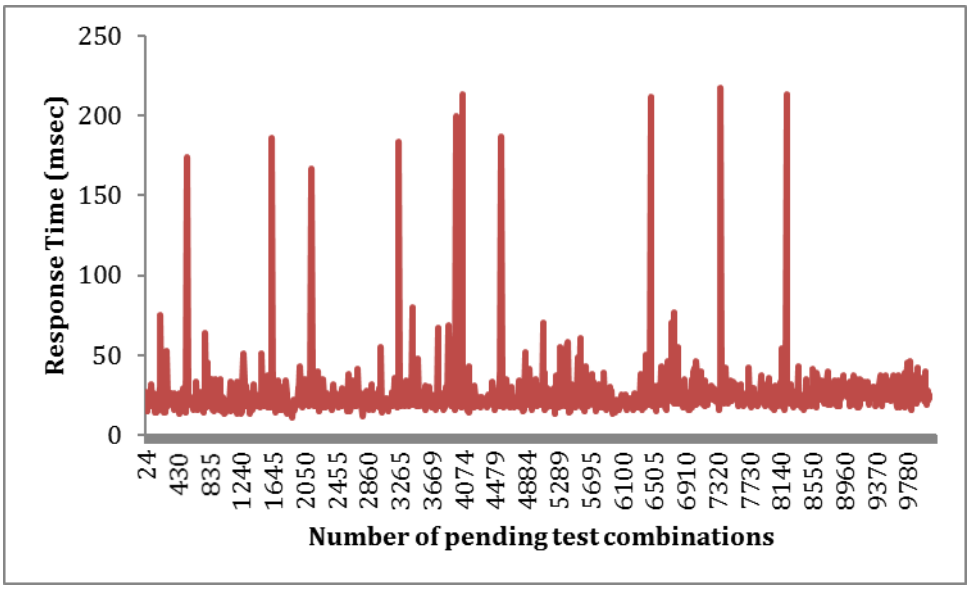


Figure 50 - ADW Bench Baseline times under anticipated normal conditions of execution (1 submitted test setup every 10 seconds with 10 test combinations to be launched)

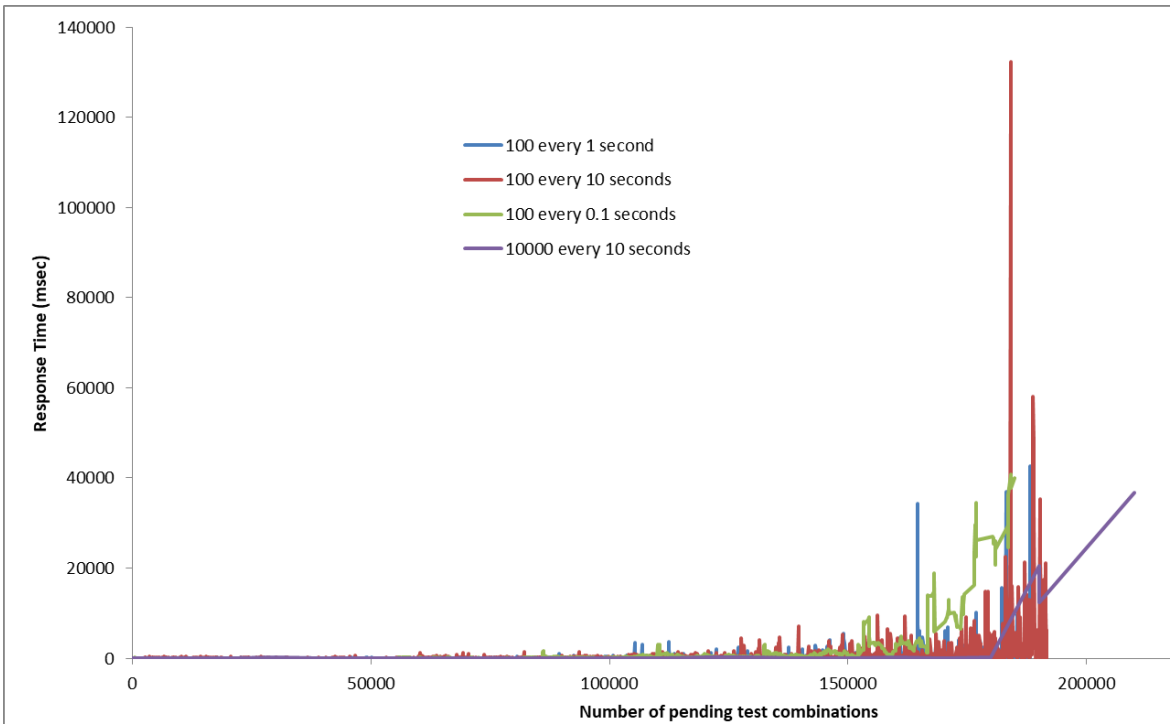


Figure 51 - Various stress testing cases for ADW Bench based on request frequency and test setup size

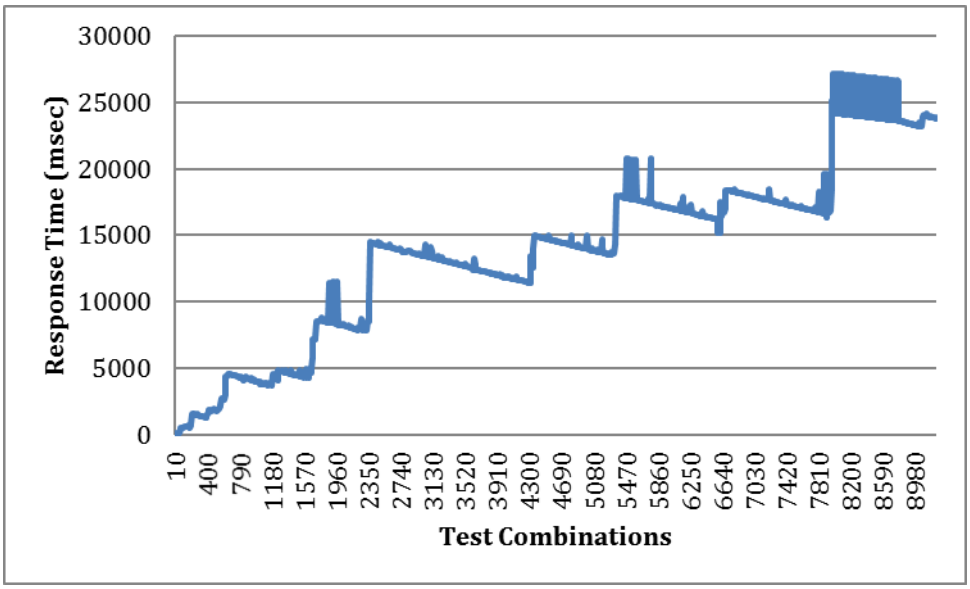


Figure 52 - ADW Bench Response Times under small job granularity and high frequency

Test case	Average Response Time at approximately half point	Average Response Time (overall) (msec)	Standard Deviation (overall) (msec)	Breaking point (pending combinations in the system)
10 every 10 seconds	-	25.25	18.96	-

