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/

D2.5 – Conceptual model and Reference architecture - II

Work Package	WP2 – Requirements, Architecture & Technical Coordination
Lead Author (Org)	Dimosthenis Kyriazis (UPRC)
Contributing Author(s) (Org)	Mauricio Fadel Argerich (NEC), Orlando Avila-García (ATOS), Ainhoa Azqueta (UPM), Bin Cheng (NEC), Ismael Cuadrado-Cordero (ATOS), Christos Doulkeridis (UPRC), Kostas Giannakopoulos (SILO), Gal Hammer (RHT), Ricardo Jimenez (LXS), Konstantinos Kalaboukas (SILO), Sophia Karagiorgou (UBI), Miki Kenneth (RHT), Pavlos Kranas (LXS), Nikos Lykousas (UBI), Richard McCreadie (GLA), Stavroula Meimetea (UPRC), Yosef Moatti (IBM), Konstantinos Papadimitriou (SILO), Marta Patino (UPM), Stathis Plitsos (DANAOS), Dimitrios Poulopoulos (UPRC), Bernat Quesada Navidad (ATOS), Amaryllis Raouzaiou (ATC), Martí Sanchez-Juanola (ATOS), Anestis Sidiropoulos (ATC/BAC), Paula Ta-Shma (IBM), Giannis Tsantilis (UBI)
Due Date	01.07.2019
Date	03.07.2019 (Re-submission: 04.10.2019)
Version	2.1

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
0.1.0	20.05.2019	Dimitris Poulopoulos (UPRC)	Initiated document
0.2.0	03.06.2019	Dimitris Poulopoulos (UPRC)	Updated architecture section
0.3.0	05.06.2019	George Kousiouris (UPRC)	Updated ADW section
0.3.1	05.06.2019	Maurizio Megliola (GFT)	GFT data structure & description
0.3.2	10.06.2019	Pavlos Kranas (LXS)	Updated Seamless Analytics
0.3.3	17.06.2019	UBI inputs	Updated Data Toolkit
0.3.4	18.06.2019	UPM inputs	Updated CEP
0.3.5	18.06.2019	Christos Doulkeridis (UPRC)	Updated Process Mapping
0.4.0	24.06.2019	Dimosthenis Kyriazis (UPRC)	Updated several sections based on overall updated architecture
0.4.1	24.06.2019	Mauricio Fadel Argerich (NEC)	Updated Dynamic Orchestrator
0.4.2	24.06.2019	RHT inputs	Updated Resources Management
0.4.3	24.06.2019	Ismael Cuadrado Cordero, Orlando Avila-García (ATOS)	Updated QoS Evaluation
0.4.4	24.06.2019	Richard McCreadie (GLA)	Updated ADS Ranking & Deployment
0.4.5	24.06.2019	Jean-Didier Totow (UPRC)	Updated Triple Monitoring
0.4.6	26.06.2019	Orlando Avila-García (ATOS)	Updated QoS Evaluation
0.5.0	26.06.2019	Amaryllis Raouzaiou (ATC)	Updated Process Modelling and Visualization Engine
0.5.1	30.06.2019	Richard McCreadie (GLA)	Review Part 1
0.6.0	03.07.2019	Dimosthenis Kyriazis (UPRC)	Pre-final version
1.0	03.07.2019	Dimosthenis Kyriazis (UPRC)	Final version
1.1	27.09.2019	Stathis Plitsos (DANAOS), Bernat Quesada Navidad (ATOS Worldline), Maurizio Megliola (GFT)	Updated version to address GDPR-related comments (Appendix 1, Appendix 2 and Appendix 3)
2.0	01.10.2019	Dimosthenis Kyriazis (UPRC)	Candidate final version
2.1	03.10.2019	Maurizio Megliola (GFT)	Anonymization fix

Disclaimer

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

Table of Contents_Toc13055840

List of tables

List of figures

BY NC ND

BY NC ND

1. Executive Summary

BigDataStack aims to deliver a complete stack including an infrastructure management solution that drives decisions according to live and historical data, thus being fully scalable, runtime adaptable and highly performant. The overall objective is for BigDataStack to address the emerging needs of big data operations and data-intensive applications. The solution will base all infrastructure management decisions on data aspects (for example the estimation and provision of resources for each data service based on the corresponding data loads), monitoring data from deployments and logic derived from data operations that govern and affect storage, compute and network resources. On top of the infrastructure management solution, "Data as a Service" will be offered to data providers, decision-makers, private and public organisations. Approaches for data quality assessment, data skipping and efficient storage, combined with seamless data analytics will be realised holistically across multiple data stores and locations.

To provide the required information towards enhanced infrastructure management BigDataStack will provide a range of services, such as the application dimensioning workbench, which facilitates data-focused application analysis and dimensioning in terms of predicting the required data services, their interdependencies with the application micro-services and the necessary underlying resources. This will allow the identification of the applications data-related properties and their data needs, thereby enabling BigDataStack to provision deployment with specific performance and quality guarantees. Moreover, a data toolkit will enable data scientists to ingest their data analytics functions and to specify their preferences and constraints, which will be exploited by the infrastructure management system for resources and data management. Finally, a process modelling framework will be delivered, to enable functionality-based modelling of processes, which will be mapped in an automated way to concrete technical-level data analytics tasks.

The key outcomes of BigDataStack are reflected in a set of main building blocks in the corresponding overall architecture of the stack. This deliverable is a refinement of the key functionalities of the overall architecture, the interactions between the main building blocks and their components, as they were described in the previous version of the architecture (Deliverable D2.4 - Conceptual model and Reference architecture). Comparing to the previous version of the architecture, key changes refer to the interplay between the application and data dimensioning and the components that manage the deployment lifecycle (i.e. deployment patterns generation and ranking and deployment management), the dynamic orchestrator and the overall quality and performance assessment during runtime. Additionally, there are changes in the specifications of several components (reflecting their latest implementation status) and as such their associated sections have received updates in this document as well (e.g. seamless analytics framework). It should be noted that additional design details and evaluation results for all components of the architecture will be delivered in the corresponding follow-up (WP-specific) deliverables addressing the user interaction block, the data as a service block and the infrastructure management block. It should be noted that v2.0 of this deliverable has been released to include relevant GDPR-related information (updates in Appendix 1, Appendix 2 and Appendix 3).

2. Introduction

The new data-driven industrial revolution highlights the need for big data technologies, to unlock the potential in various application domains (e.g. transportation, healthcare, logistics, etc). In this context, big data analytics frameworks exploit several underlying infrastructure and cluster management systems. However, these systems have not been designed and implemented in a "big data context". Instead, they emphasise and address the computational needs and aspects of applications and services to be deployed.

BigDataStack aims at addressing these challenges (depicted in Figure 1) through concrete offerings, that range from a scalable, runtime-adaptable infrastructure management system (that drives decisions according to data aspects), to techniques for dimensioning big data applications, modelling and analysing of processes, as well as provisioning data-as-a-service by exploiting a seamless analytics framework.

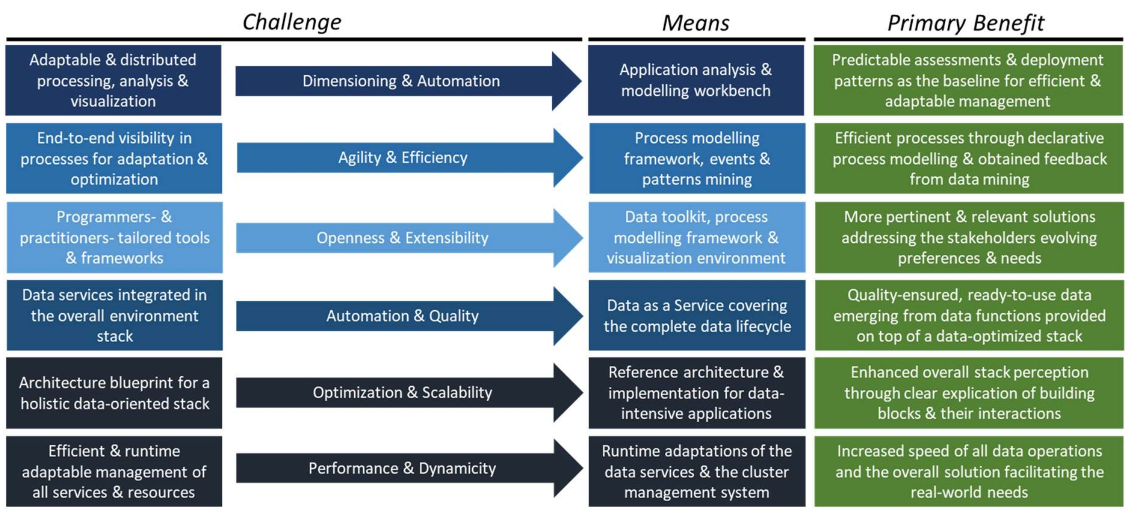


Figure 1 - Technical challenges

2.1. Terminology

The following table summarises a set of key terms used in BigDataStack, not regarding acronyms but regarding actual usage, given the big number of concepts and technologies addressed by the envisioned stack.

Term	Description
Application services	Components/micro-services of a user's application
Data services	"Generic" services such as cleaning, aggregation, etc.
Dimensioning	Analysis of a user's application services to identify the data and resources needs/requirements
Toolkit	Mechanism enabling ingest of data analytics tasks & setting of requirements (from an end-user point of view)
Graph	An overall graph including the application services and the data services
Process modelling	"Workflow" modelling regarding business processes
Process mining	Analytics tasks per process of the "workflow"
Process mapping	Mapping of business processes to analytics tasks to be executed

Interdependencies between application / data services	Data flows between application components and data services

Table 1 - Terminology

2.2. Document structure

The document is structured as follows:

- Section 3 provides an overview of the capabilities offered by the BigDataStack environment, including the key offerings and the main stakeholders addressed by each offering.
- Section 4 introduces the identified main phases, to showcase the interactions between different key blocks and offerings of the stack.
- Section 5 presents the overall project architecture.
- Section 6 provides descriptions of the main architecture components.
- Finally, in Section 7, a detailed sequence of events depicting the information flows is provided. It should be noted that these sequence diagrams capture the interactions on the overall architecture level and are not supposed to provide details of the interactions on lower levels given that these are provided by the corresponding design and specification reports of the work package deliverables and will be refined in later reports accordingly.

3. BigDataStack Capabilities

This section provides an overview of the capabilities that will be offered by BigDataStack, in terms of offerings towards an extensive set of stakeholders. The goal is to present a set of "desired" capabilities as the key goals of BigDataStack. The components providing and realising these capabilities are thereafter described in the overall architecture.

3.1. Key offerings

BigDataStack offerings are depicted through a full "stack", that aims not only to facilitate the needs of data operations and applications (all of which tend to be data-intensive), but also promote these needs in an optimized way.

As depicted in Figure 2, BigDataStack will provide a *complete infrastructure management system*, which will base the management and deployment decisions on data from current and past application and infrastructure deployments. A representative example would be that of a service-defined deployment decision by a human expert (current approach), where he chooses to deploy VMs on the same physical host, to reduce data transfer latencies over the network (e.g. for real-time stream processing). On the other hand, the BigDataStack approach instead will base the decision making according to information from current and past deployments (e.g. generation rates, transfer bottlenecks, etc.), which may result in a superior deployment configuration. To this end, the BigDataStack infrastructure management system would propose a data-driven deployment decision resulting in containers/VMs placed within geographically distributed physical hosts. This simple case shows that the trade-off between service and data-based decisions on the management layer should be re-examined nowadays, due to the increasing volumes and complexity of data. The envisioned "stack" is depicted in Figure 2, which captures the key offerings of BigDataStack.

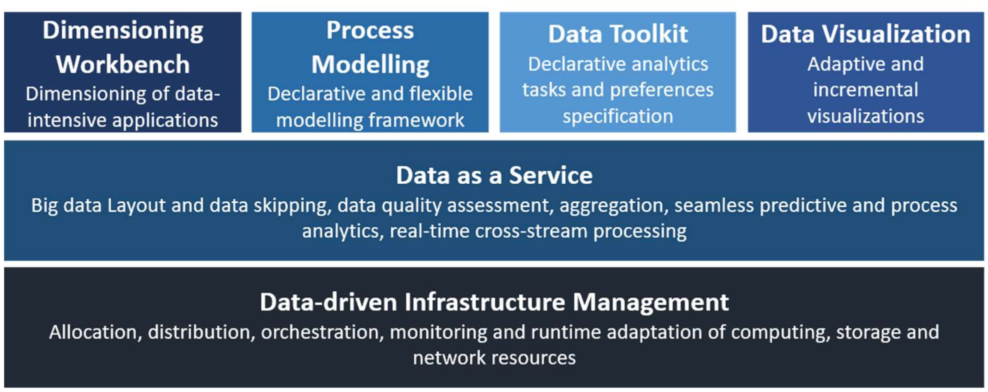


Figure 2 - Key offerings

The first core offering of BigDataStack is *efficient and optimised infrastructure management,* including all aspects of management for the computing, storage and networking resources, as described before.

The second core offering of BigDataStack exploits the underlying data-driven infrastructure management system, to provide *Data as a Service in a performant, efficient and scalable way*. Data as a Service incorporates a set of technologies addressing the complete data path: data quality assessment, aggregation, and data processing (including seamless analytics, real-time Complex Event Processing - CEP, and process mining). *Distributed storage* is realised through a layer, enabling data to be fragmented/stored according to different access patterns in different underlying data stores. A *big data layout and data skipping* approach is used to minimize the data that should be read from the underlying object store to perform the corresponding analytics. The *seamless data analytics framework* analyses data in a holistic fashion across multiple data stores and locations and operates on data irrespective of where and when it arrives at the framework. A *cross-stream processing engine* is also included in the architecture to enable distributed processing of data streams. The engine considers the latencies across data centres, the locality of data sources and data sinks, and produces a partitioned topology that will maximise the performance.

The third core offering of BigDataStack refers to *Data Visualization*, going beyond the presentation of data and analytics outcomes to *adaptable visualisations in an automated way*. Visualizations cover a wide range of aspects (interlinked if required) besides *data analytics*, such as *computing, storage and networking infrastructure data*, *data sources* information, and *data operations* outcomes (e.g. data quality assessment outcomes, application analytics outcomes, etc.). Moreover, the BigDataStack *visualisations will be incremental*, thus providing data analytics results as they are produced.

The fourth core offering of BigDataStack, the *Data Toolkit, aims at openness and extensibility*. The toolkit allows the *ingestion of data analytics functions* and the *definition of analytics*, providing at the same time *"hints" towards the infrastructure/cluster management system for the optimised management* of these analytics tasks. Furthermore, the toolkit allows data scientists to *specify requirements and preferences* as service level objectives (e.g. regarding the response time of analytics tasks), which are considered by *infrastructure management* both during deployment time and during runtime (i.e. triggering adaptations in an automated way).

The *Process Modelling* offering provides a *framework allowing for flexible modelling of process analytics* to enable their execution. Process chains (as workflows) can be specified through the framework, along with overall workflow objectives (e.g. accuracy of predictions, overall time for the whole workflow, etc) that are considered by mechanisms mapping the aforementioned processes to data analytics that can be executed directly on the BigDataStack infrastructure. Moreover, process mining tasks realize a feedback loop towards overall process optimisation and adaptation.

Finally, the sixth offering of BigDataStack, the *Dimensioning Workbench* aims at enabling the dimensioning of applications in terms of predicting the required data services, their interdependencies with the application micro-services and the necessary underlying resources.

3.2. Stakeholders addressed

BigDataStack provides a set of endpoints to address the needs of different stakeholders as described below:

1. *Data Owners:* BigDataStack offers a unified *Gateway* to obtain both streaming and stored data from data owners and record them in its underlying storage infrastructure that supports SQL and NoSQL data stores.
2. *Data Scientists:* 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 response time of a specific analytics task).
3. *Business Analysts:* BigDataStack offers the *Process Modelling Framework* allowing business users to define their functionality-based business processes and optimise them based on the outcomes of process analytics that will be triggered by BigDataStack. Mapping to specific process analytics tasks will be performed in an automated way.
4. *Application Engineers and Developers:* BigDataStack offers the *Application Dimensioning Workbench* to enable application owners and engineers to experiment with their application and obtain dimensioning outcomes regarding the required resources for specific data needs and data-related properties.

These actors interact with the corresponding offerings and provide information that is exploited thereafter by the infrastructure/cluster management system of BigDataStack. It should be noted that on top of these offerings, the *Visualization Environment* is also an interaction point with end users, providing the outcomes of analytics as well as the monitoring results of all infrastructure and data-related operations.

BY NC ND

4. Main phases

The envisioned operation of BigDataStack is reflected in four main phases as depicted in Figure 3 (and further detailed in the following sub-sections): Entry, Dimensioning, Deployment and Operation.

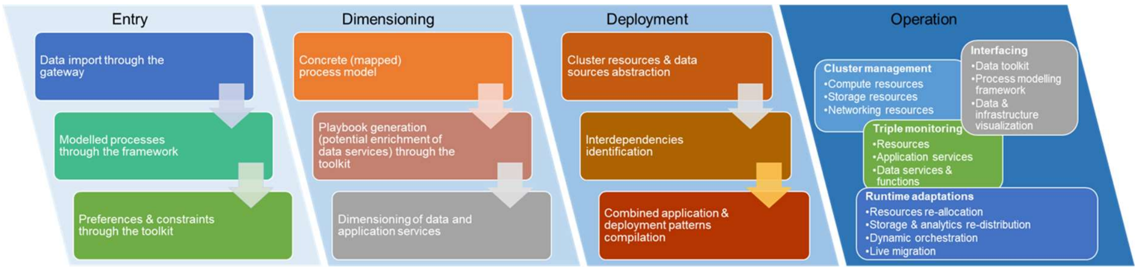


Figure 3 - BigDataStack Main Phases

During the entry phase, data owners ingest their data through a unified gateway. Analysts design business processes by utilising the functionalities of the Process Modelling framework in order to describe the overall business workflows, while data scientists can specify their preferences and pose their constraints through the Data Toolkit.

During the dimensioning phase, the individual processes / steps of the overall process model (i.e. workflow) are mapped to analytics tasks, and the graph is concretized (including specific analytics tasks and application services to be deployed). The whole workflow is modelled as a playbook descriptor and is passed to the Dimensioning Workbench. In turn, the Dimensioning Workbench provides insights regarding the required infrastructure resources, for the data services and application components, through an envisioned elasticity model that includes estimates for different Quality of Service (QoS) requirements and Key Performance Indicators (KPIs).

The goal of the deployment phase is to deliver the optimum deployment patterns for the data and application services, by considering the resources and the interdependencies between application components and data services (based on the dimensioning phase outcomes).

Finally, the operation phase facilitates the provision of data services including technologies for resource management, monitoring and evaluation towards runtime adaptations.

4.1. Entry phase

During the entry phase, data is introduced into the system, the Business Analysts design and evaluate their business processes, and the Data Scientists specify their preferences and constraints through the Data Toolkit. Thus, the Entry Phase consists of three discrete steps:

- Data owners ingest their data in the BigDataStack-supported data stores, through a unified gateway. They can directly choose if they want to store (non-) relational data or use the BigDataStack's object storage offering. The seamless analytics framework brings together the LeanXcale database and the Object Store into a new entity, permitting the definition of rules for automatic balancing of datasets between these two basic data storage components (e.g. data older than 3 months should be moved

BY NC ND

to the object store), as well as to describe and use a dataset, which may be spread over the two data storage components seamlessly. Streaming data can also be processed, leveraging the BigDataStack's CEP implementation.

- Given the stored data, Business Analysts can design processes utilising the intuitive graphical user interface provided by the Process Modelling framework, and the available list of "generic" processes (e.g. customer segmentation process). Overall, they compile a business workflow, ready to be mapped to concrete executable tasks. These mappings are performed by a mechanism incorporated in the Process Modelling framework, the Process Mapping component.
- Based on the outcomes of process mapping, the graph of services (representing the corresponding business workflow) is made available to the Data Scientists through the Toolkit. The scientists can specify preferences for specific tasks, for example, what the response time of a recommendation algorithm should be or 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 Kubernetes-like configuration template file describing the graph/workflow (which includes all relevant information for the application graph with concrete "executable" services). We refer to this as a *BigDataStack Playbook*. This is passed to the dimensioning phase in order to identify the resource needs for the identified services.

4.2. Dimensioning phase

The dimensioning phase of BigDataStack aims to optimize the provision of data services and data-intensive applications, by understanding not only their data-related requirements (e.g. related data sources, storage needs, etc.) but also the data services requirements across the data path (e.g. the resources needed for effective data storage, analytics, etc.), and the interdependencies when moving from an atomic / single service to an application graph. In this context, dimensioning includes a two-step approach that is realised through the BigDataStack Application Dimensioning Workbench:

- In the first step, the input from the Data Toolkit is used to define the composite application (consisting of a set of micro-services) needs with relation to the required data services. The example illustrated in Figure 4 shows that 3 out of the 5 application components require specific data services for aggregation and analytics.
- The second step is to dimension these identified/required data services, as well as all the application components, regarding their infrastructure resource needs. That is achieved by exploiting load injectors generating different loads, to benchmark the services and analyse their resources and data requirements (e.g. volume, generation rate, legal constraints, etc.).

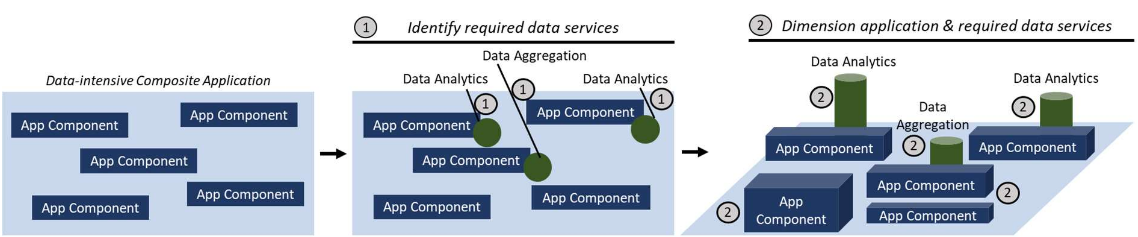


Figure 4 - Dimensioning phase

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.).

4.3. Deployment phase

The deployment phase of BigDataStack aims at determining the optimum deployment configuration and deployment resources for the application and data services in terms of cluster resources. The need for such configuration emerges from the fact that to deploy the application and data services in a way such that it will meet the user's needs, BigDataStack needs to account for the application and data services complexity/efficiency, the workload (e.g. requests per second) and the user-defined quality of service requirements/preferences (e.g. <100ms response time).

To this end, the deployment phase of BigDataStack includes a four-step process:

- In a first step of the deployment phase, the application and data services compositions (as represented by a BigDataStack playbook) is analysed, and the independent sub-structures comprised of application and data services (referred to as "pods") are identified.
- Second, a set of resource templates are used to convert each pod into a series of candidate deployment patterns (CDPs), where each CDP is comprised of a pod and resource template.
- Third, for each CDP, performance estimations are obtained from the Dimensioning phase (based on prior application benchmarking and analysis) given expected data and application workload or workloads.
- Finally, each CDP is scored with respect to the user's quality of service requirements and/or preferences to determine the suitability of each. The best configuration for each pod is then selected, either for immediate deployment or to be shown to the user for prior approval.

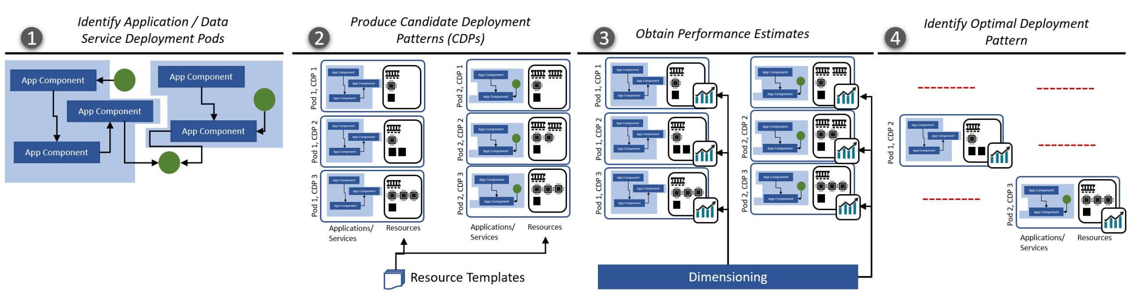


Figure 5 - Deployment phase

4.4. Operations phase

The operation phase of BigDataStack is realised through different components of the BigDataStack infrastructure management system and aims at the management of the complete physical infrastructure resources, in an optimised way for data-intensive applications.

The operation phase includes a seven-step process as depicted in Figure 6:

- Based on the deployment phase, outcomes regarding the optimised deployment pattern, computing resources are reserved and allocated.
- According to the allocated computing resources, storage resources are also reserved and allocated. It should be noted that storage resources are distributed.
- Data-driven networking functions are compiled and deployed to facilitate the diverse networking needs between different computing and storage resources.
- The application components and data services are deployed and orchestrated based on "combined" data and application-aware deployment patterns. An envisioned orchestrator mechanism compiles the corresponding orchestration rules according to the deployment patterns and the reserved computing, storage and network resources.
- Data analytics tasks will be distributed across the different data stores to perform the corresponding analytics, while analytics on top of these stores is performed through the seamless analytics framework.
- Monitoring data is collected and evaluated for the resources (computing, storage and network), application components and data services and functions (e.g. query execution status).
- Runtime adaptations take place for all elements of the environment, to address possible QoS violations. These include resource re-allocation, storage and analytics re-distribution, re-compilation of network functions and deployment patterns.

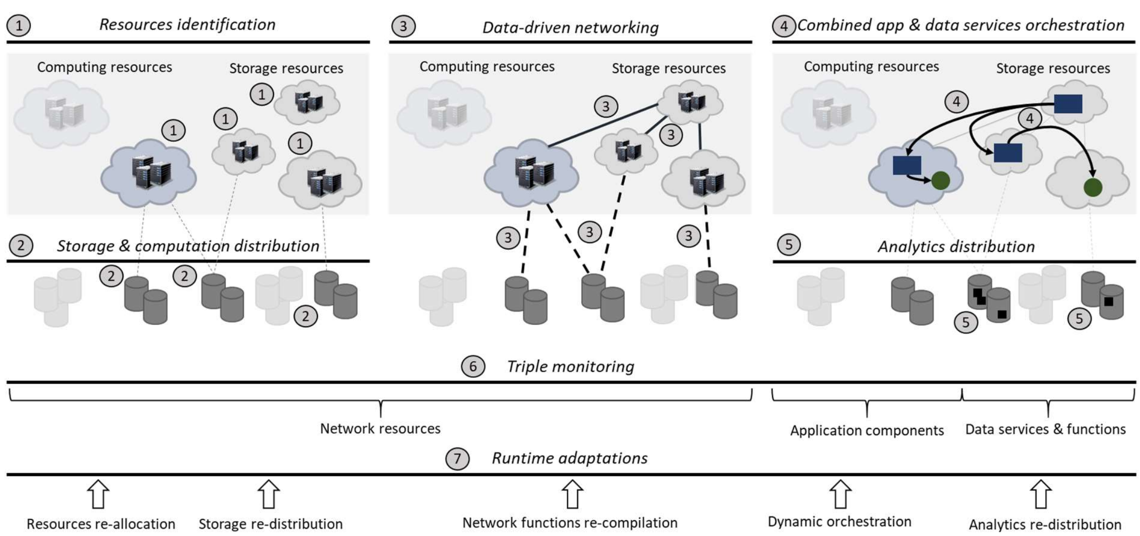


Figure 6 - Operations phase

5. Architecture

The following figure presents the overall conceptual architecture of BigDataStack, including the main information flows and interactions between the key components.

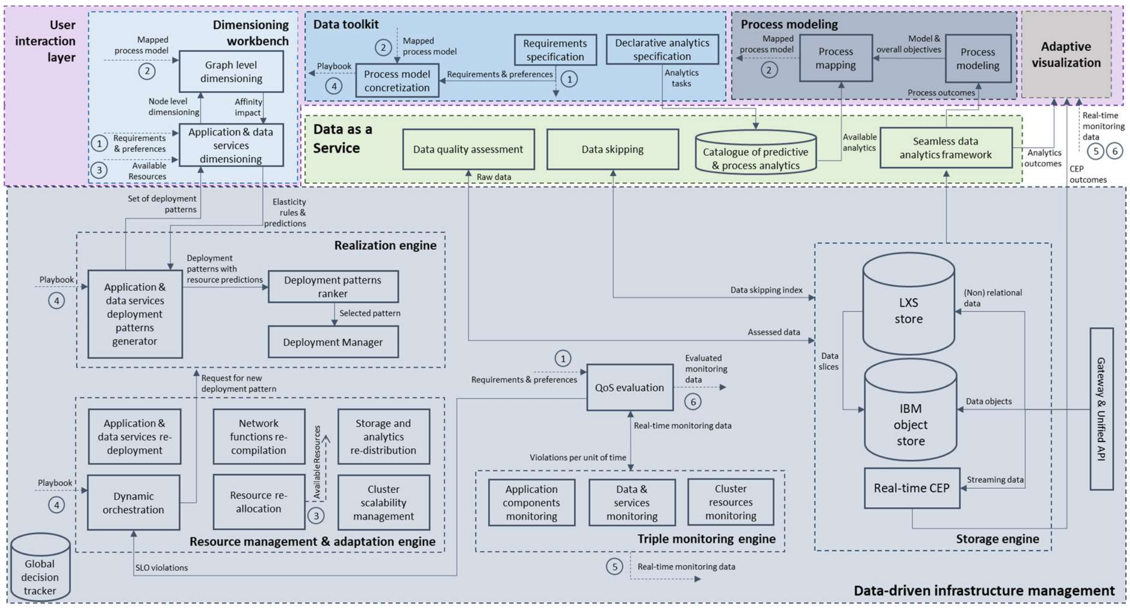


Figure 7 - BigDataStack architecture model

First, raw data are ingested through the *Gateway & Unified API* component to the *Storage* engine of BigDataStack, which enables storage and data migration across different resources. The engine offers solutions both for relational and non-relational data, an *Object Store* to manage data as objects, and a CEP engine to deal with streaming data processing. The raw data are then processed by the *Data Quality Assessment* component, which enhances the data schema in terms of accuracy and veracity and provides an estimation for the corresponding datasets in terms of their quality. Data stored in *Object Store* are also enhanced with relevant metadata, to track information about objects and their dataset columns. Those metadata can be used to show that an object is not relevant to a query, and therefore does not need to be accessed from storage or sent through the network. The defined metadata are also indexed, so that during query execution objects that are irrelevant to the query can be quickly filtered out from the list of objects to be retrieved for the query processing. This functionality is achieved through the *Data skipping* component of BigDataStack. Moreover, slices of historical data are periodically transferred from the *LeanXcale* database to the *Object Store,* to free-up space for fresh tuples.

Given the stored data, decision-makers can model their business workflows through the *Process Modelling framework* that incorporates two main components: the first component is *Process modelling*, which provides an interface for business process modelling and the specification of an end-to-end optimisation goals for the overall process (e.g. accuracy, overall completion time, etc). The second component refers to *Process Mapping.* Based on the analytics tasks available in the *Catalogue of Predictive and Process Analytics* and the specified overall goals, the mapping component identifies analytics algorithms that can realise the corresponding business processes. The outcome of the component is a model in a structural representation e.g. a JSON file that includes the overall workflow, and the mapped business processes to specific analytics tasks.

Following, through the *Data Toolkit*, data scientists design, develop and ingest analytic processes/tasks to the *Catalogue of Predictive and Process Analytics*. This is achieved by combining a set of available or under development analytic functions into a high-level definition of the user's application. For instance, they define executables/scripts to run, as well as the execution endpoints per workflow step. Data scientists can also declare input/output data parameters, analysis configuration hyper-parameters (e.g. the *k* in a *k*-means algorithm), execution substrate requirements (e.g. CPU, memory limits etc.) as service level objectives (SLOs), as well as potential software packages / dependencies (e.g. Apache Spark, Flink etc.). The output of the *Data Toolkit* component enriches the output of the previous step (i.e. *Process Modelling*) and defines a BigDataStack Playbook.

