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/

D2.4 – Conceptual model and Reference architecture

Work Package	WP2 – Requirements, Architecture & Technical Coordination
Lead Author (Org)	Dimosthenis Kyriazis (UPRC)
Contributing Author(s) (Org)	Orlando Avila-García (ATOS), Konstantinos Chaidogiannos (UBI), Bin Cheng (NEC), Ismael Cuadrado-Cordero (ATOS), Christos Doulkeridis (UPRC), Eleni Fotopoulou (UBI), Jonathan Fuerst (NEC), Konstantinos Giannakakis (ATC), Kostas Giannakopoulos (SILO), Gal Hammer (RHT), Ricardo Jimenez (LXS), Miki Kenneth (RHT), Pavlos Kranas (LXS), Richard McCreadie (GLA), Stavroula Meimetea (UPRC), Yosef Moatti (IBM), Kostas Papadimitriou (SILO), Marta Patino (UPM), Stathis Plitsos (DANAOS), Dimitrios Poulopoulos (UPRC), Bernat Quesada Navidad (ATOS), Amaryllis Raouzaiou (ATC), Martí Sanchez-Juanola (ATOS), Paula Ta-Shma (IBM), Giannis Tsantilis (UBI), Constantinos Vassilakis (ATC), Valerio Vianello (UPM), Anastasios Zafeiropoulos (UBI)
Due Date	31.08.2018
Date	07.09.2018
Version	1.0

Dissemination Level

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

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

Versioning and contribution history

Version	Date	Author	Notes
0.5	16.07.2018	Dimitrios Poulopoulos (UPRC)	Added UBI and IBM contributions
0.6	20.07.2018	Dimosthenis Kyriazis (UPRC)	Updated contributions, comments, identification of points to be addressed
0.7	30.07.2018	Sophia Karagiorgou (UBI)	Comments addressed & figures updated
0.7	30.07.2018	Nikos Lykousas (UBI)	Comments addressed & figures updated
0.8	10.08.2018	Peter-Jason Gould (UPRC)	Comments addressed & figures updated
0.9	03.09.2018	Dimitrios Poulopoulos (UPRC)	Integration of additional inputs received from partners
0.10	04.09.2018	Dimosthenis Kyriazis (UPRC)	Final version following internal review
1.0	07.09.2018	Dimosthenis Kyriazis (UPRC)	Final version

Disclaimer

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

Table of Contents

List of tables

List of figures

1. Executive Summary

BigDataStack aims to deliver a complete stack including an infrastructure management system that drives decisions according to the live and historical data, thus being fully scalable, runtime adaptable and high-performant. In this way, BigDataStack will address the emerging needs of big data operations and data-intensive applications. The system will base all infrastructure management decisions on the data analytics, monitoring data from deployments and logic derived from data operations that govern and affect storage, compute and network resources, as well as their interdependencies. On top of the infrastructure management system, “Data as a Service” will be offered to data providers, decision-makers, private and public organisations. Approaches for data cleaning, data layout 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 underlying resources necessary. 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 process mining analytics.

The aforementioned key outcomes of BigDataStack are reflected in a set of main building blocks in the corresponding overall architecture of the stack. This deliverable describes the key functionalities of the overall architecture, the interactions between the main building blocks and their components, while also providing a first version of the internals of these components regarding research approaches to be realised during the course of the project. Additional information and detailed specification of the components will be delivered through the corresponding design and specification reports of the relevant work packages of the project.

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”, and they instead 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 analyzing of processes, as well as provisioning data-as-a-service, by exploiting a seamless analytics framework.

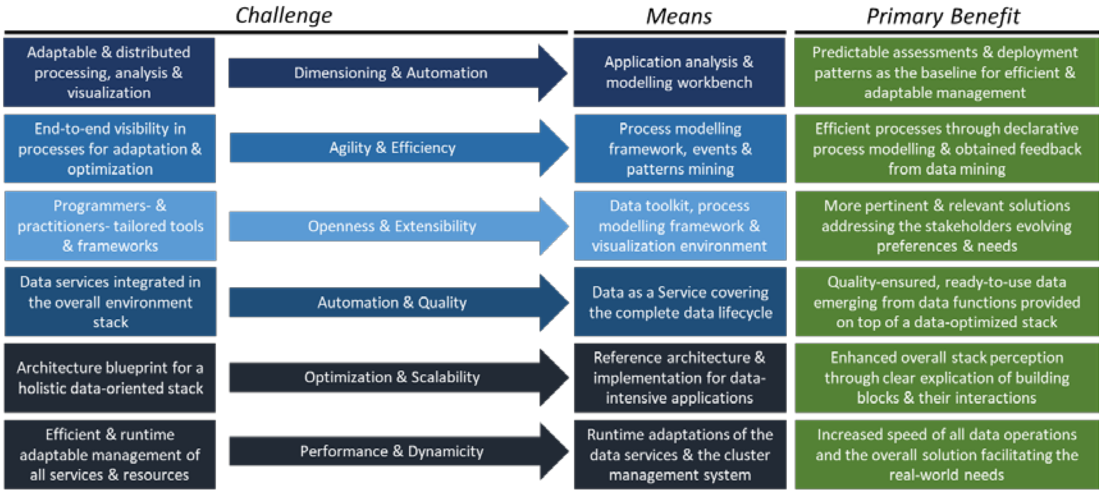

Challenge
Means
Primary Benefit
Adaptable & distributed processing, analysis & visualization
Dimensioning & Automation
Application analysis & modelling workbench
Predictable assessments & deployment patterns as the baseline for efficient & adaptable management
End-to-end visibility in processes for adaptation & optimization
Agility & Efficiency
Process modelling framework, events & patterns mining
Efficient processes through declarative process modelling & obtained feedback from data mining
Programmers- & practitioners- tailored tools & frameworks
Openness & Extensibility
Data toolkit, process modelling framework & visualization environment
More pertinent & relevant solutions addressing the stakeholders evolving preferences & needs
Data services integrated in the overall environment stack
Automation & Quality
Data as a Service covering the complete data lifecycle
Quality-ensured, ready-to-use data emerging from data functions provided on top of a data-optimized stack
Architecture blueprint for a holistic data-oriented stack
Optimization & Scalability
Reference architecture & implementation for data-intensive applications
Enhanced overall stack perception through clear explication of building blocks & their interactions
Efficient & runtime adaptable management of all services & resources
Performance & Dynamicity
Runtime adaptations of the data services & the cluster management system
Increased speed of all data operations and the overall solution facilitating the real-world needs

Figure 1 - Technical challenges

2.1. Terminology

Table 1 summarises the terms used in BigDataStack, not regarding acronyms but regarding actual usage, given the vast set of concepts and technologies addressed by the 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. Descriptions of the architecture components are provided thereafter in
- Section 6 provides descriptions of the main architecture components.
- Finally, in Section 7, detailed sequence of events depicting the information flows are 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 will be provided by the corresponding design and specification reports of the work package deliverables.

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 at not only facilitating the needs of data operations and applications (all of which tend to be data-intensive), but facilitating 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 a service-defined deployment decision by a human expert (current approach) where they choose 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.

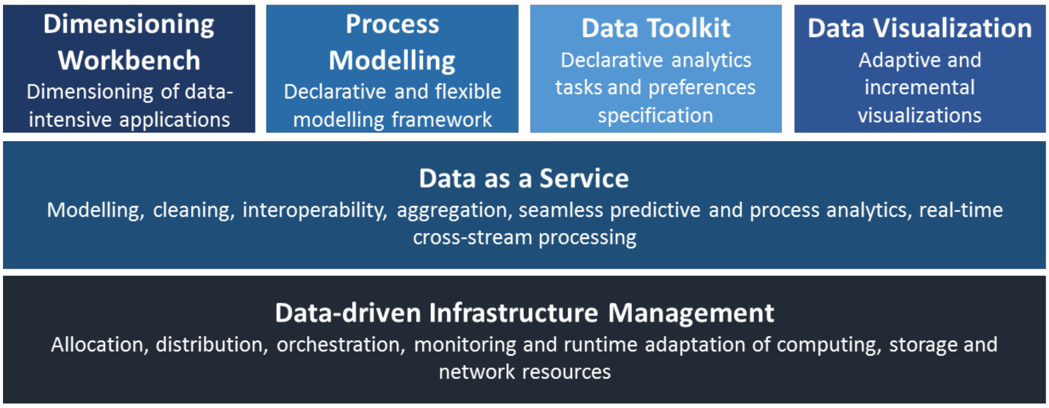

Dimensioning Workbench
Dimensioning of data-intensive applications
Process Modelling
Declarative and flexible modelling framework
Data Toolkit
Declarative analytics tasks and preferences specification
Data Visualization
Adaptive and incremental visualizations
Data as a Service
Modelling, cleaning, interoperability, aggregation, seamless predictive and process analytics, real-time cross-stream processing
Data-driven Infrastructure Management
Allocation, distribution, orchestration, monitoring and runtime adaptation of computing, storage and network resources

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 will incorporate a set of technologies addressing the complete data path: modelling and representation, cleaning, aggregation, and data processing (including seamless analytics, real-time Complex Event Processing - CEP, and process mining). *Distributed storage* will be realised through a layer enabling data to be fragmented/stored according to different access patterns and allowing the efficient expression of that data for database storage and subsequent retrieval. *Advanced modelling* will be provided to enable the definition of flexible schemas for both data in flight and at rest. These schemas will be then utilised by the introduced *seamless data analytics framework,* which 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* will be provided that can be executed in federated environments. The engine will consider the latencies across data centres, the locality of data sources and data sinks, and produce 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,* according to application analysis and data semantics. Visualizations will 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. cleaning outcomes, aggregation 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 will allow 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 will allow data scientists and administrators to *specify requirements and preferences* both for the *infrastructure management* (e.g. application requirements) and for the *data management* (e.g. data quality goals, incremental analytics, information aggregation "levels", etc.).

The ***Process Modelling*** offering will provide a *framework allowing for flexible modelling of process analytics* to enable their execution. Functionality-based process modelling will then be concretised to technical-level process mining analytics, while a feedback loop will be implemented 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 preferences for the data cleaning service).
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 previously mentioned actors provide information that is exploited thereafter by the infrastructure/cluster management pillar 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.

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.

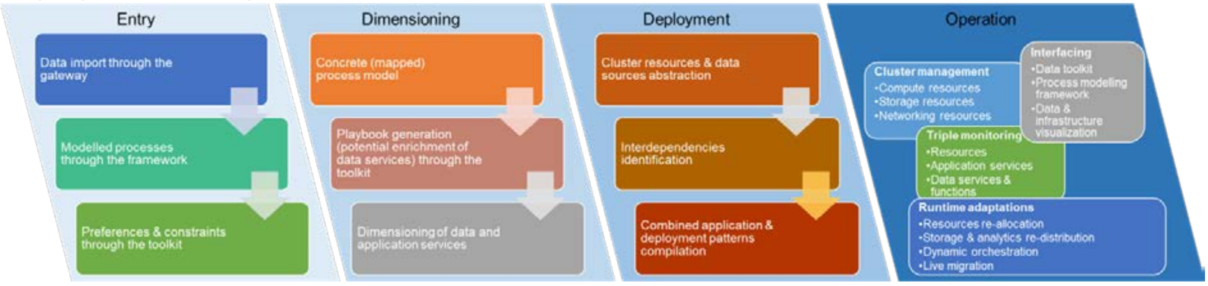

Entry
Data import through the gateway
Modelled processes through the framework
Preferences & constraints through the toolkit
Dimensioning
Concrete (mapped) process model
Playbook generation (potential enrichment of data services) through the toolkit
Dimensioning of data and application services
Deployment
Cluster resources & data sources abstraction
Interdependencies identification
Combined application & deployment patterns compilation
Operation
Cluster management
•Compute resources
•Storage resources
•Networking resources
Interfacing
•Data toolkit
•Process modelling framework
•Data & infrastructure visualization
Triple monitoring
•Resources
•Application services
•Data services & functions
Runtime adaptations
•Resources re-allocation
•Storage & analytics re-distribution
•Dynamic orchestration
•Live migration

Figure 3 - BigDataStack Main Phases

During the entry phase, data owners ingest their data through a 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 API. They can directly choose if they want to store (non-) relational data or use the BigDataStack's object storage offering. Moreover, historical data can periodically move from the operational database to object storage, keeping only recent data on the database and providing a backup mechanism. 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 tasks that will be executed. These mapping is performed by a mechanism incorporated in the Process Modelling framework, which is called Process Mapping.
- 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, how a data cleaning service should treat missing values or ingest a complete algorithm in the case this has not been mapped (and as a result made available) by the Process Mapping mechanism.