100 every 1 seconds	47.98	466.02	2723.97	~170,000
100 every 10 seconds	101.60	701,70	4012,14	~190,000
100 every 0.1 seconds	59.59	2275,34	6141,43	~150,000
10000 every 10 seconds	68.10	3248.22	8979.61	~210,000
10 every 0.01 seconds	9314,42 (on 5,000 pending combinations)	13905,31	6621.13	~10,000

Table 58 - Statistics for the various test cases

8.4.1.2 Real-time Stream Processing Plugin Analysis

Earlier in Section 8.3.1 we introduced the real-time stream processing plugin. This is the first of three plugins that will be developed for the application simulator – which aims to solve the issue of having precollected data on various application level component types without benchmarking during the actual deployment process. The development of this plugin is complete, and in this section we report the initial analysis of this plugin when deploying it on cluster hardware, using a mono-transformer application. In particular, we performed experiments in order to evaluate two main research questions:

1. How accurately does the real-time stream processing plugin simulate the specified properties/configuration? This is useful as it gives us an idea of the expected error bounds when using this plugin.
2. How much overhead is added by containerized deployment on OpenShift? This is valuable as we need to know the degree of resource overhead when estimating the amount of resources to request during application deployment.

We summarize the outcome of our experiments below:

Simulation Accuracy, Memory Usage: We first examine the accuracy of memory allocation within the application simulation. Ideally, when we specify that we want a transformer that uses 128Mb of memory, we would expect that to be the amount of memory that is allocated. However, due to the underlying implementation, there may be some variance in actual allocation. To test this, we deploy an instance of the plugin on a local machine, while monitoring the memory foot-print of the transformer JVM object. We run eight tests, each requesting a transformer with a different amount of memory, while keeping CPU usage at zero so that each test lasts only as long as it takes to allocate and randomise the memory for the transformer.

Figure 53 shows the total memory usage of the experiments in contrast to the amount of memory requested by the transformer (array size) for different request sizes. As we can see from Figure 53, the requested and used memory are not always equal, particularly when requesting small amounts of memory. This is indicative of notable overheads in memory usage by Flink transformers, which appear to add around 40Mb in memory usage to any deployment.

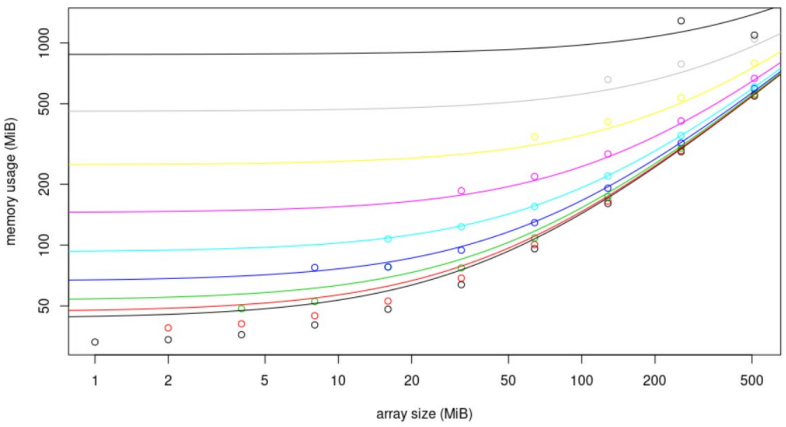


Figure 53 - Total memory usage comparison with transformer memory usage

Simulation Accuracy, CPU Usage: We also ran the equivalent to the above experiments when requesting different CPU loads and monitoring resultant application CPU usage. The results from this experiment showed that on a local deployment, CPU simulation accuracy is 100% accurate.

Overheads, CPU Usage: Having shown that CPU load simulation is accurate on a local deployment, we next examine any overheads when deployed on a containerized Flink cluster on top of OpenShift. To test this, we run a series of experiments, where we deploy a Flink JobManager and TaskManager (forming a small Flink cluster) as containers running on Openshift, and then deploy the real-time stream processing plugin onto that Flink Cluster. We request 100% CPU usage by the transformer and monitor CPU usage by the TaskManager container that is running the transformer every 1 second. A separate container, co-located with the TaskManager, feeds the transformer with a fixed number of records without any added delay. Each experiment is run for 60 seconds, although the records sent for processing will not need all that time before they are processed. This experiment is replicated twenty times to provide information about performance variance. With each experiment, we recreate all OpenShift objects from scratch, to avoid contamination of results stemming from cached data. We might expect that we would see 100% CPU usage for the container (potentially with a ramp-up and ramp-down period where the transformer starts up and later shuts down).

Figure 54 illustrates the CPU usage over time for these experiments. The blue curve represents the mean CPU usage across experiments, while the red curve represents the median CPU usage. As we can see from Figure 54, each experiment does not reach the desired 100% CPU usage on the TaskManager container. Indeed, while usage spikes to around 80% at some time points, mean CPU usage during the active periods is much lower (around 20% CPU usage). This indicates that there are significant overheads being added by the combination of Flink and the containerized deployment. In particular, the lower CPU usage can be explained by data transfer latencies between the container that is sending the records and the transformer receiving those records, in addition to internal buffering by Flink itself (which is not present on a single-container local deployment). As a result, the transformer is left idle

for a significant period of time while waiting for data. Discrepancies like this illustrate why we need real deployment data from components like the ADW to train models such as that used by ADS-Ranking, as there are multiple factors that can influence the actual resources needed by an application that are not part of the application itself.