The generated playbook is utilized by the *Application and Data Services Deployment Patterns Generator.* The component creates different arrangements (i.e. patterns / configurations) of deployment resources for each application and data service Pod. These candidate deployment patterns (CDPs) are passed to the *Application Dimensioning Workbench,* along with an end-to-end optimization objective and the information on the available resources, in order to estimate resource usage and QoS performance prior to actual deployment. The primary output of the *Application Dimensioning Workbench* is an elasticity model, which defines the mapping of the input QoS parameters to the concrete resource needed (such as the number of VMs, bandwidth, latency etc.). These decisions are depended on data-defined models. Thus, based on the obtained dimensioning outcomes, deployment patterns are ranked by the *Deployment Patterns Ranker* and the optimum pattern is selected for deployment, making the concluding arrangement of services data-centric. The *Deployment Manager* administers the setup of the application and data services on the allocated resources.

During runtime, the *Triple Monitoring engine* collects data regarding resources, application components (e.g. application metrics, data flows across application components, etc.) and data operations (e.g. analytics / query progress, storage distribution, etc.). The collected data are evaluated through the *QoS Evaluation* component to identify events / facts that affect the overall quality of service (in comparison with the SLOs set in the toolkit). The evaluation outcomes are utilised by the *Runtime adaptation engine*, which includes a set of components (i.e. cluster resources re-allocation, storage and analytics re-distribution, network functions re-compilation, application and data services re-deployment, and dynamic orchestration patterns), to trigger the corresponding runtime adaptations needed for all infrastructure elements to maintain QoS.

Moreover, the architecture includes the *Global decision tracker*, which aims at storing all the decisions taken by the various components. The overall BigDataStack system takes advantage of this recorded historical information to perform future optimisations. The key rationale for the introduction of this component is the fact that decisions have a cascading effect in the proposed architecture. For example, a dimensioning decision affects the deployment patterns compilation, the distribution of storage and analytics, etc. The information about whether these decisions are altered during runtime will be exploited for optimised future choices across all components through the decision tracker. Moreover, the tracker holds additional information such as application logging data, Candidate Deployment Patterns, QoS failures, etc. Thus, as a global state tracker, provides the ground for cross-component optimisation, as well as tracking the state and history of BigDataStack applications.

Finally, the architecture includes the *Adaptive Visualisation* environment, which provides a complete view of all information, including raw monitoring data (for resource, application and data operations) and evaluated data (in terms of SLOs, thresholds and the evaluation of monitoring in relation to these thresholds). Moreover, the visualization environment acts as a unique point for BigDataStack for different stakeholders, actors, thus, incorporating the process modelling environment, the data toolkit and the dimensioning workbench. These accompany the views for infrastructure operators (e.g. regarding deployment patterns).

6. Main architectural components

Based on the overall architecture presented in the previous chapter, this chapter provides additional information regarding the individual components of the BigDataStack architecture.

6.1. Resources Management

The Resource Management sub-system provides an Enterprise grade platform which manages Container-based and Virtual Machine-based applications consistently on cloud and on-premise infrastructures. This sub-system makes the physical resources (e.g. CPUs, NICs and Storage devices) transparent to the applications. The application's requirements will be computed based on the input from the Realisation Engine and by a constant monitoring using the Triple Monitor. The applications' required resources are automatically allocated from the available existing infrastructures and will be dismissed upon execution completion. Thus, the Resource Management sub-system serves as an abstraction layer over today's infrastructures, physical hardware, virtual hardware, private and public clouds. This abstraction allows the developing of compute, networking and storage management algorithms which can work on a unified system, rather than dealing with the complexity of a distributed system.

BigDataStack will build on top of the open source OpenShift Kubernetes Distribution (OKD) project [1] for its Resource Management sub-system. The OKD project is an upstream project used in Red Hat's various OpenShift products. It is based and build around Kubernetes and operators and is enhanced with features requested by commercial customers and Enterprise level requirements. According to Duncan et al. [2] ODK is "an application platform that uses containers to build, deploy, serve, and orchestrate the applications running inside it". OKD simplifies the whole process [3] of the deployment of a "fine-grained management over common user applications" and management of the containerized software (the lifecycle of the applications). Since its initial release in 2011, it has been adopted by multiple organizations and has grown to represent a large percentage of the market. According to IDC [4], OKD aims at accelerating the application delivery with "agile and DevOps methodologies"; moving the application architectures toward micro-services; and adopting a consistent application platform for hybrid cloud deployments.

As a base technology, OKD uses Docker and/or CRI-O for containerization and Kubernetes [5] for their orchestration, including packaging, instantiation and running the containerized applications. It also implements "geard" or "gear daemon" [6], a command-line client for the management of containers and its linkage to systems across multiple hosts, used for the installation and management of application components [7]. On top of the above described technologies, OKD adds [8]:

- Source code management, builds, and deployments for developers
- Managing and promoting images at scale as they flow through your system
- Application management at scale
- Team and user tracking for organizing a large developer organization
- Networking infrastructure that supports the cluster

OKD integrates in the DevOps and users' operation following a hierarchical structure, as shown in Figure 8. A master node centralizes the API/authentication, data storage, scheduling, and management/replication operations, while applications are run on Pods (following the Kubernetes philosophy).

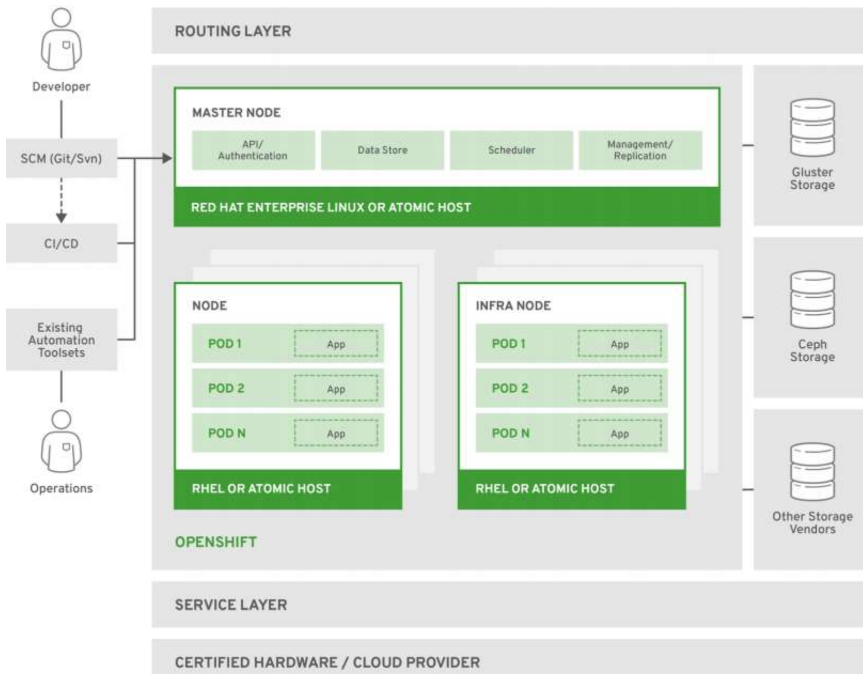


Figure 8 - OKD architecture overview inside the DevOps operation [8]

Following this layered architecture, users access the API, web-services and command line directly from the master node, while the applications and data services are accessed through the routing layer where the services are located, that is, in the physical machine the pod was deployed. Finally, the integrated container registry includes the set of container images which can be deployed in the system.

Another important point for the project is the protection of security and privacy of the user. On top of the security provided by Kubernetes, OKD also offers granular control on the security of the cluster. As shown in [4], users can choose a whitelist of cipher suites to meet security policies; and share PID between containers to control the cooperation of containers.

By building on top of OKD, we ensure that BigDataStack components are easily portable to different cloud offerings, such as Amazon, Google Compute Engine, Azure, or any On-Premise deployment based on OpenStack.

To ensure a more transparent and simple resource management we are working on several fronts that will be present on our architecture:

- *Kuryr:* Network speed up by better integrating OKD on top of OpenStack cloud deployments. Working on Kuryr OpenStack upstream project to integrate OpenShift SDN networking into OpenStack SDN networking, simplifying the operations, as well as achieving remarkable performance boost (up to 9x better). By using Kuryr at the OKD level we connect the containers directly into the OpenStack networks, instead of having 2 different SDNs and the performance problem of double encapsulation.

- *Kernel Driver*: New (NVMe) Kernel driver that speeds up access to NVMe devices from VMs without guest image modification, achieving up to 95% of native performance – compare to standard 30% with existing VirtIO drivers.
- *Network Policies:* Network Management through declarative API. As part of the Kuryr upstream work, we have also extended its functionality to support Kubernetes Network Policies, which allows user to define the access control to the different components of their applications in a fine grained manner. These policies are defined in a declarative way, i.e., by stating the desired status, rather than the steps to accomplish it. Then Kuryr will make sure that the isolation level desired at the OKD (containers) level is translated and enforced through OpenStack Security Group rules.
- *Operators:* Development of operators for easy life cycle management of infrastructure and applications. In addition to the performance improvements, we are also pursuing the use of the operators design pattern. This entails the use and development of certain operators (containers) which have their business logic integrated and react to the current status of the system/applications until they match the desired status. This helps to install the applications in an easy/reproducible manners, as well as to deal with day two operations, such as scaling or upgrades. In this regard we are working on a Kuryr SDN operator that allows easy installation and scaling of OKD cluster on top of OpenStack environments. This network operator takes care of creating everything needed on the OpenStack side, as well as installing anything required by Kuryr both at the initial deployment time and upon OKD cluster scaling actions. Another example of operators being used are the Spark Operator and the Cluster Monitoring Operator
- *Infrastructure API:* Unified API for infrastructure resources to make infrastructure management easy, and abstracted from the real infrastructure. To achieve this, the upstream community created the Kubernetes Cluster API project. We have been working on the support for the OpenStack abstraction together with its operator/actuator: Cluster API Provider OpenStack. This allows us to automate the creation/scaling actions regarding OKD nodes when running on top of OpenStack too. Thus, we can easily extend an OKD cluster as needed, just by modifying an object in *Kubernetes/OKD:* Similarly, this give us further advantages regarding resource management, e.g., if any of the VMs where our OKD is running dies (or the physical server that has it dies), the developed operator/actuator will automatically recreate the needed VMs in a different compute node, automatically recovering the system until it maps the desired status.

Note that while the first two points are related to infrastructure performance, the later 3 are key points for managing infrastructure as code, as well as to enable easy configuration/adaptation by upper layers, such as the Data-Driver Network Management or the Deployment Orchestration components.

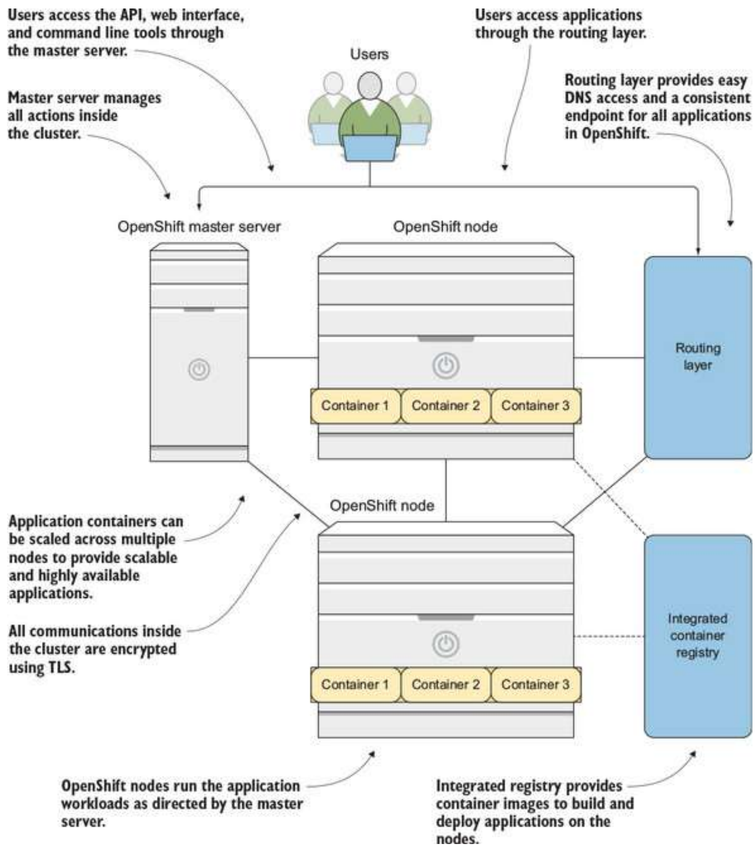


Figure 9 - OKD architecture overview in the users operations

6.2. Data-Driven Network Management

The Data-Driven Network Management component will efficiently handle network management and routing introspection, computing and storage resources, by collectively building intelligence through analytics capabilities. The motivation is to optimise computing and storage mechanisms to improve network performance. This component can obtain data from different BigDataStack layers (i.e. from storage layer to applications layer) and will be used to extract knowledge out of the large volumes of data to facilitate intelligent decision making and what-if analysis. For example, with big data analysis, the data-driven network management will know which storage or computing resource has high popularity. Based on the analysis result, the component will be able to produce insights on how to redistribute storage and/or computing resources to reduce network latency, improve throughput and satisfy access load and thus response time.

Monitoring mechanisms over the storage layer will provide information to adjust the network parameters (e.g. by enforcing policies to achieve a significant reduction in data retrieval and response time). Also, monitoring mechanisms over the computing layer will enable the development of functionalities and trigger policies that will satisfy users' requirements regarding runtime and performance.

To serve data-driven network management, we will analyse the data coming from storage and computing resources within a workflow which is depicted in Figure 10. The workflow is composed of three components namely: *ingest*, which consumes network data, *process*, which computes network metrics and *analyse*, which produces network insights. The lifecycle of the analysis task includes a set of algorithms which enable computational analytics over the data, conduct a set of control mechanisms and infer knowledge related to resources optimisation. Taking advantage of data-driven network management, big data applications will be able to access the global network view and programmatically implement strategies to leverage the full potential of the physical storage and computing resources.

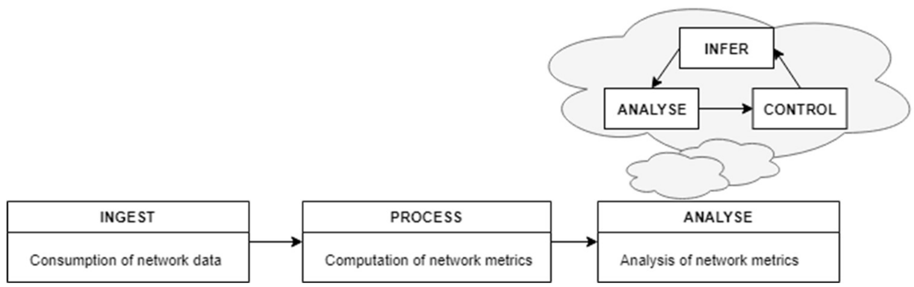


Figure 10 - Data-Driven Network Management components

6.3. Dynamic Orchestrator

The Dynamic Orchestrator (DO) assures that scheduled applications conform to their Service Level Objectives (SLOs). Such SLOs reflect Quality of Service (QoS) parameters and might be related to throughput, latency, cost or accuracy targets of the application. For example, to generate recommendations for online customers of an e-commerce website, the recommender has to analyse the customer profile and provide the recommendation in a limited amount of time (e.g., 1 sec.), otherwise, the page load will be too slow and customers might leave the website. If the number of online customers increases, then the recommender will need to improve its recommendations throughput in order to keep up serving the recommendations in less than 1 second. The DO will then modify the deployment in order to improve throughput, so that the recommender does not violate the corresponding SLO.
The DO assures conformation to SLOs by applying various dynamic optimisation techniques throughout the runtime of an application at multiple layers across various components of the data-driven infrastructure management system. As such, the DO knows about the adaptation actions that can be carried out for an application and when these actions should be carried out, i.e. what actions will affect each SLO.

Figure 11 depicts the high-level interactions of the dynamic orchestrator with other components. Newly scheduled applications are deployed through the Application and Data Service Ranking component (ADS-Ranking).[1] The ADS-Ranking scores possible deployment patterns/configurations (CDPs) and selects the one which it predicts to best satisfy the SLOs. After an application is deployed, the DO monitors its performance through the triple monitoring engine. In case there are SLO violations, the QoS component sends a message with the violation to the DO, which has two choices: (i) Initiate a re-deployment of the application through ADS (this choice will be made when SLOs can only be reached with *major*

[1] ADS-Ranking is also sometimes referred to as the *Deployment Recommender*, as in many scenarios its practical application is to recommend a deployment configuration for the user.

*deployment changes***,** e.g., selecting another ADS ranking option), (ii) Performing more *fine-grained adaptations* at different components of the system (e.g., the DO might perform "small" changes in the deployment configuration such as the number of replicas).

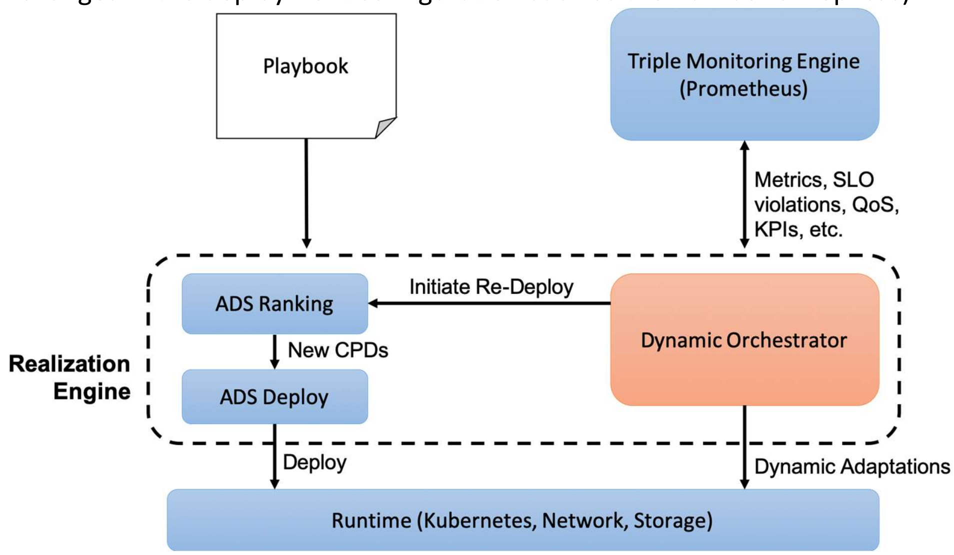


Figure 11 - High-Level Interaction with other Components

Note, that each of the other components also have their internal control loop and their internal logic for performing (high-responsive) actions, independently of the orchestrator or any of the other components. The primary challenge of the dynamic orchestrator is to reach a (close-to) optimal adaptation decision quickly, i.e., with a small overhead. This is a difficult goal, because application tasks will be distributed and adaptation can be achieved at different components (application, platform, network). The relationship between an adaptation technique and how it affects an SLO is not clear in advance and two adaptation techniques at different components might lead both to conformation of an SLO. Likewise, two adaptations at two components, might also conflict with each other. As such, the main challenges of the dynamic orchestrator are:

- Conflicting adaptations in different components
- Overhead for adaptation decisions
- Optimal adaptation

The orchestration logic itself is not implemented using hardcoded rules, but instead, uses Reinforcement Learning (RL). RL allows the DO to dynamically change its adaptation logic over time based on the outcome (feedback) from previous decisions. In RL, this means that the orchestration problem is broken down into:

- *States:* These are system and application metrics (e.g. CPU usage and throughput) and the current and past SLOs fulfillment.
- *Actions:* These change in deployment (e.g. add/remove a replica).
- *Reward:* The reward value is positive and proportional to resource utilization (to avoid underutilization) if SLOs are met, negative otherwise.

Figure 12 depicts a more detailed view of the dynamic orchestrator. Each application has its own BigDataStack application, RL Agent and RL Environment; while the Manager is unique for all applications. The Manager is in charge of the communication with the other components, receiving the Playbook, receiving the metrics and passing them to the corresponding BigDataStack application, and receiving the action to be taken from the RL Agent, and sending it to the ADS-Ranking or the platform for performing dynamic adaptations.

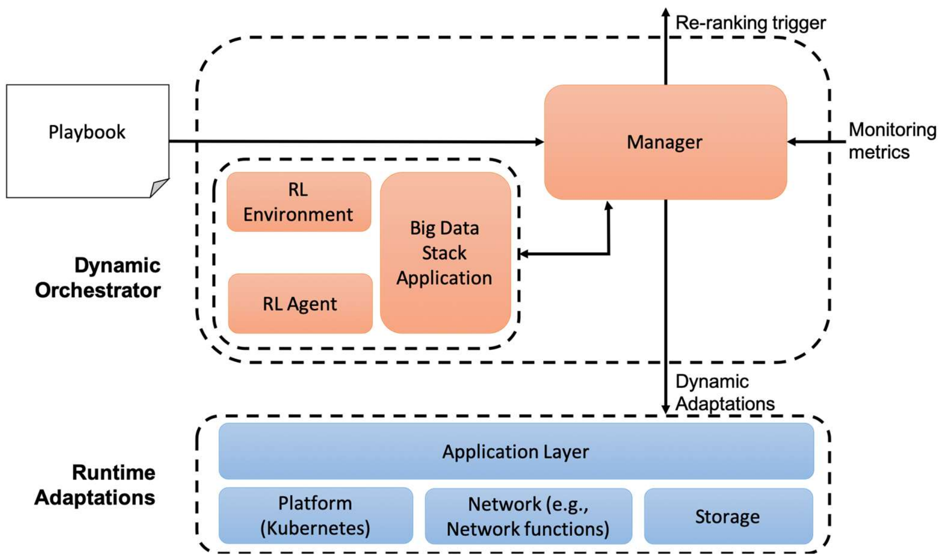


Figure 12 - Dynamic Orchestrator Detailed View

Moreover, Figure 13 depicts the different classes of the DO. Their inner working, step by step, is the following:

1. The Manager handles the communication with all the other components, using RabbitMQ and creates one instance of BigDataStackApplication for each application to be monitored.
2. The BigDataStackApplication creates the RLEnvironment, with its actions and state spaces, and the RLAgent that will be in charge of learning and deciding the best adaptation actions to take when an SLO is violated.
3. Each time a new message comes in, the Manager sends the information to the corresponding BigDataStackApplication, which updates the RLEnvironment state.
4. If a message with an SLO violation comes in, the Manager triggers the RLAgent, to decide which action should be taken according to the current RLEnvironment state.
5. Then, the Manager sends a message to the ADS-Ranking requesting the identification of a new deployment configuration or to ADS-Deploy to directly change the deployment.

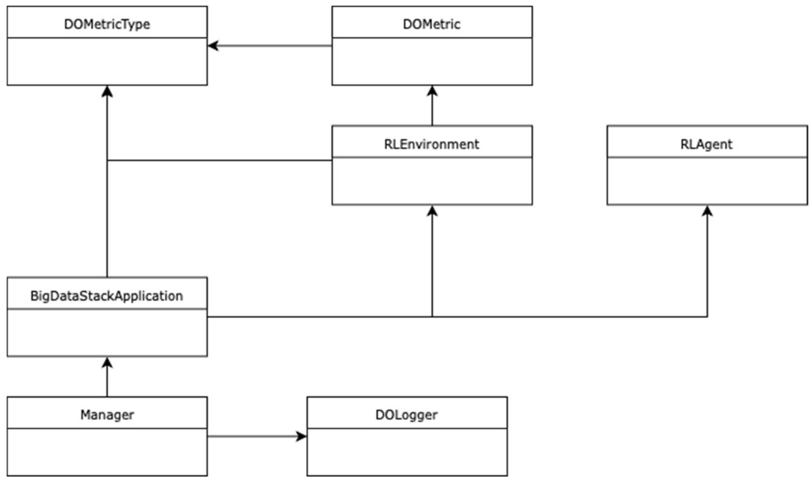


Figure 13 - High-level class diagram of the Dynamic Orchestrator

6.4. Triple Monitoring and QoS Evaluation

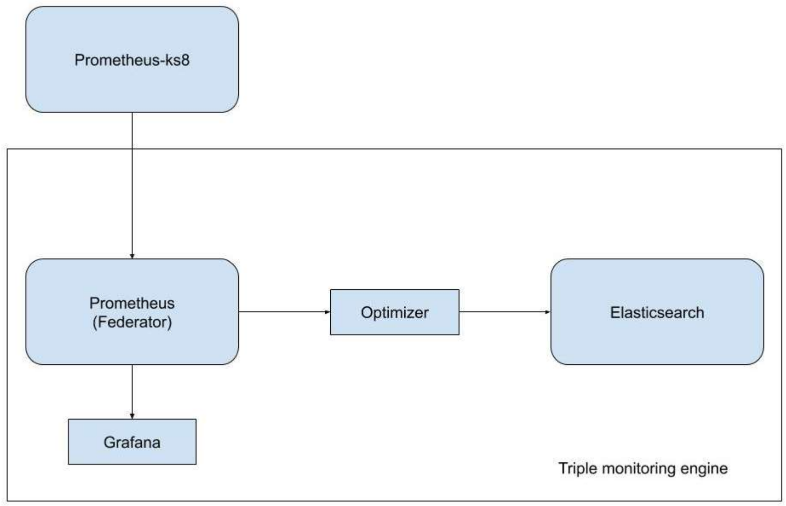
The *Triple Monitoring* and *QoS Evaluation* are two closely related components with clearly separated responsibilities:

- The objective of the Triple Monitoring is to collect, store and serve metrics at three levels of the platform: application, data services and infrastructure (cluster) resources.
- The goal of the QoS Evaluation is to continuously evaluate those metrics against constraints (thresholds) or objectives imposed by certain BigDataStack platform users.

6.4.1. Triple Monitoring

The monitoring engine manages and correlates/aggregates monitoring data from different levels to provide a better analysis of the environment, the application and data; allowing the orchestrator to take informed decisions in the adaptation engine. The engine collects data from three different sources:

- Infrastructure resources of the compute clusters such as resource utilisation (CPU, RAM, services and nodes), availability of the hosts, data sources generation rates and windows. This information allows the taking of decisions at a low level. These metrics are directly provided by the infrastructure owner or through specific probes, which track the quality of the available infrastructures. In the context of bigdatastack, the infrastructure's metrics are collected by Kubernetes. Those metrics will be ingested to the triple monitoring engine by federating Prometheus instances.



- Application components such as application metrics, data flows across application components, availability of the applications etc. This information is related directly to the data-driven services, which are deployed in the infrastructure. These metrics are associated with each application, and they should be provided by those applications. For application related to BigDataStack infrastructure, the most suitable method is to embed Prometheus exporter to each of those applications. Use case application will be sending metrics via a http method for flexibility reason.

- Data functions/operations such as data analytics, query progress tracking, storage distribution, etc. This is a mix of data and storage infrastructure information providing additional information for the “data-oriented” infrastructure resources.

The component will cover both raw metrics (direct measurements provided by the infrastructure deployed sensors or external measurement systems like the status of infrastructure) and aggregated metrics (formulas to exploit metrics already collected and produce the respective aggregated measurements that can be more easily used for QoS tracking). The collection of metrics will be based on both solutions: the direct probes in the system that should be monitored and the direct collection of the data from the monitoring engine.

- The probe approach will cover the information systems, where the platform will be able to deploy and collect direct information. In this case, the orchestration engine must manage the deployment of the necessary probes. This approach can cover other cases, where the probe is included directly in the application, and the orchestration only needs to deploy the associated application, which can provide the metric information to the monitoring engine.
- The direct collection will cover the scenarios where the platform cannot deploy any probe, but the infrastructures or the applications expose some information regarding these metrics. In this case, the monitoring engine will be responsible for collecting the metrics data that are exposed by a third party via a REST_API (Exporter).

After collecting and processing the data, the monitoring engine will be responsible for notifying other components when an event happens based on the metrics that it is tracking and specific attributes such as computing, network, storage or application level. Moreover, it will expose an interface to manage and query the content. This functionality is implemented in the QoS Evaluator (SLA Manager). Figure 14 depicts the Triple Monitoring Engine and their components.

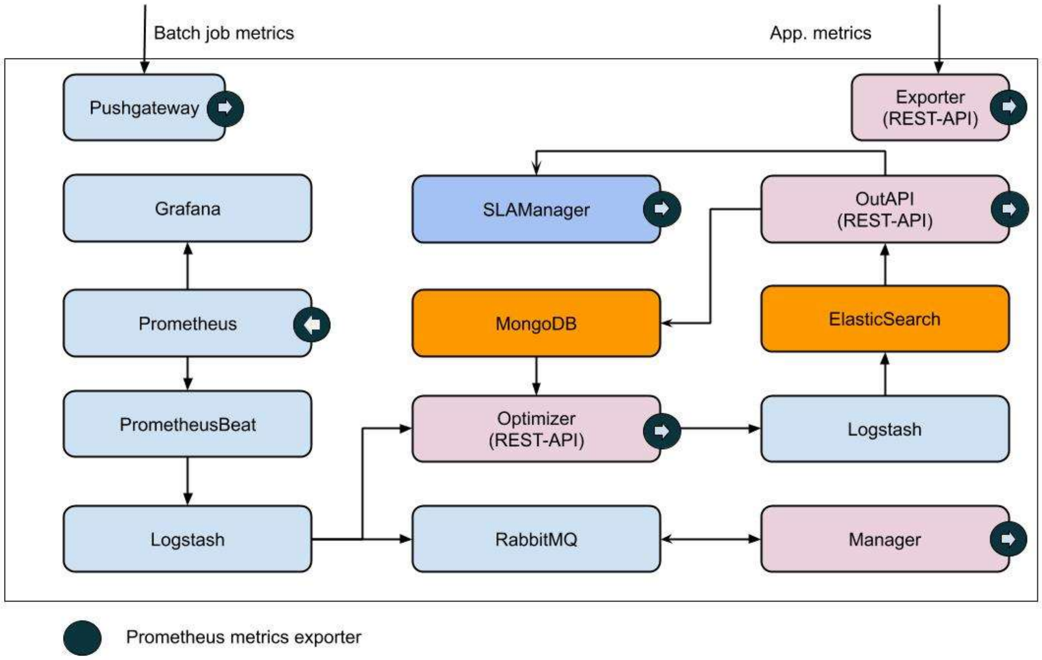


Figure 14 - Triple Monitoring Engine architecture diagram

The Triple Monitoring Engine will be based on the Prometheus monitoring solution (see [9] for more details) and is composed of the following components:

- Monitoring Interface: This is responsible for exposing the interface to allow other components to communicate. The interface will manage two ways of interaction with other components: i) exposing a REST API (outAPI, Figure 14) that will enable other components to know specific information, for example, if another component wants to know more details about one violation, to take the correct decision, or if they need to configure new metrics to collect directly by the monitoring engine. Therefore, the interface will consist of both a REST interface and a publish/subscribe notification interface. The publish/subscribe mechanism is implemented with RabbitMQ. This allows any components to consume in real-time information.
- Monitoring Manager: This component handles subscriptions by storing the queue, the list of metrics and metadata related to the subscription. The manager consumes all metrics collected by Prometheus. Based on the subscriptions list, they are redirected to the component subscribed by the queue declared.
- Monitoring Databases: ElasticSearch is currently used as the metrics database. MongoDB is also used to store all metrics requested via the outAPI in order to keep a track of metrics' utilization.

cc BY NC ND

- PrometheusBeat: Since Prometheus has a small retention period, BigDataStack optimization loops in various components (e.g. deployment patterns generation) raised the need for a solution that would allow accessing and holding the collected metrics. To this end, this component receives the metrics collected by Prometheus, and ingests them to a pipeline (Logstash) for being stored.
- Optimizer: Since the Triple Monitoring Engine of BigDataStack collects monitoring data from different sources and all those data are utilized at specific time periods by different BigDataStack architecture components, storage optimization is required. Based on the information stored in the MongoDB (metrics utilization) this component decides about the time period for which the monitoring data should be kept.
- Push gateway: The push gateway is a Prometheus exporter. It is used in BigDataStack specially for collecting monitoring data obtained after each Spark driver execution.
- Collector Layer: This component is responsible for obtaining the data to be moved to the Monitoring manager. There are two ways to collect the data, either through a probe or through direct collection:
    - Probe API exposes an interface to allow different kinds of probes to send the monitoring data to the monitoring engine.
    - Direct collection is realized through a component that collects directly the monitoring data, by invoking other systems or components. For example, it receives the data directly from the Resource management engine or invoke the third-party libraries to obtain the state of the application and data services.