The output of the Entry Phase is a playbook descriptor (that includes all relevant information for the application graph with concrete "executable" services) that is passed to the dimensioning phase in order to identify the resource needs for the services and as a result for the overall application.

4.2. Dimensioning phase

The dimensioning phase of BigDataStack aims at optimizing 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 representation, aggregation, etc.) and the interdependencies between application components and data services (both included as processes through the process modelling approach described in the previous paragraph). 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 (i.e. playbook) 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 a load injector generating different loads, to benchmark the services and analyse their resources and data requirements (e.g. volume, generation rate, legal constraints, etc.).

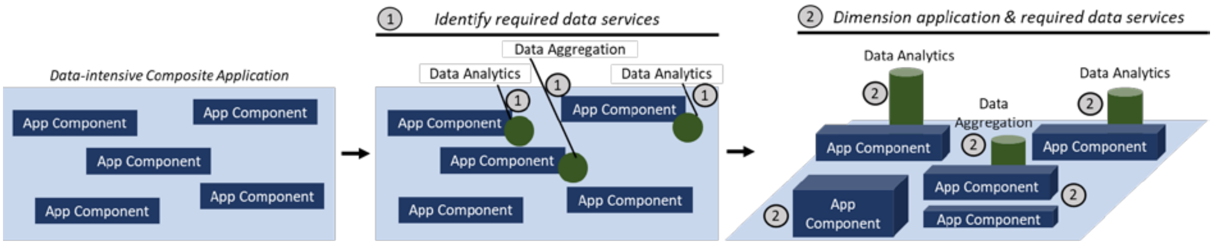


Figure 4 - Dimensioning phase

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 QoS) to needed resource parameters (such as the number of VMs, bandwidth, latency etc.).

4.3. Deployment phase

The deployment approach of BigDataStack aims at orchestrating the optimum deployment patterns and practices for both data services and applications. The need for such optimisation emerges from the fact that all services to be deployed have interdependencies that need to be considered, to obtain a practical deployment, as well as to account for the user's preferences (e.g. minimisation of cost). To this end, the deployment approach of BigDataStack includes a two-step phase and is realised through the Deployment mechanism of the BigDataStack infrastructure/cluster management system:

- Deployment receives the dimensioning information for components from the Dimensioning phase and decides on the overall optimum deployment pattern through a ranking mechanism.
- Following the identification and analysis of the interrelations and the impact of the components, in terms of computation, storage and networking resources (considering data characteristics such as volumes, application components and data services I/O rates, legal constraints, etc.), optimum deployment patterns will be compiled as shown in Figure 5.

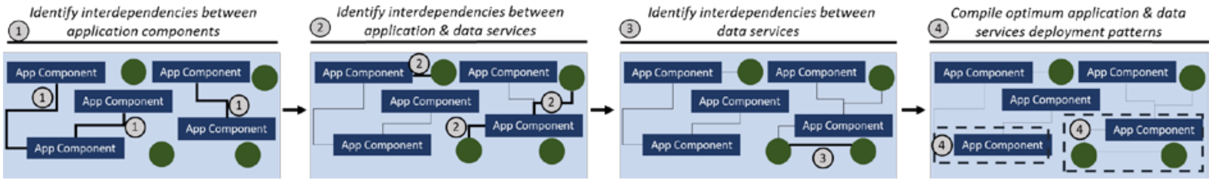


Figure 5 - Deployment phase

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.

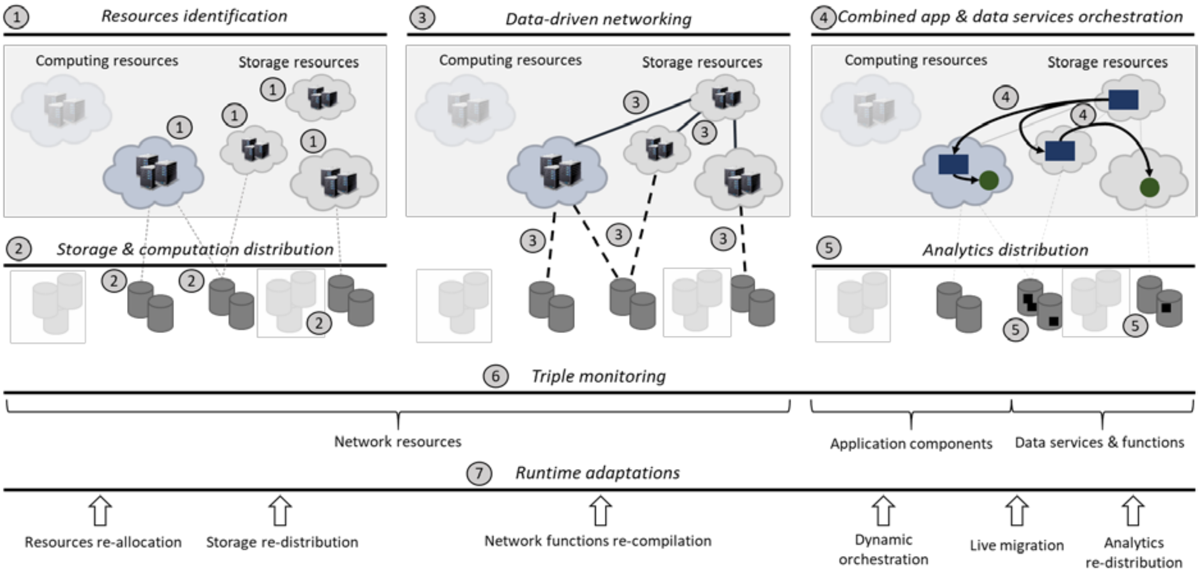


Figure 6 - Operations phase

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 will compile 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 orchestration of application components and data services is also performed.
- 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 including resource re-allocation, storage and analytics re-distribution, re-compilation of network functions and deployment patterns.

5. Architecture

The following figure (Figure 7) presents the initial architecture of BigDataStack, depicting the main information flows and interactions between key components.

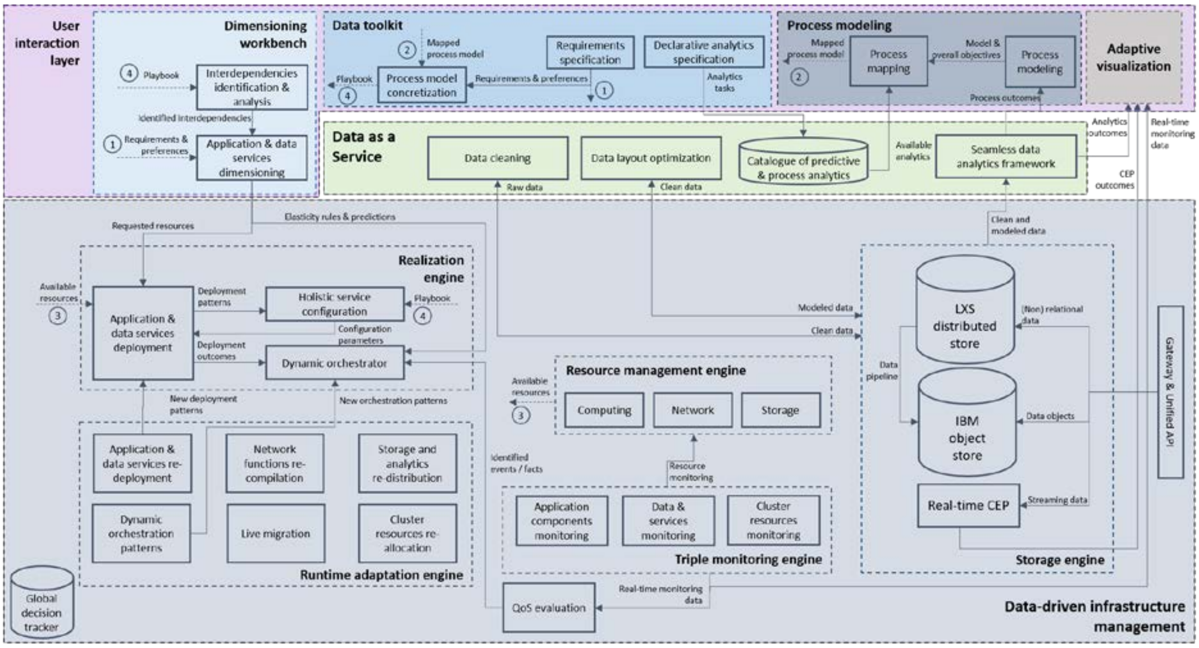

User interaction layer
Dimensioning workbench
4 Playbook
Interdependencies identification & analysis
Identified interdependencies
1 Requirements & preferences
Application & data services dimensioning
Data toolkit
2 Mapped process model
Requirements specification
Declarative analytics specification
Playbook 4
Process model concretization
Requirements & preferences 1
Analytics tasks
Process modeling
Mapped process model 2
Process mapping
Model & overall objectives
Process modeling
Process outcomes
Adaptive visualization
Data as a Service
Data cleaning
Raw data
Data layout optimization
Clean data
Catalogue of predictive & process analytics
Available analytics
Seamless data analytics framework
Analytics outcomes
Real-time monitoring data
CEP outcomes
Requested resources
Elasticity rules & predictions
Clean and modeled data
Realization engine
Available resources 3
Application & data services deployment
Deployment patterns
Holistic service configuration
Playbook 4
Configuration parameters
Deployment outcomes
Dynamic orchestrator
New deployment patterns
New orchestration patterns
Modeled data
Clean data
LXS distributed store
(Non) relational data
Data pipeline
IBM object store
Data objects
Real-time CEP
Streaming data
Gateway & Unified API
Storage engine
Resource management engine
Available resources 3
Computing
Network
Storage
Identified events / facts
Resource monitoring
Application & data services re-deployment
Network functions re-compilation
Storage and analytics re-distribution
Dynamic orchestration patterns
Live migration
Cluster resources re-allocation
Runtime adaptation engine
Application components monitoring
Data & services monitoring
Cluster resources monitoring
Triple monitoring engine
Global decision tracker
QoS evaluation
Real-time monitoring data
Data-driven infrastructure management

Figure 7 - BigDataStack architecture model

First, the raw data is ingested through the *Gateway & Unified API* component to the *Storage* engine of BigDataStack, which enables storage and data migration across different resources. The engine includes both stores for relational and non-relational data, as well as an object store and a CEP engine for streaming data processing. The raw data can be processed by the *Data Cleaning* component, which enhances its quality in terms of completeness, accuracy and volatility. Thereafter, the cleaned data is passed to the *Data Layout Optimization* framework where it is modelled (using flexible schemas to describe streaming and stored data), as well as annotated with metadata.