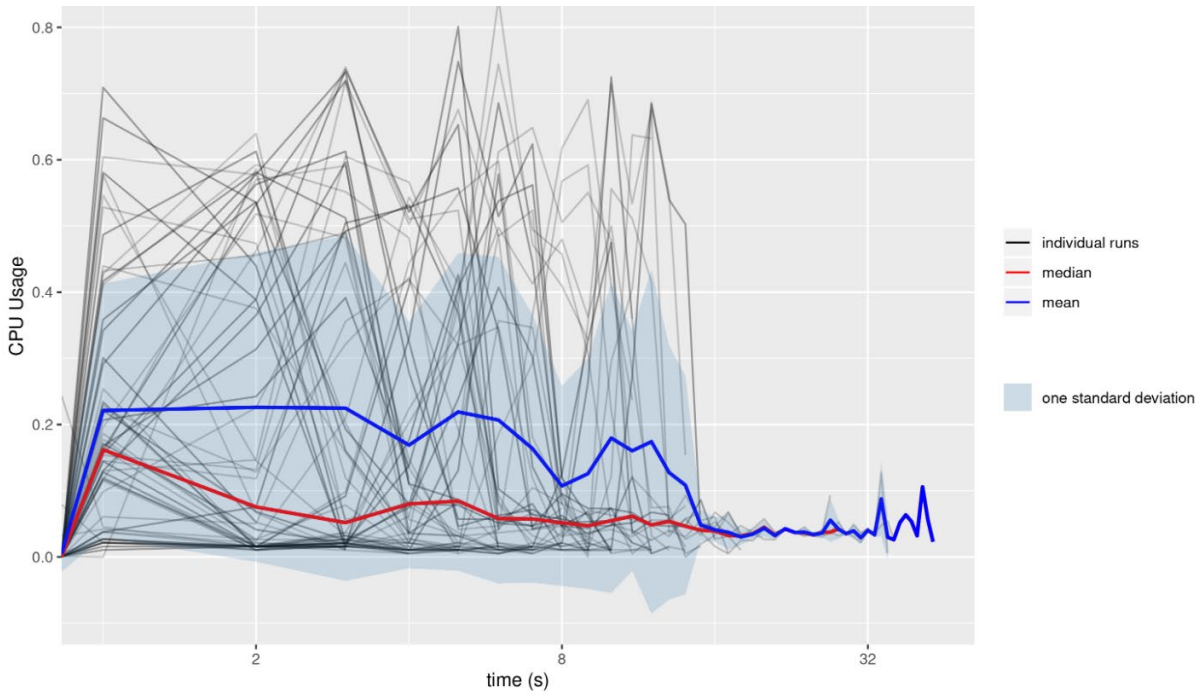
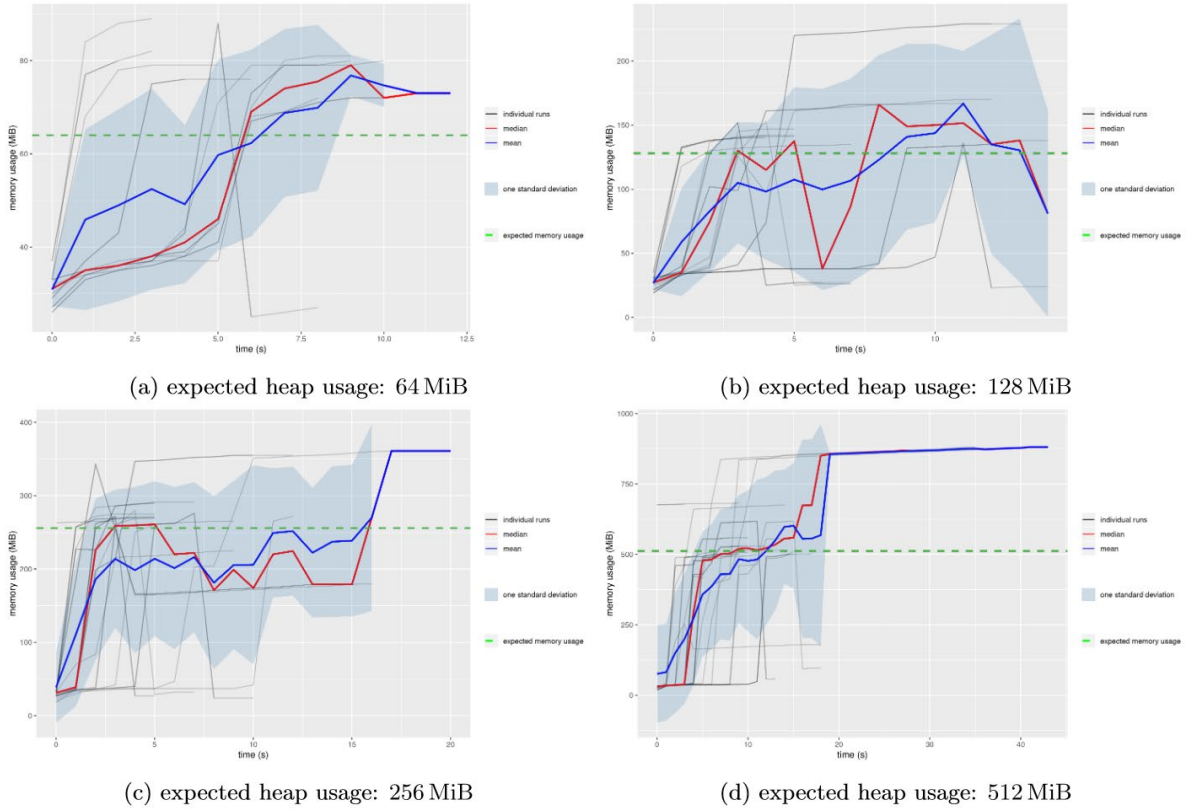


Figure 54 - TaskManager CPU usage when running a 100% Load Transformer

Overheads, Memory Usage: Earlier, we showed that the transformer object itself exhibited about 40Mb higher memory usage than expected, likely due to the Flink object and its input/output buffers. We next examine overall memory usage for the real-time stream processing plugin when running containerized on OpenShift. We follow the same setup as for the Overheads, CPU Usage experiments, with the exception that we vary the target memory usage of the transformer to either 64Mb, 128Mb, 256Mb and 512Mb, respectively. As transformers generate output records, we run multiple experiments with different output sizes. Specifically, we explore every power-of-two number of Mb that is compatible with the target memory of the current transformer. We measure the Java Heap size within the TaskManager container that is running the transformer. We expect that there will be a memory overhead introduced here by the Flink management software that was not present within the local deployment.

Figure 55 (a-d) reports the memory usage of the TaskManager for each of the four memory requests. The horizontal dashed green line represents the requested memory amount, while the blue and red lines show mean and median Java Heap size respectively. From Figure 55 we observe three main trends. First, we see that there is a notable start-up period where the transformer has not yet reached the desired memory consumption. This is expected, as it takes time for the transformer to start-up as well as time to generate and randomise an array of the desired size. Second, again as expected, the full heap usage is higher than the transformer alone (during the period where the transformer is active), representing

additional overhead. Furthermore, the size of this overhead is related to the transformer size. This represents the additional memory that is needed to generate the output records from the transformer and appears to be close to a linear combination of target memory and output record size. Third, if we look closely at the individual experiment curves, we see a drop in memory consumption after processing as commenced in many experiments. This indicates that once the records have been processed the JVM quickly garbage collects the transformer and all container objects (although sometimes the experiment ends before this occurs). In general, we see from these results that there is much variance in memory usage over time for this type of application, and hence we should use longer experiment durations to get a true picture of the required peak memory for this type of application.



(a) expected heap usage: 64 MiB

(b) expected heap usage: 128 MiB

(c) expected heap usage: 256 MiB

(d) expected heap usage: 512 MiB

Figure 55 - TaskManager Memory usage over time as we vary Transformer Memory Usage

In summary, these experiments tested the real-time stream processing plugin within the application simulator. Through local testing, we showed that the implementation is quite accurate at simulating transformer-level CPU and memory consumption. However, when subject to containerized deployment, we observed significantly higher divergence in both CPU and Memory usage in comparison to the target values. This illustrates the value that the application simulator can bring, as quantifying these discrepancies on different types of hardware will provide valuable data when estimating resources to request during application deployment.

8.4.2 Comparison with other approaches

As mentioned previously the main purpose of ADW Bench is not to introduce another baseline benchmark test, since a number of them are already available and tailored for various

application domains (an interesting analysis of these appears in [22]), but to enable a higher layer benchmark and load injection execution, coordination and management framework that will simplify and abstract this process for the performance engineers/application owners. In this section an analysis of relevant work is presented in order to compare against existing solutions and identify advantages and disadvantages of each approach. The comparison is performed against the following axis:

- User interfaces and whether these are based on web, command line interface (CLI) and/or REST API availability.
- Ability to support multiple platforms of execution with multiple benchmarks or the ability to extend these.
- Nature of the tool, in the sense of whether it is primarily designed as a benchmarking tool scope, a load injector scope or both. For example, for the load injector scope one should be able to handle arbitrarily defined workloads as well as distributed load generation.
- Ability of the tool to cover the complete test lifecycle, meaning resource setup, test setup, test execution, result acquisition and resource cleanup and whether this ability exists for multiple variations (parameter sweep fashion), trace driven simulations or isolated/parallel modes of operation). Given that this aspect is primarily met in cloud-targeting and measuring frameworks, since the cloud domain offers extensive capabilities for API based resource manipulation, the majority of the examined tools are from the cloud domain.
- General scope of the tool as well as availability as an open source project.