Integration with resource management engines

The Triple Monitoring Engine provides APIs for receiving metrics from different sources (infrastructure, application and data services) and expose them for consumption. Although different APIs will be available due to the great diversity of monitoring data sources, the recommended API is the "Prometheus exporters" model. Some of the technologies that are being considered for BigDataStack are already integrated within Prometheus, as shown in Table 2.

Technology component	Monitoring aspect	Prometheus exporter availability	Method
Kubernetes	Computing infrastructure	Yes	Federation
OpenStack	Computing infrastructure	Yes	Exporter
Spark/Spark SQL	Data functions/operations	Yes	Exporter (SparkMeasure)
IBM COS (Cloud Object Store)	Data infrastructure	No	
LeanXcale database	Data infrastructure	For some metrics	Federation
CEP	Data Infrastructure	Yes	Federation

Table 2 - Prometheus integration

Federation of Prometheus instances

Federation is used to pull monitoring data from another Prometheus instance. This model is introduced in the BigDataStack Triple Monitoring Engine for two main reasons. Firstly, the platform uses Kubernetes as containers orchestrator, which embedded by default a

Prometheus (prometheus-ks8) instance. This instance collects monitoring data related to the cluster, nodes and services running. For security reasons it is not efficient to use prometheus-k8s for collecting application- and data- related monitoring data. Secondly, the LeanXcale database and the CEP are independent systems and have their own Prometheus instances. For reusability reason and improvement (collect only monitoring data directly used by BigDataStack components) the proposed federation model is the most suitable method to achieve this requirement.

In the federation mode, the master instance should be configured appropriately by specifying the interval of time where metrics will be collected, the source job also if needed, the metrics to collect can be specified.

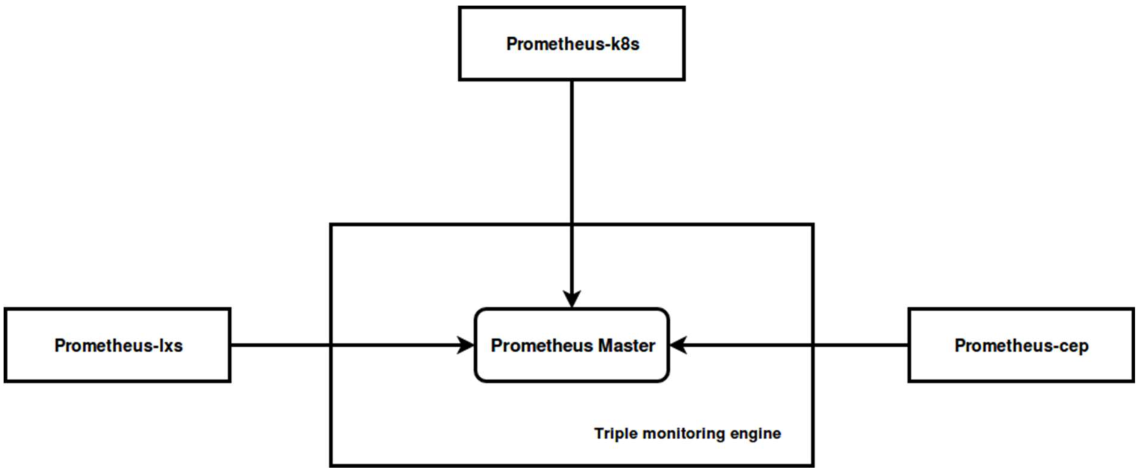


Figure 15 - Triple Monitoring Engine Federation Model

6.4.2. QoS Evaluation

The Quality of Service (QoS) Evaluation component is directly connected with the Triple Monitoring Engine to evaluate the quality of the application and data services deployed on the platform. To do so, it compares service metrics (key performance indicators) with the objectives set by the owner of the service and thus imposed over the BigDataStack platform when the service was deployed. The QoS Evaluation component is also responsible for notifying if the quality objectives are not met by the running the service. Therefore, the component is not responsible for obtaining the metrics (delegated to the monitoring engine) but to apply evaluation rules upon those metrics and notify when quality failures occur.

The main entities within the QoS Evaluation are the following:

- *Agreement*: it is a description of the QoS evaluation task to be carried out by the QoS Evaluation. It describes the creation and expiration time of the task, the provider and consumer of the application or service whose quality needs to be guaranteed, and the list of QoS constraints or guarantees to be evaluated.
- *SLO* (Service Level Objective) or QoS *guarantee*: it is a set of thresholds for the value of a given metric, representing increasing levels of criticality. The last threshold is always the last limit or final objective to be meet. The other thresholds are used as checkpoints to better understand and control the dynamics of the indicator. The SLO belongs to the agreement.

- *Violation*: it is generated when the value of a the QoS metric trespasses any of the SLO thresholds. The QoS Evaluation component notifies each violation to other components of the platform subscribed to the event; perhaps the most important of the subscribers is the Dynamic Orchestrator, which is responsible for the service deployment adaptation decisions.

The QoS Evaluation is made of the following components:

- *Interface component* (REST API): through this interface the consumers of the QoS evaluation service can start/stop the evaluation of certain application metrics.
- *QoS database:* it is responsible for storing all the content agreements, violation, service level objectives. This will be stored in the Global Decision Tracker.
- *Evaluator:* it is responsible for performing QoS evaluation. A periodic thread is started to check the expiration date of agreements. For each enabled agreement, it starts a task to check agreement evaluation by getting needed metrics from the adapter. The task is also started when metrics are received from the Notifier.
- *Adapter:* it is responsible for calling the monitoring system to obtain the metrics data. It will be different for each monitoring system, so it will be accountable for building the specific request to the Triple Monitoring System to gather and transform metrics to have them ready to compare with SLOs by the Evaluator.
- *Notifier:* It is responsible for notifying to third parties that want to be alerted if something happens in the defined agreements, such that corrective actions can be taken.

In the BigDataStack platform, application and data services QoS constraints (objectives are specified by the ***Data Scientist*** trough the ***Data Toolkit*** (see Section 6.13) together with the rest of information describing the application to be deployed. This is compiled in the so-called application *playbook*, which serves as the specification for the BigDataStack platform to deploy and operate the application. The following table shows and example of QoS constraints imposed over the response time of an online service called “recommendation-provider”. Notice the Data Scientist can specify not only required response times but also recommended response time[2]:

```
- name: recommendation-provider
  metadata:
    qosRequirements:
    - name: "response_time"
      type: "maximum"
      typeLimit: null
      value: 900
      higherIsBetter: false
      unit: "miliseconds"
    qosPreferences:
    - name: " response_time"
      type: "maximum"
      typeLimit: null
      value: 300
      higherIsBetter: false
      unit: "miliseconds"
```

2 Notice this is an extract of the *playbook* showing just one of the QoS constraints imposed on one service.

When a service deployment is requested, The Dynamic Orchestrator (i.e. the component in charge of making deployment adaptation decisions to satisfy QoS constraints) breaks down the QoS objective into thresholds of increasing levels of criticality. Depending on the nature of the QoS metric (indicator) to control and both the recommended and required values, the Dynamic Orchestrator may produce an arbitrary number of thresholds between the fist (related to recommended value) and last (related to the required value) thresholds.

With every deployment, the Dynamic Orchestrator will request the QoS Evaluation component to create/start a task to continuously compare the service performance metric against those thresholds. This request is made asynchronously through a messages queue. This is implemented as topic within the RabbitMQ service (which acts as the message broker between BigDataStack components). In the previous example, the Dynamic Orchestrator may send the following message to the QoS Evaluation[3]:

```
"qosIntervals": {
  "reponse_time": [
    ">300",
    ">500",
    ">700",
    ">900"
  ]
}
```

The QoS Evaluation component incorporates the thresholds or intervals to be monitored (requested by the Dynamic Orchestrator) as a *guarantee* object in the *agreement* for the actual service deployment. In that way, all QoS constraints to be evaluated and guaranteed for the same service deployment are maintained together. In the previous example, the *agreement* and *guarantee* created from the Dynamic Orchestrator request may be like the following:

```
{
  "id": "TEST-ATOSWL-NormServ-19022019-1",
  "name": "TEST-ATOSWL-NormServ-19022019-1_agreement",
  "details": {
    "id": "TEST-ATOSWL-NormServ-19022019-1",
    "type": "agreement",
    "name": "TEST-ATOSWL-NormServ-19022019-1_agreement",
    "provider": {
      "id": "a-provider-01",
      "name": "ATOS Wordline"
    },
    "client": {
      "id": "a-client-01",
      "name": "Eroski"
    },
    "creation": "2019-05-30T07:59:27Z",
    "expiration": "2020-01-17T17:09:45Z",
    "guarantees": [
    {
```

[3] Notice this is an extract of the *enhanced playbook* showing the QoS thresholds (intervals) for the evaluation of just one of the metrics (indicators) of one service.

CC BY NC ND

```
      "name": "response_time",
      "constraint": "[response_time>50]",
      "importance": [
      {
        "Name": "0",
        "Type": "warning",
        "Constraint": ">300"
      },
      {
        "Name": "1",
        "Type": "warning 2",
        "Constraint": ">500"
      },
      {
        "Name": "2",
        "Type": "warning 3",
        "Constraint": ">700"
      },
      {
        "Name": "3",
        "Type": "error",
        "Constraint": ">900"
      }
    ]}
  ]}
}
```


The QoS Evaluation will continuously assess the value of all guaranteed QoS attributes (metrics or indicators) and detect violations, that is, when the value trespasses the different thresholds that have been specified. QoS violations are notified to any interested component of the BigDataStack platform through a publisher/subscriber mechanism implemented as topic within the RabbitMQ service (which acts as the message broker between BigDataStack components). Following the previous example, the following violation notifications may be published[4]:


```
{
  "Application": "TEST-ATOSWL-NormServ",
  "Message: "QoS_Violation",
  "Fields": {
    "IdAggrement": "TEST-ATOSWL-NormServ-19022019-1",
    "Guarantee": "response_time",
    "Value": "351",
    "ViolationType: {
      "Type": "warning",
      "Interval": "0"
    },
    "ViolationTime": {
      "ViolationDetected": "2019-06-30T07:59:27Z",
      "AppExpiration": "2020-01-17T17:09:45Z"
    }
  }
}
```

```
{
  "Application": "TEST-ATOSWL-NormServ",
```


[4] Notice that the first violation notification example is that of the lowest level of criticality (meaning a simple warning) while the second example if that of the highest criticality (meaning an error).

```
   "Message: "QoS_Violation",
   "Fields": {
     "IdAggrement": "TEST-ATOSWL-NormServ-19022019-1",
     "Guarantee": "response_time",
     "Value": "920",
     "ViolationType: {
       "Type": "error",
       "Interval": "3"
     },
     "ViolationTime": {
       "ViolationDetected": "2019-06-30T09:34:21Z",
       "AppExpiration": "2020-01-17T17:09:45Z"
     }
   }
 }
```


Perhaps the most important of the subscribers is the Dynamic Orchestrator itself, which will respond to different violation alerts depending on the criticality of the threshold trespassed.

The QoS Evaluation displays the warning (lowest criticality) and error (highest criticality) thresholds on the interface of the Triple Monitoring Engine, superimposed to the metrics evolution graphs to which apply. The following figure is an example of the *Response Time* evolution graph on the Triple Monitoring Engine.

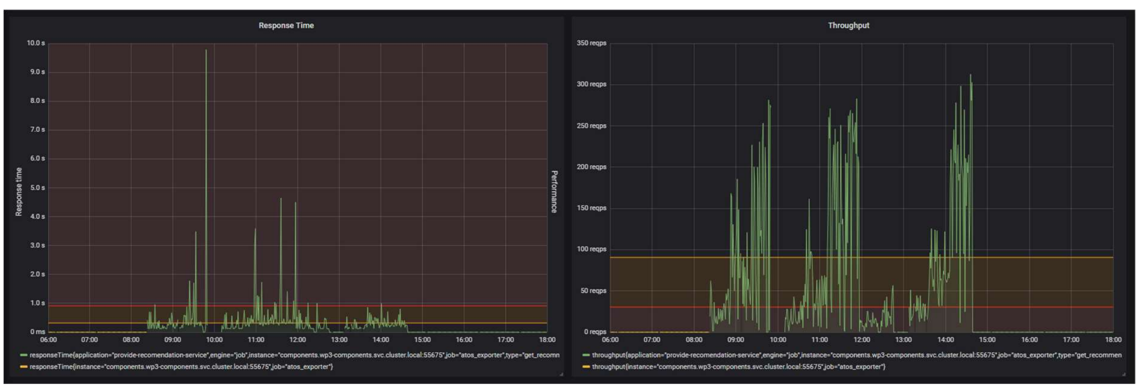


Figure 16 - SLO guarantees thresholds shown over the *Response Time* (left) and Throughput (right) metrics graphs: warning (lowest criticality) and error (highest criticality) thresholds as orange and red lines, respectively.

6.5. Applications & Data Services Ranking / Deployment

Application and Data Services Ranking/Deployment is a top-level component of the BigDataStack platform, as defined in the central architecture diagram (see Section 5). It belongs within the realisation engine of the platform and is concerned with how best to deploy the user's application to the cloud, based on information about the application and cluster characteristics. From a practical perspective, its role is to identify which - of a range of potential deployment options - is the best for the current user, given their stated (hard) requirements and other desirable characteristics (e.g. low cost or high throughput), as well as operationalize the deployment of the user's application based on the selected option.

In practice, the Application and Data Services Ranking/Deployment is divided into three main sub-components, namely: the main component ADS-Ranking; and two support components ADS-Deploy and ADS-GDT, which we describe in more detail below:

- *Application and Data Services Ranking* (ADS-Ranking): This is dedicated to the selection of the best deployment option. Note that this component is sometimes referred to as the 'deployment recommender service', as from the perspective of a BigDataStack Application Engineer, it produces a recommended deployment for them on-demand.
- *Application and Data Services Deployment* (ADS-Deploy): This is concerned with the physical scheduling/deployment of the application for the selected deployment option via Openshift.
- *Application and Data Services Global Decision Tracker* (ADS-GDT): This stores information about the state of different applications and decision made about them.

Application and Data Services Ranking (ADS-Ranking)

ADS-Ranking is tightly coupled to the Application & Data Services Dimensioning (ADS-Dimensioning) component of BigDataStack that sits above it. The main output of ADS-Dimensioning is a series of candidate deployment patterns (ways that the user's application might be deployed) including resource usage and quality of service predictions. It is these deployment patterns that ADS-Ranking takes as input (see REQ-ADSR-01 [10]) and subsequently selects one or more 'good' options for the Application Engineer. Each candidate deployment pattern represents a possible configuration for one 'Pod' in the user's application (a logical grouping of containers, forming a micro-service) [11]. User applications may contain multiple pods.

Communication to and from ADS-Ranking is handled via the Publisher-Subscriber design pattern. In this case, 'messages' are sent between components, which trigger processing on the receiving component. More precisely, ADS-Ranking subscribes to the ADS-Dimensioning component to receive packages of pod-level candidate deployment patterns (CDPs), one package per-pod in the application to deploy. On-receive, this triggers the ranking of the provided deployment patterns, as well as the filtering out of patterns that either do not meet the user's requirements, or that are otherwise predicted to provide unacceptable performance. After ranking/filtering is complete, ADS-Ranking will select a single deployment pattern per-pod to send to the BigDataStack Adaptive Visualisation Environment. Within this environment, the user can either choose to deploy their application using the recommended patterns directly, customise the patterns and then deploy, or otherwise cancel the deployment process. Upon choosing to deploy with a set of patterns, those patterns are sent to ADS-Deploy for physical scheduling on the available hardware.

Figure 17 illustrates the data flow between the components around ADS-Ranking. As we can see, ADS-Dimensioning first gets information about the user's application and preferences from a BigDataStack Playbook and uses it to produce packages of candidate deployment patterns (CDPs). Each CDP represents a deployment configuration that we could use to deploy the user's application pod (where some CDPs will produce more efficient or effective deployments than others). These pattern packages are sent as messages to ADS-Ranking, which ranks and filters those patterns, finally selecting one per-pod, which is predicted to efficiently and effectively satisfy the user's requirements. These top patterns are aggregated, then placed in a message envelope and sent back to the BigDataStack Adaptive Visualisation Environment, where the application engineer can accept those patterns and use them directly for deployment, or otherwise customise them first. Once the application engineer is happy with the deployment, they can then send the final patterns via the visualisation environment to ADS-Deploy, which will schedule deployment on OpenShift.

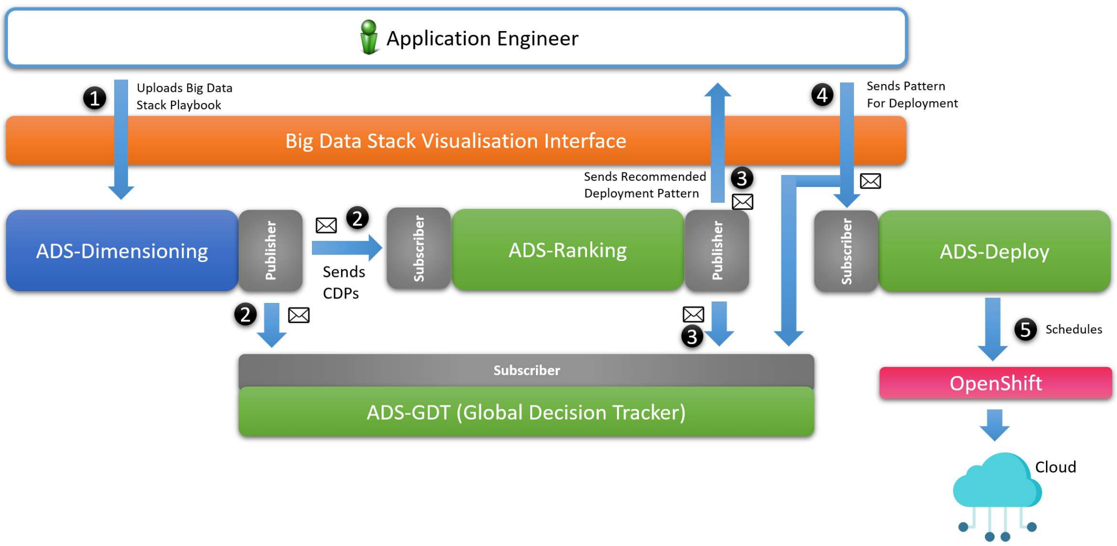


Figure 17 - Process Flow for ADS Ranking/Deploy during First Time Deployment

Internally, ADS-Ranking supports two central operations: 1) the first-time ranking/filtering of CDPs; and 2) re-ranking of CDPs in scenarios where the previous deployment is deemed unsuitable. The first operation (CDP ranking and filtering) is comprised of three main processes. These three processes are:

- *Pod Feature Builder:* This takes as input a set of CDPs, and for each CDP in that package, it builds a single vector representation of that CDP, which combines all the information provided by dimensioning. It can also filter out CDPs that do not meet minimal Quality of Service (QoS) requirements, saving computation time later in the process. The output of this component is the (filtered) list of CDPs along with their new vector representations. This process targets REQ-ADSR-02 [10].
- *Pod Scoring:* This process takes the CDPs and vector representations as input and ranks those CDPs based on their predicted suitability, with respect to the user's desired quality of service. To achieve this, it uses either a rule-based model or a supervised model [12] trained on previous CDP deployments and their observed fitness. The output of this process is a ranking of scored CDPs. This process targets REQ-ADSR-03 and 04 [10].
- *Pod Selection:* This process takes as input the ranking of CDPs and selects one of these CDPs. This may be a simple process that takes the top CDP and filters out the rest. However, it may include more advanced techniques to better fit with user needs, such as making sure the selected CDP will provide sufficient extra processing capacity, in the case of applications that process data streams with fluctuating data rates. The

output of this process is a single CDP (per-pod), which is the recommended deployment that is shown to the user. This process targets REQ-ADSR-05 [10].

If the user's application is comprised of multiple pods, then the recommended CDP for each pod are then collected and aggregated together to form a recommendation for the entire application. The aforementioned processes are implemented using Apache Flink [13] to facilitate low-latency real-time processing. The overall flow for first-time ranking/filtering of CDPs is shown in Figure 18. In this simplified example, three CDPs are used as input for a single application (A1), which is comprised of two pods (P1 and P2). Pod 1 has two CDPs (A1-P1-1 and A1-P1-2), while Pod 2 has one CDP (A1-P2-1). As we can see from Figure 18, these CDPs are first grouped by pod, to create parallel processing streams for each. For each CDP, these are then subject to feature extraction, to create the representation vectors. In this case, features from the overall pod (e.g. total cost) and features from each container (e.g. container latency) are extracted here. These CDPs and feature vectors are sent to pod scoring, to produce a numerical estimate of overall suitability of the CDP. The best CDP per-pod (A1-P2-2 and A1-P2-1 here) are then grouped by application (A1) and then output (to the visualisation environment for viewing by the application engineer).

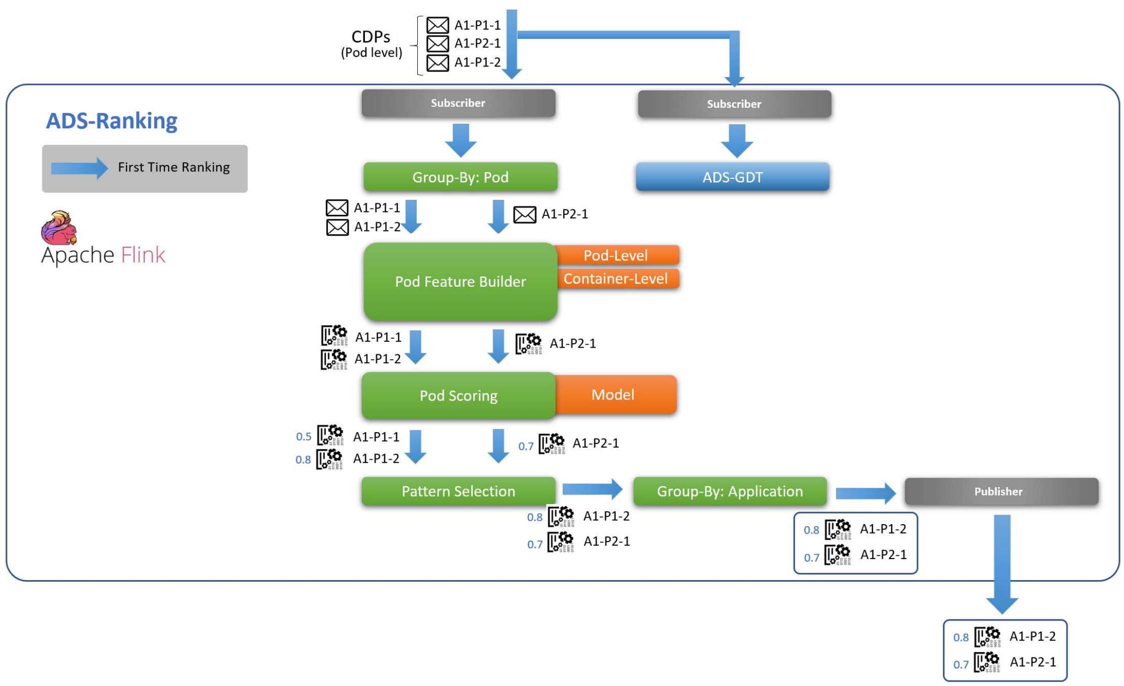


Figure 18 - ADS-Ranking, First Time Deployment Internal Process Flow

The second function (CDP Re-Ranking) is similar to the primary function, with the exception that it takes in a CDP that has been deemed to have failed the user in terms of quality of service along with context about that CDP (e.g. why it failed), and it introduces an additional 'Failure Encoding' process:

- *Failure Encoding:* This process examines the context of a failed CDP and encodes that failure into the CDP structure as features, such that they can be used by the Pod Feature Builder when generating the CDP vectors. In this way, properties that promote other CDPs that will not suffer from the same issues as the failed CDP can be upweighted during ranking. This process targets REQ-ADSR-07 [10].

Figure 19 illustrates the main processes and data flow within ADS-Ranking. In this case, re-ranking is triggered by sending a set of CDPs representing a quality of service (QoS) failing user application deployment to ADS-Ranking. For this example, the application has two pods and hence two CDPs (A1-P2-2 and A1-P1-1), where a QoS failure has been detected for A1-P1-2 (denoted by ✗). The first step that ADS-Ranking takes is to collect all the alternative CDPs that were not selected from the user's application. These were stored in ADS-GDT (Global Decision Tracker), which will be described later. Once these CDPs have been collected, any CDPs for pods that were not subject to QoS failures are discarded, as these do not need to be considered for re-deployment (A1-P2-1). The remaining CDPs are then subject to failure encoding, which converts the failure information into a feature vector that can be used during ranking (<x>). The CDPs are then sent to the Pod Feature Builder in a similar manner to first-time ranking, where the normal process is followed, with the exception that the additional features obtained from the failure encoding are used to enhance ranking effectiveness.

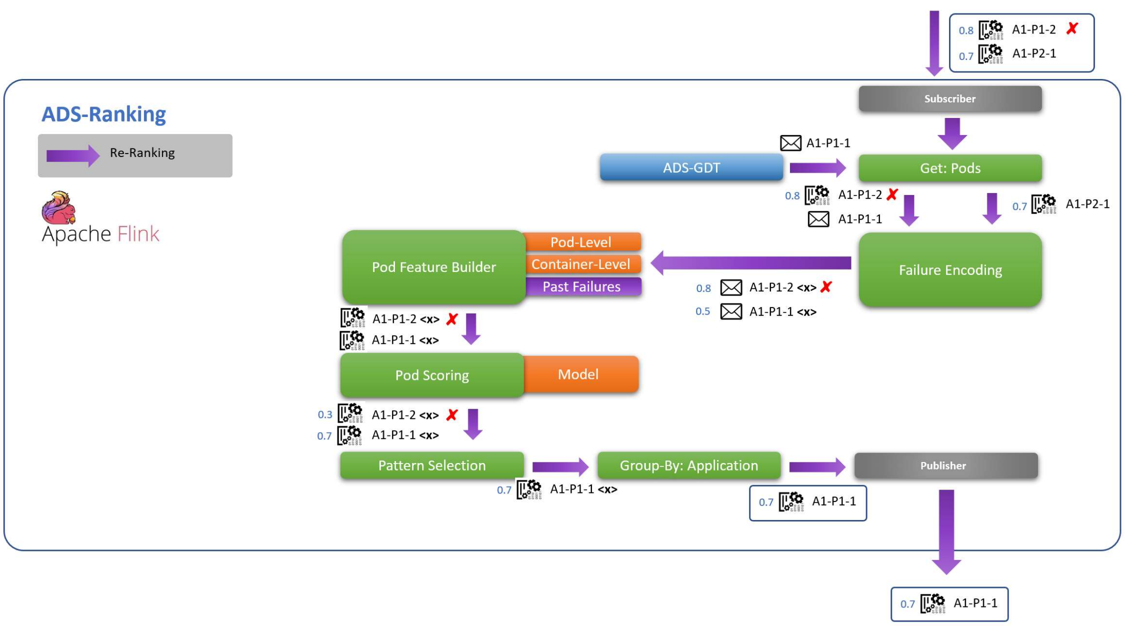


Figure 19 - ADS-Ranking, Re-Ranking Internal Process Flow

Application and Data Services Deployment (ADS-Deploy)

This process is triggered by the BigDataStack Adaptive Visualisation Environment and takes as an input the selected CDP(s). The aim of this component is two-fold. First, to use the given CDP(s) to launch the user's application pods on the cloud infrastructure. Second, to notify relevant BigDataStack components of the deployment status, such that follow-on processes (such as monitoring) can commence. To achieve this, the ADS-Deploy component interacts with a container orchestration service (e.g. OpenShift), translating the CDP into a sequence of deployment instructions.

This task is divided into the following steps:

bigdatastack.eu

1. *Receive and check CDP.* The component checks that the CDP triggering the deployment process is structurally correct.
2. *Translate CDP.* The CDP is translated to an ontology that the orchestrator will understand.
3. *Interpretation and deployment.* The orchestrator interprets the file received and starts the containers and rules.
4. *Communication with the user.* The result of the process (either success or fail) is communicated to the rest of the architecture (and ultimately, to the user) as an event by means of a publisher-subscriber model. The main subscribers to this event will be the Dynamic Orchestrator, ADS-GDT components, along with the BigDataStack Adaptive Visualisation Environment.

Application and Data Services Global Decision Tracker (ADS-GDT)
The role of the Global Decision Tracker is (as its name suggests) to keep track of any state or decisions made about a user's application related to its deployment or run-time performance. In effect, it is a data store that holds both the current configuration (BigDataStack Playbook and associated CDPs) for each deployed user application, along with relevant events generated by other components (e.g. ADS-Deploy reporting a successful deployment or the dynamic orchestrator reporting a quality of service failure).

Like the other ADS-* components, ADS-GDT uses the publisher-subscriber pattern to enable asynchronous one-to-many communication flows in a standardised and reliable manner. In this case, it subscribes to all the message queues that are relevant to deployment or application run-time activities and saves them within a local database. It also hosts a RESTful API service that provides bespoke access to the collected data for both BigDataStack services (e.g. ADS-Ranking during re-ranking) but also to the BigDataStack Adaptive Visualisation Environment, where application state information is needed for visualisation.

6.6. Data Quality Assessment

The data quality assessment mechanism aims at evaluating the quality of the data prior to any analysis on them to ensure that analytics outcomes are based on datasets of specific quality. To this end, BigDataStack architecture includes a component to assess the data quality. The component incorporates a set of algorithms to enable domain-agnostic error detection, in a given dataset. The domain-agnostic approach followed aims at facilitating the goals of data quality assessment without prior knowledge of the application domain / context, thus making it "generalised" and applicable to different application domains and as a result to different datasets. While current solutions in data cleaning are quite efficient when considering domain knowledge (for example in eHealth regarding the correlation between different measurements of different health parameters), they provide limited results regarding data volatility, if such knowledge is not utilised. BigDataStack will provide a data quality assessment service that exploits Artificial Neural Networks (ANN) and Deep Learning (DL) techniques, to extract latent features that correlate pairs of attributes of a given dataset and identify possible defects in it.

The key issues that need to be handled by the Data Quality Assessment service are:

- Work in a context-aware but domain-agnostic fashion. The process should be adaptable to any dataset, learn the relationships between the data points and discover possible inconsistencies.
- Model the relationships between data points and reuse the learned patterns. The system should store the models learned by the machine learning algorithms, and reuse them through an optimisation component, which checks if the raw data have similar patterns, dataset structure or sources. In that case, already existing models should be activated, to complete the process in an efficient manner.

The way to learn and predict the relationships between data points, to discover possible deviations, is to exploit the recent breakthroughs in Deep Learning, and the idea of an embedding space. Figure 20 depicts a serial architecture, which tries to predict if two entities are related to each other.