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 optimisation goals for the overall process. Based on the analytics tasks available in the Catalogue, *Process mapping* identifies the specific mining and analytics tasks needed to realise the corresponding business processes. The outcome of the component is a model in a structural representation e.g. a Docker Compose file [1]. 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 parameters, (e.g. the *k* in a *k*-means algorithm), execution substrate requirements (e.g. CPU, memory limits etc.) and software packages/dependencies (e.g. Apache Spark, Flink etc.). During this step, data scientists and DevOps/QA engineers can also give their feedback on the entire process design and proceed to optimisation. Finally, the output of the Data Toolkit component is a 'Playbook' descriptor, enriching the output of the previous step (i.e. Process Modelling), in the form of an enriched Docker Compose file. The latter is passed to the *Application dimensioning workbench* to estimate the resource requirements by also considering interdependencies between services. The primary output of the component 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.). Thus, it enables the investigation of optimisation options by the next step of the process, the *Candidate Deployment Patterns Generations* in the *Realization engine*. Based on the obtained dimensioning outcomes (i.e. resource needs and dependencies) and the availability of resources (information received from the *Resource management engine*), deployment patterns are compiled and used for deployment on the selected resources (through the *Application & data services deployment* mechanism), along with potential configuration parameters obtained by the *Holistic services configuration* component. The *Dynamic orchestrator* performs orchestrations of the application and data services on the allocated resources. The execution of the data analytics tasks (triggered by the *Dynamic orchestrator*), is performed on the *Seamless data analytics framework* that facilitates analytics across multiple resources and locations.

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 is evaluated through a *QoS Evaluation* component to identify events/facts that affect the overall quality of service. These 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. These adaptations are aligned with the elasticity model compiled from the Application dimensioning workbench during the analysis of the application.

Moreover, the architecture includes the so-called *Global decision tracker*, which aims at storing all the decisions taken by the various components. We plan to take 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, it will provide 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 will provide completely and dynamically (regarding sources and data artefacts being presented), all information, including raw data, cleaned and modelled data, analytical queries, and resources monitoring information.

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 Origin project for its Resource Management sub-system. The OpenShift Origin project is an upstream project used in Red Hat's various OpenShift products. It is based and build around Kubernetes and CRI-O and is enhanced with features requested by commercial customers and Enterprise level requirements. More specifically, OpenShift Kubernetes Distribution (OKD) [2] will be used (note that OKD is released under an Apache License 2.0). According to Duncan et al. [3] ODK is "an application platform that uses containers to build, deploy, serve, and orchestrate the applications running inside it". OKD simplifies the whole process [4] 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 [5], 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 CRI-O for containerization and Kubernetes [6] for their orchestration, including packaging, instantiation and running the containerized applications. It also implements "geard" or "gear daemon" [7], a command-line client for the management of Docker containers and its linkage to systems across multiple hosts, used for the installation and management of application components; and Project Atomic [8] an operating system designed to run Docker containers. On top of the above described technologies, OKD adds [9]:

- 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 [6] and run on a federated cloud).

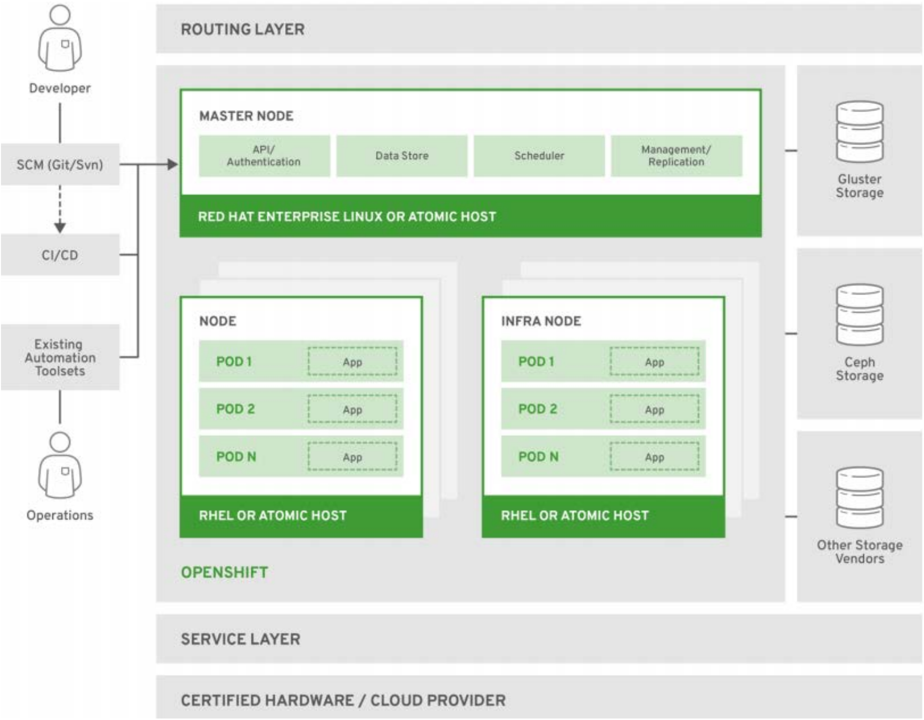


Figure 8 - OKD architecture overview inside the DevOps operation [9]

Following this layered architecture, users access the API, web-services and command line directly from the master node, while the applications are accessed through the routing layer where the application is location, 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 [6], users can choose a whitelist of cipher suites to security meet security policies; and share PID between containers to control the cooperation of containers.

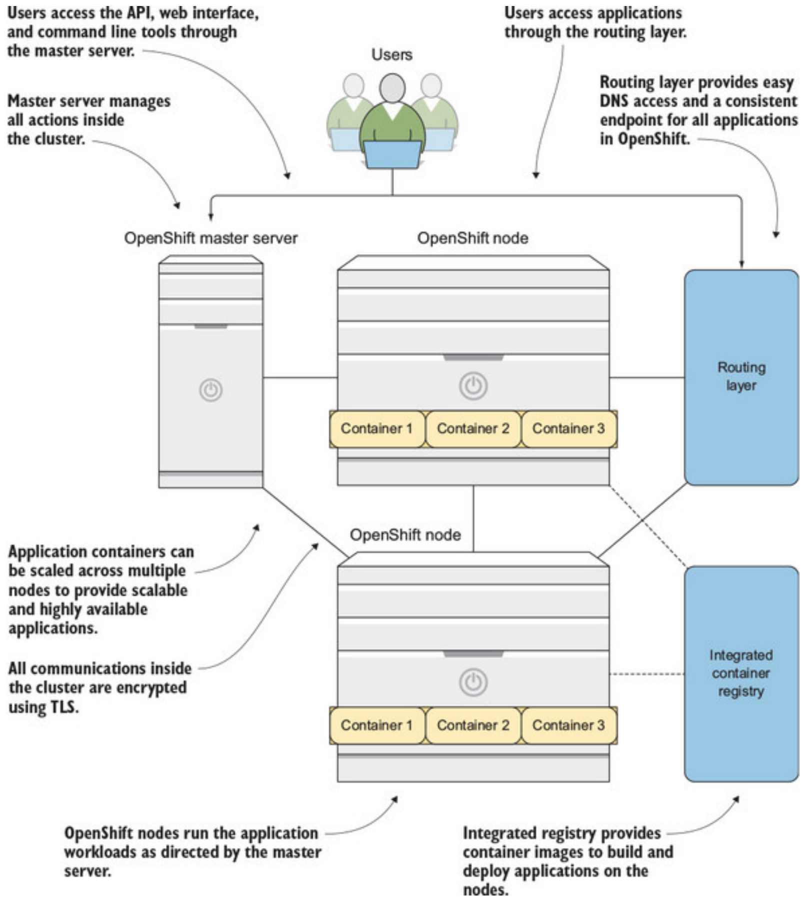

Users access the API, web interface, and command line tools through the master server.
Users access applications through the routing layer.
Users
Master server manages all actions inside the cluster.
Routing layer provides easy DNS access and a consistent endpoint for all applications in OpenShift.
OpenShift master server
OpenShift node
Container 1
Container 2
Container 3
Routing layer
OpenShift node
Application containers can be scaled across multiple nodes to provide scalable and highly available applications.
All communications inside the cluster are encrypted using TLS.
Container 1
Container 2
Container 3
Integrated container registry
OpenShift nodes run the application workloads as directed by the master server.
Integrated registry provides container images to build and deploy applications on the nodes.

Figure 9 - OKD architecture overview in the users operations [3]

6.2. Data-Driven Network Management

The Data-Driven Network Management component will efficiently handle 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 and satisfy access load.

Monitoring mechanisms over the storage layer will provide information to adjust the network parameters (e.g. my 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 enforce 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 that 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.

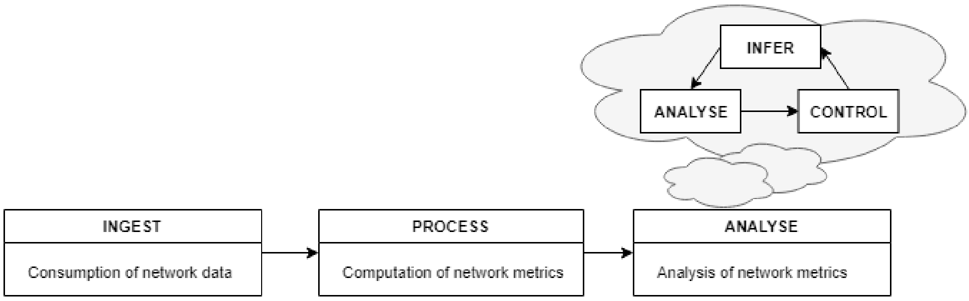

INFER
ANALYSE
CONTROL
INGEST
Consumption of network data
PROCESS
Computation of network metrics
ANALYSE
Analysis of network metrics

Figure 10 - Data-Driven Network Management components

6.3. Dynamic Orchestrator

The Dynamic Orchestrator assures that scheduled applications conform to 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 predict potential vessel collisions, an application processing vessel sensor data might have hard latency targets that need to be met by the system, so that vessels can react in time. The dynamic orchestrator 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 Dynamic Orchestrator is the component that maintains:

- The knowledge about potentially complementary (or conflicting) actions that other components can apply.
- The intelligence for deciding which of those actions shall be taken, when, by which component, and to which extent.

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). ADS ranks possible deployment patterns/configurations (CDPs) and selects the one which is predicted to best satisfy the SLOs. After an application is deployed, the orchestrator monitors its performance through the triple monitoring engine. In case of (predicted) SLO violations, it has two choices: (i) Initiate a re-deploy of the application through ADS (we envision that this choice will be made when SLOs can only be reached with *major deployment changes*, e.g., selecting another ADS ranking option), (ii) Performing more *fine-grained adaptations* at different components of the system (e.g., the orchestrator might reconfigure infrastructure services such as networking resources).