The overall comparison between ADW Bench and the main comparable tools appears in Table 59 for the aforementioned aspects. The comparison was based on the retrieved available information (published papers, GitHub repositories or tool description pages). In the following paragraphs details of each tool are presented.

In [23] the ARTIST benchmarking tool is presented, which has the aim of measuring and monitoring the performance of Cloud services. The tool is centered around 3 main baseline tools and can automate the launch and execution of the tests (even in a periodic manner), as well as result collection and filtering against various cloud platforms. However, its main rationale is to implement a continuous and stable experiment execution (e.g. with the same workload conditions for the tests and on a single node) for comparability purposes in an attempt to monitor cloud services performance evolution and variation. As such, it does not enable the arbitrary input of any type of workload needed by the end user in these tests, nor does it enable different business cases such as the private cloud isolation case.

In [24] Cloudbench Tool (CBTOOL) is a multi-benchmark framework that aims primarily at infrastructure as a service (IaaS) cloud stress and scalability testing. It enables running controlled experiments with workloads designed by the user/contributor, through experiment plans based on a scripting syntax. In CBTool applications are by default bundled with the load generation, hence its approach is primarily a benchmark one and not a generic load injector towards any given external endpoint, although it could indirectly serve towards that purpose. CBTool has also a feature for parallel vs isolated mode, through waitfor,

waituntil and waiton capabilities that can hault test submission, thus being able to apply the specific mode requirement in a test scenario. One difference on this aspect is that in our case this mode is enforced across the overall tool scope (thus if multiple users are trying to use the stress test cluster/resource they will all be blocked in an isolated requirement), whereas in the CBTool's case it seems that blocking relates to the specific experiment plans submitted by this user. A step that appears increased in needs is the number of steps needed to adapt a given application to the benchmark framework automation. In the case of ADW Bench it would be almost the same specification (e.g. Docker compose file) used for the actual deployment of the target application, due to its primary application centric nature. Another comparison point with ADW Bench is the latter's intuitive flow based programming style (due to Node-RED) and portability (the actual source code is a JSON file of a few KB), compared to the 20k python lines of code of CBTool. Overall, CloudBench is the state of the art tool that presents the majority of features, being also the basis for the SPEC Cloud IaaS 2018 benchmark. In a nutshell, CBTool is a more centralized, production grade, VM oriented and infrastructure/benchmark centered tool, whereas ADW Bench can be considered as a container oriented, more lightweight, portable tool that bundles all the needed functionalities in one package.

Smart Cloudbench [25] is another tool that aims to simplify benchmarking execution against Cloud services through the use of application level benchmarks such as TPC-W. The tool covers aspects such as (web) UI based configuration and setup, test lifecycle management for VMs, result aggregation and reporting on application and resource level metrics. Its primary use is for benchmarking purposes and comparisons (similarly to the ARTIST tool already mentioned) through the supported benchmarks (with ability to apply user defined aspects of workload). There is no mentioning for availability of a REST based API (or other forms of automated insertion of parameter sweep tests) through which further automation could be achieved. On the other hand, it supports direct selection through querying incorporation of cost, performance and KPIs specification.

Google Perfkit [26] is an approach that aims to facilitate the execution of various benchmarks against Cloud services in order to capture cloud related metrics (similar to CBTool) but also scores related to a wide set of primarily microbenchmarks. Thus it mainly acts as a benchmarking comparison between service offerings and not at the level of generic load injection that can be used to analyse or stress test an arbitrary application. It has advanced templates for multiple benchmark results gathering and processing, parsing for the incorporated benchmarks as well as enabling the test lifecycle management (resource creation, test installation and execution, result gathering and resource cleanup). Its primary UI is through a command line interface (CLI). Extensions (PerfkitExplorer) have been built for providing web based interfaces, although only as a Google App Engine project.

Octoperf [27] is a commercial offering that features either a privately deployed or SaaS based solution and it is based on Apache Jmeter (one of the supported benchmarks in ADW Bench). It comes with a number of features such as load testing execution design, extensive web UI, inclusion of Service Level Agreement rules etc. One of the drawbacks of Octoperf is its closed source and purely commercial nature, thus it is not freely available to all audiences. Furthermore, it does not seem to have means of abstracting from Jmeter in order to include different baseline benchmarks/load injectors nor does it apply scenarios such as a trace driven simulation or an isolated vs concurrent execution. In essence it is completely integrated with

the baseline tool. On the other hand, it features the ability to use multiple platforms (at least AWS and Digital Ocean) as well as a RESTful API through which automations may be performed, in addition to features such as result reporting and presentation.

µSuite [28] and DeathStarBench [29] are tools to target microservices and containers, however their main goal is to define and implement a baseline benchmark test that can be used in containerized environments. As such they could be seen as a potential baseline test for future inclusion in ADW Bench rather than a directly comparable tool. In other cases, specialized benchmark management solutions have appeared, including test lifecycle management processes, targeting at specific use cases and domains (e.g. cloud hosted DBMS systems elasticity aspects in [30])

Overall what can be observed from the related work analysis is that there is a large number of tools available. However typically each one has its own features and drawbacks. In the case of ADW Bench, as differentiating factors the following can be summarized:

- It enables its usage either as a benchmarking framework or as a generic load injection tool for arbitrary loads, thus serving a dual nature.
- It enables the implementation of rich test definitions and diverse modes of execution.
- It targets containerized environments for its operation, deployment and test lifecycle.
- It is tailored and oriented around the easy acquisition of related datasets towards further processing such as machine learning models training.
- It is focused around portability, lightweight nature, all-in-one characteristics for minimizing dependencies while being based on a flow programming style that can be more attractive and abstract for developers.

Furthermore, with relation to usage in containerized environments, a complete description of requirements posed in such cases appears in [31]. Even though these are targeted primarily at baseline benchmarks for containerized environments, we consider it important also for the above benchmark/execution management layer (such as ADW Bench) to cover at least a part of them, since it can be anticipated that when such tools appear they will need to be managed in a similar fashion. ADW Bench meets related requirements such as inclusion of a software repository easily accessible from a public version control system (ADW Bench uses Gitlab), support for reusable container images, support for automated deployment and orchestration and easy to use interfaces among others.