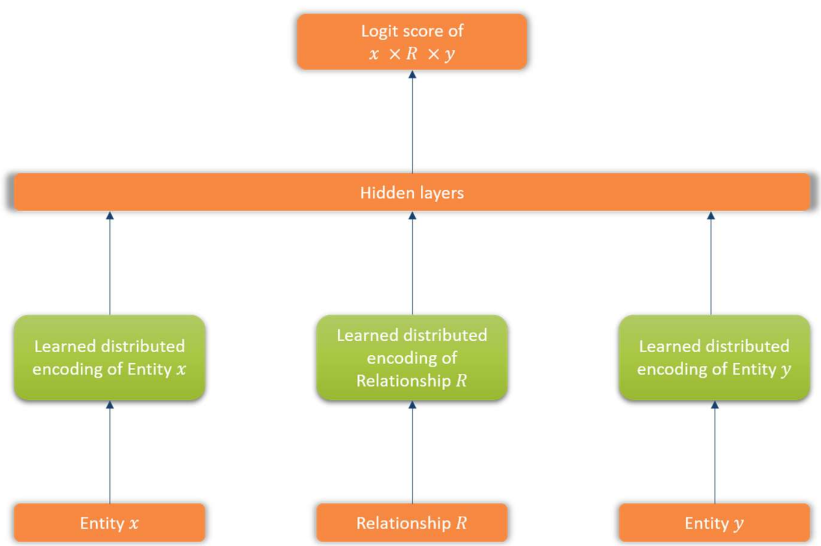


Figure 20 - Domain agnostic data cleaning model architecture

Given the learned distributed encodings of each entity x, y or, in our case any data point, we can discover if these two candidate entities or data points are related. Thus, considering the DANAOS use case, if the temperature sensor emits a value that is illogical given other rpm sensor readings, the relationship between these two data points would be associated with a low score (or probability). This could provide significant improvements in the results of an analytical task that the data scientist wants to execute, and is part of a general business process.

To optimize the data quality assessment process, we introduce a subcomponent that retrieves previously learned models, when a similar dataset structure arrives in the system, or the same data source sends new data.

Data quality assessment component inputs:

- The raw data ingested by the data owner through the Gateway & Unified API
- The data model provided by the optimizer if exists

- User preferences and specifications, ingested through the Data Toolkit

Data cleaning component outputs:

- Assessed data, establishing data veracity
    - A probability score for each tuple in the database column
- Trained, reusable ML models, stored in a repository for later use

The main structure of the Data Quality Assessment component is depicted in Figure 21. Based on this figure the flow is as follows:

- The Data Pre-processing unit takes raw data and converts them in a form that the machine learning algorithms can work with
- The main pillar of the service is the data cleaning component, which takes the pre-processed data as input, trains a new model and stores it in the model repository
- During the assessment phase, a scheduler pulls newly ingested data to be assessed
- The data quality assessment module retrieves the learned model from the repository and makes the necessary predictions
- The assessed data are updated into the distributed storage

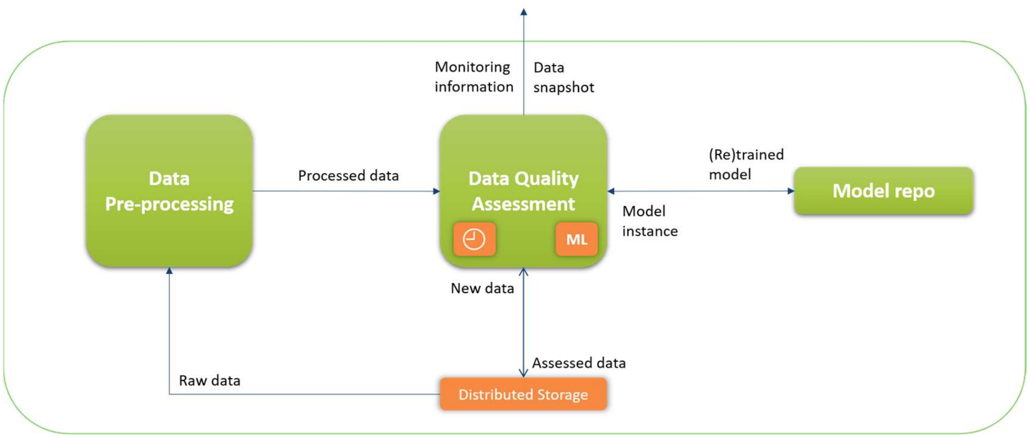


Figure 21 - Data Cleaning Module Architecture

6.7. Real-time CEP

Streaming engines are used for real-time analysis of data collected from heterogeneous data sources with very high rates. Given the amount of data to be processed in real-time (from thousands to millions of events per second), scalability is a fundamental feature for data streaming technologies. In the last decade, several data streaming systems have been released. StreamCloud [14], was the first system addressing the scalability problem allowing a parallel distributed processing of massive amount of collected data. Apache Storm [15] and later Apache Flink [13] followed the same path providing commercial solutions able to distribute and parallelise the data processing over several machines to increase the system throughput in terms of number of events processed per second. Apache Spark [16] added streaming capability onto their product later. Spark's approach is not purely streamed, it divides the data stream into a set of micro-batches and repeats the processing of these batches in a loop.

The complex event processing for the BigDataStack platform will be a scalable complex event processing (CEP) engine able to run in federated environments with heterogeneous devices with different capabilities and aggregate and correlate real-time events with structured and non-structured information stored in the BigDataStack data stores. The CEP will take into account the features of the hardware, the amount of data being produced and the bandwidth in order to deploy queries. The CEP will also consider redeploy and migrate queries if there are changes in the configuration, increase/decrease of data, changes in the number of queries running or failures.

Data enters the CEP engine as a continuous stream of events, and is processed by continuous queries. Continuous queries are modeled as an acyclic graph where nodes are streaming operators and edges are data streams connecting them. Streaming operators are computational units that perform operations over events from input streams and outputs resulting events over its outgoing streams. Streaming operators are similar to relational algebra operators, and they are classified into three categories according with their nature, namely: stateless, stateful and data store.

- Stateless operators are used to filter and transform individual events. Output events, if any, only depend on the data contained in the current event.
- Stateful operators produce results based on state kept in a memory structure named sliding window. Sliding windows store tuples according to spatial or temporal conditions. The CEP provides aggregates and joins based on time windows (e.g., events received during the 20 seconds) and size windows (e.g. the last 20 events).
- User defined operators. They implement other user defined functions on streams of data.
- Data store operators are used to integrate the CEP with the BigDataStack data stores. These operators allow to perform correlation among real time streaming data and data at rest.

The main components of BigDataStack CEP are:

- Orchestrator: It oversees the CEP. It registers and deploys the continuous queries in the engine. It monitors the performance metrics and decides reconfiguration actions.
- Instance Manager (IM): It is the component that runs a continuous query or a piece of it. They are single threaded and run in one core.
- Reliable Registry: It stores information related to query deployments and components status. It is implemented by Zookeeper.
- Metric Server: It handles all performance metrics of the CEP. The collected metrics are load, throughput, latency of queries, subqueries and operators, CPU, memory and IO usage of IMs. These metrics are handled by Prometheus time series database.
- Driver: The interface between the CEP and other applications. Applications use the CEP driver to register/unregister or deploy/undeploy a continuous query, subscribe with the output streams of the queries to consume results and mainly to send events to the engine.

Figure 22 shows the different components of the CEP and their deployment in several nodes. Each node can run several Instance Managers (one per core). The registry and metric server are deployed in different nodes although they can be collocated in the same node. The client

bigdatastack.eu

and receiver applications are the ones producing and consuming the CEP data (shown as dashed black lines). The rest of the communication is internal to the CEP. The Orchestrator communicates with the IMs to deploy queries (configuration messages) and registers this information in Zookeeper (Zookeeper communication). All components send performance metrics to the metric server (yellow dashed lines).

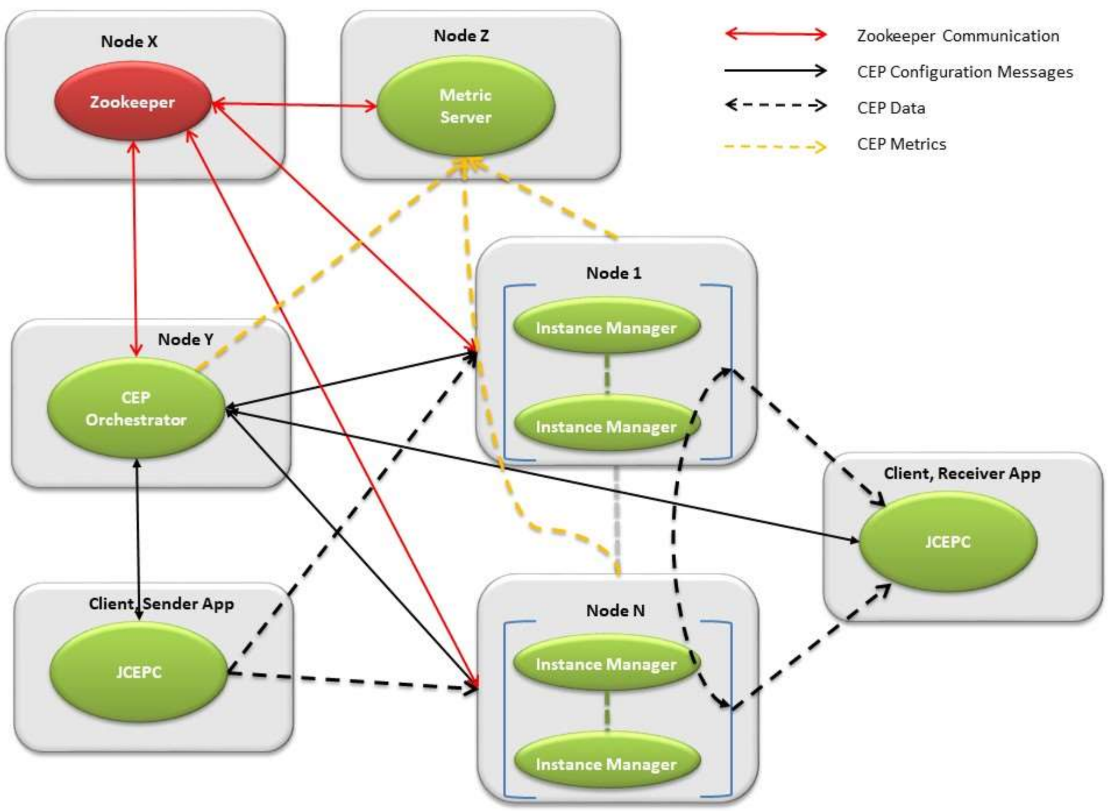


Figure 22 - CEP Components and Deployment

6.8. Process mapping and Analytics

The *Process mapping and analytics* component of the BigDataStack architecture consists of two separate sub-components: *Process Mapping* and *Process Analytics*.

- The objective of the Process Mapping sub-component is to predict the best algorithm from a set of algorithms available in the Predictive and Process Analytics Catalogue, given a specific dataset D and a specific analysis task T.
- The goal of the Process Analytics sub-component is to discover Processes from event logs and apply Process Analytics techniques to the discovered process models in order to optimize overall processes (i.e., workflows).

6.8.1. Process Mapping

The inputs of the Process Mapping sub-component consist of:

- The analysis task T (e.g., Regression, Classification, Clustering, Association Rule Learning, Reinforcement Learning, etc.) that the user wished to perform

- Additional information that is dependent on the analysis task T (e.g., the response – predictor variables in the case of Supervised Learning, the desired number of clusters in the case of Clustering, etc.).
- A dataset D that is subject to the analysis task T

Table 3 provides an overview of the main symbols used in the presentation of the Process Mapping sub-component.

Symbol	Description
T	An analysis task (e.g., clustering, classification…)
D	A dataset
T(D)	The analysis task T applied on dataset D
A(T)	An algorithm that solves the analysis task T (e.g., A(T)=K-means for T=Clustering)
A(T,D)	An algorithm applied on D to solve the task T
M(D)	A model describing a dataset D
T	An analysis task (e.g., clustering, classification…)
D	A dataset
T(D)	The analysis task T applied on dataset D

Table 3 - Main symbols used in Process Mapping

The output of the Process Mapping sub-component is an algorithm A(T) that is automatically selected as the best for executing the data analysis task T at hand. The best algorithm can be based on various quantitative criteria, including result quality or execution time, and combinations thereof.

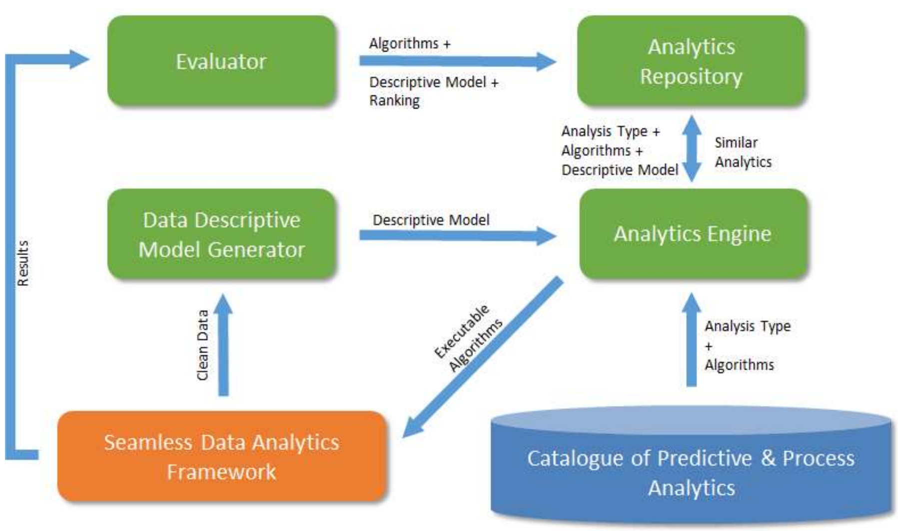


Figure 23 - High-level architecture of Process Mapping sub-component

High-level Architecture

Figure 23 provides an overview of the different modules and their interactions. The Process Mapping sub-component comprises the following four main modules:

- *Data Descriptive Model:* This module takes as input a dataset in a given input form and performs automatically various types of data analysis tests and computation of different statistical properties, in order to derive a model M(D) that describes the dataset D. Based on the relevant research literature, examples of information that is typically captured by the model M(D) include: dimensionality and the intrinsic (fractal) dimensionality, set of attributes, types of attributes, statistical distribution per numerical attribute (mean, median, standard deviation, quantiles), cardinality for categorical attributes, statistics indicating sparsity, correlation between dimensions, outliers, etc. The exact representation of the model M(D) is going to be presented in the following more concretely, but it can be considered as a feature vector. Thus, in the following, the terms model and feature vector are used interchangeably. Subsequently, the produced feature vector M(D) is going to be used in order to identify previously analysed datasets that have similarities with the given dataset. This is achieved by defining a similarity function $sim(M(D_1),M(D_2))$ that operates at the level of feature vectors $M(D_1)$ and $M(D_2)$.
- *Analytics Engine:* The main role of this module is to provide an execution environment for analysis algorithms. Given a specific dataset D and a task T, the Analytics Engine can execute the available algorithms A(T) on the specific dataset, and obtain its result A(D,T). The available algorithms are retrieved from the Predictive and Process Analytics Catalogue for algorithms available in BigDataStack. In this way, evaluated results of analysis algorithms executed on datasets are kept along with the model description of the dataset. Separately, we implement in the analytics engine the functionality of computing similarities between models of datasets, thereby enabling the retrieval of the most similar datasets to the dataset at hand.
- *Analytics Repository:* The purpose of this repository is to store a history (log) of previous evaluated results of data analysis tasks on various datasets. Each record in this repository corresponds to one previous execution of a specific algorithm on a given dataset. It contains the model of dataset that has been analysed in the past, along with the algorithm executed, and its associated parameters. In addition, the record keeps one or more quality indicators, which are numerical quantities (evaluation metrics) that evaluate the performance of the specific algorithm when applied to the specific dataset.
- *Evaluator:* Its primary role is to evaluate the results of an algorithm that has been executed, and provide some numerical evaluations indicating how well the algorithm performed. For example, for clustering algorithms, several implementations of clustering validity measures can be used to evaluate the goodness of derived clusters. For classification algorithms, the accuracy of the algorithm can be computed. For regression algorithms, R-Squared, p-values, adjusted R-Squared and other metrics will be computed to evaluate the quality of the result. Apart from these quality metrics, performance-related metrics are also recorded, with execution time being the most representative such metric.

Once the Process Mapping sub-component has received the required inputs, the data is ingested into the Data Descriptive Model where characteristics and morphology aspects of the dataset D are analysed, in order to produce the model M(D). Then, together with user requirements are forwarded to the Analytics Engine. At this point a query is made from the Analytics Engine to the Analytics Repository, a storage of previously executed analysis models and the final algorithms that were executed in each case. We distinguish two cases:

- *No similar models can be found:* In this case, the available algorithms from the Predictive and Process Analytics Catalogue that match the user requirements are executed, and the results are returned and evaluated in the Evaluator (where quality metrics are computed for each run depending on its performance). The results are stored in the Analytics Repository.
- *A similar model can be found:* In this case, the corresponding algorithm (that performed well in the past on a similar dataset) is executed on the dataset at hand, and the results are again analysed in the Evaluator. The results are again stored in the Analytics Repository. In case the result is not satisfactory, the process can be repeated for the second most similar model, etc.

<u>Example of Operation</u>

The operation of Process Mapping entails two discrete phases: (a) the learning phase, and (b) the in-action phase.

In the learning phase, the system executes algorithms on datasets and records the evaluations of the results in the analytics repository. Essentially, the system learns from executions of algorithms of different datasets.

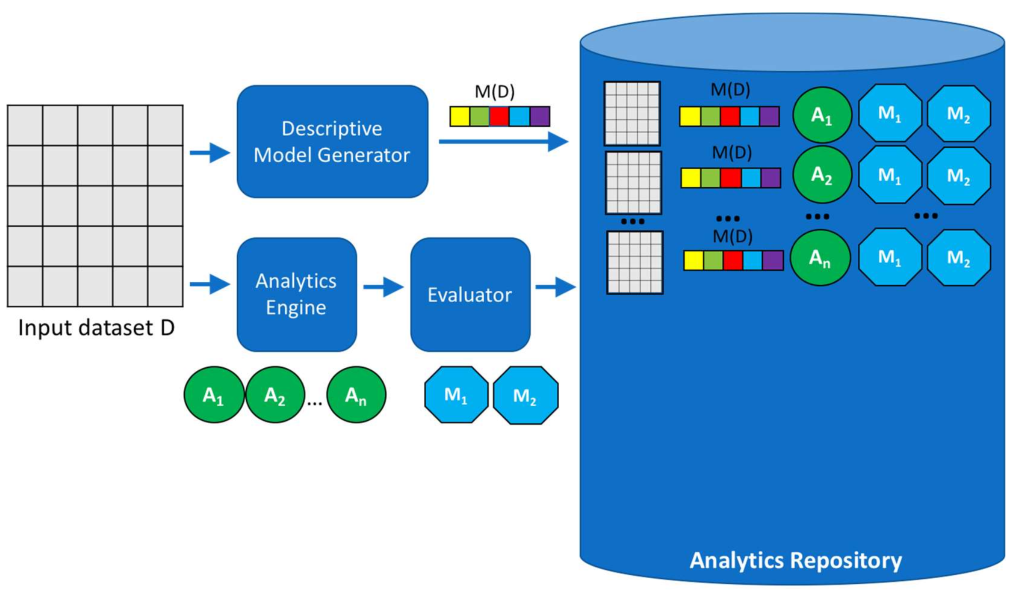


Figure 24 - Learning phase of Process Mapping: Processing the first dataset D

The learning phase starts without any evaluated results in the analytics repository. As shown in Figure 24, when the first dataset D is given as input, the Descriptive Model Generator produces the model M(D). In parallel, the available algorithms A_1, A_2, …, A_n are executed on D and their result is given to the Evaluator, which computes the available metrics M_1 and M_2. Examples of metrics could be accuracy and execution time. Then, this information is stored in the analytics repository: the model M(D), the algorithm A_i, and the values of metrics M_1 and M_2. Notice that the actual dataset is not stored, however it is shown in the figure just for illustration purposes.

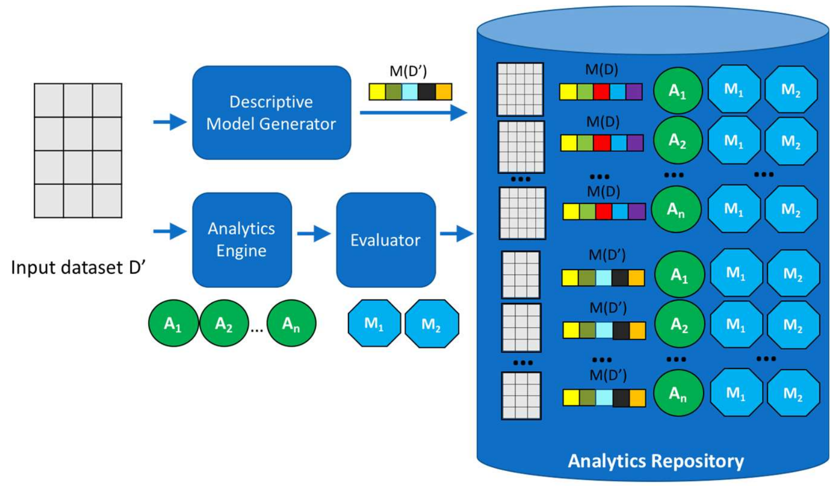


Figure 25 - Learning phase of Process Mapping: Processing the second dataset D'

Figure 25 shows the processing of a second dataset D', still in the learning phase. The same procedure as described above is repeated, and the results are added to the Analytics Repository.

The in-action phase corresponds to the typical operation of Process Mapping in the context of BigDataStack, namely to perform the actual mapping from an abstract task T (which is present as a step of a process designed in the process modelling framework) to a concrete algorithm A(T) that can be executed on the dataset D at hand, i.e., A(T,D). The following example aims at clarifying the detailed operation.

Figure 26 shows a new dataset which is going to be processed based on the specification received from the process modelling framework. Next, the Process Mapping automatically suggests the best algorithm (A_*) from the pool of available algorithms A_1, A_2, …, A_n.

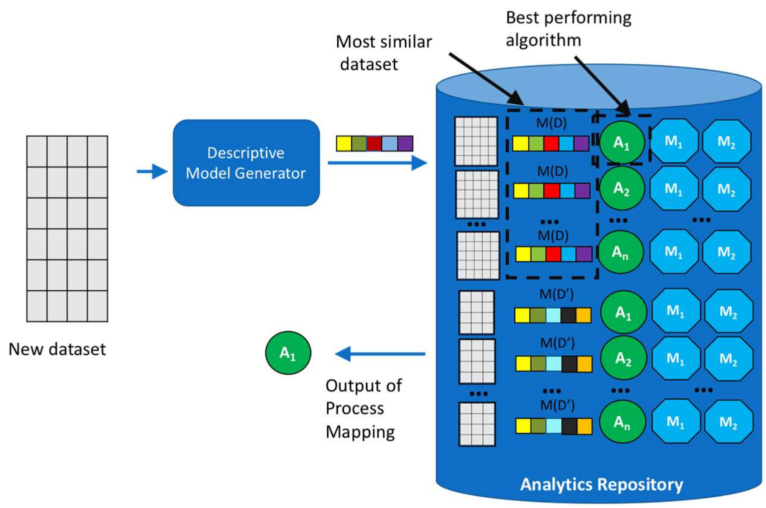


Figure 26 - The in-action phase of Process Mapping

As depicted in the figure above, the Descriptive Model Generator produces the model for the new dataset, and then this model is compared against all available models in the analytics repository in order to identify the most similar dataset. In this example, M(D) is the most similar model. Then, the best performing algorithm is selected from the results kept for M(D). The values of available metrics (M_1 and M_2) are used to identify the best algorithm based on an optimization goal, which could rely to one metric or a combination of metrics, according the needs of the application. In the example, the output of Process Mapping is depicted as algorithm A_1.

Technical Aspects of Prototype Implementation

At the time of this writing, which corresponds to the first half of the project, we have a prototype implementation of Process Mapping in place. The prototype targets a specific class of analysis algorithms, namely Clustering algorithms, in order to be focused. In the second half of the project, this functionality is going to be extended. Below, we provide the technical details and individual techniques used by Process Mapping.

First, the Descriptive Model Generator follows two alternative approaches for model generation (i.e., feature extraction) from the underlying dataset, based on the state-of-the-art methods for automatic clustering algorithm selection. The first approach, called *attribute-based*, generated eight (8) features from the dataset: logarithm of number of objects, logarithm of number of attributes, percentage of discrete attributes, percentage of outliers, mean entropy of discrete attributes, mean concentration between discrete attributes, mean absolute correlation between continuous attributes, mean skewness of continuous attributes, and mean kurtosis of continuous attributes. The second approach, called *distance-based*, computes the vector of pairwise distances **d** of all pairs of objects in the dataset. Then, it generates nineteen (19) features from **d**. The first five (5) features are the mean, variance, standard deviation, skewness and kurtosis of **d**. The next ten (10) features are the ten percentiles of distance values in **d**. The last four (4) features are based on the normalized Z-score, namely they correspond to the percentage of normalized Z-score values in the range:

BY NC ND

[0,1), [1,2), [2,3), [3,infinity). Determining the best approach between attribute-based and distance-based is a subject of experimental evaluation in the context of BigDataStack. A recent paper reports that distance-based approach is better for clustering tasks.

Second, the Analytics Engine is implemented as a wrapper around WEKA, a library for machine learning tasks. In the current implementation three clustering algorithms are used (Kmeans, FarthestFirst, and EM) for the proof-of-concept prototype. In the second half of the project, we are going to replace WEKA with Spark's MLlib. Also, we are going to extend the functionality to other machine learning and analysis tasks, other than clustering.

Last, but not least, the Evaluator uses metrics both for the quality of data analysis as well as for performance. The result quality for clustering is evaluated using Silhouette coefficient, a metric for clustering quality assessment that is based on intra-cluster distances and inter-cluster distances. In terms of performance, the Evaluator records the execution time needed by the algorithm to produce the results. The application that runs in BigDataStack can select whether algorithm selection will be based on optimizing result quality, performance, or an arbitrary (application-defined) combination of these two.

6.8.2. Process Analytics

The Process Analytics sub-component comprises the following four main modules:

- *Discovery:* The main objective of this component is via a given event log to create a process model.
- *Conformance Checking/Enhancement:* This component's role is dual. Firstly, in the Conformance Checking Stage a process model is evaluated against an event log for missing steps, unnecessary steps, and many more (process model replay). Secondly, in the Enhancement Stage user input is considered (e.g. cost-effectiveness or time effectiveness of a process) to create an according model of a process. Also, in this stage dependency graphs will be created and through metrics, such as direct succession and dependency measures to be utilized by the Predictions component.
- *Log Repository:* A repository consisting of any changes to a model during Conformance Checking/Enhancement stage.
- *Prediction:* Dependency graphs and weighted graphs of process models, created in the Enhancement phase will be used in collaboration with an active event log to predict behaviour of an active process.
- *Model Repository:* A storage unit of all process models, user-defined or created in the Discovery stage.

The input variables of this mechanism are:

- Event logs.
- Process models (not obligatory).

The output of the mechanism is as follows:

- Discovered process models.
- Enhanced process models.
- Diagnostics on process models.

- Predictions - Recommendations on events occurring in process models.

The main structure of the predictive component is depicted in Figure 27:

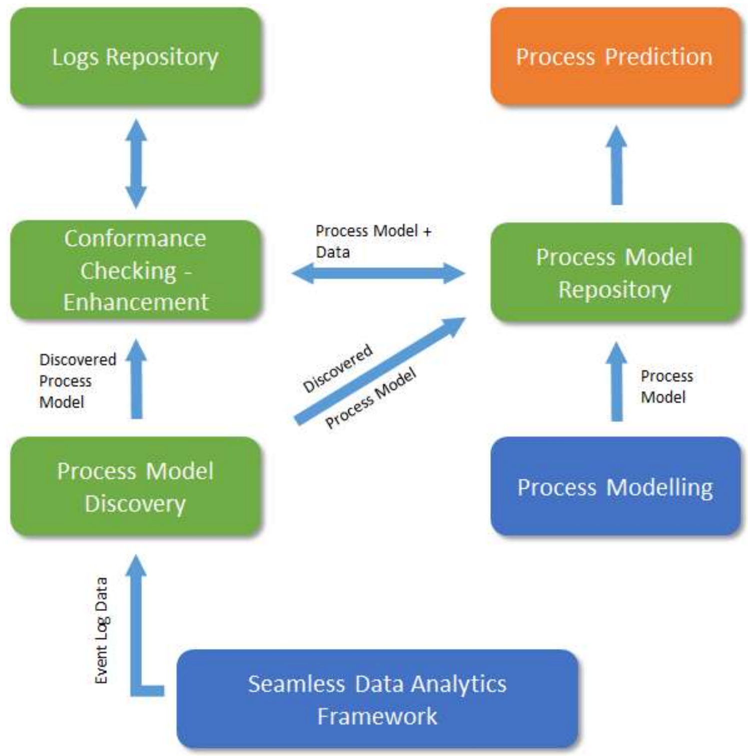


Figure 27 - Internal architecture of Process Analytics sub-component

6.9. Seamless Analytics Framework

A single logical dataset can be stored physically in many different data stores and locations. For example, an IoT data pipeline may involve an ingestion phase from devices via a message bus to a database and after several months the data may be moved to object storage to achieve higher capacity and lower cost. Moreover, within each lifecycle phase, we may find multiple stores or locations for reasons such as compliance, disaster recovery, capacity or bandwidth limitations etc. Our goal is to enable seamless analytics over all data in a single logical dataset, no matter what the physical storage organization details are.

In the context of BigDataStack, we could imagine a scenario where data would stream from IoT devices such as DANAOS ship devices, via a CEP message bus, to a LeanXcale data base and eventually, under certain conditions be migrated to the IBM COS Object Store. This flow makes sense since LeanXcale provides transactional support and low latency but has capacity limits. Therefore, once the data is no longer fresh it could be moved to object storage to vacate space for newer incoming data. This approach is desirable when managing Big Data.

The seamless analytics framework aims to provide tools to analyse a logical dataset which may be stored in one or more underlying physical data stores, without requiring deep knowledge of the intricacies of each of the specific data stores, nor even awareness of where the data is exactly stored. Moreover, the framework provides the tools to automatically migrate data from the relational datastore to the object store, without the interference of a database administrator, with no downtime or expensive ETLs, ensuring data consistency during the migration process at the same time.

A given dataset may be stored within multiple data stores and the seamless analytics framework will permit analytics over it in a unified manner. LXS Query Engine is extended in order to support queries over a logical database that might be split across different and heterogeneous datastores. This extended query engine will serve as the federator of the different datastores and will a) push down incoming queries to each datastore b) retrieve the intermediate results and merge them in order to return the unified answer to the caller. Therefore, the data user will have the impression of executing a query against a single datastore which hosts the logical dataset, without having to know how the dataset is fragmented and split within the different stores. Finally, the federator will provide a standard mechanism for retrieving data: JDBC, thus allowing for a variety of analytical frameworks such as Apache Spark to make use of the Seamless Analytical Framework to perform such tasks.

The data lifecycle is highlighted in the following figure:

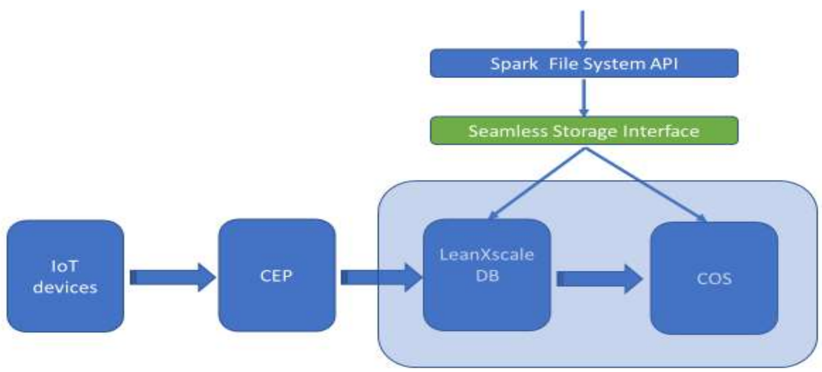


Figure 28 - Seamless Interface

Data is continuously produced in various IoT devices and forwarded to the CEP engine for an initial real-time analysis. This analysis might identify potential alerts or challenges which are triggered by submitting specific rules which use data coming from a combination of sources and are relevant under a specific time window. CEP later ingests data to the LeanXcale relational datastore, which is the first storage point due to its transactional semantics that ensure data consistency. After a period, data can be considered historical and are of no use for an application. However, they are still invaluable as they can participate in analytical queries that can reveal trends or customer behaviours. As a result, data are transferred to the Object Store that is the best candidate for such type of queries. Due to this, data is continuously migrating between stores, and the seamless interface provides the user with a holistic view, without needing to keep track of what was migrated when.