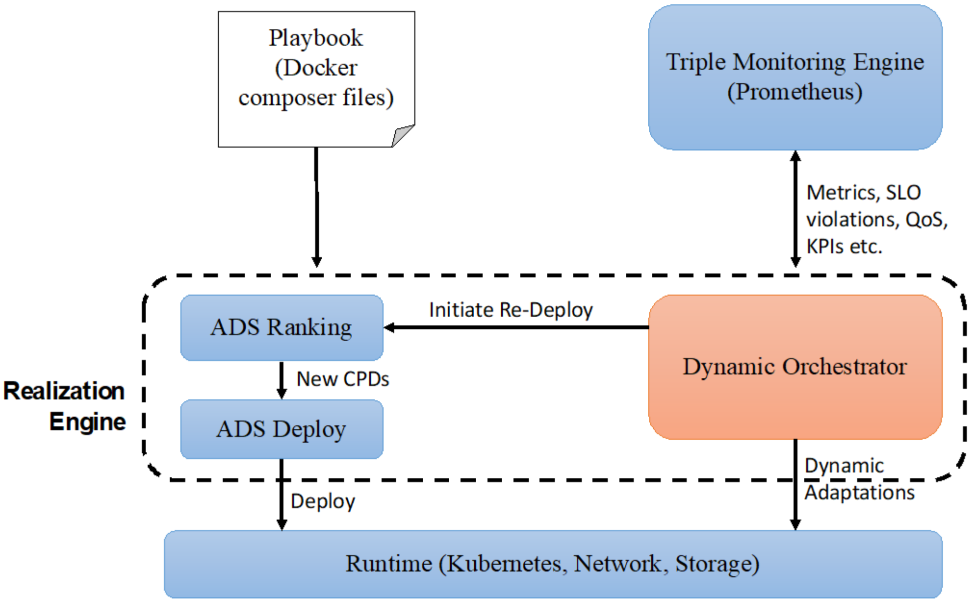

Playbook (Docker composer files)
Triple Monitoring Engine (Prometheus)
Metrics, SLO violations, QoS, KPIs etc.
Realization Engine
ADS Ranking
Initiate Re-Deploy
Dynamic Orchestrator
New CPDs
ADS Deploy
Deploy
Dynamic Adaptations
Runtime (Kubernetes, Network, Storage)

Figure 11 - High-Level Interaction with other Components

Note, that each of the other components also has its internal control loop and its 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, storage). In practice, two adaptation techniques at different components might lead both to conformation of SLOs. 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
- Small overhead for adaptation decisions
- Optimal adaptation

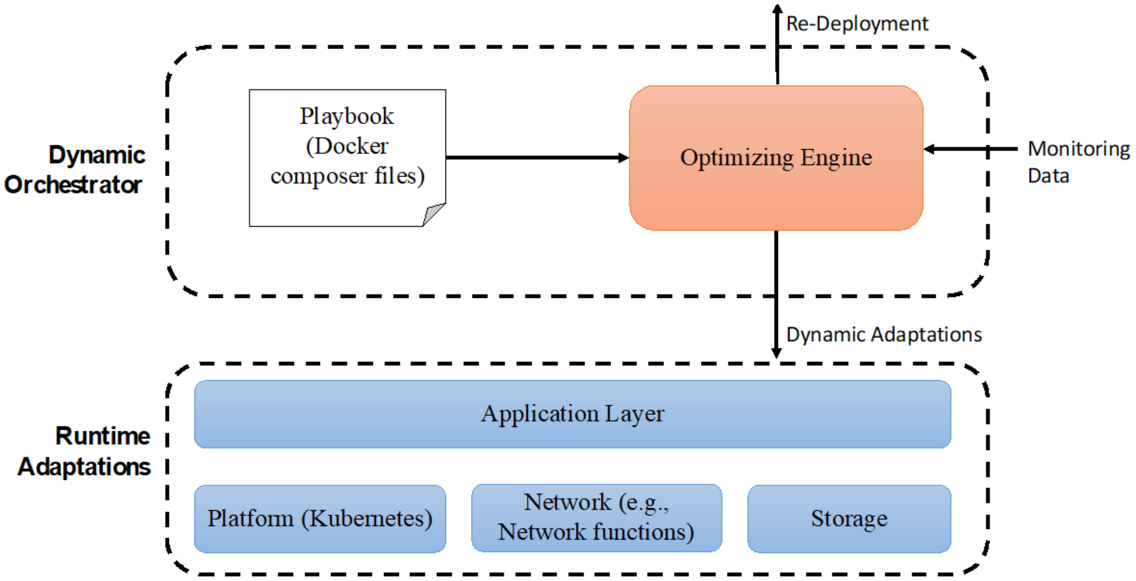


Figure 12 - Dynamic Orchestrator Detailed View

Figure 12 depicts a more detailed view of the dynamic orchestrator. As described in the previous section, the dynamic orchestrator can either initiate a re-deployment, through a re-ranking of deployment patterns via ADS ranking, or perform more fine-grained adaptations to keep a running application inside its SLOs (e.g., latency requirements). These dynamic adaptations are realised through runtime adaptations at multiple controllers (a.k.a., layers or components) of the BigDataStack platform.

6.4. 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 and RAM), 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.
- 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.
- 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.

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.

Figure 13 depicts the Triple Monitoring Engine and their components.

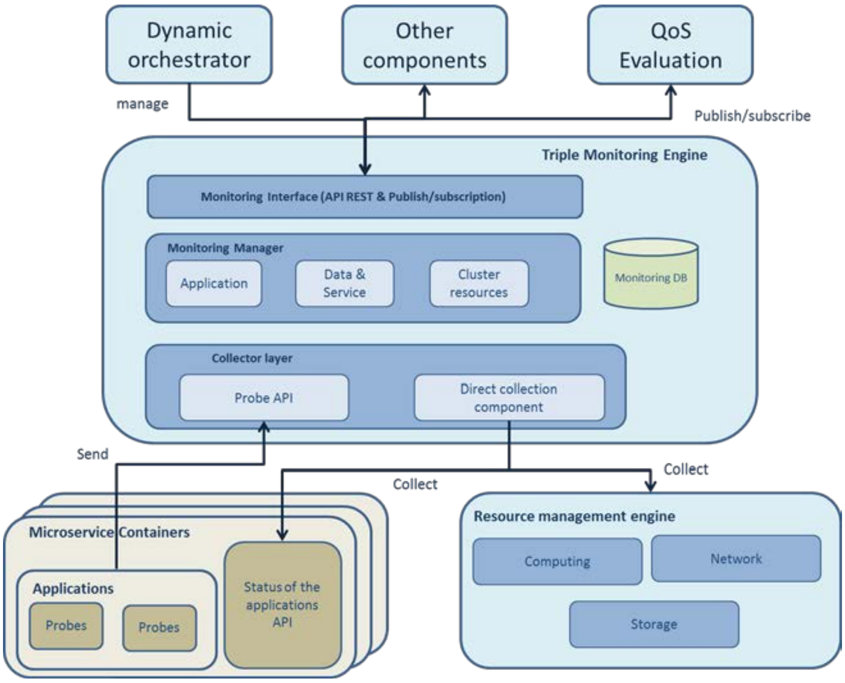


Figure 13 - Triple Monitoring Engine architecture diagram

The Tripe Monitoring Engine will be based on the Prometheus monitoring solution (see [11] 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 that will enable other components to know specific information, for example, if other 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.

- Monitoring Manager: This is the core of the engine. It is responsible for receiving all the data, process and aggregate it, depending on the type, such as application components, Data & Services and cluster resources, as well as persist it within the monitoring database. Therefore, all the logic of the engine will be incorporated in this component.
- Monitoring DB: This is responsible for persisting all the data. The database will be dimensioned depending on historical requirements, the kind of aggregation and the expected volume of data produced by the metrics. The monitoring DB component will either be a separate component or provide all the information to the global decision tracker of the architecture.
- Collector Layer: This is responsible for obtaining the metrics data to be moved to the Monitoring manager. There are ways collect data; through a probe or direct collection:
    - Probe API will expose an interface to allow different kinds of probes to send the metrics to the monitoring engine.
    - Direct collection component: it will collect directly the metrics data, by calling other systems or components. For example, it will receive the data directly from the Resource management engine or call the third-party libraries to obtain the state of the application.

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	Prometheus exporter name
Kubernetes	Computing infrastructure	Yes	Kubernetes
OpenStack	Computing infrastructure	Yes	Prometheus OpenStack exporter
Spark/Spark SQL	Data functions/operations	Yes	SparkMeasure, jmx
IBM COS (Cloud Object Store)	Data infrastructure	No	
LeanXcale database	Data infrastructure	For some metrics	

Table 2 - Prometheus integration

QoS Evaluation

The QoS is directly connected with the Tripe Monitoring Engine to evaluate the quality of the services on the platform. The component will be responsible for notifying if something happens in some of the conditions defined for the services at the different levels, infrastructure performance, storage and application components. Therefore, the component is not responsible for obtaining the metrics (delegated to the Monitoring Engine), and neither to apply the defined rules when something happens. It focuses on the identification of rules violations and their notification.

The component evaluates whether the Service Level Objectives (SLOs) which are set for a given application or data service are being fulfilled. An SLO specifies a constraint on Non-Functional Requirements. An SLO may describe a business penalty to apply in case of violation.

The component is based on the following components

- *Interface component (REST* API*):* the component is responsible for managing the lifecycle of the QoS component.
- *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.

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. 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 two main sub-components which we describe in more detail below:

- **Application and Data Services Ranking** (ADS-Ranking): Which is dedicated to the selection of the best deployment option
- **Application and Data Services Deployment** (ADS-Deploy): Which is concerned with

the physical scheduling/deployment of the application for the selected deployment option

Application and Data Services Ranking (ADS-Ranking)

ADS-Ranking is strongly 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 prediction models. It is these deployment patterns that ADS-Ranking takes as input and subsequently selects one or more 'good' options.

It is envisaged that communication to and from ADS-Ranking will be 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 ADS-Dimensioning to receive packages of deployment patterns (one package per-user 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 to send to ADS-Deploy for physical scheduling on the available hardware.

Figure 14 illustrates the data flow between the components around ADS-Deploy. As we can see, ADS-Dimensioning first gets information about the user's application and preferences and uses these to produce a package of candidate deployment patterns (CDPs), that we might consider instantiating to deploy the user's application. This pattern package is sent as a message to ADS-Ranking, which ranks and filters those patterns, and finally selecting one (Selected CDP), which is predicted to efficiently and effectively satisfy the user's requirements. This top pattern is placed in a message envelope and sent to both ADS-Deploy for physical scheduling and the dynamic orchestrator such that deployment-based activities can be triggered.

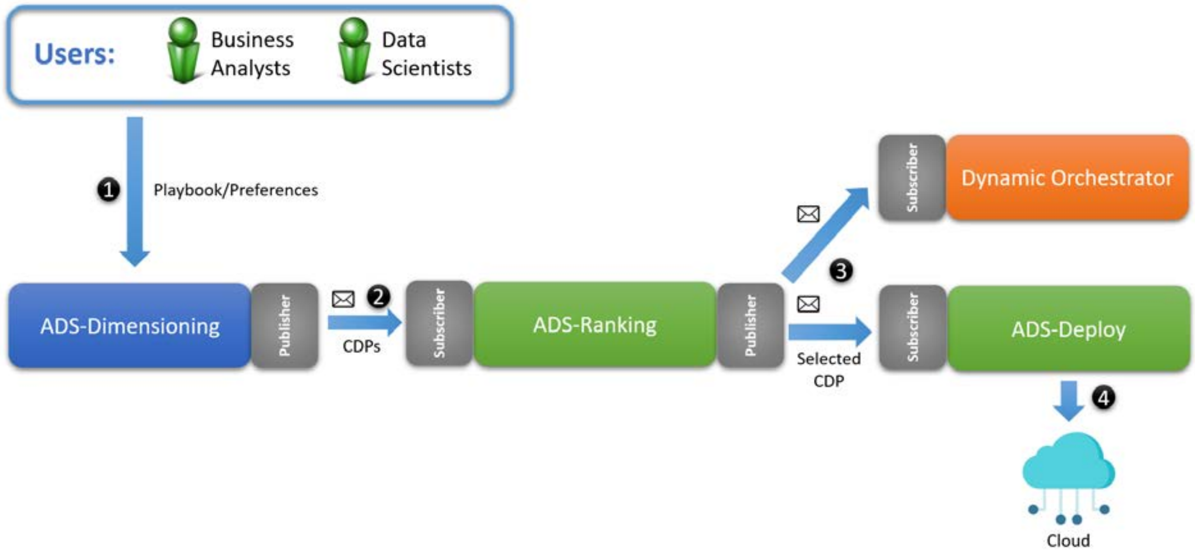


Figure 14 - Process Flow for ADS Ranking/Deploy

Internally, ADS-Ranking supports three central operations: 1) the first-time ranking/filtering of CDPs; 2) re-ranking of CDPs in scenarios where the previous deployment is deemed unsuitable, and 3) the learning of models to operationalise CDP ranking. The first operation (CPD ranking and filtering) is comprised of three main processes. These three processes are:

- **CDP Aggregate Representation Builder**: This takes as input a set of CDPs, and for each CDP in that package, it builds a single vector representation for that CDP, which combines all the information provided in its Predicted Performance block. The output of this component is the list of CDPs along with their new vector representations.
- **Deployment Pattern Ranking**: 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 characteristics. To achieve this, it uses a supervised model trained on previous CDP deployments and their observed fitness. The output of this process is a ranking of CDPs.
- **Deployment Pattern Selection**: This process takes as input the ranking of CDPs and selects one of these CDPs. That 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 CDPs that can be deployed.