Framework /Tool	User Interface	Application encapsulation	Multiplatform/ benchmark	Benchmark/ Load Injection Scope	Test Lifecycle Management	Test and mode variations	General scope	Open Source
ADW Bench	Web, REST API	Containers	Yes/Yes	Yes/Yes	Yes	Yes	lightweight, portable, single package, intuitive flow based programming style	Yes

ARTIST	Web	VMs	Yes/Yes	Yes/No	Yes, periodically	No	Same load, benchmark metrics over time	Yes
M-suite	No	Containers	Yes/No	Yes/No	No	No	Baseline benchmark definition	Yes
Smart Cloudbench	Web	VMs	Yes/Yes	Yes/No	Yes	No	Application and Resource metrics	N/A
Perfkit	With extensions but with vendor lock-in	VMs	Yes/Yes	Yes/No	Yes	No	Benchmark score, resource metrics	Yes
Octoperf	Web, REST API	N/A	Yes/No	No/Yes but heavily integrated with Jmeter	Yes		Application Scope	No
Cloudbench	Web, CLI, XML RPC	VMs	Yes/Yes	Yes/Partial	Yes	Test variations through test plans	centralized, production grade, VM oriented and infrastructure/ benchmark centered	Yes

Table 59 - Comparative Table of Features between ADW Bench and other tools

8.5 Next Steps

8.5.1 ADW Bench

In terms of ADW Bench, the next steps include:

- Finalization of integration with the Openshift Application Simulator adapter for generic submission testing and launch.
- Extension to new adapters for being able to deploy against external platforms (e.g. AWS Elastic Container service).
- Incorporation of data services in the benchmarked graph.
- Collection of necessary data that may be used for prediction model creation which will aid us in providing responses for cases that have not been benchmarked.

8.5.2 ADW Model Creation and Runtime Usage

In terms of ADW Runtime, the next steps include:

- Creation of baseline prediction models with relation to data services, application elementary types and the various aspects that affect their QoS.
- Propagation of this information at the level of the service graph, by understanding the structure of the graph (e.g. as a directed graph), using the initial workload input and combining the baseline models in one overall end to end QoS prediction.
- Implementation and testing of a RESTful service through which the respective models can be queried during runtime and the playbook population method.

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.

Apart from that Adaptive Visualizations will provide a multi-view/multi-role unified and structured User Interface that either consumes or integrates various components such us Process Modeller, Data Toolkit, Process Mapping, Benchmarking, Application Dimensioning Workbench, CEP, Triple Monitoring Engine, Data Quality Assessment and Predictive Maintenance components.

9.1 Requirements

The anticipated functionalities / requirements are described in the following tables (Table 60 -Table 73), that are compiled together with the rest of requirements of BigDataStack in D2.3.

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-01	System and Software	USE	ROL-2/ROL-03/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	N/A				
Status	Not Fulfilled				

Table 60 – 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	N/A				
Status	Fulfilled				

Table 61 – 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.				
Status	Fulfilled				

Table 62 – System Requirement (3) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-04	System and Software	FUNC	ROL-03	MAN
Name	The system should be able to consume/integrate Process Modeller Component.				
Description	Business Analyst can create graphs and export/edit/import them in latter time.				
Additional Information	N/A				
Status	Fulfilled				

Table 63 – System Requirement (4) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-05	System and Software	FUNC	ROL-02	MAN
Name	The system should be able to consume/integrate Data Toolkit Component.				
Description	Data Analyst can use the output provided by Process Modeller Component to apply Data Analyst Logic and enrich it.				
Additional Information	N/A				
Status	Fulfilled				

Table 64 – System Requirement (5) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-06	System and Software	FUNC	ROL-04	MAN

Name	The system should be able to integrate Benchmarking API.
Description	Application owner can define tests by providing the proper playbook and extract metrics.
Additional Information	N/A
Status	Fulfilled

Table 65 – System Requirement (6) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-07	System and Software	FUNC	ROL-04	MAN
Name	The system should be able to integrate Application Dimensioning Workbench.				
Description	Application owner imports a playbook produced by the Data Toolkit Component and choose assisted Mode Deployment to get Deployment Recommendations. These recommendations can be automatically deployed and monitored.				
Additional Information	N/A				
Status	Fulfilled				

Table 66 – System Requirement (7) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-08	System and Software	FUNC	ROL-04	MAN
Name	The system should be able to integrate Application Dimensioning Workbench.				
Description	Application owner imports a playbook produced by the Data Toolkit Component and choose Manual Mode Deployment to get Deployment Recommendations. These recommendations can be automatically deployed and monitored.				
Additional Information	N/A				
Status	Not Fulfilled				

Table 67 – System Requirement (8) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-09	System and Software	FUNC	ROL-04	MAN
Name	The system should be able to monitor the Deployed Application and receive notifications regarding QoS that are violated by the Dynamic Orchestrator and ADS Ranking Recommendator.				

Description	Monitoring Screen is provided in the Application Owner View for each application deployed.
Additional Information	N/A
Status	Fulfilled

Table 68 – System Requirement (9) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-10	System and Software	FUNC	ROL-04	MAN
Name	The system should be able consume and visualize datasets provided by the Data Quality Assessment Component.				
Description	User can extract information provided by the above component.				
Additional Information	N/A				
Status	Fulfilled				

Table 69 – System Requirement (10) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-11	System and Software	FUNC	ROL-04	MAN
Name	The system should be able consume and visualize datasets provided by the Maintenance component.				
Description	User can extract information provided by the above component.				
Additional Information	N/A				
Status	Fulfilled				

Table 70 – System Requirement (11) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-12	System and Software	FUNC	ROL-04	MAN
Name	The system should be able consume and visualize graphs from the CEP.				
Description	User can extract information provided by the above components.				
Additional Information	N/A				
Status	Fulfilled				

Table 71 – System Requirement (12) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-13	System and Software	FUNC	ROL-04	MAN
Name	The system should be able to provide different Views of the UI depending on the user role.				
Description	Different Components should be visible and accessed depending of the role that each user has. This role can be Business Analyst, Data Analyst, Application Owner and Administrator. Access to any view of the above requires authentication of the user.				
Additional Information	N/A				
Status	Fulfilled				

Table 72 – System Requirement (13) for Adaptable Visualizations

	Id	Level of detail	Type	Actor	Priority
	REQ-AV-14	System and Software	FUNC	ROL-02 ROL-03 ROL-04	MAN
Name	User Authentication				
Description	Authentication of the User should be performed once upon logging in the platform. Any additional authentication for individual components should happen in the background without further user interaction.				
Additional Information	N/A				
Status	Not Fulfilled				

Table 73 – System Requirement (14) for Adaptable Visualizations

9.2 Design Specifications

Figure 56 depicts the most commonly used architecture for visualizing big data.