6.10. Application Dimensioning Workbench

The goal of the dimensioning phase is to provide insights regarding the required infrastructure resources primarily for the data services components, linking the used resources with load and expected QoS levels. To this end, it needs to link between the application/service-related information (such as KPIs and workload, parameters of the data service etc.) and the used resources to be able to provide recommendations towards the deployment mechanisms, through e.g. prediction and correlation models. Benchmarking against these services is necessary in order to concentrate the original dataset that is needed in a variety of business scenarios, such as sizing the required infrastructure for private deployments of the data services or consulting deployment mechanisms in a shared multitenant environment where multiple instances of a data service offering may reside.

The main issues that need to be handled by the Dimensioning Workbench are:

- The target trade-off that needs to be achieved between a generic functionality and an adapted operation. For example, benchmarking for each individual application request would lead to very high and intolerable delays during the deployment process. Thus, one would need to abstract from the specifics of an application instance through the usage of suitable workload features, benchmark in advance for a variety of these workload features and thus only need to query for the most suitable results during the deployment stage.
- The achieved abstraction and automation for easily launching highly scalable and multi-parameter benchmarks against the data services, with minimal user interaction and need for involvement. This would require the rationale of a benchmarking framework inside ADW that will be able to capture the needed variations between the configuration parameters (workload, resource etc), adapt to the needed client types per data service as well as the target execution environment of the tests (e.g. different execution platforms such as OpenShift, Docker Swarm, external public Cloud offerings such as AWS etc).
- The workflow/graph-based nature of the application, which implies that application (and data service) structure should be known and taken under consideration by the analysis. To this end, needed annotations are required so that the generic structure which is provided as input to the Workbench through the Data Toolkit contains all the necessary information such as expected QoS levels (potentially for different metrics), links between the service components etc. On top of this structure, the workbench can quantify the expected QoS per component and then propagate through the declared dependencies.
- While application structure is provided to the workbench, this will often not imply a particular deployment configuration for the application (e.g. what node types will be suitable for the user’s application). Multiple trade-offs in this domain could also be given to the users, enabling them to make a more informed final decision based on cost or other parameters. For this reason, the dimensioning workbench needs to receive this input of available deployment patterns from the Pattern Generation in order to populate them with the expected QoS, information that is taken under consideration in the process for final ranking and selection.
- Adaptation of benchmarking tests in a dockerized manner in order to be launched through the framework in a coordinated and functional manner, based on each test’s requirements and needed sequences.

Dependencies of the dimensioning component especially in the form of anticipated exchange of information (in type and form) are presented in the following bullets. Inputs include:

- Structure of the application along with the used data services is considered an input, as concretized by the Data Toolkit component (in the form of a playbook file, the BigDataStack Playbook) and passed on to the Dimensioning component, following its enrichment with various used resource types from the Pattern Generator, and including expected workload levels inserted by the user in the Data toolkit phase. This is the structure upon which the Dimensioning workbench needs to append information regarding expected QoS per component.
- Types of infrastructure resources available in terms of size, type, etc (referred to as resource templates). This information is necessary at the Pattern Generator side in order to create candidate deployments.
- Different types of Data Services will be provided by BigDataStack to the end users. Each of these services may have different characteristics and functionalities, affected in a different manner and quantity by the application input (such as the data schema used). Consideration of these features should be included in the benchmarking workload modelling of the specific service (e.g. number of columns in the schema tables, types of operations, frequency of them etc.), as well as inputs that may be received by the application developer/data scientist, such as needed quality parameters of the service (such as latency, throughput needed etc.) or other preferences declared through the Data Toolkit.
- Application related current workload and QoS values should be available to enable the final creation of the performance dataset, upon which any queries or modelling will be performed. This implies a collaboration and adaptation with the used benchmark tests and/or infrastructure monitoring components such as the Triple Monitoring Engine, in case the used benchmarks do not report on the needed metrics.
- Language and specification used by the Deployment component, or any other provisioned execution environment, given that ADW needs to submit such descriptors for launching the benchmarking tests.
- Exposure of the necessary information, such as endpoints, configuration, results etc to the Visualization components of the project, in order to be embedded and controlled from that side as well. Thus relevant APIs and JSON schemas need to be agreed and implemented based on this feature.

Necessary outputs:

- The most prominent output of the Dimensioning phase is the concretized (in terms of expected QoS) playbook for a candidate deployment structure for the used data services in the format needed by the ADS-Ranking component that utilizes the dimensioning outcomes. This implies that the format used by Dimensioning to describe these aspects should be understood by the respective components and thus was agreed in collaboration, defined currently as a Kubernetes configuration template type of file structure called a BigDataStack Playbook. More concretely, this is operationalized as a series of candidate deployment patterns (CDPs), which describe the different ways that the user’s application might be deployed along with the expected QoS levels per defined metric. CDPs are provided in the respective file format, such that they can be easily used to perform subsequent application deployment. The Dimensioning phase will augment each CDP with estimated performance metrics and/or quality of service metrics, providing a series of indicators that can be used to judge the potential suitability of each CDP. These estimates are used later to select the CDP that will best satisfy the user's deployment requirements/preferences.

- Intermediate results include the benchmarking results that are obtained through the benchmarking framework of ADW. These need to be exposed either to internal ADW components for subsequent stages (e.g. modelling or population of the playbook) or external such as Visualization panels towards the users for informative purposes.

The main structure of the Dimensioning is depicted in Figure 29. The component list is as follows:

- *Pattern Generation:* The role of pattern generation is to define the different ways that a user's application might be deployed. In particular, given the broad structure of a user's application provided by the Data Toolkit, there are typically many ways that this application might be deployed, e.g. using different node types or utilizing different replication levels. We refer to these different ways that a user's application might be deployed as 'candidate deployment patterns' (CDPs). CDPs are generated automatically through analysis of the user's application structure provided in the form of a 'BigDataStack Playbook' file from the Data Toolkit, as well as the available cloud infrastructure. Some CDPs will be more suitable than others once we consider the user's requirements and preferences, such as desired throughput or maximum cost. Hence, different CDPs will encode various performance/cost trade-offs. These CDPs define the configurations that are used as filters for retrieving the most relevant benchmarking results during the Dimensioning phase, producing predicted performance and quality of service estimations for each. Even though Pattern Generation is part of Dimensioning, it is portrayed as an external component given that for each CDP the core Dimensioning block will be invoked.
- *ADW Core:* The ADW Core is the overall component that is responsible for the main functionalities of Dimensioning. It is split into two main parts, the ADW Core Benchmarking, which is responsible for implementing and storing benchmarking runs with various setups, and the ADW Core Runtime that is used during the assisted deployment phase of BigDataStack in order to populate the produced CDPs with the predicted QoS levels. Following, a highlight of the various functionalities of each element is described, split into more fine-grained parts.
- *Bench UI:* The Bench UI is used by the Data Service owner in order to define the parameters of the benchmarking process, which is performed "offline", thus not in direct relationship to a given application deployment during runtime. It is necessary for this user to investigate the performance considerations of their service and proceed with this stage, during the incorporation of their data service in the BigDataStack ecosystem, in order to have gathered the necessary data a priori and not need to benchmark during the actual application deployment. The latter would create serious timing considerations and limitations that would not be tolerated by the end users. Through the Bench UI, multiple parameters can be defined, leading to a type of parameter sweep execution of a test, in order to automate and enable an

easier result gathering process. The UI includes a visual element for selection of the parameters, as well as a relevant REST endpoint in which the user can submit a JSON description of the test (thus enabling further automation through multiple REST submissions). It can also be used to monitor the progress of the test. Result viewing and relevant queries can also be performed via the central visualization component of BigDataStack, while a workload definition tab is expected to be supported also in Y3 of the project.

- *Test Control:* Test control is used in order to prepare, synchronize and configure test execution. A number of steps are needed for this process based on the user's selected options, such as running tests in a serial or parallel manner, preparing shared volumes and networks and so on.
- *Deployment Description Adapter:* In order to enable launching of the defined tests in an execution platform (such as Openshift, Docker Swarm, external Clouds etc), relevant deployment descriptors should be created. For example, for Openshift a relevant playbook file needs to be created and populated with the parameters selected for the benchmark tests, such as input arguments, selected resources etc and then forwarded to ADS Deploy. A playbook template structure is created beforehand for each bench test type based on the execution needs of each test (e.g. number and type of containers, needed shared volumes and networks etc), necessary included data service etc, that is then populated with the specific instantiation's details. Different execution platforms can be supported through the inclusion of relevant plugins that implement the according formats of that platform or the relevant API calls to setup the environment (a Docker Swarm version is already supported at this time). Through this setup the system under stress (data service) is automatically deployed, as well as the necessary number of bench test clients in order to cover the desired load levels.
- *Image repository:* While this refers to the main image repository across the project, its inclusion here is used to indicate the necessary inclusion of the bench tests images, appropriately adapted based on the benchmarking framework's needs, in terms of execution, configuration and result storing.
- *Results/Model repository:* This component is intended to hold the benchmarking results obtained through the test execution process as well as hold the created regression models used during the Result Retrieval queries in the Runtime phase (Y3).
- *Structure Translator:* This component acts as an abstraction layer and is responsible for obtaining the output of the Data Toolkit containing the application structure in the format this is expressed (e.g. playbook service structure) and extracting the parameters that are needed in order to instantiate the query towards the result retrieval phase. Furthermore, in cases of multi-level applications, it is responsible for propagating the process across the service graph.
- *Result Retrieval:* This component is responsible for obtaining the specified deployment options from the CDPs, the anticipated workload and produce the predicted QoS levels of the service. This may happen either through direct querying of the stored benchmarked results (y2) or through the creation and training of predictive regression models (Y3) that will also be able to interpolate for cases that

BY NC ND

have not been investigated, based on the training of the regressor and the depiction of the outputs (QoS) dependency from the predictor inputs (workload and h/w-s/w configuration used).

- *Output Adaptor:* This component acts as an abstraction layer and is responsible for generating the output format needed for the communication with ADS Ranking (in the particular case enriching the inputed playbook file with the extra QoS metrics).

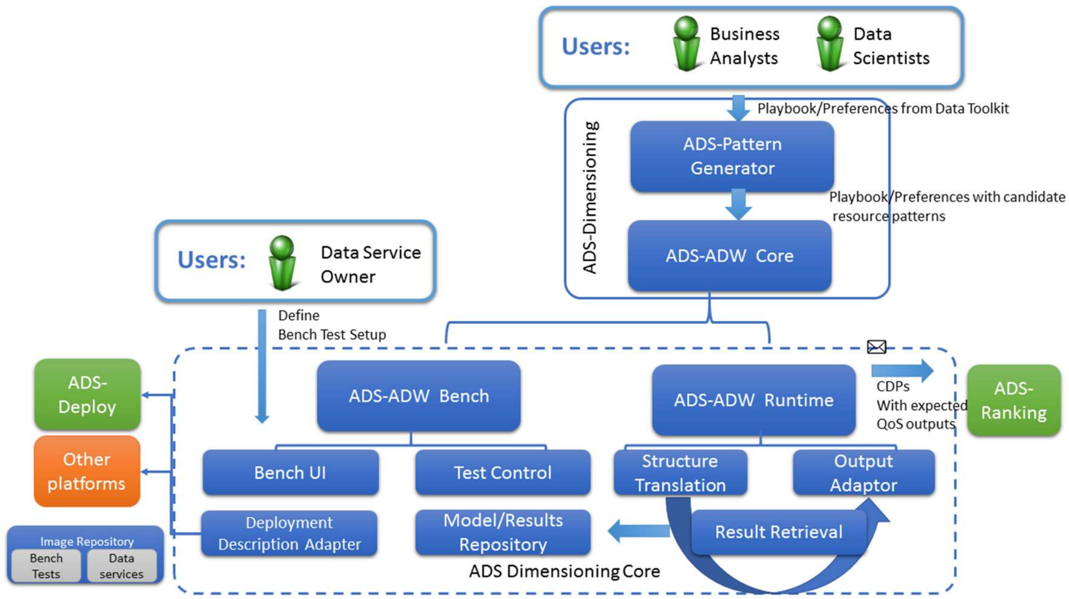


Figure 29 - Application dimensioning internal structure and link with external components

6.11. Big Data Layout and Data Skipping

Here we focus on how to best run analytics on Big Data in the cloud. Today's best practices to deploy and manage cloud compute and storage services independently leaves us with a problem: it means that potentially huge datasets need to be shipped from the storage service to the micro-service to analyse data. If this data needs to be sent across the WAN then this is even more critical. Therefore, it becomes of ultimate importance to minimize the amount of data sent across the network, since this is the key factor affecting cost and performance in this context.

We refer the reader to the BigData Layout section (8.10) of the D2.1 BigDataStack deliverable which surveys the main three approaches to minimize data read from Object Storage and sent across the network. We augmented these approaches with a technique called Data Skipping, which allows the platform to avoid reading unnecessary objects from Object Storage as well as avoiding sending them across the network (also described in D2.1). As explained there, in order to get good data skipping it is necessary to pay attention to the Data Layout.

In BigDataStack data skipping provides the following added-value functionalities:

1. Handle a wider variety of datasets, go beyond geospatial data
2. Allow developers to define their own data skipping metadata types using a flexible API.

3. Natively support arbitrary data types and data skipping for queries with UDFs (User Defined Functions)
4. Handle continuous streaming data that is appended to an existing logical dataset.
5. Continuously assess the properties of the streaming data to possibly adapt the partitioning scheme as needed
6. Handle general query workloads. This is significant because often different queries have different, even conflicting, requirements for data layout.
7. Handle query workloads which change over time.
8. Build a benefit/cost model to evaluate whether parts of the dataset should be partitioned anew (thus rewritten) to adapt to significant workload changes.

Previous research focused on the HDFS, whereas we plan to focus on Object Storage, which is of critical importance in an industrial context. Object Storage adds constraints of its own: once an object has been put in the Object Store, it cannot be modified, where even appending to an existing object is not possible, neither can it be renamed. This means that it is important to get the layout right as soon as possible and avoid unnecessary changes. Moreover, it is important for objects to have roughly equal sizes (see our recent blog on best practices [17]), and we are researching the optimal object size and how it depends on other factors such as data format. Moreover, the cost model for reorganizing the data layout is likely to be different for Object Storage than for other storage systems such as HDFS.

6.12. Process modelling framework

Process modelling provides an interface to business users to model their business processes and workflows as well as to obtain recommendations for their optimization following the execution of process mining tasks on the BigDataStack analytics framework. The outcome of the component is a model in a structural representation – a JSON formatted file. The latter is actually a descriptor of the overall graph reflecting the application and data services mapped to specific executables that will be deployed to the BigDataStack infrastructure. To this end, the descriptor is passed to the Data Toolkit component and then to the *Application Dimensioning Workbench* to identify their resource requirements prior to execution.

The main issues that need to be handled by the Process modeling framework are:

- *Declarative process modelling approach*: Processes may be distinguished in Routine (Strict) and Agile. Routine processes are modelled with the imperative method that corresponds to imperative or procedural programming, where every possible path must be foreseen at design time and encoded explicitly. If a path is missing, then it is considered not allowed. Classic approaches like the BPEL or BPMN follow the imperative style and are therefore limited to the automation type of processes. The metaphor employed is the flow chart. Agile processes are modeled with the declarative method according to which declarative models concentrate on describing what must be done and the exact step-by-step execution order is not directly prescribed; only the undesired paths and constellations are excluded so that all remaining paths are potentially allowed and do not have to be foreseen individually. The metaphor employed is rules/constraints. Agility at the process level, entails “the ability to redesign and reconfigure Individual business process components, combining individual tasks and capabilities in response to the environment” [18].

Declarative process modeling or a mixed approach seems to fit well in our environment providing the necessary flexibility in process modelling, mapping and optimization.

- *Structure to output to the Data Toolkit and subsequently to the application dimensioning framework, workflow/reference to executables/execution logic:* The output of the process modeling framework should be a structure to feed the Data Toolkit and later on the dimensioning framework. The structure should provide for reproducing the process graph, the tasks mapping to executables and the logic in terms of rules/constraints that govern the execution flow and the execution of the process tasks. Process Modelling outputs the structure of the developed process model to Data Toolkit component.

The main structure of the Process modelling framework is described below. The component list is as follows:

- *Modeling toolkit:* This component provides the *interface for business analysts* to design their processes in a non-expert way, the *interface for developers* to provide in an easy way predefined tasks and relationship types as selectable and configurable tools for business analysts and *the core engine* to communicate with all the involved components towards *design, concretization, evaluation, simulation, output and optimization of a business process*.
- *Rules engine:* The engine *provides all the logic for defining rules and constraints, evaluating and executing* them. The aim is the business analyst to be provided with a predefined set of rules offered as a choice through the tasks and relations toolbox.
- *ProcessModel2Structure Translator:* This component generates the structure from the developed model that will feed the Data Toolkit and subsequently the dimensioning framework. This structure must be able to instantiate and run as an application. It will include the *workflow*, *the logic* in terms of relationships and rules regarding the execution of process tasks, *reference and configuration* of the *involved analytics tasks* (contained in the catalogue) and reference to *other application tasks and services* (which are not contained in any catalogue) (i.e. a task that generates a report from collected values, a task that finds the maximum value of a set of values, or a task that when triggered communicates using an API and turns off a machine, if we consider a process that controls the operation of machines).

Process Modelling Framework Capabilities

bigdatastack.eu

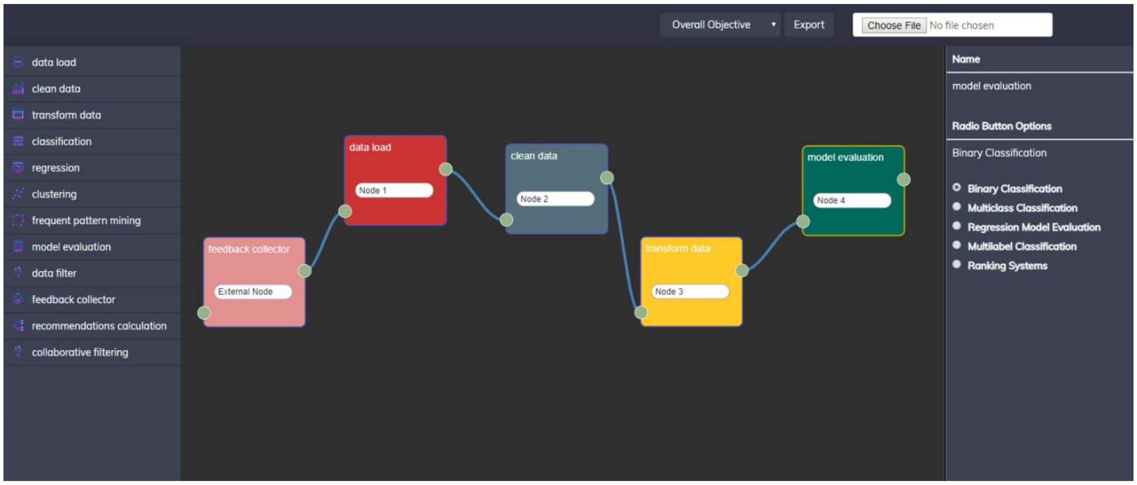


Figure 30 - Process modeling framework

The Process Modeler component is the first link in the chain. The Business Analysts have the ability to design their processes in a straightforward graphical way by using a visual editor. The user can create a graph containing nodes from a list provided and assign options to each node. In detail these nodes and their respective options are:

- Data Load
  - Distributed Store
  - Object Store
- Clean Data
  - Yes
  - No
- Transform Data
  - Normalizer
  - Standard Scaller
  - Imputer
- Classification
  - Binomial Logistic Regression
  - Multinomial Logistic Regression
  - Random Forest Regression
- Regression
  - Linear Regression
  - Generalized Linear Regression
  - Random Forest Regression
- Clustering
  - K Means
  - LDA
  - GMM

- Frequent Pattern Mining
  - FP Growth
- Model Evaluation

- Binary Classification
- Multiclass Classification
- Regression Model Evaluation
- Multilabel Classification
- Ranking Systems

- Data Filter
  - Yes
  - No
- Feedback Collector (External Service)
- Recommendations Calculation (External Service)
- Collaborative Filtering
  - ALS

Additionally, the business analyst can define the overall objective of the graph which can be:

- Analytics Algorithm Accuracy
- Analytics Algorithm Time Performance
- Save Computing Resources
- Overall Time Efficiency
- Overall Cost Efficiency
- Decrease Average Throughput
- Decrease Average Latency

Finally, the Process Modeller Component provides the capability to import, export, save and edit the generated graphs.

6.13. Data Toolkit

The main objective of the data toolkit is to design and support data analysis workflows. An analysis workflow consists of a set of data mining and analysis processes, interconnected among each other in terms of input/output data streams or batch objects. The objective is to support data analysts and/or data scientists to concretize the business process workflows created through the *process modelling framework*. This can be done by considering the outputs of the *process mapping* component or choosing among a set of available or under development analytic functions, while parametrizing them with respect to the service-level objectives defined in the corresponding process. A strict requirement regards the capacity to support various technologies/programming languages for development of analytic processes, given the existence and dominance of set of them (e.g. R, Python, Java, etc).

Towards this direction, the data toolkit is going to be modelled in a way that will enable data scientists to declare and parametrize the data mining/analytics algorithms, as well as the required runtime adaptations (CPUs, RAM, etc.), data curation operations associated with the high-level workflow steps of the business process model.

At its core, the data toolkit will incorporate an environment which supports the design of *graph-based workflows*, and the ability to annotate/enrich each workflow step with algorithm or processes specific parameters and metadata, while respecting a predefined set of rules to which workflows must conform on in order to guarantee their validity.

BY NC ND

There is a wide range of versatile flow-based programming tools that fit well the requirements for constituting the basis for the data toolkit, such as Node-Red [19]. Also a custom workflow-design environment tailored for the specific needs of the data toolkit could be developed, supported by libraries such as D3.js [20] and NoFlo [21], which will allow for fine-grained control over all the elements associated with the data analytics workflow.

Figure 31 depicts the core configuration user interface per functional component and/or service in the BigDataStack context. Therefore, the Data Scientist can parameterise her components providing details on the elasticity profile, the Docker images, the minimum execution requirements, the required environmental variables, the exposed interfaces and required interfaces (if any), existing attributes (i.e. lambda functions, etc.) and the corresponding health checks regarding the services.

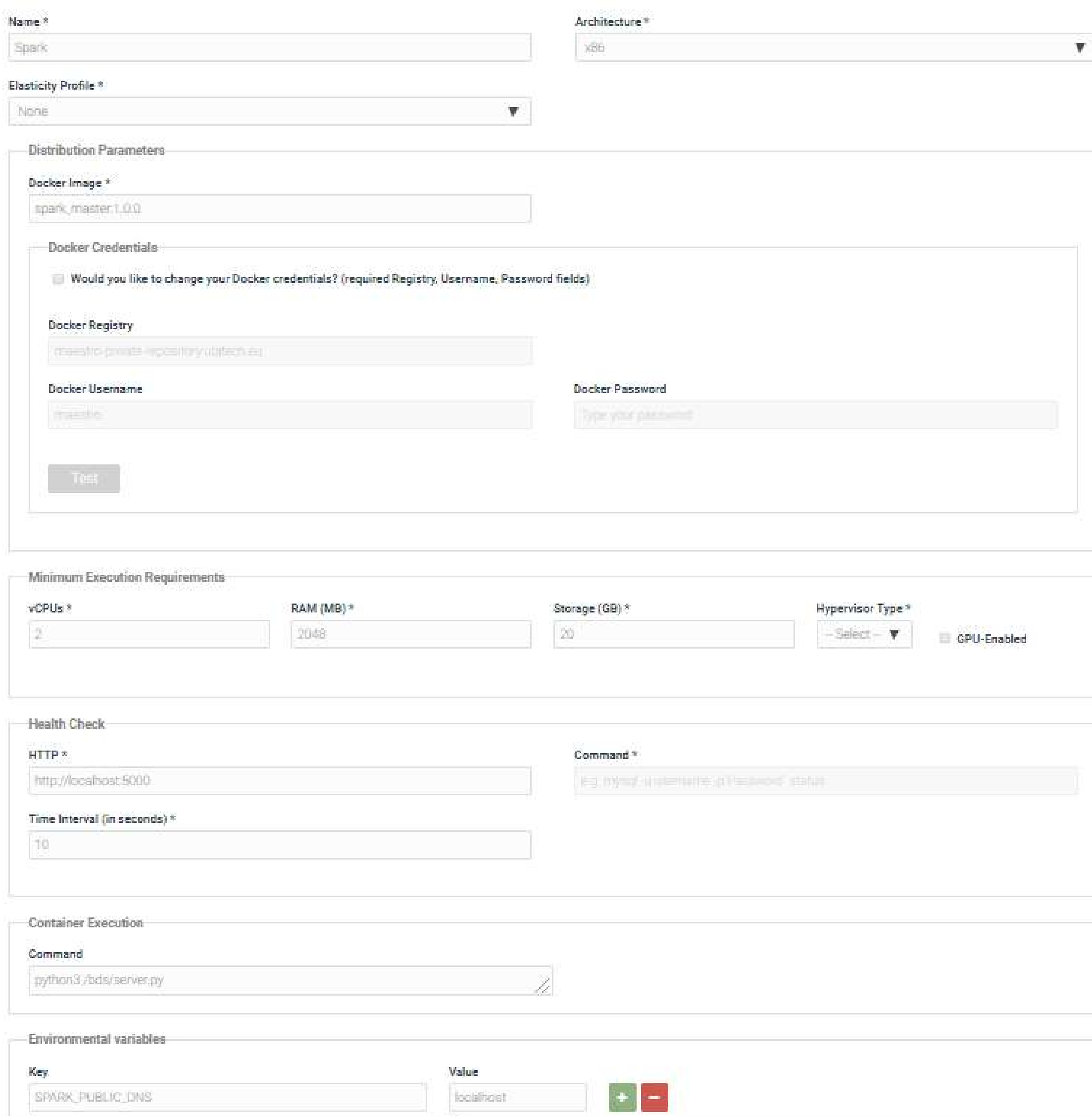


Figure 31 - Application configuration per graph components

6.14. Adaptable Visualizations

The adaptable visualization layer has multiple purposes: (i) to support the visualization of data analytics for the applications deployed in BigDataStack, (ii) to provide a visual application performance monitoring dashboard of the data operations and the applications during benchmarking, dimensioning workbench and during operation and (iii) to integrate and facilitate various components such us Process Modeller, Data Toolkit, Benchmarking, Dimensioning Workbench, Triple Monitoring Engine, Data Quality Assesment and Predictive Maintenance. Importantly, the dashboard will be able to monitor the application deployed over the infrastructure. For the visualization of data analytics, it will provide a reporting tool that will enable to build visual analytical reports. The reporting will be produced from analytical queries and will include summary tables as well as graphical charts.

The main issues that need to be handled by the adaptable visualizations framework are:

- User authentication
- KPIs definition and integration: Definition of a KPI must be possible through the framework if not supported elsewhere in the architecture
- Triggering of events and production of visual notifications. Event handling and triggering of alarms or responses to the event must be supported.
- Different views of the UI platform depending on the user role. 4 roles are defined:
    - Administrator (full UI View)
    - Business Analyst (Process Modeller View)
    - Data Analyst (Data Toolkit View)
    - Application Owner/Engineer (BenchMarking, Dimensioning Workbench, Analytics View)
- Integration of Process Modeller, Data Toolkit and Benchmarking Components.
- Deployment of playbooks towards the Dimensioning Workbench Component, visualization of the configurations recommended and deployment of the selected application.
- Management of the Deployed Applications and handling of the Deployment Adaptation Decisions. Decisions are consumed from the Global Decision Tracker.
- Ability to redeploy applications when QoS Warnings are received and Deployment Alterations are considered.
- Visualisation of the Predictive maintenance for both cases of full datasets and exclusively quality assessed data.
- Visualisation of the Data Quality Assessments in summary customizable tables.

The foreseen I/O and the structure of the visualization framework in terms of definition of the subcomponents and their interactions are listed in the following bullets.

Necessary inputs:

- Analytic outcomes as input from the seamless data analytics framework
- Real-time monitoring data as input from the triple monitoring engine. Data will refer Application components monitoring, to Data & Services monitoring and to Cluster resources monitoring
- CEP outcomes as input from the real-time CEP of the Storage engine

- Input from exposed data sources to facilitate KPIs definitions and event triggering rules.

Necessary Outputs:

- Output of visual reports

The main structure of the Adaptable visualizations framework is depicted in Figure 32. The component list is as follows:

- *Visualization toolkit:* this component connects all the components (Process Modeller, Data Toolkit, BenchMarking, Dimensioning Workbench) and makes available a tool set of offered capabilities (e.g. types of graphs, reports, tables)
- *Rights management module (Admin Panel):* this component handles the permissions to modify views to components, editors and event triggers
- *Data connector:* this component makes possible to retrieve data schemas and data from the exposed data sources to assist in defining KPIs and set event triggers. Furthermore, it could provide the same way access to historical data or reports
- *Events processing:* this component makes possible to define event triggers that will produce visual notifications, warnings or generation of specific reports

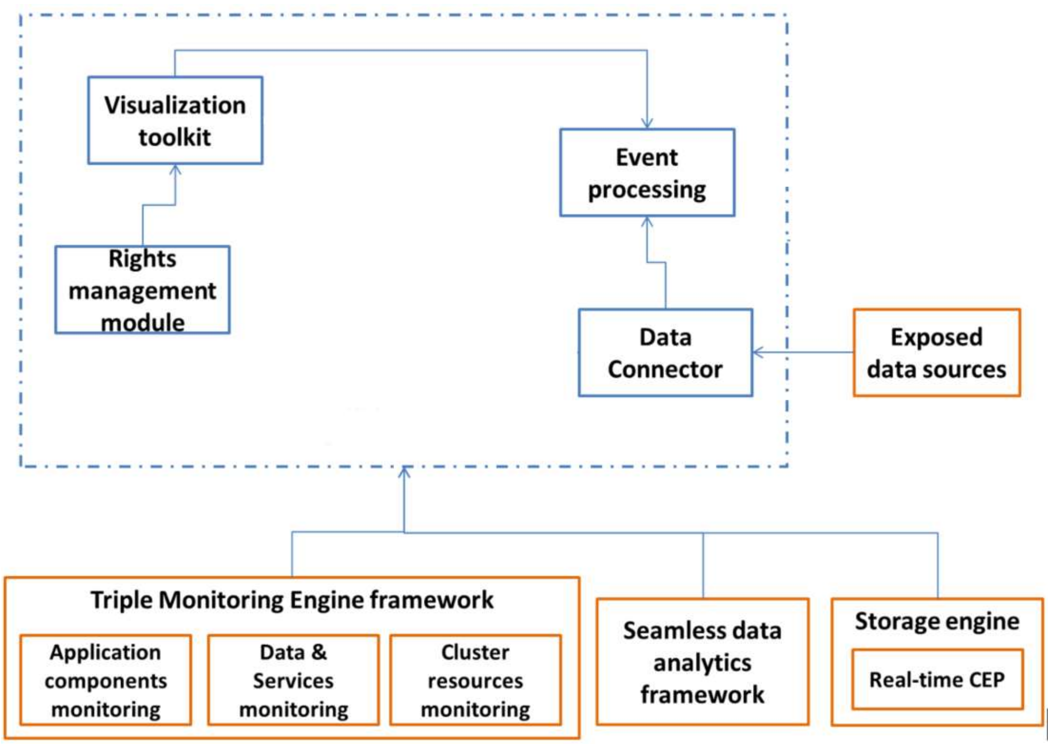


Figure 32 - Visualization framework building blocks

7. Key interactions

7.1. User Interaction Layer

User Interaction within the BigDataStack ecosystem plays an important role in the entire lifecycle of a big data application / operation. There exist the following user roles: Business Analysts, Data Analysts and/or Data Scientists.