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 for some reason along with context about that CDP (e.g. why it failed and the other CDPs that were not selected for the same application), 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, such that it can be used by the CDP Aggregate Representation Builder when representing candidate CDPs. In this way, properties that promote other CDPs, which will not suffer from the same issues of the failed CDP encountered, can be up-weighted during ranking.

ADS-Ranking's third function enables it to update the ranking model used within Deployment Pattern Ranking. This secondary function involves two main processes:

- **Model Manager**: This process is responsible for managing the creation/updating of the learning to rank models that are used within ADS-Ranking. The Model Manager has two modes of operation. First, it may trigger an update of the underlying learning to rank model once the previous model has reached a pre-defined age. For instance, we might refresh the model once each day. Second, model updating may be triggered by other components by sending an update message to the Model Manager's Subscriber. That might be used in the case where a deployed CDP has been identified as non-viable by the monitoring services, and hence this should be fed into the learning framework to improve the ranking.
- **Learning to Rank**: This process is responsible for performing the learning of the underlying ranking model. When triggered by the Model Manager, it will source

training examples from the Global Decision Tracker and start learning a new ranking model. This new model will replace the old one when training completes.

Figure 15 illustrates the main processes and data flow within ADS-Ranking.

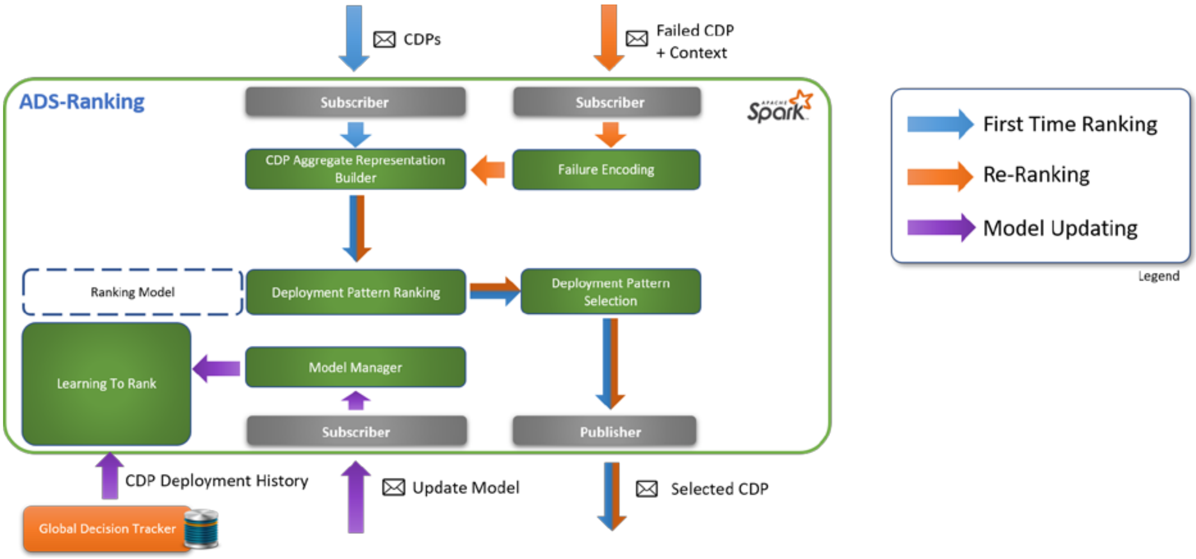


Figure 15 - Internal Components of ADS-Ranking

Application and Data Services Deployment (ADS-Deploy)
This process is triggered by the ADS-Ranking and takes as an input the selected CDP. The output of this component is both starting a specific configuration on the cloud infrastructure for the user, that is, one or more containers configured according to the information included in the CDP and informing the user about the success of the creation operation. To achieve this, the ADS-Deploy component interacts with a container orchestration service (e.g. Kubernetes), translating for the latter the CDP into instructions for its correct deployment on it.

This task is divided into the following steps:

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 (an 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 and ADS-Ranking components.

6.6. Data Cleaning

The data cleaning mechanism will incorporate a set of algorithms to enable domain-agnostic cleaning and harmonisation of data. The latter is proposed as an approach that results to a mechanism being "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 cleaning service that exploits Artificial Neural Networks (ANN) and Deep Learning (DL) techniques, to extract latent features that correlate different fields (of datasets) and identify possible defects in these datasets.

Furthermore, the mechanism will be enhanced to evaluate and assess the corresponding quality levels of the data sources. To this end, by monitoring the frequency of the errors that arrive from a specific source, the data cleaning mechanism will identify possible deterioration and levels of data veracity. This will be correlated with the information regarding the obtained data and the corresponding quality / verification outcomes properties, to provide a complete, holistic snapshot of the data source and the actual data veracity.

The key issues that need to be handled by the Data Cleaning 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.
- Enhance the quality of the raw data, in terms of completeness, accuracy and veracity. In other words, the service should address the problem of missing values and identify distorted or erroneous data points.
- Identify source function deterioration. If a source constantly measures and transmits distorted signals, or even produce no data at all, the system should take notice and raise a suitable alert. It should also check against previous datasets, to verify data veracity.
- Model the relationships between data points and reuse the learned patterns. The system should save the models learned by the machine learning algorithms, and reuse them through an optimisation component, that 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 16 depicts a serial architecture, which tries to predict if two entities are related to each other.

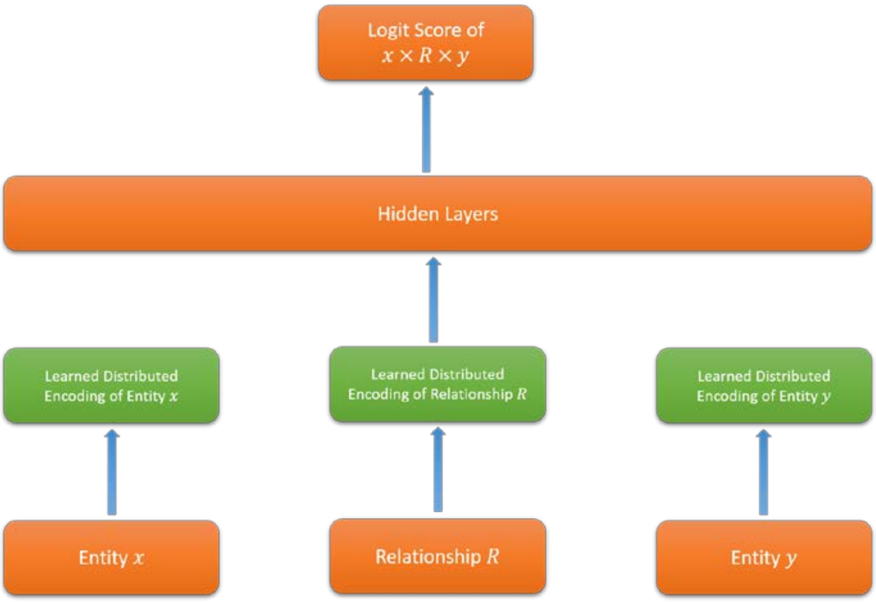

Logit Score of $x \times R \times y$
Hidden Layers
Learned Distributed Encoding of Entity x
Learned Distributed Encoding of Relationship R
Learned Distributed Encoding of Entity y
Entity x
Relationship R
Entity y

Figure 16 - 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 a medical example, if the temperature sensor emits a value that is illogical given other patient examinations, the relationship $R = temperature$ between two data points, the actual measurement and the patient's condition, would be associated with a low score. To optimize the data cleaning 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 cleaning component inputs:

- The raw data ingested by the data owner through the Gateway & Unified API
- The corresponding data schema (type, rates etc.)
- The data model provided by the optimizer if exists
- User preferences and specifications, ingested through the Data Toolkit

Data cleaning component outputs:

- Cleaned and harmonised data, in terms of completeness, accuracy and veracity
- Trained, reusable ML models, stored in a repository for later use
- A holistic snapshot of the data source function and the actual data veracity

The main structure of the Data Cleaning is depicted in Figure 17.

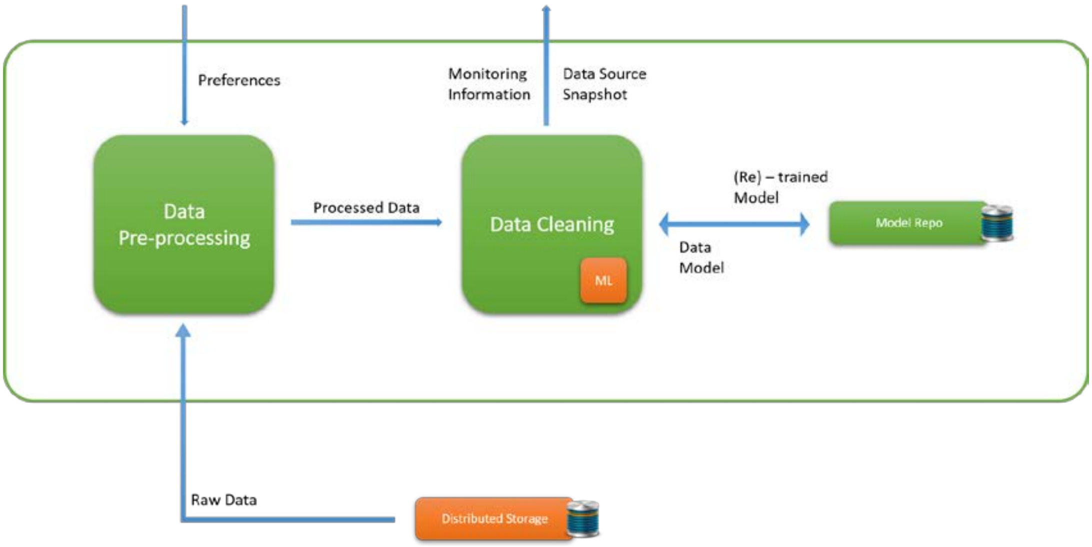


Figure 17 - Data Cleaning Module Architecture

Based on the figure above, 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 and, optionally, a learned data model through the Model Repository, for optimising the process
- The model repository stores and retrieves the learned models produced by the ML (Machine Learning) component

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 [12], was the first system addressing the scalability problem allowing a parallel distributed processing of massive amount of collected data. Apache Storm [13] and later Apache Flink [14] 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 [15] added streaming capability onto their product later. Spark's approach is not purely streamed, if fact it divides the data stream into a set of micro-batches and repeats the processing of these batches in a loop.