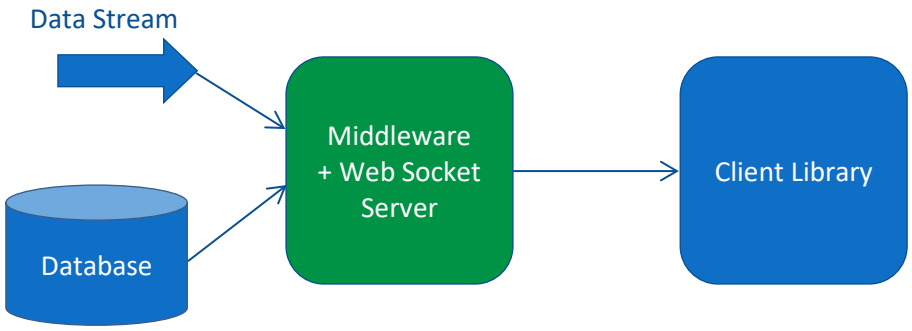


Figure 56 – Base architecture for visualizing big data

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.
The visualization of the data streams has been integrated with the CEP developed in BigDataStack. First of all, data streaming queries are designed through a visual editor so that, programmers do not need to program the CEP query. Those queries can be executed and the visualization tool for the CEP shows the live flow of data though the different query operators. Both the query editor and the online visualization are implemented using Angular 7 and Rete.js[15]. The online visualization tool uses WebSockets for communication with the CEP.
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 option selection is currently Chart js [20]: Open source javascript library that provides simple yet flexible charting for developers and designers.
All the components were integrated via Restful APIs and several graphs are directly consumed from Grafana software.

9.3 Experimentation Outcomes

An end to end scenario has already been implemented. A high level graph was produced by the business analyst using Process Modeller Component. The generated graph was propagated to the Data Analyst who extracted a playbook by invoking the Data Toolkit Component. This playbook was later on consumed by the ADW API and a recommendation was deployed and monitored. Application has access to performance metrics and QoS violation alarms and respective actions taken by the Dynamic Orchestrator Component.

Figure 57 – Monitoring QoS violations and performance of the Deployments

[15] https://rete.js.org/#/

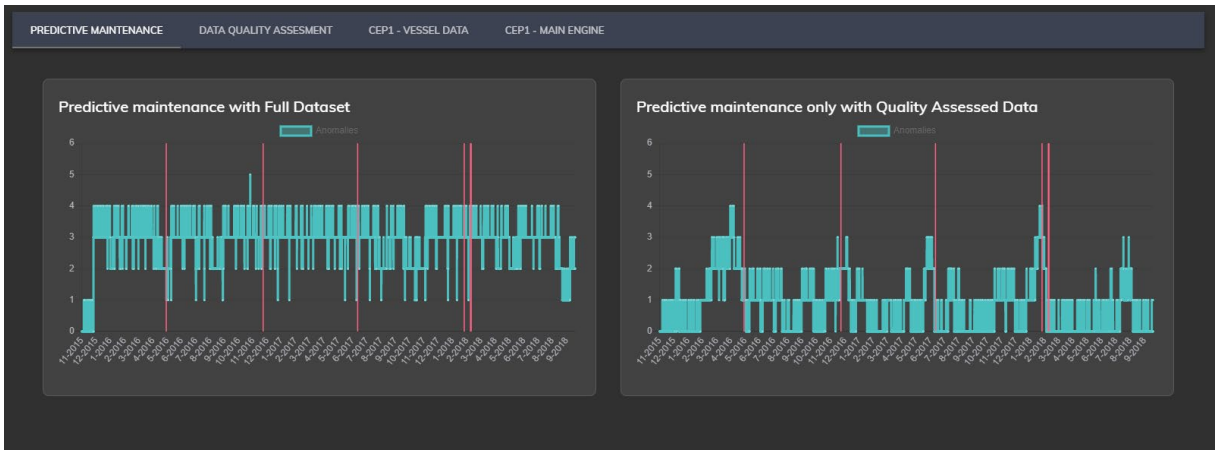


Figure 58 – Predictive maintenance Component

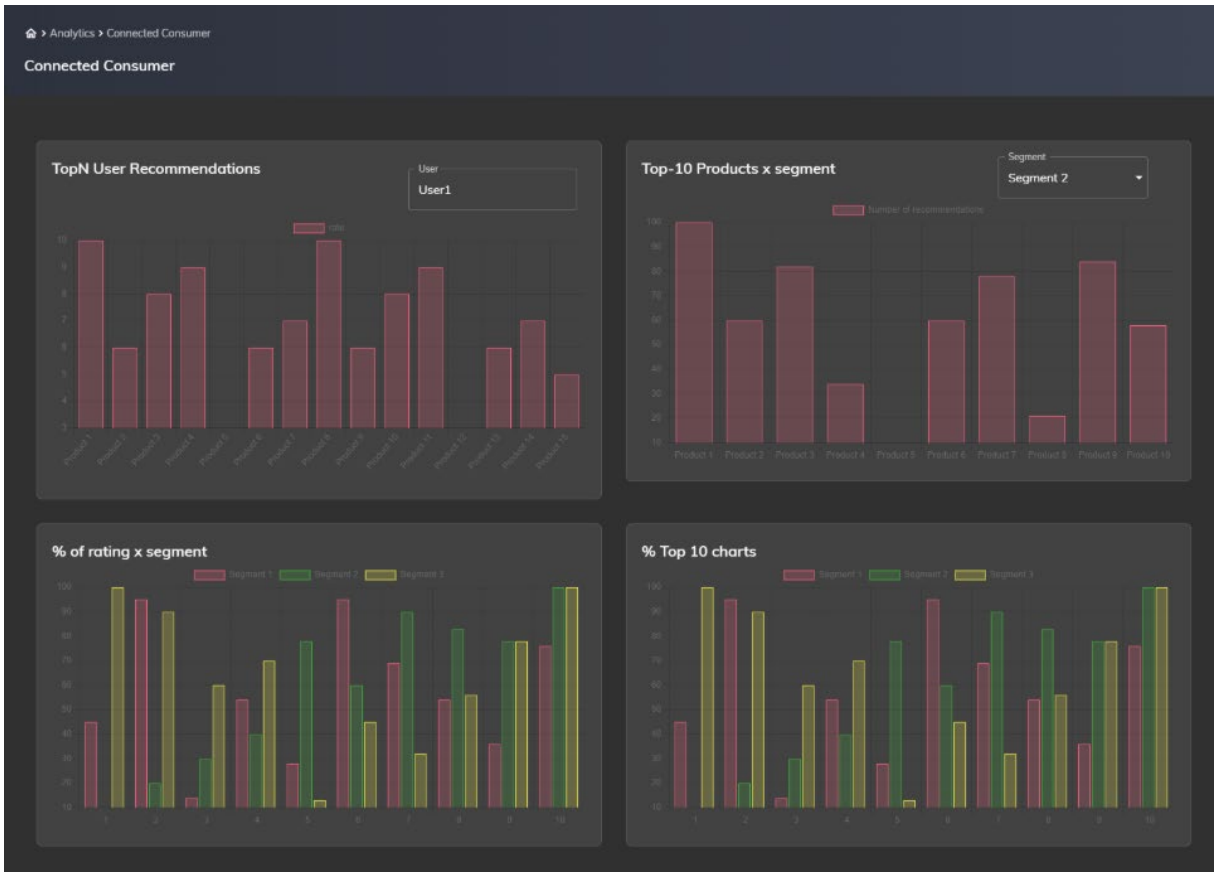


Figure 59 – Analytics Dashboard Component

The full queries for the Danaos use case are shown in Figure 60. A more detailed view is shown in Figure 61, where the boxes represent the operators with the conditions they are checking and arrows represent the stream of data and how it flows through operators.