First, the *Business Analyst* uses the *Process Modelling Framework* to define the *business processes and associated objectives* and accordingly design a BPMN-like workflow for the actualization of the business-oriented objectives and the required analytic tasks to accomplish. The analyst is able to design, model and characterize each step in the workflow according to a list of predefined rules encapsulated by a *rules engine* component of the modelling framework. The output of this process is a graph-like output (i.e. in JSON format) with a high-level description of the workflow from the business analyst's perspective along with the related end-to-end business objectives. The sequence diagram of Process Modelling is depicted in Figure 33.

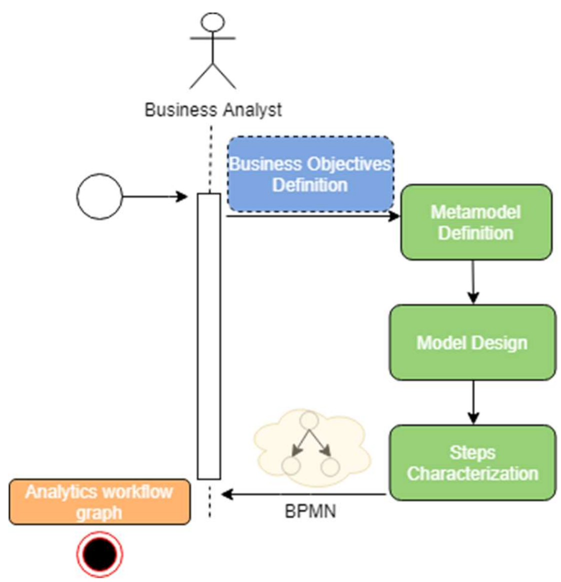


Figure 33 - Information flows in Process Modelling

Figure 34 depicts a high-level application graph designed by the Business Analyst by indicatively incorporating within the data workflow four (4) processing steps with editable fields by means of drop-down lists, namely data load, data clean, perform analytic task and evaluate result.

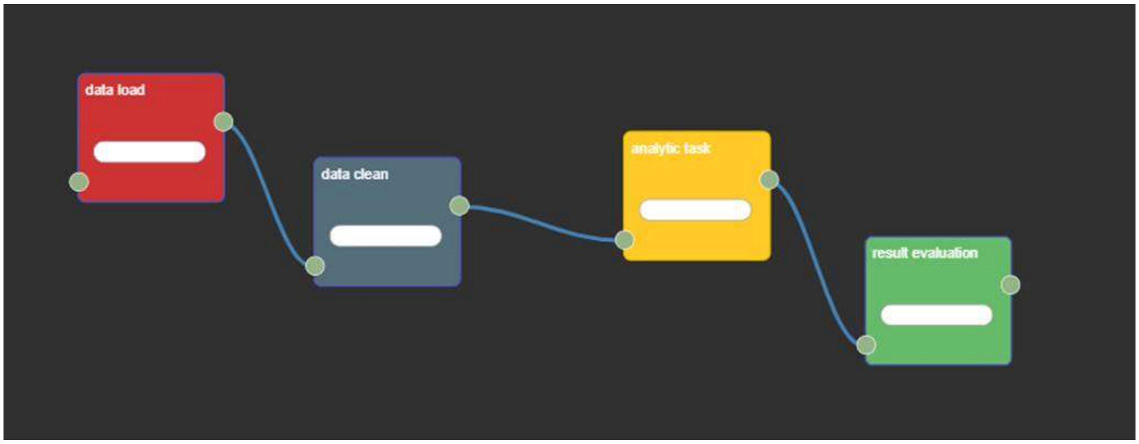


Figure 34 - Example of a high level BRMN-like application graph

Next, the *Process Mapping* component provides an association of the process steps modeled by the Business Analyst with specific analytic tasks, following a set of criteria related to each process task, while considering any constraints defined in the business objectives. These criteria may contain the characterization of required data, time, resources and/or performance parameters need to be concretized to perform the analytic tasks. The output of this step is a workflow graph (i.e. in JSON format) enriched with the mappings of the business process steps grounded to algorithms, runtime and performance parameters.

Then, the *Data Analyst* and/or the *Data Scientist* uses the *Data Toolkit*, to perform a series of tasks related to the concretization of the analytics process workflow graph produced in the process mapping step, as depicted in Figure 35, such as:

- Concretizing the business objectives in terms of selecting lower bounds for hardware, runtime adaptations, performance for which the selected algorithms perform sufficiently well.
- Defining the data source bindings from where the datasets related to the task will be ingested.
- Defining any data curation tasks (i.e. data cleaning, feature extraction, data enrichment, data sampling, data aggregation, Extract-Transform-Load (ETL) operations) necessary for the algorithms and the related steps.
- Configuring and parametrizing the data analytics tasks returned (i.e. selected) by the Processes Mapping component, and additionally providing the functionality to design and tune new algorithms and analysis tasks, which are then stored to the Catalogue of Predictive and Process Analytics and can be re-used in the future.
- Selecting and defining performance metrics for the algorithms, along with the acceptable ranges with respect to the business objectives and service-level objectives, used to evaluate the algorithm/model and resources configurations.

At the end, a *Playbook* (i.e. in YAML format) representing the grounded workflow for each business process will be generated, in the format that further feeds the *Dimensioning workbench* in order to provide the corresponding resource estimates for each node of the graph.

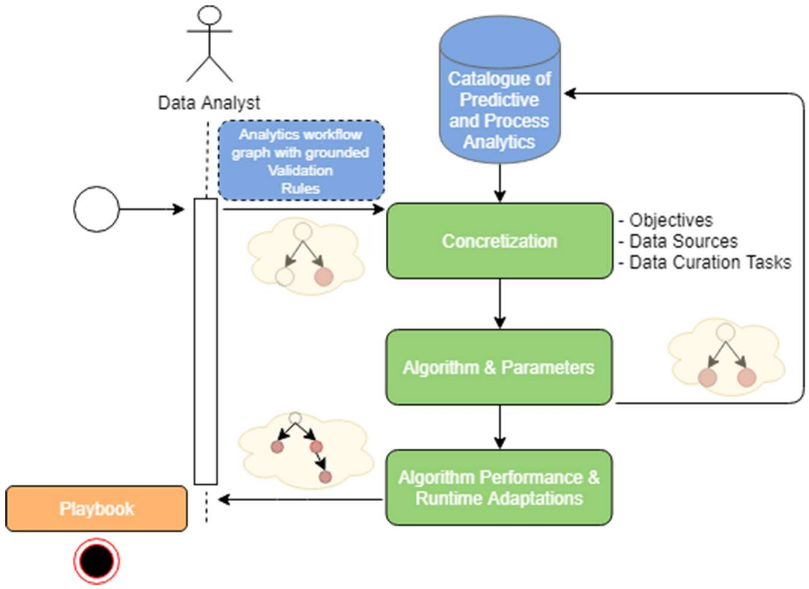


Figure 35 - Information flows in Process Mapping

The following figure (Figure 36) presents the sequence diagram, which depicts the main information flows for the User Interaction Layer of the BigDataStack architecture.

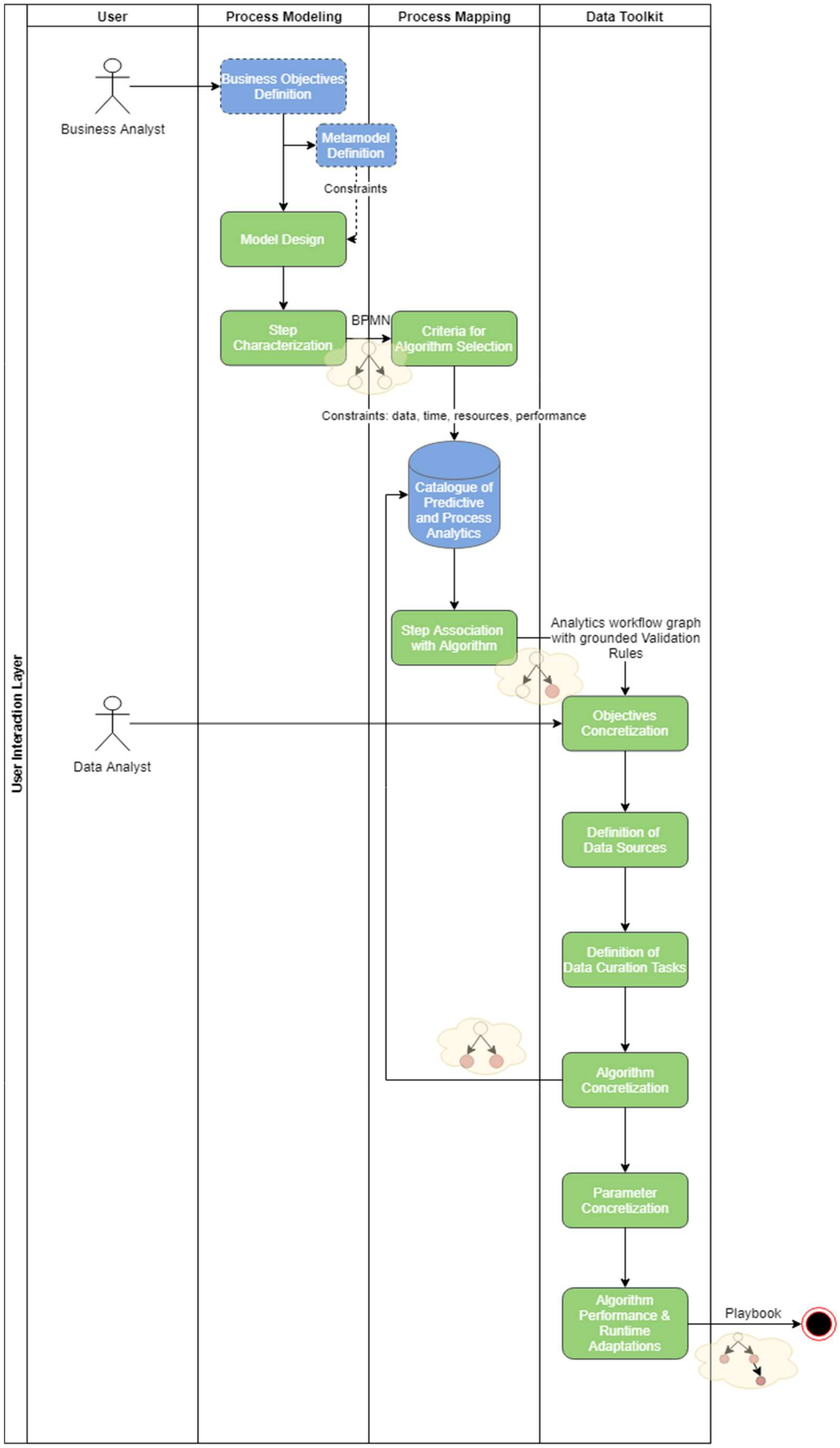


Figure 36 - User Interaction Layer Sequence Diagram

BY NC ND

Example Use Case: Predictive Maintenance

Regarding the entry phase described above, an example is presented in the following sections to link the functionalities of different components to an actual use case.

Business Analyst's View

The following figure (Figure 37) shows the perspective of a business analyst in terms of Process Modelling, which treats Real-time ship monitoring (RTSM) as a whole. This is expected to be the view (not in terms of user interface but in terms of processes and abstraction of information) of the Process Modelling Framework. Moreover, through the framework, the business analyst will be able to specify constraints (as noted with red fonts in the figure).

Overall, separate processes, actions and data required to perform RTSM. As shown, the first step is the vessel and weather data acquisition. That includes a dataset with granularity down to a minute and 2 years timespan for vessel data, along with weather data as provided by the National Oceanic and Atmospheric Administration (NOAA), i.e., granularity of weather reports up to 3 hours for every 30 minutes of a degree. Past this, given that there are plenty of attributes within both datasets, there has to be some attribute selection rule. For example, only 190 approximately are required from both datasets, because these are the most reliable and important. Following this, the data are imported into two different components. The first is the monitoring tool, which simulates and enhances the on-board tools of the Alarm Monitoring System (AMS). Given that, if an anomaly occurs a rule-based alert has to be produced close-to or in real time. The second component is the Predictive Maintenance Alert. This informs the end user that the current data under examination pinpoint a malfunction that has occurred in the past. Again, this should work close-to or even better in real-time. Consecutively, given that identifying an upcoming malfunction is achieved, spare part ordering follows. The ordered spare part has to be delivered at least 1 day before the estimated time of arrival, while ordering of spare parts should be performed only by suppliers that are to be trusted. Quality of service should not be neglected while cost criteria are also taken into account. Finally, given the delivery port of the spare part, re-routing of the vessel takes place, where the estimated time of arrival to the closest port is less than 12 hours.

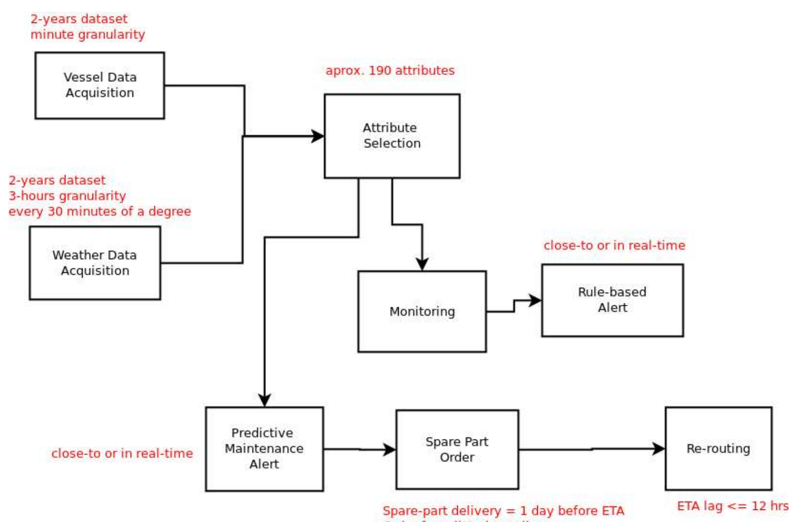


Figure 37 - Business analyst view

Data Analyst's View

Following the outcome of the process modelling (previous view), Figure 38 depicts the view for the data analyst, that is the view in the Data Toolkit. As shown in the figure, the view is different with components that have been mapped automatically from the Process Mapping mechanism of BigDataStack (e.g. “CEP monitoring” to enable the “Rule-based alert” process).

Overall the data analyst’s view is a set of system components, in-house or out-sourced processes and/or systems, actions and data required to perform RTSM. The Vessel data acquisition process is fed from an in-house database (DB) that contains vessel data (power consumption related and main engine data) along with Telegrams and past maintenance events. Given a total of 10 vessels, this requires up to 40 GB of hard disk storage. Weather data are imported from NOAA via FTP, by a weather service that loads hindcasts in GRIB format for the whole earth with a 3-hour granularity for every 30 minutes of a degree. GRIB files are parsed and stored in a database that requires up to 2.1 TB storage. Given that any trajectory of a vessel can by joined with weather data via a REST API that the weather service provides. Past this, given that there are plenty of attributes within both datasets, i.e., weather and vessel data, there has to be some attribute selection rule. For example, only 190 approximately are required from both datasets, because these are the most reliable and important such as the consumed power (kW), the rotations per minute of the main shaft (RPM) etc. In order to avoid feeding the algorithmic components of this architecture with false or null data values, a filtering component is in charge of removing null values, preferably with average values, smoothing-out the effect of data-loss. Next, given a set of defined rules, such as “if the power consumption exceeds a limit and the fuel-oil inlet pressure drops below a threshold” the CEP component is in charge to produce an alert, close-to or in-real time. In parallel, a pattern recognition algorithm tries to identify patterns on the data that looks like a past case where a malfunction occurred in the main engine. If this happens, an alert is produced, and given the upcoming malfunction that has been identified a spare-part suggestion is made. Given the Danaos-ONE platform, where orders of spare parts are placed via a REST API, the order of the suggested spare-part is placed and is accessible from the suppliers that are preferred. So, once the order is made to a supplier, a suggested place and time are provided, and given this re-routing of the vessel takes place via an external REST service provided at a specific IP address and port.

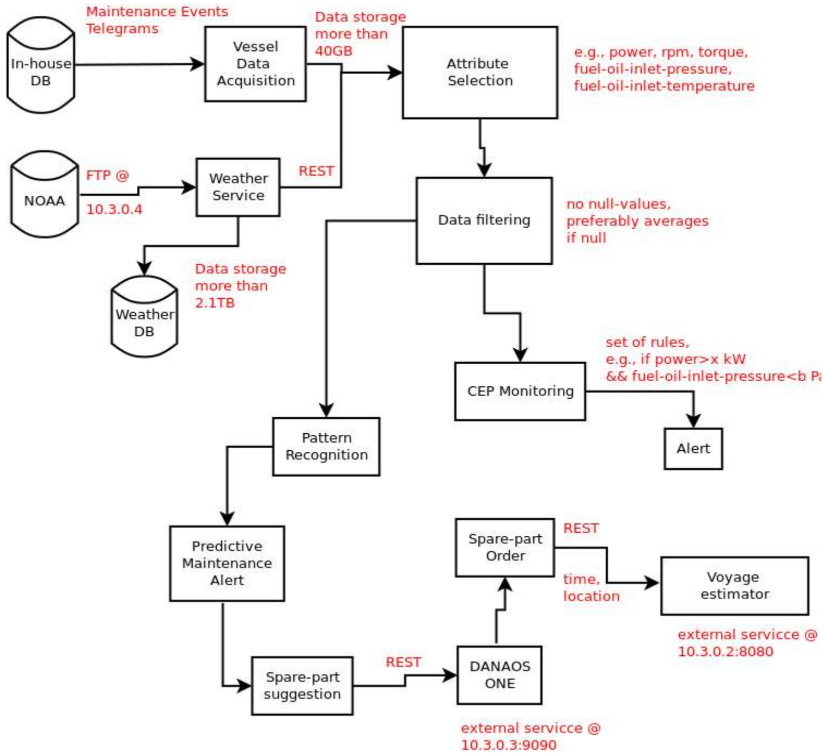


Figure 38 - Data analyst's view

7.2. Realization & Deployment

Application and Data Service Ranking

Within the Realization module, there is a series of operationalizable tasks associated to Application Data Service Ranking (ADS-Ranking). The goal of these tasks is to enable the selection of a candidate deployment pattern (CDP) which represents a complete configuration of the application (which is needed for application deployment on the cloud). There are two main tasks of interest when realizing an application's deployment:

- **First-Time Ranking of Candidate Deployment Patterns**: This task aims to select the most suitable candidate deployment pattern from a set that has previously been generated when the user first requests deployment of their application.
- **Application Deployment**: This task involves the practical deployment of the user application on the cloud through interaction with Openshift.

Below we discuss each of these two tasks in more detail and provide an interaction sequence diagram for each. For legibility of the interaction diagrams, we use short names for each component. A mapping between components and their short names are shown in the following table.

Full name	Sub-component	Short name (interaction diagrams)
Application and Data Services Dimensioning	N/A	Dimensioning
Application and Data Services Ranking	Pod Feature Builder	ADS-R Feature Builder
Application and Data Services Ranking	Pod Scoring	ADS-R Scoring
Application and Data Services Ranking	Model	ADS-R Model

Application and Data Services Ranking	Pattern Selector	ADS-R Pattern Selector
Application and Data Services Deploy	N/A	ADS-Deploy
Dynamic Orchestrator	N/A	Orchestrator
Application and Data Services Global Decision Tracker	N/A	ADS-GDT
BigDataStack Adaptive Visualisation Environment	N/A	BigDataStack UI

Table 4 - Short-name Component Mapping Table

First-Time Ranking of Candidate Deployment Patterns

The first task is concerned with the ranking of candidate deployment patterns when the user first requests their application to be deployed. Candidate deployment patterns are generated by the Dimensioning component of BigDataStack. The output of this task is a selected deployment pattern, which can be passed to Application and Data Services Deployment for physical deployment.

This task is triggered by the Dimensioning component once it has finished generating the different candidate deployment patterns (CDPs) and producing the quality of service estimations for each. The Dimensioning component sends a package of CDPs to the Application and Data Services Ranking (ADS-Ranking) component, or more specifically the Feature Builder sub-component of it. This component analyses and aggregates the different quality of service estimations into a form that can be used for ranking (referred to as features). Once this transformation is complete, the CDPs and aggregated features are sent to the Scoring sub-component, which uses a ranking model to score and hence rank each CDP based its suitability with respect to the user's requirements. Once the CDPs have been ranked, that ranking is sent to the Pattern Selection sub-component, which selects the most suitable one. This selected CDP is then sent to the BigDataStack Adaptive Visualisation Environment component for the user to decide whether to deploy with this configuration. At the same time, a notification is sent to the Dynamic Orchestrator to specify that deployment is underway for the user's application. Moreover, the selected CDP, other CDPs not selected and ranking information/features are sent to the Global Decision Tracker (ADS-GDT) for persistence.

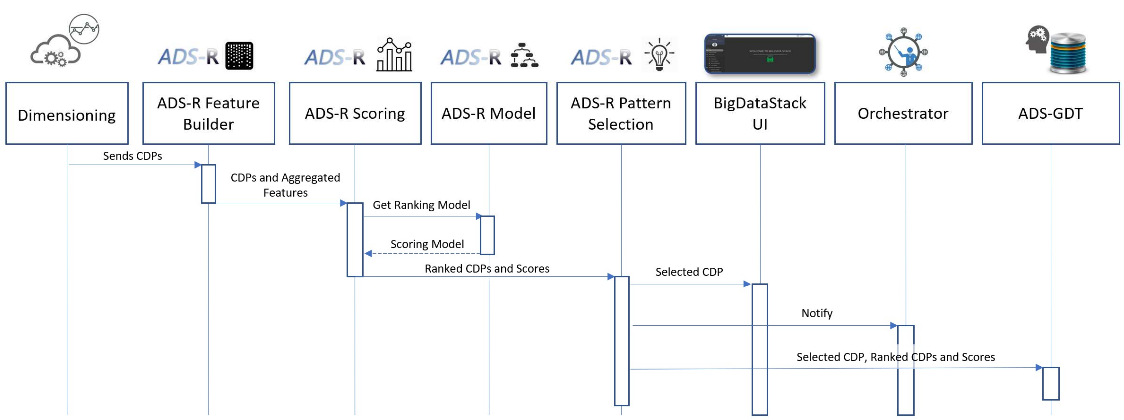


Figure 39 - Interaction Diagram for First-Time Ranking

Application Deployment

The ADS-Deploy component interacts with Openshift through Kubernetes' OpenAPI v1 [1]. Once the candidate deployment pattern has been obtained, it is sent to the deployment component. This is parsed by the ADS-Deploy component, which extracts information on the three main objects of importance to the deployment process (Pods, Services and Routes). ADS-Deploy maps these into a series of independent Openshift-managed objects representing each, enabling incremental deployment and more fine-grained control. However, all those objects are grouped into a single logical application, in order to maintain the internal coherence and keep relations between the objects. These objects are:

- Pods: A Pod represents an atomic object in Openshift, and includes one or more containers. Each pod can be replicated according to the configuration values or due to Quality-of-Service requirements. Pods have been represented as DeploymentConfig objects in BigDataStack. [11]
- Services: A Service provides access to a pod from the outside, and is in charge of vital actions such as load balancing. Services can also be replicated, so that they are scaled in/out independently or together with the pods. ADS-Deploy, creates a configuration file for each service and sends it to Openshift.
- Routes: A route gives a service a hostname that is reachable from outside the cluster. Routes are not replicable, but they are closely related with the services. In BigDataStack, a configuration file is created for each route, and information on the service and application to which they relate is contained in there.

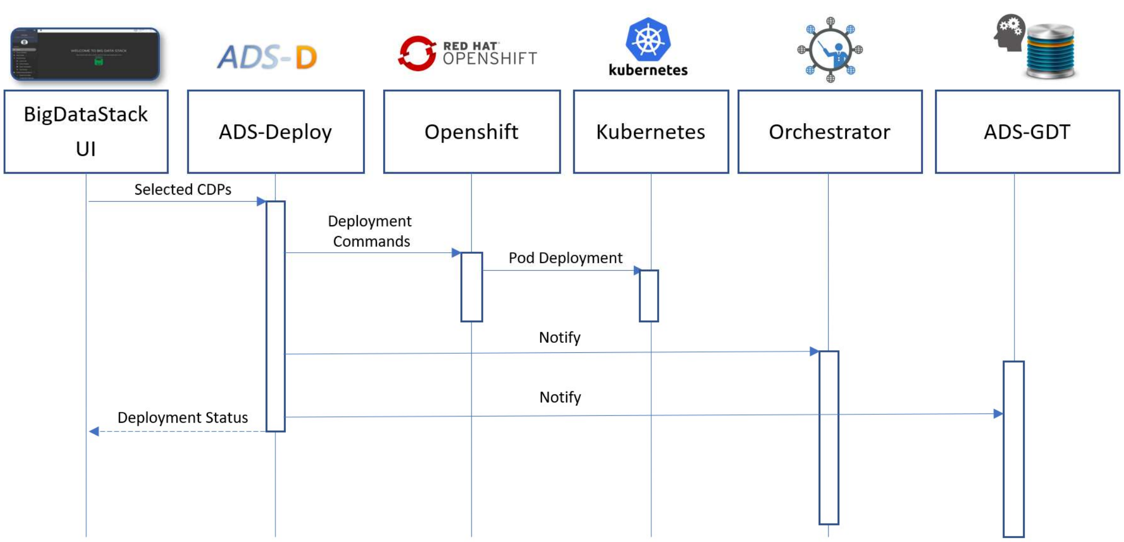


Figure 40 - Interaction Diagram for Application Deployment

7.3. Data as a Service & Storage

The Data as a Service and the Storage offerings of BigDataStack cover different cases. As base data stores, the LeanXcale data store and the Cloud Object Storage (COS) are considered as depicted in the following figure (Figure 31).

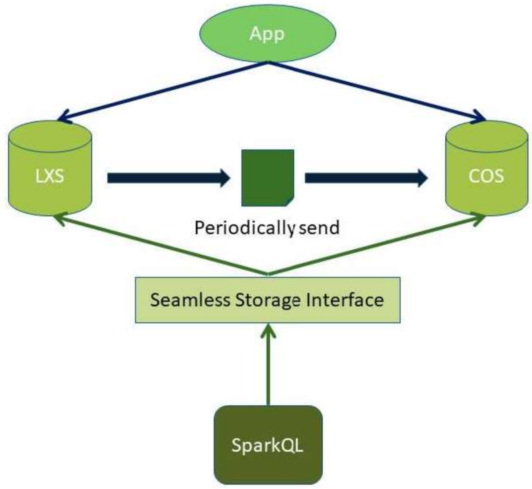


Figure 41 - Architecture of data stores

From the above, it can be considered that the two components that are able to persistently store data are: LeanXcale's relational data store, and IBM's Cloud Object Store. The former is a fully transactional database which will serve operational workloads, while in the meantime can execute analytical operations on the runtime, providing a JDBC implementation, thus being able to execute SQL compliant queries. The latter is a cloud Object Store capable of storing numerous terabytes of data but lacking transactional nor SQL capabilities. Fresh data will be first inserted in the LeanXcale database (LXS) in order to benefit from its transactional capabilities. Once data is no longer considered as fresh, (e.g. several months have passed), data will be moved to the Cloud Object Store (COS) while analytical processing over COS is provided by Apache Spark.

On top of the datastores the Seamless Storage Interface (SSI) provides an entry point for seamlessly executing queries over a logical dataset that can be distributed over different datastores which themselves may provide different interfaces. The SSI provides a common JDBC interface and is capable of executing standard SQL statements. The SQL queries will be pushed down to both stores, and retrieved intermediate results will be merged and returned. Offering a JDBC interface, SSI can be exploited by data scientists through the usage of well-known analytical tools such as SparkSQL. As a result, the end-user can write SparkSQL queries and have the SSI locate the various parts of the dataset and retrieve the results. Direct execution of the queries to a specific data store is also permitted. As a result, we have the following five scenarios:

- Direct access to the LeanXcale database
- Direct access the Cloud Object Store (COS)
- Request data using a simple SparkSQL query
- Insert data to BigDataStack
- Insert streaming data to BigDataStack

Direct access the LeanXcale (LXS) database

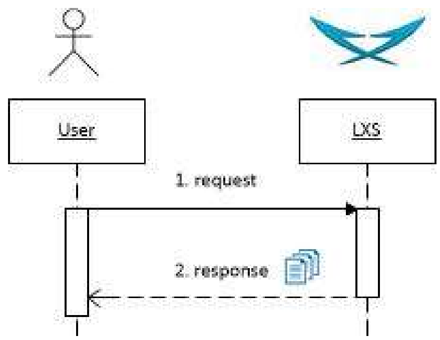


Figure 42 - Direct access the LXS

User executes an SQL query, requesting data directly from LXS using a standard JDBC interface, and the latter returns the resultSet as the response.

Direct access the Cloud Object Store (COS)

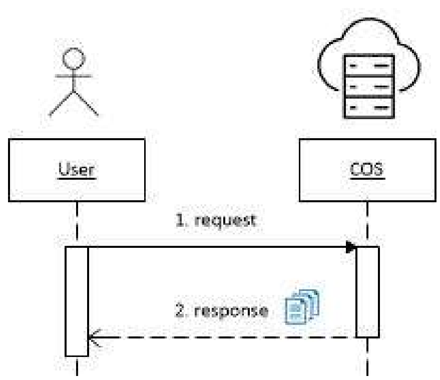


Figure 43 - Direct access the COS

User executes a query from Apache Spark, requesting data directly from COS, using the *stocator* open source connector which permits the connection of Object stores to Spark, and the COS returns back the result as the response.

Request data using a simple SparkSQL query

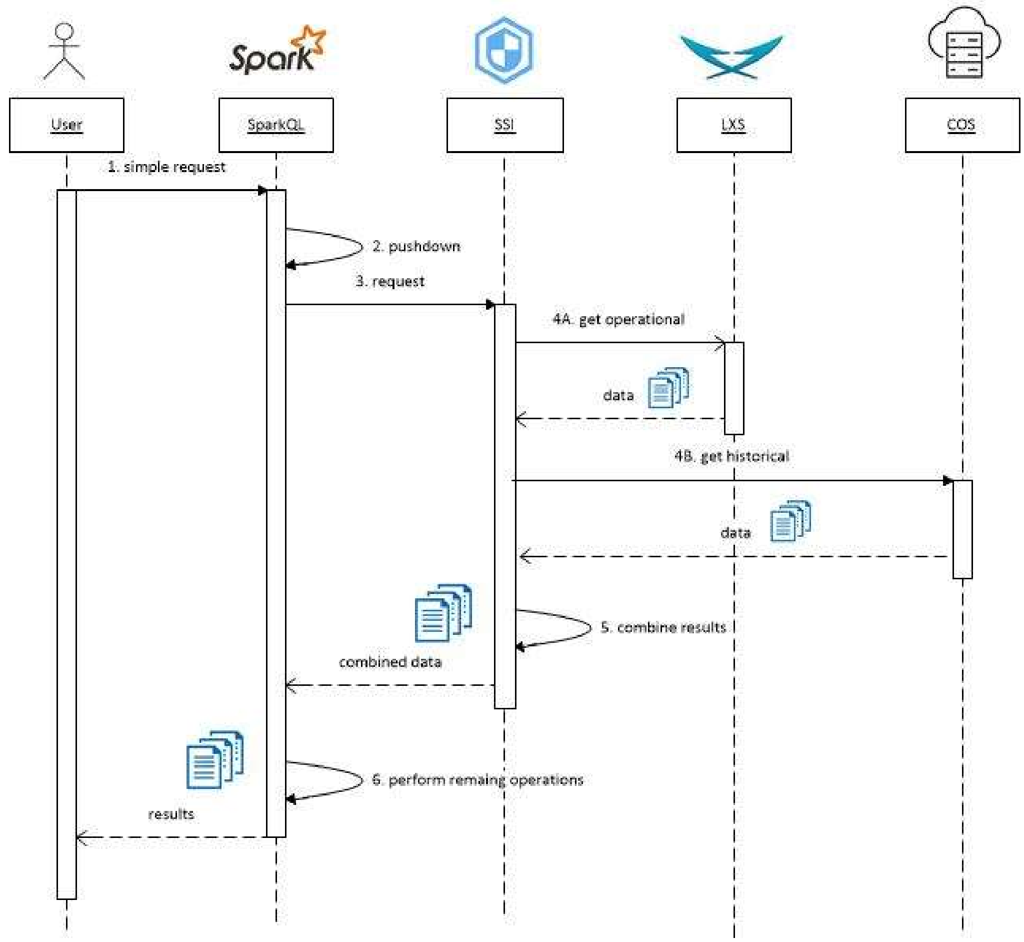


Figure 44 - Request data using a simple SparkSQL query

User sends a request for executing an analytical task by writing a SparkSQL query. The SSI, which is an extension of the LXS Query Engine provides a JDCB functionality, and as a result, is already integrated with SparkSQL. Due to this, SparkSQL will *pushdown* all operations to be executed by the SSI itself. The SSI is aware of the location of the data over the distributed dataset that is split into the two different datastores and is integrated with both of them. As a result, it translates the query to each data store's internal language and requests the data from both of them. It finally aggregates the results and returns the data back to SparkSQL, which returns the results to the user. It is important to notice that the SSI supports various query operations such as table scans, table selections, projections, ordered results, data aggregations (min, max, count, sum, avg) either grouping them by specific fields or not. From the above figure it can be also noticed that steps 4A and 4B might be in parallel according to the type of the query operators.