The streaming engine for the BigDataStack platform will be a scalable complex event processing (CEP) engine able to run in federated environments and to aggregate and correlate real-time events with structured and non-structured information stored in the BigDataStack datastores.

Data enters the CEP engine as a continuous stream of events, and it is processed by continuous queries. In contrast with a point in time queries (typical in database systems) that process existing data and deliver the result, a continuous query is deployed once and continuously delivers the result until it is decommissioned.

Continuous queries are modeled as 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 datastore.

- 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.
- Datastore operators are used to integrate the CEP with the BigDataStack datastores. 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. Register and deploy the continuous queries in the engine.
- Instance Manager: it is the component that runs a continuous query or a piece of it.
- Edge Worker: It is the components that connect with the sensors and performs pre-processing on the edge of the system.
- Reliable Registry: A store information related to the query deployments and components status.
- Driver: The interface between the CEP and other applications. Applications can use the CEP

Driver to register/unregister or deploy/displace a continuous query, discover where a source streams are deployed, subscribe with the output streams of the queries to consume results and mainly to send events to the engine.

6.8. Process mapping

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

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

Symbol	Description
T	An analysis task (e.g., clustering, classification…)
D	A data set
T(D)	The analysis task T applied on data set 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 data set D
T	An analysis task (e.g., clustering, classification…)
D	A data set
T(D)	The analysis task T applied on data set D

Table 3 - Main symbols used in process mapping

Predictive Analytics

The inputs of the Predictive Analytics 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 data set D that is subject to the analysis task T

The output of the Predictive Analytics sub-component is a selected algorithm A(T) that is predicted to be the most suitable for executing the data analysis task T at hand.

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

- *Data Descriptive Model:* This module takes as input a data set 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 data set D. Examples of information that could be 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 a model M(D) will be determined explicitly during the course of the project, but it can be considered as a set of features with corresponding values. The produced model M(D) is going to be used subsequently in order to identify previously analysed data sets that have similarities with the given data set.
- *Analytics Engine:* The role of this module is twofold: (a) to query the Predictive and Process Analytics Catalogue for algorithms available in BigDataStack that can be used for the Analysis task at hand, and (b) to query the Analytics Repository for matching data models M_i to the data model M(D) of the current data set.
- *Analytics Repository:* The purpose of this repository is to store a history (log) of previous results of data analysis tasks on various data sets. Each record in this repository corresponds to one previous execution of a task. It contains the model of data set 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 result quality of the specific algorithm when applied to the specific data model.

- *Results Filter:* 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.

Figure 18 provides an overview of the different modules and their interactions. Once the Predictive Analytics sub-component has received the required inputs, the data is ingested into the Data Descriptive Model where characteristics and morphology aspects of the data set 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 Results Filter (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 data set) is executed on the data set at hand, and the results are again analysed in the Results Filter. 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.

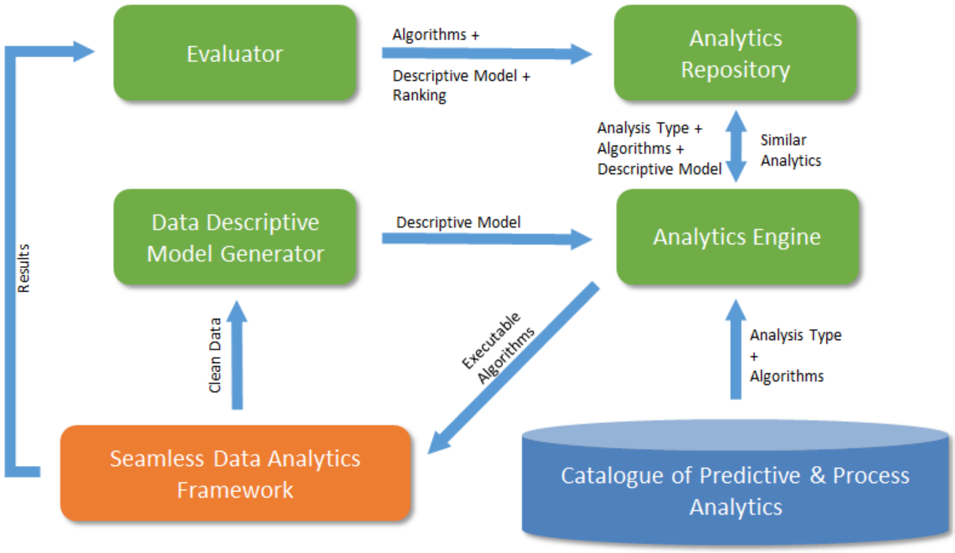


Figure 18 - Internal architecture of Predictive Analytics sub-component

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 19:

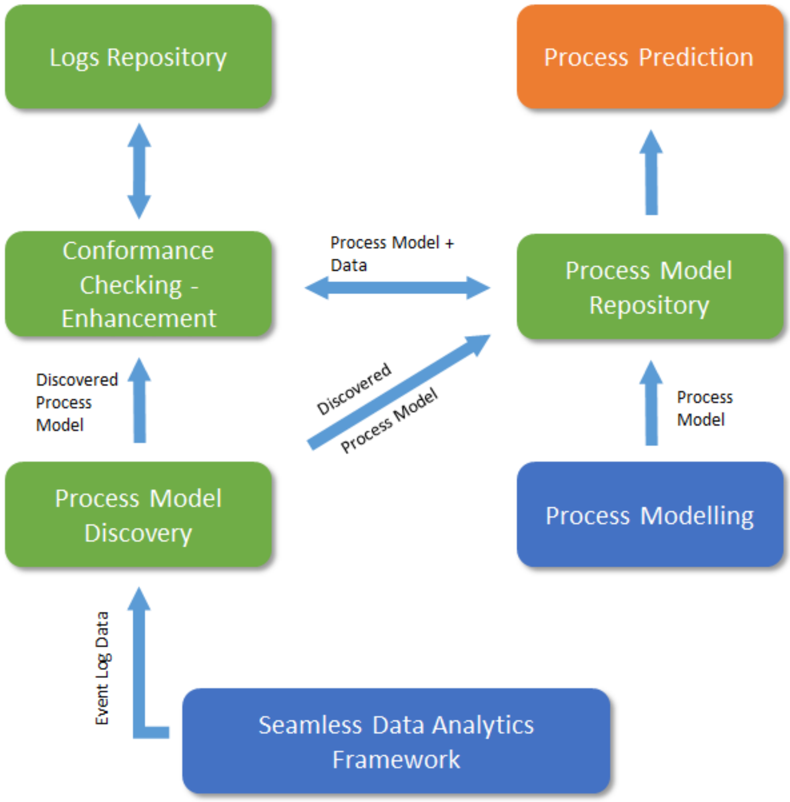


Figure 19 - Internal architecture of Process Analytics sub-component

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 data set 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.

A given data set may be stored in multiple data stores and the seamless analytics framework will permit analytics over it in a unified manner. We plan to adopt Apache Spark as the framework and analytics engine to demonstrate this approach. Figure 20 depicts this concept.

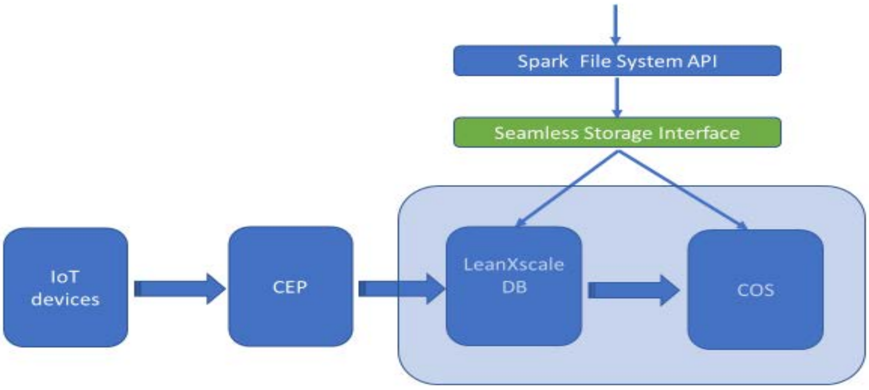


Figure 20 - Seamless Interface

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. To do this, we propose to use the Apache Spark Data Source API which allows data store developers to easily integrate with the Spark framework. To date, Spark supports many and varied data stores via the Data Source API. File systems and object stores supporting the Hadoop File System API are also supported, together with various data formats, as depicted in Figure 21.

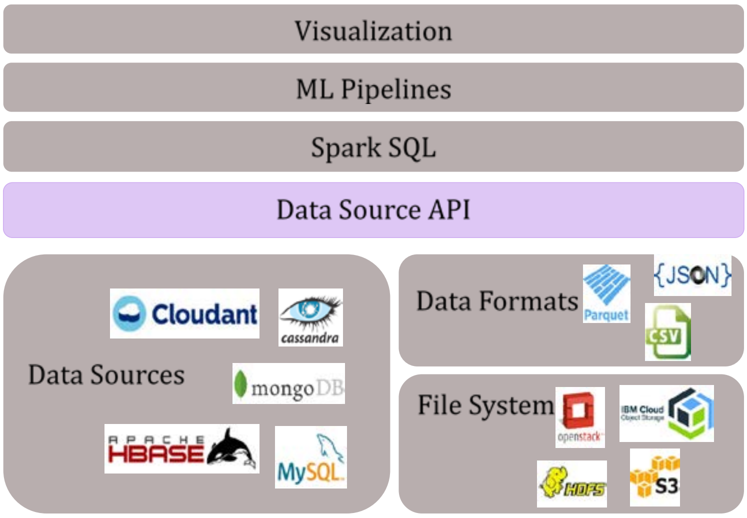


Figure 21 - Spark Data Source API Ecosystem

Moreover, we plan to study the additional capabilities offered by the new Data Source API V2. This API was released with Spark 2.3 but will continue to evolve in the upcoming releases of Spark.

6.10. Application Dimensioning Workbench

The goal of the dimensioning phase is to provide insights regarding the required infrastructure resources for the data services and application components (micro-services), linking the used resources with load and expected QoS levels. To this end it needs to cater for both cases of resources needed, creating prediction/correlation models between the application/service related information (such as KPIs and workload, parameters of the data service etc.) and the used resources to be able to provide recommendations towards the deployment mechanisms. Benchmarking against these services is an option that may help in concentrating the original dataset that is needed for the creation of such supervised models, as well as historical data from previous runs (especially for the case of the data services existing in the BigDataStack environment).

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

- The workflow 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, the generic structure needs to be provided as input to the Workbench, through the Data Toolkit. On top of this structure, the workbench can generate the model that will capture the identified quantified 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). For this reason, the dimensioning workbench receives this input from the Pattern Generation in order to create candidate deployment patterns, for which expected QoS will be returned to the deployment process for final ranking and selection.
- Insertion of monitoring code patches to extract relevant information with relation to the current monitored metrics of the application (per component), via exploiting the aforementioned endpoints that the application needs to offer (in order to extract application related information such as workload or application level QoS metrics).

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 Docker compose file) 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 experienced QoS per component. Thus, the modelling approach selected needs to be able to handle such workflow-based descriptions
- Types of infrastructure resources available in terms of size, type etc. 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 modelling of the specific service (e.g. usage of a more or less strict SQL schema, usage of document DBs against key value stores 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 percentage of data cleaning, throughput needed etc.) or other preferences declared through the Data Toolkit. For the latter, the declaration format should also be defined and is a dependency of the Dimensioning workbench.

- Application related current workload and QoS values should be available to enable dynamic adaptation of the application dimensioning workbench in terms of feedback during the load injection phase for the training set. This would either be obtained from application exposed endpoints or through the Triple Monitoring Engine of the Infrastructure Management block.