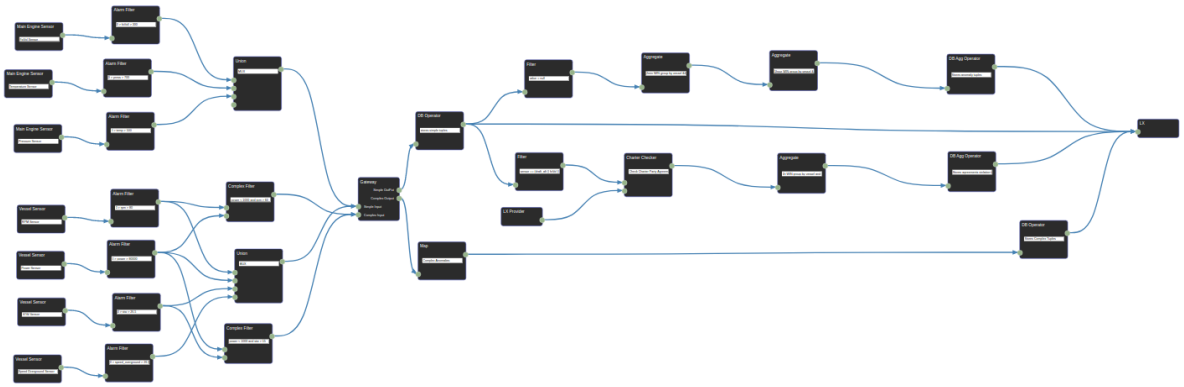


Figure 60 - Danaos CEP query

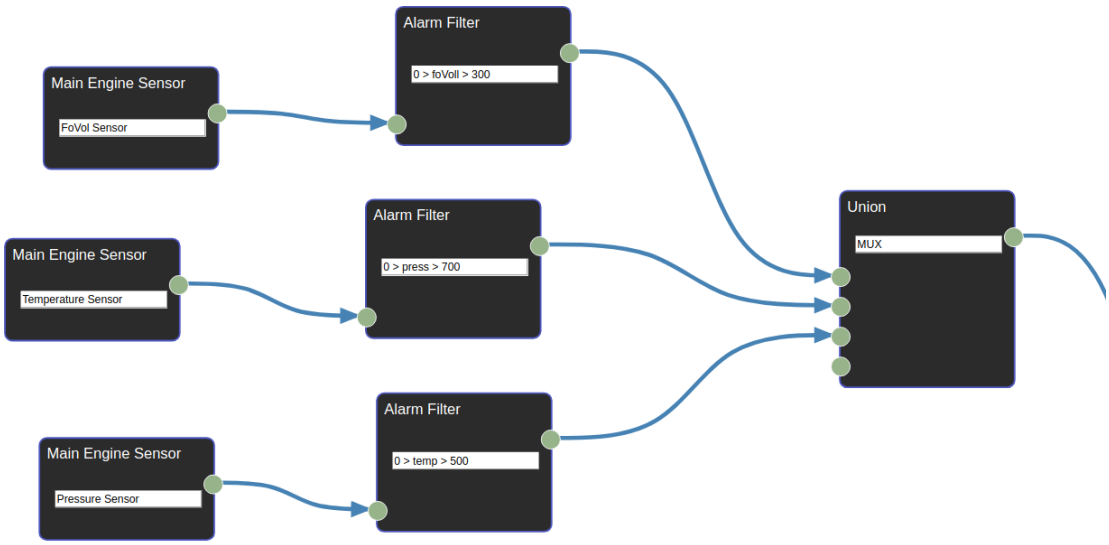


Figure 61 – Detail of one the Danaos queries

A first functional prototype of the online visualization of streams is already available. Figure 62 shows the visualization of CEP queries; more concretely, it presents the query run by CEP1 at the ship. For each query operator (represented as boxes), the name of the operator and part of the functionality is shown. For instance, the condition that checks whether the pressure is between 0 and 700 is shown. When the visualization is running, for each operator the last received tuple is also shown. A more detailed view of the same execution can be found in Figure 63. This figure presents a snapshot of the execution the Danaos query when real data is received from several sensors. On top of the figure a message is presented when an alarm is triggered. That is, one of the rules the query checks is not fulfilled.