The architecture of the seamless analytical framework and the main interactions between its components can be shown in Figure 45:

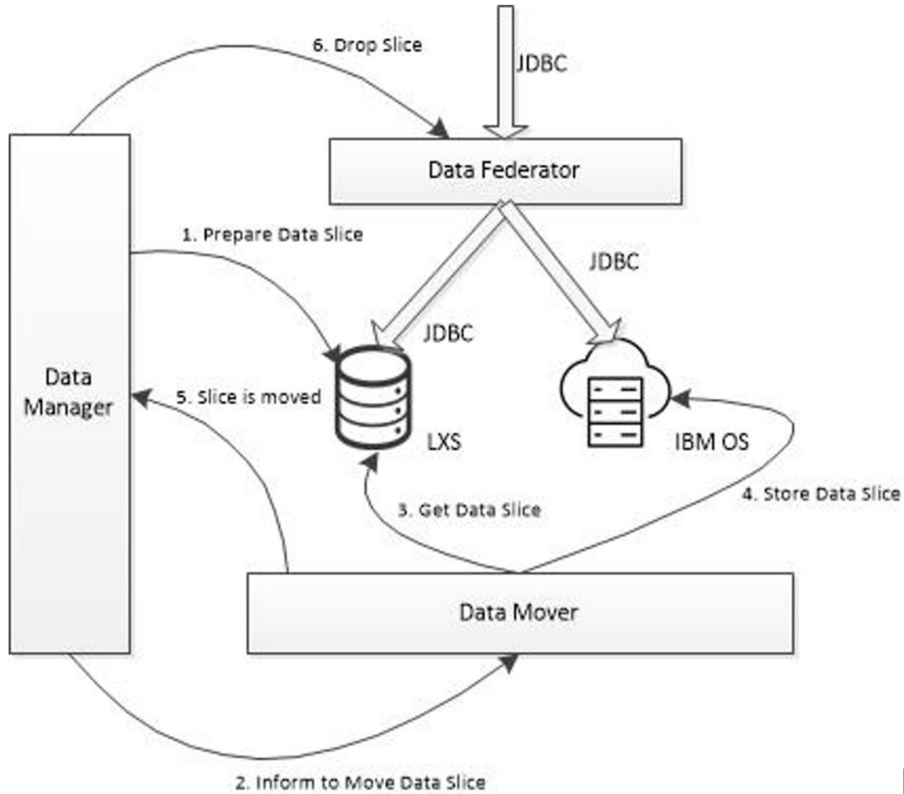


Figure 45 - Seamless Analytical Framework

The Data Manager component, as shown in Figure 45, keeps track of the data ingested in the framework. For each dataset the data user can configure the period of time after which data can be considered as historical and can safely be moved to a data warehouse such as the Object Store. When a data movement action is triggered, it first informs the relational database that a data slice should be moved to the COS. LXS is getting prepared to drop that slice (internally it marks it as read-only and splits it to a data region that can be easily dropped later on). The Data Manager then informs the Data Mover to move the slice. The latter requests the data slice by executing one or many standard JDCB statements to LXS and then uploads the data slice as one or many objects into the objects store. When the whole slice is eventually persisted into the Object Store, it informs the Data Manager which forwards this acknowledgment to the data Federator. The data Federator internally keeps track of a timestamp which records the latest successful data movement. When a query is submitted for data retrieval, it creates the query tree and pushes down a selection based on this timestamp on each operation for a table scan. Then it rebuilds the query by interpreting it according to the target datastore and retrieves the results. Finally, in accordance with the query operation, it merges the results and builds the result set. When the Data Manager acknowledges a data movement and informs the Data Federator, the Data Federator will move accordingly the internal timestamp (the splitting point). At this point, the data corresponding to the moved data slice co-exists in both stores. However, the Data Federator thanks to the timestamp will hide the replicated data first at the Object Store and after the timestamp is updated at the relational store. When it receives the acknowledgement, it updates this timestamp (split point) so that the next transactions can scan the tables accordingly. Pending transactions however will continue to scan the tables based on the value that they received when the transaction first started. The transactional semantics of LXS ensure the data consistency when the split point is updated. When this happens, the Data Federator can order the LXS to safely drop the data slice that has now been moved to the object store. However, it will wait until all pending transactions has been finished, and thus, no scan operation is performed on the data slice that is about to be dropped. By doing so, the Data Federator ensures data consistency and the validation of the results during the process of data movement: Data will exist either on LXS or the COS, or both, but they will be always scanned only once.

Instert data to BigDataStack

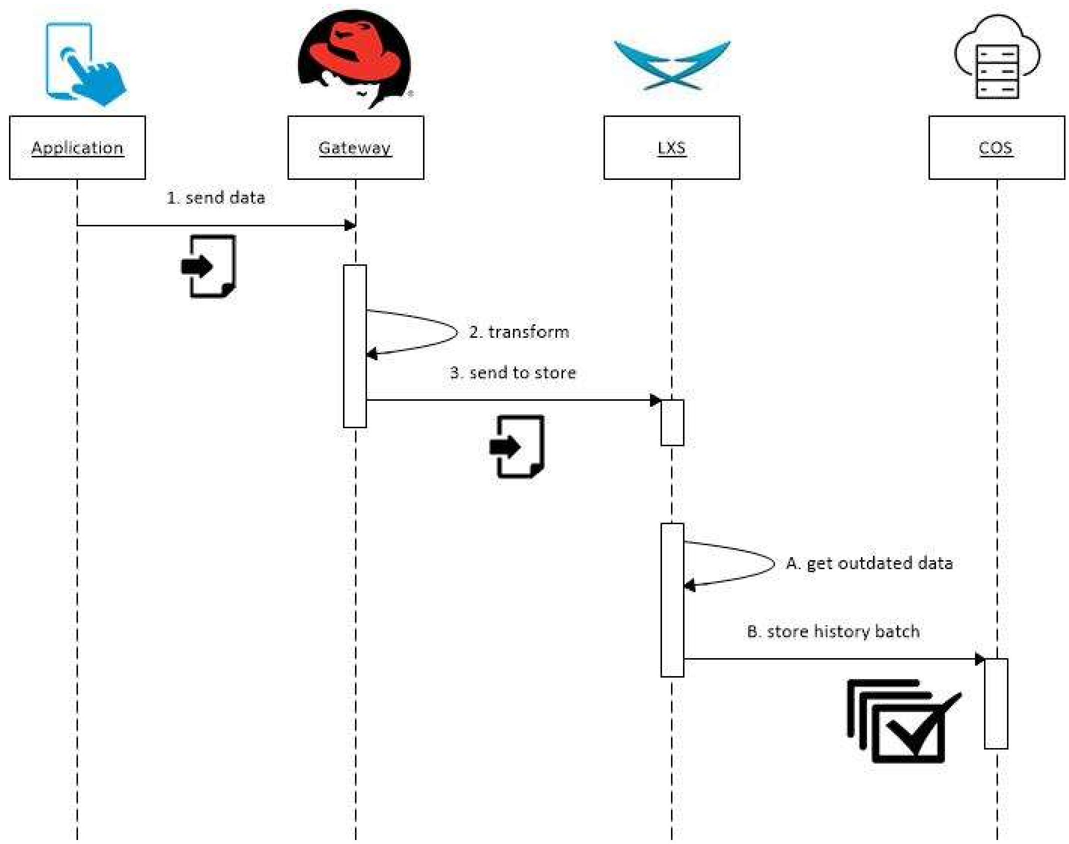


Figure 46 - Inserting data

An integrated application produces data to be stored in the BigDataStack platform. The data are being sent to the *Gateway*: the entry point of the platform. Its responsibility is to transform data coming from external sources in various formats, to the platform's internal schema. Then, it forwards the data to the operational data store to permanently store them. The latter periodically moves data that has been inserted from more than a constant period of time, to the COS.

Insert streaming data to BigDataStack

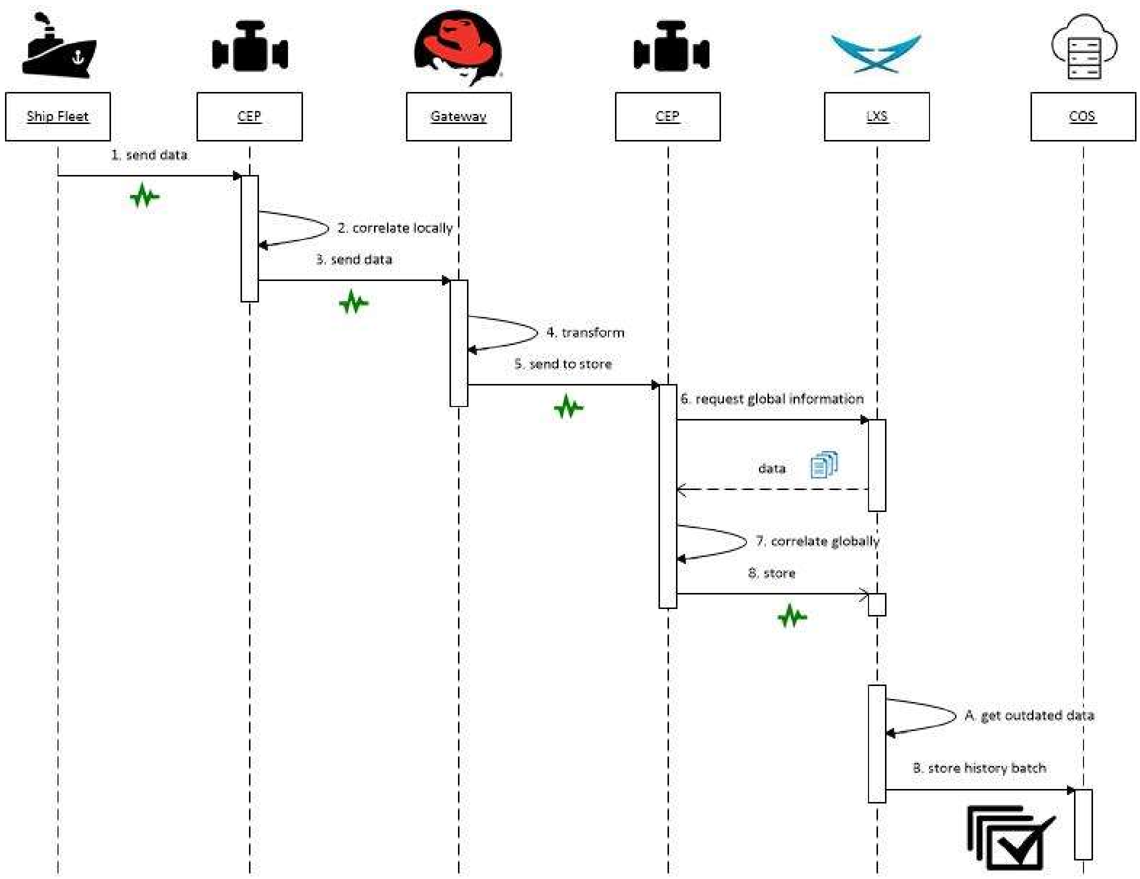


Figure 47 - Inserting streaming data

In this specific use case, a ship from the DANAOS fleet streams data coming from one of its sensors. Data is being first sent to a local installation of the CEP which correlates them and identifies possible threats, producing alerts. Then, data is sent to the platform´s Gateway which is responsible of transforming the data to the platform's internal format. A CEP cluster inside the platform receives data from the Gateway. It further analyses data to detect possible rules infringement. Data coming from all the fleet vessels is merged. This second CEP cluster processing involves querying LXS to retrieve data in rest that has been already been stored in the data store. Finally, it stores the incoming data to the relational datastore which eventually will move the data to the Object Store.

7.4. Monitoring & Runtime Adaptations

When considering the process of monitoring and adapting user applications on the cloud, it is useful to divide the discussion into three parts: 1) the interactions required to perform the actual monitoring of a running application; 2) how this monitoring process can be used to track quality of service; and 3) the interactions needed to adapt the user's application to some new configuration when a quality of service deficiency is identified or predicted. We summarize each below.

7.4.1. Triple Monitoring Engine

The triple monitoring system provides APIs for receiving metrics from different sources and exposes them for consumption. Metrics are obtained mainly by exporters and federation. In the case of the deployment of an exporter is impossible for some reason, the monitoring engine implements a system that can receives metrics by get and post methods and exposes them to Prometheus. This component of the triple monitoring is expected to behave as a REST API and Prometheus exporter. The following diagram describes its functionality.

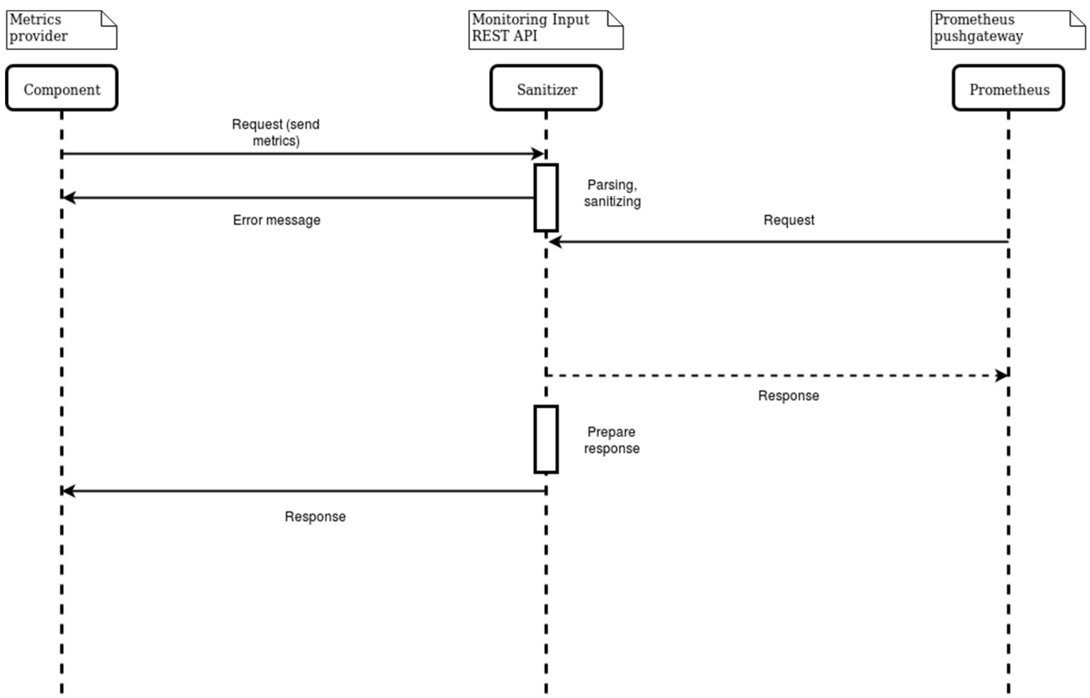


Figure 48 - Prometheus exporters

An application provider sends its metrics in JSON format by http get or post, the API parses the json structure, sanitizes metrics to convert them to Prometheus's format and saves them in a temporally list. A response is then returned to the application provider. The Prometheus engine scrapes the REST API by http get metrics, to get available metrics. This scraping operation is iteratively performed at intervals based on the amount of time specified in the Prometheus configuration.

The triple monitoring engine implements two different exposition system methods. The first is a REST API where applications consumers ask for a metric, the REST API translates this request to an Elasticsearch query and returns a result. The following sequence describes this process.

bigdatastack.eu

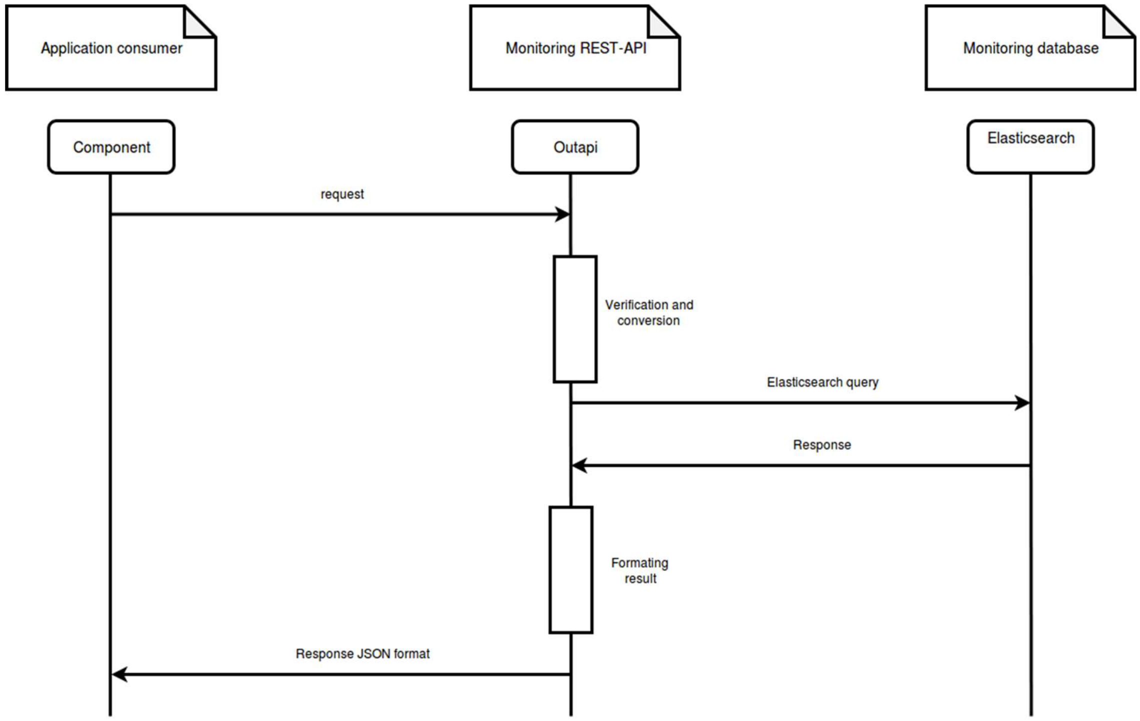


Figure 49 - Prometheus REST API

The second output interface implemented in the triple monitoring system is the publish/subscription mechanism.

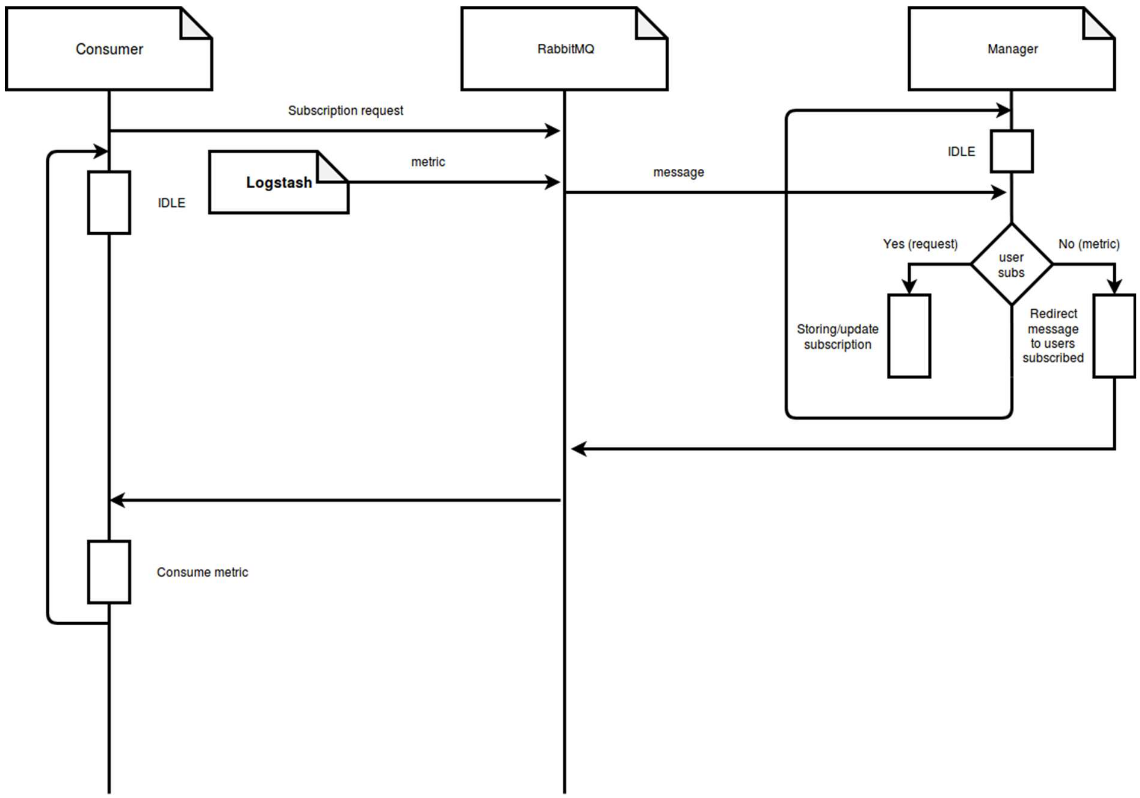


Figure 50 - Publish/subscription mechanism

An application that needs steaming data can through this component subscribe and receive metrics in real-time. Four different types of requests are available.

- The first request type is the “subscription”, the consumer after having created his queue, it is going to send to the pub/sub system a subscription request that contains the name of its queue, its name (application name) and a list a metrics. The consumer sends its request in the “manager” queue so that to be consumed by the manager of the triple monitoring system. The manager receives the subscription request, creates a subscription object and adds it into the subscription list. A confirmation message is then returned to the consumer. The manager reads the subscription list each time it receives a metric from its queue, it redirects this metric to the declared queue.
- The second request is the “add_metrics” request type, the consumer sends a message that contains its name, queue name and a metric to add to its subscription list, the manager verifies the request, updates the subscription and returns a message.
- The third request type is “my_subscription”, the consumer sends its name and queue name. The manager returns the corresponding subscription list.
- The last request is the heart_beat, the manager has no way to detect disconnection by a consumer. The consumer should confirm its presence each specific interval of time. The heart_beat interval is declared in the subscription request.

7.4.2. Quality of Service (QoS) Evaluation

QoS properties (parameters) to be evaluated by the QoS Evaluation component should correspond to the kind of quality of service (QoS) requirements coming from the Application Dimensioning Workbench and defined within the BigDataStack Playbook.

- An example of a QoS requirement is the "throughput."
- There should be a trivial mapping between Playbooks' KPIs and the "guaranteed" of "agreements".

The QoS Evaluation component will be responsible for translating the Playbooks' QoS requirements into SLOs (Service Level Objectives).

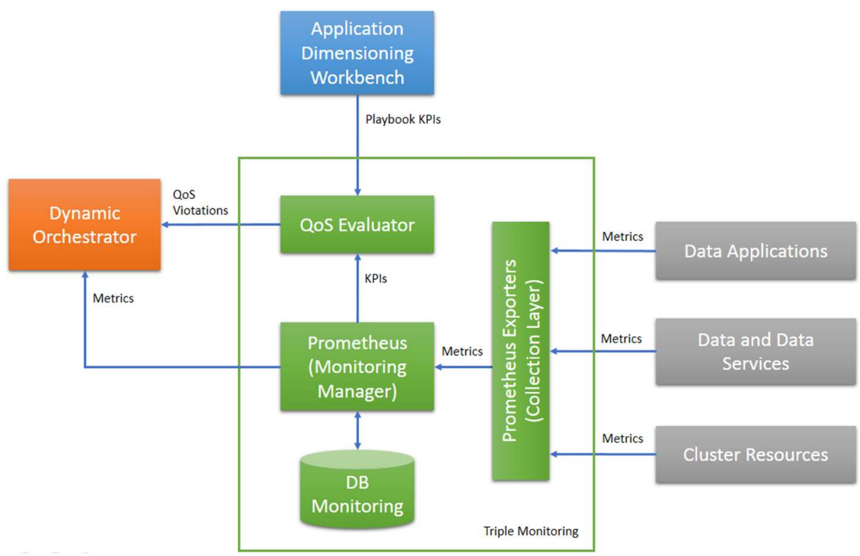


Figure 51 - QoS Evaluation component

The QoS Evaluation component will periodically query the Triple Monitoring Engine (based on Kubernetes) to recover the metrics related to the monitored QoS parameters.

Once a violation of a given SLO is detected, a notification is sent to the Dynamic Orchestrator to trigger the data-driven orchestration of application components and data services. The standard sequence of interactions will be the following:

- Evaluator calls the Adapter to recover a certain set of QoS metrics from Prometheus.
- The Evaluator calls the Notifier when an SLO violation is detected.
- Notifier calls the Dynamic Orchestrator passing a message describing the violation through publisher/subscriber mechanism implemented as a topic within the RabbitMQ service (which acts as the message broker between BigDataStack components)

The Dynamic Orchestrator communicates with the ADS-Ranking component to trigger the dynamic adaptation (re-configuration) of the application or data service deployment patterns.

Adapting at Runtime

If a user's application is identified or predicted to have some deficiency with respect to the quality of service targets, then that application's configuration needs to be altered to correct for this. For instance, this might involve moving data closer to the machines performing the computation to reduce IO latency, or in more extreme cases it might require the complete re-deployment of the user's application on new more suitable hardware. BigDataStack supports a range of adaptations that might be performed , such as Pattern Re-Deployment, where the goal is to select an alternative candidate deployment pattern (hardware configuration) after the user's application has been deployed. This is used in cases where the original deployment pattern was deemed unsuitable and this could not be rectified without changing the deployment infrastructure. In this case, a new candidate deployment pattern will be chosen, and the application services will be transitioned to this new configuration. This may result in application down-time as services are moved.

The components involved for this adaptation are the Dynamic Orchestrator (DO) and the Triple Monitoring. When a new application is deployed, the Playbook is sent to the DO on the queue OrchestratorPlaybook. The DO reads the playbook and enriches it, adding more information about the SLOs: it splits the values of the metrics related to SLOs in different intervals that the QoS component will monitor, e.g. response time can be divided in the intervals 0.5-1s, 1-1.5s, etc. In addition, the DO subscribes to the Triple Monitoring Engine and creates a new queue, using which it will consume the metrics from the application.

The Enriched Playbook is sent to the QoS Evaluator on the queue EnrichedPlaybook. The QoS registers this and will start monitoring the application to detect when an SLO is violated, and in this case, a message will be sent to the DO on the queue OrchestratorQOSFeed. The DO will read this message and based on the current state (as defined by the metrics consumed from the Triple Monitoring Engine, the QoS information and its experience), will decide what is the most likely action to resolve the violation is and subsequently send it to the ADS-Ranker on queue Lv3-ADSRanking-RR to start adaptation.

In the remainder of this section we provide more detail on how Pattern Re-Deployment is operationalized within BigDataStack.

Pattern Re-Deployment

The aim of the pattern re-deployment task is to facilitate the selection of a new candidate deployment pattern (CDP) if a previously selected CDP is no longer considered viable. This might occur if a deployed application fails to meet minimum service requirements and this cannot be resolved through data service manipulation. In this case, we need to take into account why the current pattern is failing and based on that information, re-rank the CDPs for the user application and select a new alternative that will provide better performance. This new CDP can then be used to transition the user's application to the new configuration by the Application and Data Services Deployment component.

This task is triggered by the Dynamic Orchestrator when the orchestrator detects that an application deployment is failing. It sends a notification to the Application and Data Services Ranking component. More precisely, this notification is processed by the Failure Encoder sub-component. This component first contacts the Global Decision Tracker to retrieve the other

CDPs that were not selected for the failing user's application (as it is from these that a new pattern will be selected). These patterns are then sent into the same process pipeline as for first-time ranking (see Section 6.5), with the exception that the previously selected deployment is excluded (we know that it is insufficient) and the Pattern Selector sub-component will also consider the reason that the previously selected CDP failed.

When the ADS-Selector chooses the new CDP, this information is sent to the ADS-Deploy, together with the instruction to redeploy. Then, the deployment component translates the CDP, and communicates it to the container orchestrator using the same process as defined in Section 6.5. The orchestrator will then start a re-dimensioning process. If the process is successful, then the user's process continues normally. However, if the re-dimensioning was unsuccessful, then the container orchestrator needs to destroy the current deployment, stopping the processes and starting a new deployment from scratch. This situation has the setback that users have their processes interrupted and/or restarted and ultimately impair the availability of application and data services (downtimes).

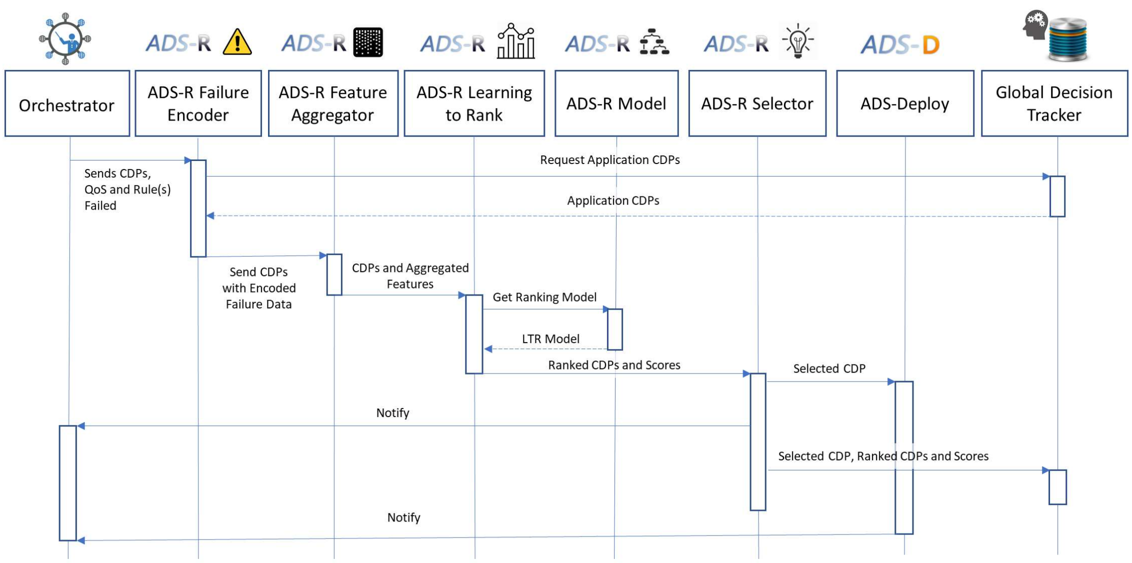


Figure 52 - Interaction Diagram for CDP Re-Ranking

8. Conclusions

This document refines the initial version of the BigDataStack architecture presented in deliverable D2.4 - Conceptual model and Reference architecture. It captures the updated version of the overall conceptual architecture in terms of information flows and capabilities provided by each one of the main building blocks. Additional refinements for each component are also detailed on the corresponding sections, as well as the changes in the main interactions between them.