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 application components and 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 should be agreed in collaboration, defined currently as a Docker compose file structure. 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 Docker Compose 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.

The main structure of the Dimensioning is depicted in Figure 22. 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 Docker Compose file (known as the 'Playbook') 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 of the user's application that are benchmarked 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.

- *Structure2Model 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. Docker compose service structure) and transforming it to the common representation needed in order to serve as input to the Modelling Engine. The latter needs to have a standardized input to create the performance model corresponding to the structure.
- *Modelling Engine:* This component is the core of the Dimensioning workbench and is responsible for building the main model structure and training it to be able to give out concretizations in terms of the needed resources based on dynamic inputs
- *Load Injector:* The injector is responsible for load generation of different load patterns to benchmark the application and/or data services. It is also an abstraction layer since it needs to adapt to different types of data services. Existing abstraction layers (such as YCSB's multi-driver logic) may help alleviate much of the burden, at least for the database domain, however potentially other types of data services such as streaming will be needed.
- *Model repository:* This component is intended to hold the created models from the Modelling Engine for future usage or for usage during the Runtime Control.
- *Output Adaptor:* This component acts as an abstraction layer and is responsible for generating the output format needed for the deployment of the service to the target platform (in this particular case enriching the inputed Docker compose file with the extra QoS metrics). Thus, it needs to reverse the process followed during the Structure2Model translator.

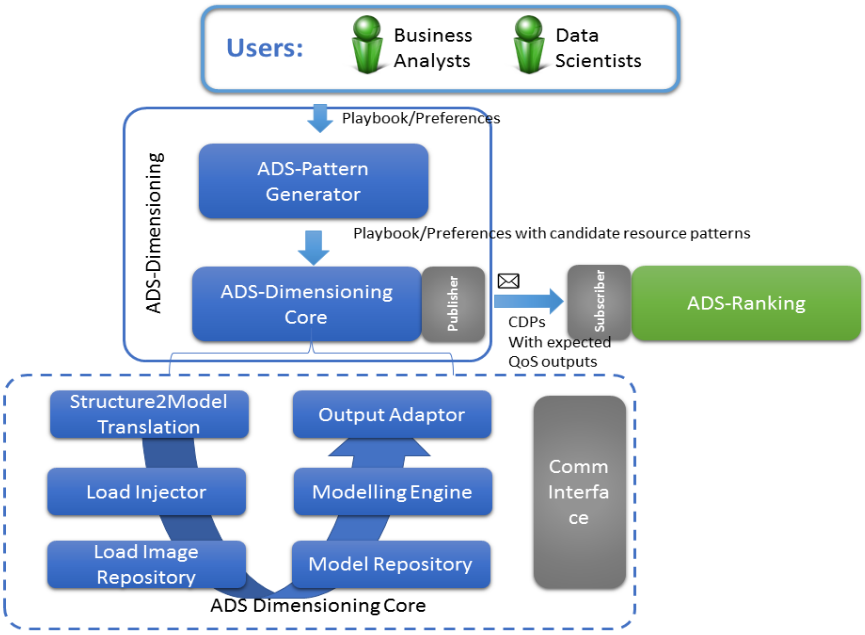


Figure 22 - Application dimensioning internal structure and link with external components

6.11. Big Data Layout

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 the IOStack H2020 project [19], we developed a data skipping method, which fits well with Object Storage and is based on a partitioning scheme using the k-d-tree data structure in a novel way.

This initial work has certain limitations such as:

1. It handles only static data
2. Our work focused on geospatial data from the GridPocket use case partner.
3. Our work focused on optimization of one specific query for nearest neighbor energy comparisons

In the BigDataStack project, we plan to significantly enhance this initial research in the following directions:

1. Handle a wider variety of datasets, go beyond geospatial data
2. Handle continuous streaming data that is appended to an existing logical dataset.
3. Continuously assess the properties of the streaming data to possibly adapt the partitioning scheme as needed
4. Handle general query workloads. This is significant because often different queries have different, even conflicting, requirements for data layout.
5. Handle query workloads which change over time.
6. Build a benefit/cost model to evaluate whether parts of the data set 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 [16]), 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 Docker Compose 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 *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" [17]. Declarative process modeling or a mixed approach seems to fit well in our environment providing the necessary flexibility in process modelling, mapping and optimization.
- *Recognize workflow patterns and task types*: In process-aware information systems various perspectives can be distinguished: (i) The control-flow perspective captures aspects related to control-flow dependencies between various tasks (e.g. parallelism, choice, synchronization etc.), i.e. Sequence pattern: A task in a process in enabled after the completion of a preceding task in the same process or Parallel split pattern: The divergence of a branch into two or more parallel branches each of which execute concurrently. (ii) The data perspective deals with the passing of information, scoping of variables, etc. (iii) The resource perspective deals with resource to task allocation, delegation, etc. (iv) The exception handling perspective deals with the various causes of exceptions and the various actions that need to be taken because of exceptions occurring. The expressiveness of modelling languages is assessed according to the well-known workflow resource patterns [18] they can express/model.
- *Data connection, schema and data sources access:* Business analysts should have access to the data schema of the specified data sources and be able to specify their rules/constraints when feeding certain analytic tasks as part of the flow.

- *Simulated run, debug/evaluate/first level optimization:* A process while being designed must be able to perform a simulated run for debugging purposes but as well for optimization recommendations before feeding the dimensioning framework and head for execution.
- *Structure to output 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 dimensioning framework. The structure should provide for reproducing the process graph, the tasks mapping to executables and the logic in terms of rules/constrains that govern the execution flow and the execution of the process tasks. The foreseen I/O and the structure of the process modeling framework in terms of definition of the subcomponents and their interactions are listed in the following bullets.

Necessary inputs:

- Input from the seamless data analytics framework to assist optimization of processes towards specified goals through the collection of performance measures and events (process mining) during process runtime.
- Input from the Catalogue of predictive & process analytics to provide for the definition and concretization of process tasks utilizing already developed data analytics applications.
- Input from exposed data sources to facilitate process design and rules/constraints definition.
- Input from the Catalogue of application components to provide for the definition and concretization of process tasks utilizing already developed applications other than data analytics.

Necessary outputs:

- Output the structure of the developed process model to the application dimensioning workbench.
- Output defined tasks, relationships and process model to the repository.

The main structure of the Process modelling framework is depicted in Figure 23. 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*.
- *Data connector:* This component provides connection to the *data sources schema and data* to be exploited as *input during process design through the Modelling toolkit*. Furthermore, connection to the data sources through the data connector is required not only during design time but also during evaluation of the associated rules (*Rules engine*) and simulated run of the process (*Simulator engine*).
- *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. In example, a relation between two tasks which requires to be expressed as a rule is not to proceed in executing the 2nd task before you obtain X samples as output from the 1st task. This option could be offered to the business analyst through a configurable behavior of a relation expressed as an arrow; the analyst clicks on the arrow connecting the two tasks and selects the number of outputs that will trigger the execution of the following task. Another example could be a constraint expressed as rule such as for a task providing temperature, do not proceed to executing the following task unless the output (temperature) is higher than X degrees Celsius.

- *Model/Task repository:* This component is responsible for *storing reusable tasks, defined relationships and whole process models* and making it available to the modeling toolkit.
- *Simulator engine:* This engine is responsible for *performing a simulated run of the defined process after being mapped* in order not only to evaluate the correct execution and help design but also to collect at a first level performance measures through the process mining engine which will be used by the optimization engine to provide a first set of optimization recommendations during design.
- *ProcessStructure2Model Translator:* This component is responsible for *obtaining the offered predictive & process analytics from the Catalogue* in a format containing the application structure and transforming it to the common representation needed to serve as *input to the Modelling Toolkit* described above.
- *Process mapper:* this component provides an association of *the process tasks defined by the process modelling toolkit* with specific analytical tasks stored in the *Model/Task repository,* following a set of criteria related to each process task. These associations are later enriched, parametrized and concretized in the context of the *data toolkit*. Additionally, it is responsible for mapping specific steps of the workflows defined in the *process modelling toolkit* with *concrete tasks* that maybe required to a process but are not of the data analytics type i.e. generate a report, execute an action using the exposed API of a control application. Additional information for this engine can be found in Section 6.8 regarding predictive analytics that serve the needs of process mapping.
- *Process mining engine:* This component is responsible for collecting performance measures by setting the necessary performance hooks either during a simulation run of the process or during an actual run of the process then through the Seamless data analytics framework. These data will feed the optimization engine to provide recommendations towards reaching the specified goals. Additional information for this engine can be found in Section 6.8 regarding process analytics that serve the needs of process mining.
- *Optimization engine:* This component *provides recommendations on the optimization of a process workflow and mapping* with respect to a goal (objective) set. Optimization may be *offline or online*. *Offline does not require the process to run* and provides a first set of optimization recommendations regarding the workflow design (i.e. detect repeated steps such as a task that generates on the same data several times an output). *Online optimization requires the process to run either in simulation or run over the infrastructure* to use the collected performance measures and provide recommendations towards optimization of the objective set. Additional information

for this engine can be found in Section 6.8 regarding predictive analytics that serve the needs of overall process (i.e. workflow) optimization.

- *ProcessModel2Structure Translator:* This component generates the structure from the developed model that will feed 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 or a task that finds the maximum value of a set of values or a task that when triggered communicates using an API and turns of a machine (if we consider a process that controls the operation of machines).

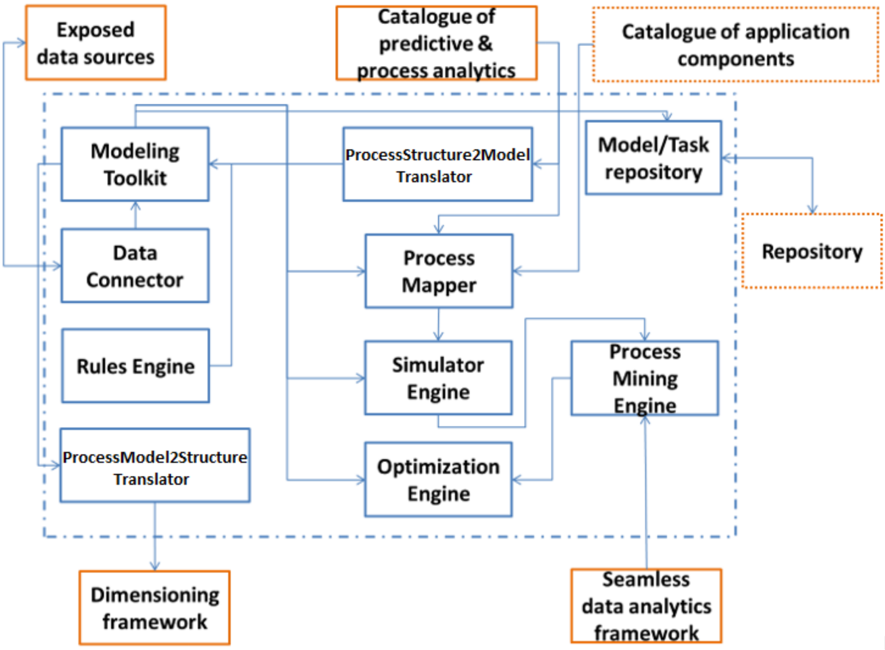


Figure 23 - Process modeling framework building blocks

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 toolkit**. This can be done by considering the outputs of the **process mapper** component or choosing among a set of available or under development analytic functions, while parametrizing them with respect to the objectives defined in the relevant process model. 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).

Towards this direction, the 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 to guarantee their validity.

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 [20]. 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 [21] and NoFlo [22], which will allow for fine-grained control over all the elements associated with the data analytics workflow.