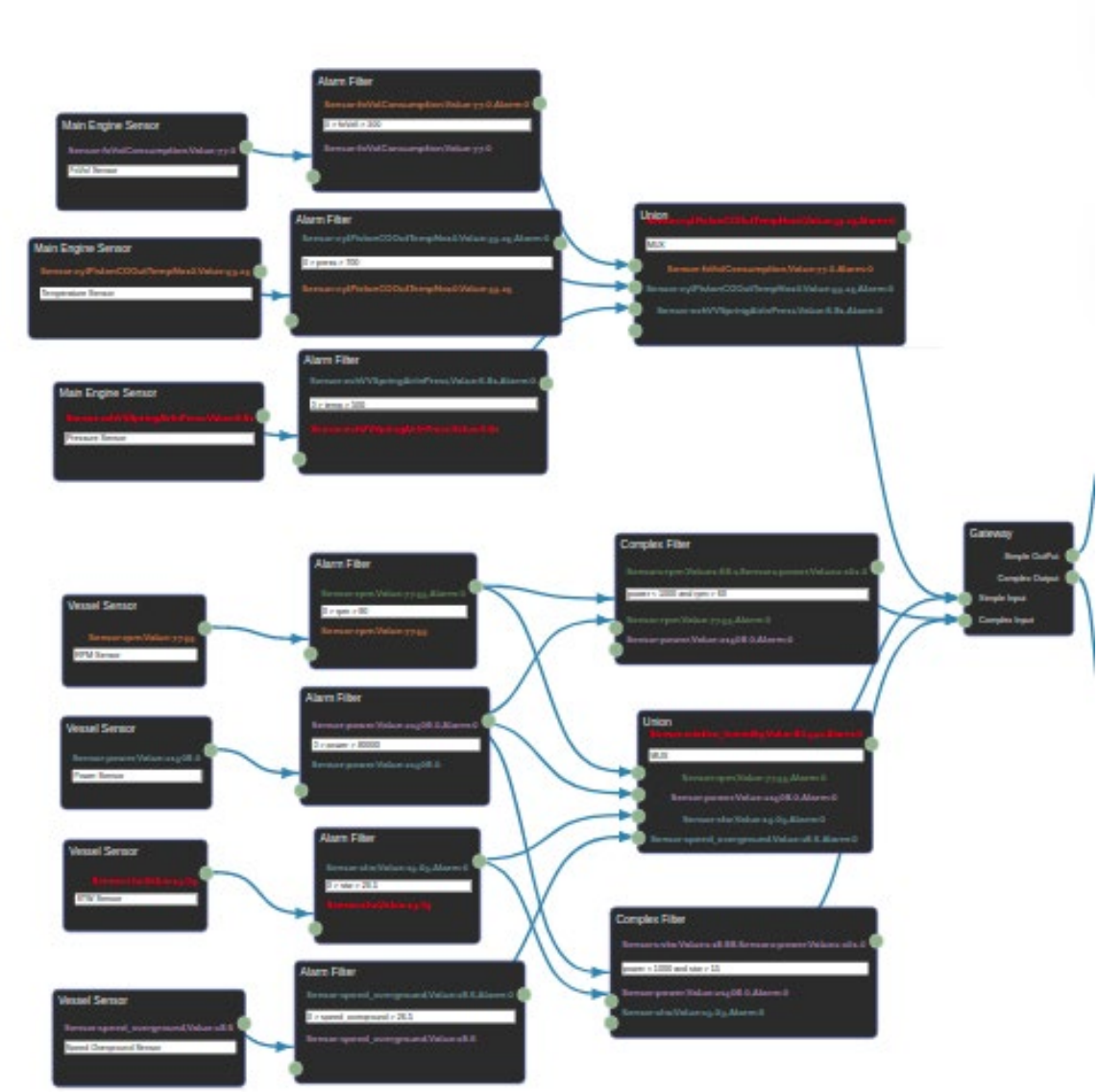


Figure 62- Visualization of CEP queries

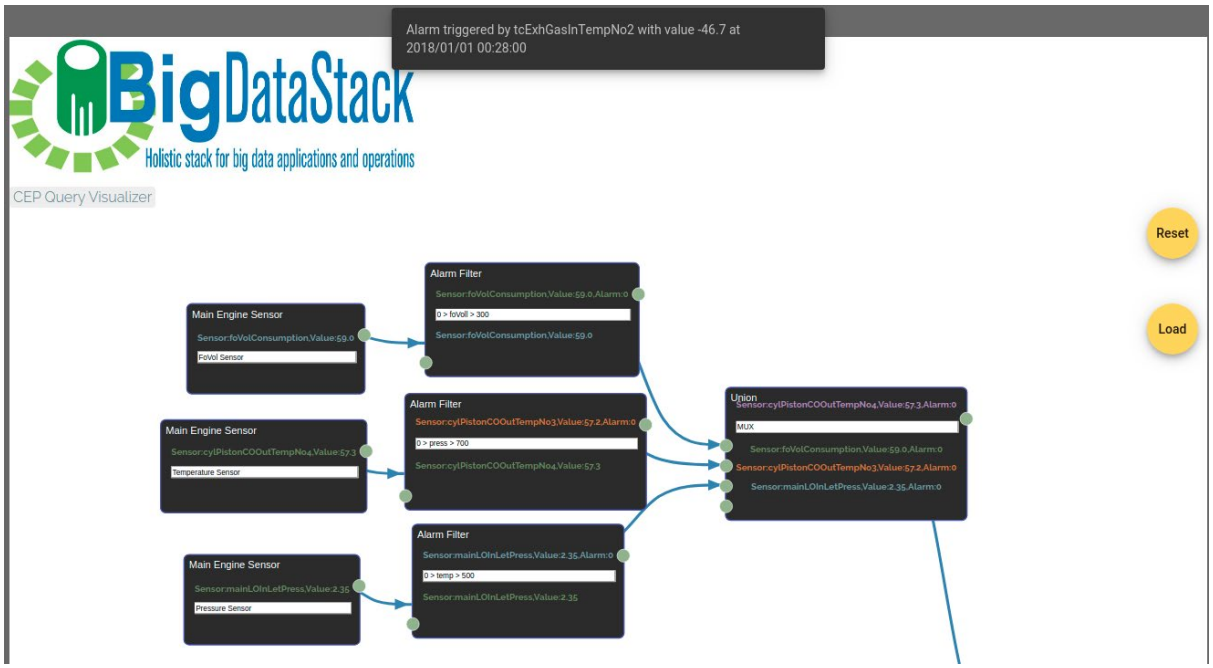


Figure 63 – Detailed visualization of the CEP queries execution

9.3.1 Evaluation results

An end-to-end scenario was successfully tested. Cross role data interaction resulted to successful deployment/ scaling/monitoring of applications.

9.4 Integration Highlights

All components have been consumed or integrated using RestFul APIs. Message queueing protocols (RabbitMQ) along with Web Socket technologies (Socket IO) were invoked for long time response services.

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:

- Integration of Data Toolkit Component. Additional login to Data Toolkit will be bypassed once the user is authenticated via JWT in the BigDS web platform.
- Integration of Dimensioning Workbench Component will be updated. Manual Deployment Mode will also be available.
- Analytics Dashboards will be more interactive in terms of filter application.
- Possible Migration towards Highchart, a Royalty-free, commercial, javascript library. Provides the implementations of hundreds of interactive graph types that can be easily integrated to any Javascript Application. This need emerges as a result of the dataset sizes.
- The dynamic visualization of the streams of data and the CEP queries requires functionality to control which elements are shown as well as mechanisms to slow down the rate at which the information flows.

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 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 experimentation. *Connected Consumer* and *Real-time ship management* UCs are used to validate the different releases of the components and their collaboration.

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] Hutter Frank, Kotthoff Lars, Vanschoren Joaquin: Automated Machine Learning.Springer.2019
[17] Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. J. Mach. Learn. Res., 13, 281-305.
[18] https://d3js.org/
[19] https://www.highcharts.com/
[20] https://www.chartjs.org/
[21] Kousiouris, G., Cucinotta, T. and Varvarigou, T., 2011. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. Journal of Systems and Software, 84(8), pp.1270-1291.
[22] Scheuner, J. and Leitner, P., 2018, July. Estimating cloud application performance based on micro-benchmark profiling. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD) (pp. 90-97). IEEE
[23] Kousiouris, G., Giammatteo, G., Evangelinou, A., Galante, N.A., Kevani, E., Stampoltas,

C., Menychtas, A., Kopaneli, A., Balraj, K.R., Kyriazis, D. and Varvarigou, T.A., 2014, April. A Multi-Cloud Framework for Measuring and Describing Performance Aspects of Cloud Services Across Different Application Types. In CLOSER (pp. 714-721)

[24] Silva, M., Hines, M.R., Gallo, D., Liu, Q., Ryu, K.D. and Da Silva, D., 2013, March. Cloudbench: Experiment automation for cloud environments. In 2013 IEEE International Conference on Cloud Engineering (IC2E) (pp. 302-311). IEEE

[25] Chhetri, M.B., Chichin, S., Vo, Q.B. and Kowalczyk, R., 2016. Smart CloudBench—A framework for evaluating cloud infrastructure performance. Information Systems Frontiers, 18(3), pp.413-428

[26] Google Perfkit Benchmarker, available at: https://github.com/GoogleCloudPlatform/PerfkitBenchmarker

[27] OctoPerf Load Testing Solution, available at: https://octoperf.com

[28] Sriraman, A. and Wenisch, T.F., 2018, September. µ Suite: A Benchmark Suite for Microservices. In 2018 IEEE International Symposium on Workload Characterization (IISWC) (pp. 1-12). IEEE.

[29] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). ACM, New York, NY, USA, 3-18. DOI: https://doi.org/10.1145/3297858.3304013

[30] Daniel Seybold, Simon Volpert, Stefan Wesner, André Bauer, Nikolas Herbst, and Jörg Domaschka. Kaa: Evaluating elasticity of cloud-hosted dbms. In Proceedings of the 11th IEEE International Conference on Cloud Computing (CloudCom), December 2019

[31] Aderaldo, C.M., Mendonça, N.C., Pahl, C. and Jamshidi, P., 2017, May. Benchmark requirements for microservices architecture research. In Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (pp. 8-13). IEEE Press