This report serves as a design documentation for the individual components of the architecture (which are further specified and detailed in the corresponding WP-level scientific reports) and presents the outcomes (in terms of design) of the initial integrated prototypes and the obtained experimentation and validation results.

BY NC ND

9. References

[1] "OpenShift Origin Kubernetes Distribution," [Online]. Available: http://www.openshift.com.

[2] J. Duncan and J. Osborne, "Chapter 1. Getting to know OpenShift," in *OpenShift in Action*, Manning Publications Co. ISBN 978-1-6172-9483-9, 2018.

[3] "How Deployments Work," [Online]. Available: https://docs.okd.io/latest/dev_guide/deployments/how_deployments_work.html.

[4] RedHat, "What's New in Red Hat OpenShift Origin 3.10 OpenShift Commons Briefing," [Online]. Available: https://blog.openshift.com/wp-content/uploads/Whats-New-in-Origin-3.10.pdf.

[5] "Geard," [Online]. Available: https://openshift.github.io/geard.

[6] "Project Atomic," [Online]. Available: https://www.projectatomic.io/.

[7] RedHat. [Online]. Available: Architecture OpenShift Container Platform 3.6 Architecture Information", https://access.redhat.com/documentation/en-us/openshift_container_platform/3.6/pdf/architecture/OpenShift_Container_Platform-3.6-Architecture-en-US.pdf.

[8] "User Requirements Notation," [Online]. Available: https://www.itu.int/rec/T-REC-Z.151-201210-I/en.

[9] BigDataStack, "D2.1 – State of the art and Requirements analysis - I," 2018.

[10] O. A. García, "Requirements & State-of-the-Art Analysis II," EC Deliverable, 2018.

[11] Kubernetes Authors, "Concepts: Pods," https://kubernetes.io/docs/concepts/workloads/pods/pod/, 2019.

[12] T.-Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval,* vol. 3, no. 3, pp. 225--331, 2009.

[13] "Apache Flink: Scalable batch and stream data processing," 2016. [Online]. Available: https://flink.apache.org/.

[14] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente and P. Valduriez, "Streamcloud: An elastic and scalable data streaming system," in *IEEE Transactions on Parallel and Distributed Systems*, 2012.

[15] "Apache Storm: Distributed and fault-tolerant realtime computation," 2013. [Online]. Available: http://storm.apache.org.

[16] "Apache Spark: Apache Spark: Lightning-fast cluster computing.," 2016. [Online]. Available: spark.apache.org/streaming.

[17] "Workflow Patterns Initiative," [Online]. Available: http://www.workflowpatterns.com/R.

[18] R. Raschke, "Process-based view of agility: The value contribution of IT and the effects on process outcomes," *International Journal of Accounting Information Systems,* vol. 11.4, pp. 297-313, 2010.

[19] "Node-RED," [Online]. Available: https://nodered.org/.

[20] "Data-Driven Documents," [Online]. Available: https://d3js.org/.

[21] "Flow-Based Programming for JavaScript - NoFlo," [Online]. Available: https://noflojs.org/.

[22] R. L. Raschke, "Process-based view of agility: The value contribution of IT and the effects on process outcomes," *International Journal of Accounting Information Systems,* vol. 11, no. 4, pp. 297-313, 2010.

Appendix 1 – Real-time Ship Management use case dataset structure and description

It should be noted that given the data schemas described below, the DANAOS datasets do not have any GDPR-related aspect.

TELEGRAMS table structure (14 attributes)

id: Telegram id,

vessel_code: The id of the vessel,

telegram_date: Telegram timestamp (UTC),

type: Telegram type: D:Departure, A:Arrival, N:Noon-telegram,

total_teus: Total Twenty-foot Equivalent Unit (TEU) (# of containers)

total_feus: Total Fourty-foot Equivalent Unit (FEU) (# of containers)

cons_ifo_static_counter: sensor-based measurement TEUs

cons_ifo_static1_counter: sensor-based measurement of FEUs,

draft_aft: Vessel draft at stern (m),

draft_fore: Vessel draft at fore (m),

sea_temperature: Sea temperature (°C),

port_name: Current port name,

next_port: The name of the next port,

eta_next_port: ETA to the next port

VESSEL_DATA table structure (23 attributes)

vessel_code: Vessel id,

datetime: Timestamp of the measurement (UTC),

power: Consumed power (kW),

apparent_wind_speed: Wind-speed (kn),

speed_overground: GPS speed (kn),

stw_long double precision: Speed through water – longitudinal (kn),

stw_trans double precision: Speed through water – transverse (kn),

rpm: rotations per minute of the main shaft,

apparent_wind_angle: Wind angle (0-359.99 degrees),

BY NC ND

total_teus: Total Twenty-foot Equivalent Unit (TEU) (# of containers),

total_feus: Total Fourty-foot Equivalent Unit (FEU) (# of containers),

cons_ifo_static_counter: Low-sulfur fuel oil consumption (metric tones),

cons_ifo_static1_counter: High-sulfur fuel oil consumption (metric tones),

port_mid_draft: Vessel draft at port-side (left-side looking to the fore) (m),

stbd_mid_draft: Vessel draft at starboard-side (right-side looking to the fore) (m),

draft_aft: Vessel draft at stern (m),

draft_fore: Vessel draft at fore (m),

stw: Speed through water – calculated by stw_trans and stw_lon (kn),

equivalent_teus: Total number of containers,

mid_draft: Vessel draft at mid-line (m),

trim: The trim of the vessel, calculated by draft_aft and draft_fore,

latitude: The latitude of the vessel's position,

longitude: The longitude of the vessel's position,

MAIN_ENGINE_DATA table structure (102 attributes)

vessel_code: The id of the vessel,

datetime: Timestamp of measurement in UTC,

airCoolerCWInLETPress: Air Cooler Cooling Water Inlet Pressure (Pa)

scavAirFireDetTempNo1: Cyllinder #1 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo2: Cyllinder #2 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo3: Cyllinder #3 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo4: Cyllinder #4 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo5: Cyllinder #5 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo6: Cyllinder #6 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo7: Cyllinder #7 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo8: Cyllinder #8 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo9: Cyllinder #9 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo10: Cyllinder #10 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo11: Cyllinder #11 Scavenge Air Fire Detection Temperature (°C),

scavAirFireDetTempNo12: Cyllinder #12 Scavenge Air Fire Detection Temperature (°C),

coolerCWinTemp: Air Cooler Cooling Water Inlet Temperature (°C)

cfWInPress: Cooling Fresh Water Inlet Pressure (Pa), controlAirPress: Control Air Pressure (Pa),

cylLoTemp: Cylinder Lube Oil Temperature (°C)

exhVVSpringAirInPress: Exhaust Valve Spring Air Inlet Pressure (Pa)

foFlow: Fuel Oil Flowrate (lt),

foInPress: Fuel Oil Inlet Pressure (Pa),

foInTemp: Fuel Oil Inlet Temperature (°C),

hfoViscocityHighLow: Heavey Fuel Oil Viscosity High Low (mm2/s)

hpsBearingTemp: HPS Bearing Temperature (°C),

jcfWInTempLow: Jacket Cooling Fresh Water Inlet Temperature Low (°C)

cylExhGasOutTempNo1: Cyllinder #1 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo2: Cyllinder #2 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo3: Cyllinder #3 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo4: Cyllinder #4 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo5: Cyllinder #5 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo6: Cyllinder #6 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo7: Cyllinder #7 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo8: Cyllinder #8 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo9: Cyllinder #9 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo10: Cyllinder #10 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo11: Cyllinder #11 Exhaust Gas Out Temperature (°C),

cylExhGasOutTempNo12: Cyllinder #12 Exhaust Gas Out Temperature (°C),

cylJCFWOutTempNo1: Cyllinder #1 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo2: Cyllinder #2 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo3: Cyllinder #3 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo4: Cyllinder #4 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo5: Cyllinder #5 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo6: Cyllinder #6 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo7: Cyllinder #7 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo8: Cyllinder #8 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo9: Cyllinder #9 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo10: Cyllinder #10 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo11: Cyllinder #11 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylJCFWOutTempNo12: Cyllinder #12 Jacket Cooling Fresh Water Outlet Temperature (°C),

cylPistonCOOutTempNo1: Cyllinder #1 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo2: Cyllinder #2 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo3: Cyllinder #3 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo4: Cyllinder #4 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo5: Cyllinder #5 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo6: Cyllinder #6 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo7: Cyllinder #7 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo8: Cyllinder #8 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo9: Cyllinder #9 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo10: Cyllinder #10 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo11: Cyllinder #11 Piston Cooling Outlet Temperature (°C),
cylPistonCOOutTempNo12: Cyllinder #12 Piston Cooling Outlet Temperature (°C),
tcExhGasInTempNo1: Turbo-Charger #1 Exhaust Gas Inlet Temperature (°C)
tcExhGasInTempNo2: Turbo-Charger #2 Exhaust Gas Inlet Temperature (°C),
tcExhGasInTempNo3: Turbo-Charger #3 Exhaust Gas Inlet Temperature (°C),
tcExhGasInTempNo4: Turbo-Charger #4 Exhaust Gas Inlet Temperature (°C),
tcExhGasOutTempNo1: Turbo-Charger #1 Exhaust Gas Outlet Temperature (°C),
tcExhGasOutTempNo2: Turbo-Charger #2 Exhaust Gas Outlet Temperature (°C),
tcExhGasOutTempNo3: : Turbo-Charger #3 Exhaust Gas Outlet Temperature (°C)
tcExhGasOutTempNo4: Turbo-Charger #4 Exhaust Gas Outlet Temperature (°C)
tcLOInLETPressNo1: Turbo-Charger #1 Lube Oil Inlet Pressure (Pa),
tcLOInLETPressNo2: Turbo-Charger #2 Lube Oil Inlet Pressure (Pa),
tcLOInLETPressNo3: Turbo-Charger #3 Lube Oil Inlet Pressure (Pa),
tcLOInLETPressNo4: Turbo-Charger #4 Lube Oil Inlet Pressure (Pa),
tcLOOutLETTempNo1: Turbo-Charger #1 Lube Oil Outlet Pressure (Pa),
tcLOOutLETTempNo2: Turbo-Charger #2 Lube Oil Outlet Pressure (Pa),
tcLOOutLETTempNo3: Turbo-Charger #3 Lube Oil Outlet Pressure (Pa),
tcLOOutLETTempNo4: Turbo-Charger #4 Lube Oil Outlet Pressure (Pa),
tcRPMNo1: Turbo-Charger #1 RPMs,
tcRPMNo2: Turbo-Charger #2 RPMs,
tcRPMNo3: Turbo-Charger #3 RPMs,
tcRPMNo4: Turbo-Charger #4 RPMs,
orderRPMBridgeLeverer: Order RPM (Bridge Lever)

rpm: Rotations per minute of the main shaft

scavAirInLetPress: Scavenge Air Inlet Pressure (Pa),

scavAirReceiverTemp: Scavenge Air Receiver Temperature (°C),

startAirPress: Starting Air Pressure (Pa),

thrustPadTemp: Thrust Pad Temperature (°C),

mainLOInLetPress: Main Lube Oil Inlet Pressure (Pa),

mainLOInTemp: Main Lube Oil Inlet Temperature (°C)

foTemperature: Fuel Oil Temperature (°C)

foTotVolume: Fuel Oil Total Volume (lt)

power: Consumed power (kW),

scavengeAirPressure: Scavenge Air Pressure (Pa)

torque: Torque of the main shaft (N/m),

coolingWOutLETTempNo1: Turbo-Charger #1 Air Cooler Cooling Water Outlet Temperature (°C),

coolingWOutLETTempNo2: Turbo-Charger #2 Air Cooler Cooling Water Outlet Temperature (°C),

coolingWOutLETTempNo3: Turbo-Charger #3 Air Cooler Cooling Water Outlet Temperature (°C),

coolingWOutLETTempNo4: Turbo-Charger #4 Air Cooler Cooling Water Outlet Temperature (°C),

foVolConsumption: Fuel Oil Consumption (lt/min)

VESSEL_DAMAGES table structure (5 attributes)

vessel_code: The id of the vessel,

defect_type: Type of damage (Main Bearing, Crosshead Bearing, Crankpin Bearing)

defect_details: Short description of damage

date_of_damage: Date of damage

cause_of_damage: Short description for cause of damage

Appendix 2 – Connected Consumer use case dataset structure and description

Introduction

This document aims at describing the main entities to be used in the implementation of the recommender system that is going to be developed in the retailer use-case of the project BigDataStack.

Having pre-analysed a sample of raw data coming from our partner Eroski, a selection of the most relevant attributes that are candidates to be used during the build of the predictive model has been done. These selected attributes are the ones contained in this document.

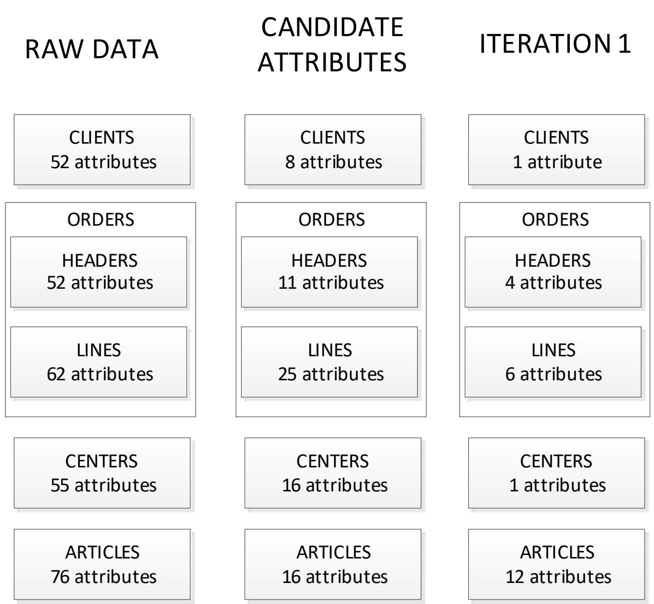


Figure 53 - Dataset structure and description

The dataset contains information about EROSKI clients. However, GDPR aspects have been taken into account before sharing the data with the consortium. Concretely:

- The only data that could be used to uniquely identify a person related to the field "ID_CLIENTE".
- ID_CLIENTE is an internal identifier of the database of EROSKI that is not known by the customers. I.e. only a person with access to the database of EROSKI could identify the customer from ID_CLIENTE.
- ID_CLIENTE has been encrypted by EROSKI with an SHA-1 algorithm. Encryption has been done before providing the dataset to BigDataStack consortium. A SHA-1 (168 bits) algorithm has been used for encryption of ID_CLIENTE.
- For each ID_CLIENTE, SHA-1 has been applied to "string_1"+ID_CLIENTE+"string_2". String_1 and string_2 are alphanumeric that contain capital and non-capital letters, numbers and special characters. These 2 values are only known by EROSKI.

The attributes for each entity have been included in this section.

CLIENTS table structure (21 attributes)

ID_CLIENTE: Client id,

TIPO_CLIENTE_ORO: Type of gold client

FLG_CLIENTE_APP: Flag if the client is an app client or not,

FLG_CLIENTE_WEB: Flag if the client is a web client or not,

FLG_CLIENTE_NUTRICIONAL: Customer shows interest in healthy products

FRANJA_GASTO_ORO_INICIAL: Initial Range of expenditure

POSIBLE_VALOR_ORO: Percentage indicating the discount given to the customer for being a gold customer

CLIENTE_1000_ORO: Flag indicating whether the client is 1000 Oro or not

FRANJA_GASTO_ORO_ACTUAL: Current Range of expenditure

TIPO_MADUREZ: Type of maturity of the client

DESC_SEG_C_CLIENTE: Description of the type of maturity of the client

DESC_SEG_G_FIDELIDAD: Segmentation of the customer according to his loyalty

DESC_INTERES_AHORRO: Segmentation of the customer according to his interest in promotions

DESC_INTERES_FRESCOS: Segmentation of the customer according to his interest in fresh food

DESC_INTERES_LOCAL: Segmentation of the customer according to his interest in local food

DESC_INTERES_SALUD: Segmentation of the customer according to his interest in healthy food

DESC_INTERES_SALUD_DETALLE: additional detail on which type of healthy food the customer is interested in

DESC_MISION_COMPRA: description of the purchase mission of the customer

DESC_SEG_SEC: segment description

DESC_SEG_SOCIODEMO: Socio-demographic segment of the client.

COD_LOC: preferred store

TICKETS (36 attributes)

ID_CLIENTE: Client id,

COD_LOC: Store's localization id,

DIA: Day,

COD_CAJA: Till id,

CC BY NC ND

NUM_TICKET: Ticket number (id),

NUM_LINEA: Line number (id),

COD_TIPO_MOVIM: Movement type,

HORA_EMISION: Timestamp of tickets emission,

COD_TIPOMARCA_HIST: Type of brand of the product

COD_F_PAGO_DET -> M_FORMA_PAGO: Type of payment procedure,

UNID_VENTA_TARIFA: Total amount of items sold in tariff's type,

UNID _VENTA_OFERTA: Total amount of items sold in offer's type,

UNID _VENTA_COMPETE: Total amount of items sold in competence's type,

UNID _VENTA_LIQUID: Total amount of items sold in liquidation's type,

UNID _VENTA_CAMPANA: Total amount of items sold in campaign's type,

IMP_VENTA_TARIFA: Total economic amount of the items sold by tariff's type,

IMP_VENTA_OFERTA: Total economic amount of the items sold by offer's type,

IMP_VENTA_COMPETE: Total economic amount of the items sold by competence's type,

IMP_VENTA_LIQUID: Total economic amount of the items sold by liquidation's type,

IMP_VENTA_CAMPANA: Total economic amount of the items sold by campaign's type,

IMP_DTO_CONSUMER: Discount amount applied for using VISA Eroski,

IMP_DTO_TRAVEL: Discount amount applied for using loyalty card Travel Club,

IMP_DTO_COUPON: Discount amount applied for the usage of coupons,

IMP_DTO_CUOTA: Discount amount applied for being member of EROSKI Club,

IMP_DTO_ONSITE: Discount amount applied after redemption of loyalty Travel points,

IMP_DTO_OTROS: Other discounts,

IMP_DTO_VALE: Amount of discounts coming from the redemption of a supplier coupon,

IMP_CONSUMO_RAP: Special discount applied in the shop,

COD_ART: Article's id,

FLG_TECLA: information about whether the product has been sold by a direct key or not

ANO_OFERTA: year of the offers applied to the order

COD_OFERTA: offer code

COD_TIPO_CENTRO: type of shop (primary/secundary)

FLG_SCANNER: has the product been scanned during the purchase (Y/N)

IMP_PVP_TARIFA: amount of the order if all of the items had been charged to the customer with catalogue prices

CENTERS structure (55 attributes)

COD_LOC: Store's localization id,

COD_PROVIN: Province id,

DESC_LOC: Center's description,

DESC_PROVIN: Province's name,

FLG_PLATAF : Indicator of distribution platform,

FEC_MODIF: Date of last modification,

COD_ZONA: Zone id,

DESC_ZONA: Zone description,

COD_REGION: Region id,

DESC_REGION: Region description,

COD_AREA: Area id,

DESC_AREA: Area's description,

COD_ENSENA: Type of center id,

DESC_ENSENA: Type of center description (Eroski City, Eroski Center…),

COD_NEGOCIO: Store's id,

DESC_NEGOCIO: Store's type,

COD_SOCIEDAD: Type of company,

DESC_SOCIEDAD: Company's description,

COD_GAMA_OBLIG: Code of mandatory catalogue,

COD_FINANZIA: financing code,

DESC_DIRECCION: address,

DESC_POBLACION: location,

FLAG_CUOTA: quota flag,

FEC_INI_LOC: opening date,

FEC_FIN_LOC: closing date,

NUM_CAJAS: number of boxes,

NUM_M2: squared meters of the store,

NUM_M_LINEA: linear meters,

COD_LOC_AME: store code in AME system,

COD_TP_LOC: type of location,

DESC_TP_LOC: description of the type of location,

COD_LOC_PADRE: father location code,

COD_MUNICIPIO: location code,

COD_TP_POTENCIAL: type of potential code,

FEC_ULT_APERTURA : last opening date,

COD_POSTAL: zip code,

COD_AGR_IMP: grouping code,

FLG_CECO_MODELO_COSTES: cost model flag,

LATITUD: latitude,

LONGITUD: longitude,

COD_ISLA: ISLA code,

FLG_LEAN: lean flag,

FLG_TRANSFORMADO: transformed flag,

FLG_PUESTA_PUNTO_PLUS: tunning flag,

COD_NIVEL_ESTR_LOC: code of local structure of sales of the center,

COD_N1: code of the level 1 of the structure of sales of the center,

DES_N1: description of the level 1 of the structure of sales of the center,

COD_N2: code of the level 2 of the structure of sales of the center,

DES_N2: description of the level 2 of the structure of sales of the center,

COD_N3: code of the level 3 of the structure of sales of the center,

DES_N3: description of the level 3 of the structure of sales of the center,

COD_N4: code of the level 4 of the structure of sales of the center,

DES_N4: description of the level 4 of the structure of sales of the center,

COD_N5: code of the level 5 of the structure of sales of the center,

DES_N5: description of the level 5 of the structure of sales of the center,

PRODUCTS structure (79 attributes)

COD_ART: product id,

DESC_ART: product description,

FLG_TECLA: exists a direct key to sell the product or not,

COD_TIPOMARCA: type of brand code,

DESC_TIPOMARCA: description of the type of brand code,

COD_N1_PPAL: Area’s id,

DESC_N1: Area’s description,

COD_N2_PPAL: Section’s id,

DESC_N2: Section’s description,

COD_N3_PPAL: Category's id,

DESC_N3: Category's description,

COD_N4_PPAL: Subcategory's id,

DESC_N4: Subcategory's description,

COD_N5_PPAL: Segment's id,

DESC_N5: Segment's description,

FEC_INI_ART: Article start time,

FEC_FIN_ART: Article finishes time,

COD_FORMATO: Format id (KG, Gr, Unities...),

COD_MARCA: Brand's id,

COD_EAN: EAN code,

COD_TALLA: Size code,

DESC_TALLA: Size code description,

COD_COLOR: Colour code,

DESC_COLOR: Colour code description,

COD_PACK : Number of items per pack,

COD_BLOQUEO: has the product blocked for the sales?,

COD_ENS_EROSKI: commercial codification in the Hypermarket,

COD_ENS_CONSUM: commercial codification in the SUPERmarket,

COD_TIPO_FORMATO: unit of measurement (related to COD_FORMATO),

COD_ART_PRIM: father product code,

COD_TIPO_MARCA2: code of EROSKI Brand (only for products belonging to a EROSKI brand)),

DESC_TIPO_MARCA2: description of EROSKI Brand (only for products belonging to a EROSKI brand)),

FEC_ULT_BLOQ: date on which the product was blocked for the sales,

COD_PORCI_CONS: product has info for the consumer related to the number of portions,

DESC_PORCI_CONS: indicator about whether the product has a description for the portions,

CC_CAPRABO: Comercial code of CAPRABO,

COD_CATEGORI_HIP: Category code hypermarket,

DESC_CATEGORI_HIP: Description of the Hypermarket Category,

COD_CATEGORI_SUP: Category code supermarket,

DESC_CATEGORI_SUP: Description of the supermarket Category,

COD_SENSIBI_HIP: SENSIBI code hypermarket, DESC_ SENSIBI HIP: Description of the SENSIBIcode of the hypermarket,

COD_ SENSIBI SUP: Category code supermarket,

DESC_ SENSIBI SUP: Description of the SENSIBIcode of the supermarket,

FLG_COMPRA: indicator about whether the product is for purchasing,

FLG_VENTA: indicator about whether the product is for sales,

COD_FAMILIA: family of the product,

DESC_FAMILIA: description of the family of the product,

COD_AMBITO_EROSKI: Scope code of the product in the hypermarkets,

DESC_AMBITO_EROSKI: Description of the scope of the product in the hypermarkets,

COD_AMBITO_CONSUM: Scope code of the product in the supermarkets,

DESC_AMBITO_CONSUM: Description of the scope of the product in the supermarkets,

COD_CODMARCA: brand code (related to COD_MARCA)

FLG_MMPP: Does the product belong to a EROSKI brand?,

COD_POSICION_MARCA: Maker brand / EROSKI Brand code,

DESC_POSICION_MARCA: Description of the code of maker Brand / EROSKI Brand code,

FLG_SALUD_BIENESTAR: health indicator,

FLG_INNOVACION: innovation indicator,

FLG_GAMA_TURISTICA: tourism product,

FLG_PODER_ADQUISITIVO: indicator about product for customer with a high purchasing power,

FLG_BLOQ_DEFINITIVO: Product definitely blocked,

COD_SUBMARCA: sub-brand code,

DESC_SUBMARCA: sub-brand description

FLG_GAMA_LOCAL: local product,

FLG_GAMA_REGIONAL: regional product,

FLG_PESO_SGA: flag product by weight,

FLG_LIQUIDABLE: flag payable,

FLG_EXDEPRECIACION: depreciation flag,

COD_TP_ART: product type,

DESC TIPO_ARTICULO: description of the product type,

CANTIDAD: number of ítems per lot,

FEC_LANZAM: launch date,

PORC_IVA: VAT rate,

COD_PROVR_GEN: code of generic supplier,

COD_PROVR_TRABAJO: code of work supplier,

NOMBRE: name of the work supplier,

PESO: weight (in grams),

PESO_NETO: net weight (in grams),

VOLUMEN: volume (in cm3)

Appendix 3 – Smart Insurance use case dataset structure and description

The datasets provided by the Insurance Company (customer of GFT) are described in the following in terms of tables and records structure and description.

Following the GDPR directive, all sensitive information of the datasets have been anonymized. For the encryption, we used a cryptographic hash function, the MD5 algorithm. It is a unidirectional function different from coding and encryption because it is irreversible. The spread of this encryption algorithm is still widespread (just think that the most frequent integrity check on file is based on MD5). This function takes as input an arbitrary length string and outputs another 128 bit output. The process happens very quickly and the output (also known as "MD5 Checksum" or "MD5 Hash") returned is such that it is highly unlikely to obtain the same hash value in output with two different input strings.

We have modeled the length of the encrypted string, based on the length of the field to be encrypted. For example, for the tax code the encrypted string is 16 characters, while for the license plate it is 8 characters. This eliminates the possibility of tracing back to the initial value. We have performed several decrypting tests present on numerous online sites and no one has been able to decrypt the string entered.

Furthermore, we have carried out a univocal check of all the encrypted keys, so that the possibility of two different string yielding identical encrypted strings is excluded.

In the following, the datasets tables and records are described. The fields highlighted in blue have been anonymized as explained above.

ana

****

id_univoco_anagrafica	string	Flow unique identifier: REGISTRY
id_univoco_master	string	
codice_fiscale	string	Subject unique identifier
tipo_anagrafica	string	Registry type (P = person, N = company)
cognome	string	Surname / company name
nome	string	Name
sesso	string	Gender (M=male, F=female, N=company)
pubblica_amministrazione	string	Public Administration (YES/NO)

ana_ptf

****

codice_fiscale	string	Subject unique identifier
idpolizza	string	Policy unique identifier
ruolo	string	Subject role
cognome	string	Surname / company name
nome	string	Name

ana_sin

****

id_univoco_anagrafica	string	Flow unique identifier: REGISTRY
id_univoco_master	string	
codice_fiscale	string	Subject unique identifier
idsinistro	string	Claim unique identifier
ruolo	string	Subject role
cognome	string	Surname / company name
nome	string	Name

ana_vei

****

codice_fiscale	string	Subject unique identifier
targa	string	License plate
cognome	string	Surname / company name
nome	string	Name

anaage

****

codice_fiscale	string	Subject unique identifier
agenzia	string	Agency ID
descrizione	string	Description

bigdatastack.eu

anaaia

****

codice_fiscale	string	Subject unique identifier
codice_anomalia	string	Anomaly identifier

anabds

****

codice_fiscale	string	Subject unique identifier
bds	bigint	
p1	bigint	
p2	bigint	
p3	bigint	
p4	bigint	
p5	bigint	
p6	bigint	

anacci

****

codice_fiscale	string	Subject unique identifier
tipo_assicurazione	string	Insurance type
ente_comunicante	string	Communicating entity
data_infortunio	string	Accident date
luogo_infortunio	string	Accident place
lesione_1	string	Injury nr 1
lesione_2	string	Injury nr 2
lesione_3	string	Injury nr 3
lesioni_ulteriori	string	Other Injuries
percentuale_inabilita	double	Disability percentage
data_decesso	string	Date of death

anacnt

****

codice_fiscale	string	Subject unique identifier

tipo_contatto	string	Contact type
contatto	string	Contact

anacontatori

****

codice_fiscale	string	Subject unique identifier
portafoglio	bigint	Total insurance policies number
portafoglio_auto	bigint	Auto insurance policies number
portafoglio_re	bigint	Elementary branches insurance policies number
portafoglio_vita	bigint	Life insurance policies number
portafoglio_cauzioni	bigint	Deposits policies number
sinistri_aperti	bigint	Open claims number
veicoli_attivi	bigint	Insured vehicles number

anafid

****

codice_fiscale	string	Subject unique identifier
tipo_soggetto	string	Subject type

anaind

****

codice_fiscale	string	Subject unique identifier
comune	string	Subject main address, city
provincia	string	Subject main address, province
nazione	string	Subject main address, country
flag_principale	string	

analnkcnt

****

tipo_contatto	string	Contact type
contatto	string	Contact
codice_fiscale_a	string	Subject unique identifier a
codice_fiscale_b	string	Subject unique identifier b

crvdlnk

****

partita_iva	string	VAT number
codice_fiscale	string	Subject unique identifier
denominazione	string	Subject / company name
cognome	string	Surname / company name
nome	string	Name

crvdsem

****

codice_fiscale	string	Subject unique identifier
semaforo	string	Traffic light

ptf

****

idpolizza	string	Policy unique identifier
agenzia	string	Agency ID
descrizione_agenzia	string	Agency description
provincia_agenzia	string	Province of the agency
ramo	string	Policy branch
tipo_polizza	string	Policy type (Individual / Collective)
stato_polizza	string	Policy state (Active/ Canceled / Suspended)
stato_coass	string	No coinsurance / Our delegation / Delegation
codice_prodotto	string	Product Code-Product Description
prodotto	string	Product
data_effetto	string	Policy effective date
data_scadenza	string	Policy effective deadline
premio	double	Policy premium

sin

****

idsinistro	string	Claim unique identifier
idpolizza	string	Policy unique identifier

data_sinistro	string	Claim occurrence date (Format: YYYY-MM-DD)
ora_sinistro	string	Claim occurrence time (Format: HH: MM)
tipo_sinistro	string	Accident type (RCA / ARD / RE)
tipo_danno	string	Damage reported type (1 = THINGS / 2 = PEOPLE / 3 = MIXED)
tipo_gestione	string	Claim management type
flag_autorita_presenti	string	Authority flag present (S - Yes, N - No)
stato_sinistro	string	Accident status
data_definizione_sinistro	string	Claim closing date (Format: YYYY-MM-DD)
numero_veicoli	bigint	Vehicles involved number
comune	string	Claim occurrence address, city
provincia	string	Claim occurrence address, province
pagato	double	Paid
riservato	double	Reserved
data_denuncia	string	Claim complaint date (YYYY-MM-DD)

sinantifrode

****

idsinistro	string	Claim unique identifier
semaforo	string	Traffic light
verifica	string	Verification
note_verifica	string	Verification notes
approfondimento	string	Deepening
note_approfondimento	string	Deepening notes
antifrode	string	Anti fraud

sinantifrodectl

****

idsinistro	string	Claim unique identifier
controllo	string	Check

vei

****

targa	string	License plate
marca	string	Vehicle brand
modello	string	vehicle model
tipo_veicolo	string	Vehicle type
tipo_targa	string	License plate type
data_immatricolazione	string	Matriculation date

vei_ptf

****

targa	string	Vehicle identifier
idpolizza	string	Policy unique identifier

vei_sin

****

targa	string	Vehicle identifier
idsinistro	string	Claim unique identifier