6.14. Adaptable Visualizations

The adaptable visualization layer has a dual purpose: (i) supporting the visualization of data analytics for the applications deployed in BigDataStack, and (ii) provide a visual application performance monitoring dashboard of the data operations and the applications, both during benchmarking and during operation. 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 application and infrastructure performance monitoring dashboard will be adaptable, since it will enable to build a custom dashboard that can include charts with the KPIs chosen by the user, out of a set of chart catalogue and a set of transformation operators, which can aggregate and correlate the received metrics in different manners.

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

- Adaptable dashboards and reports: Dashboards and reports must be possible to be adapted according to the needs of the viewer and provide a personalized view
- 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 or else: Event handling and triggering of alarms or responses to the event must be supported

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
- Output of historical reports data to repository

The main structure of the Adaptable visualizations framework is depicted in Figure 24. The component list is as follows:

- *Visualization toolkit:* this component connects all the components and makes available a tool set of offered capabilities (e.g. types of graphs, reports)
- *Rights management module:* this component handles the permissions to modify views, dashboards, reports, KPIs and event triggers
- *Dashboard module:* this component offers the capability to design and adapt dashboards
- *Reports module:* this component offers the capability to design and adapt reports to be generated
- *KPIs module:* this component offers the capability to define KPIs
- *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

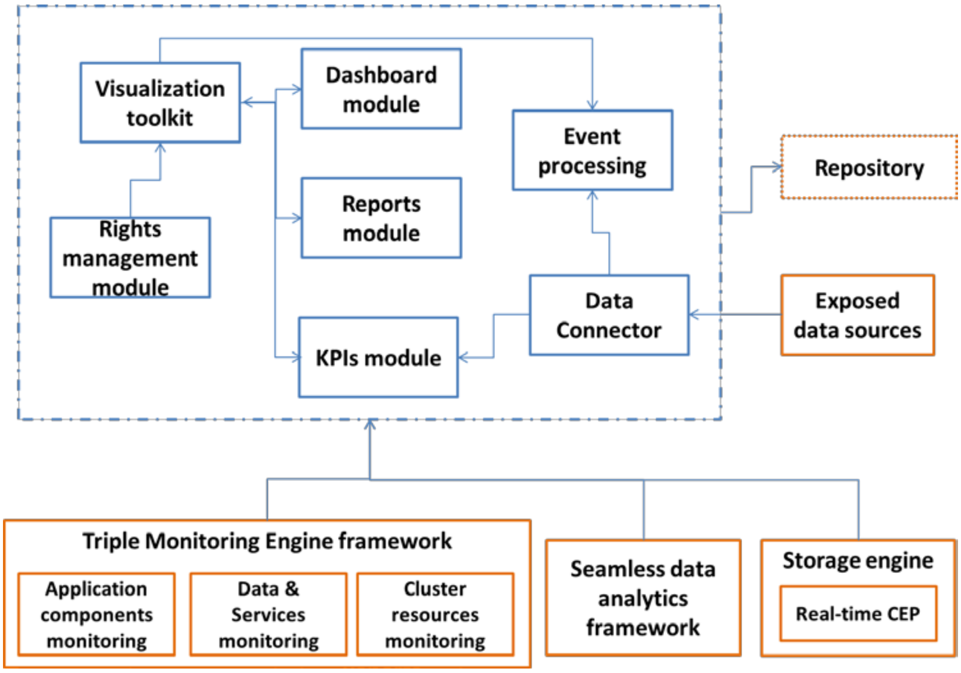


Figure 24 - Visualization framework building blocks

7. Key interactions

7.1. User Interaction Layer

User Interaction with 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 with a high-level description of the workflow from the business analyst's perspective. The sequence diagram of Process Modelling is depicted in Figure 25.

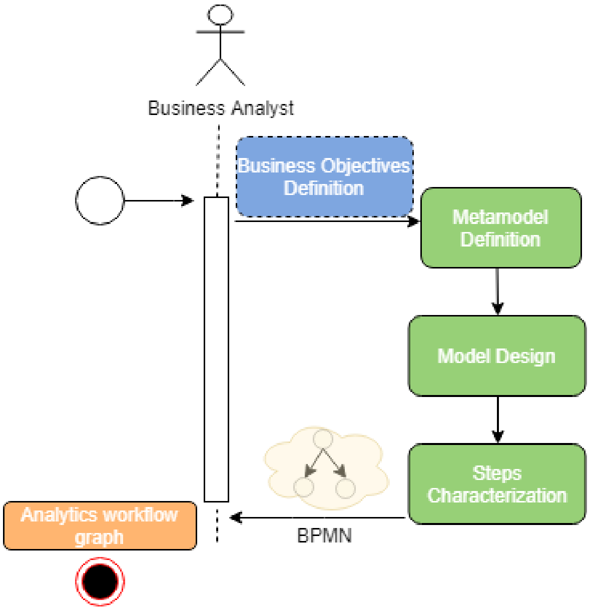


Figure 25 - Information flows in Process Modelling

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 taking into account 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 enriched with the mappings of the business process steps 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, 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, used to evaluate the algorithm/model configurations.

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

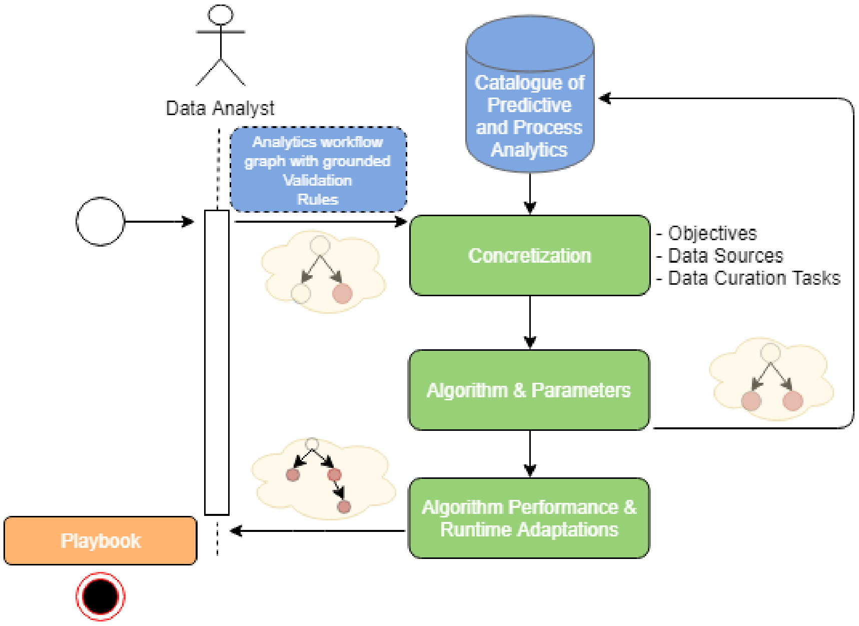

Data Analyst
Catalogue of Predictive and Process Analytics
Analytics workflow graph with grounded Validation Rules
Concretization
- Objectives
- Data Sources
- Data Curation Tasks
Algorithm & Parameters
Algorithm Performance & Runtime Adaptations
Playbook

Figure 26 - Information flows in Process Mapping

In the following, Figure 27 presents the sequence diagram, which depicts all flows in the User Interaction Layer along with inputs, outputs and steps.

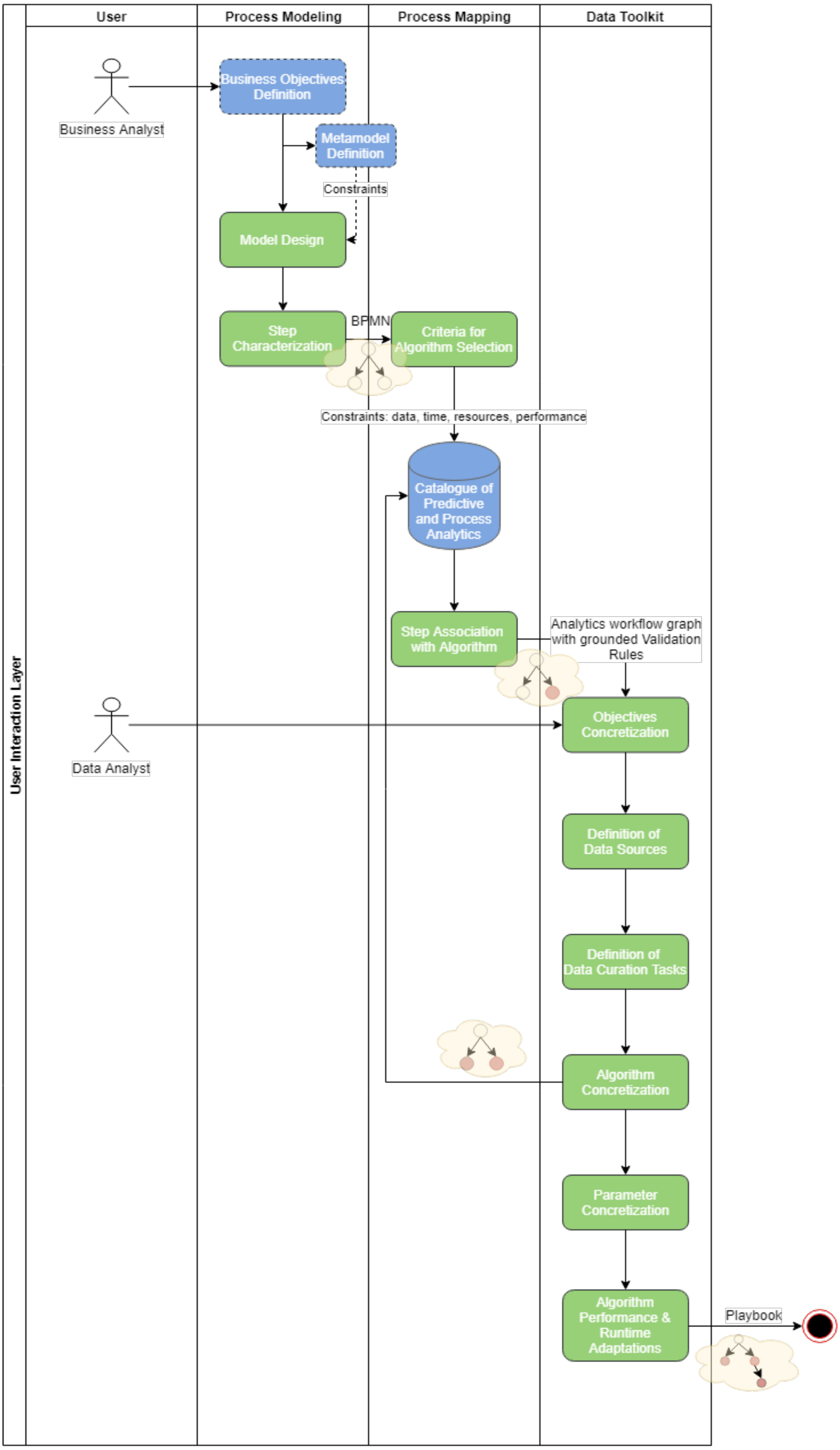

User Interaction Layer
User
Process Modeling
Process Mapping
Data Toolkit
Business Analyst
Business Objectives Definition
Metamodel Definition
Constraints
Model Design
Step Characterization
BPMN
Criteria for Algorithm Selection
Constraints: data, time, resources, performance
Catalogue of Predictive and Process Analytics
Step Association with Algorithm
Analytics workflow graph with grounded Validation Rules
Data Analyst
Objectives Concretization
Definition of Data Sources
Definition of Data Curation Tasks
Algorithm Concretization
Parameter Concretization
Algorithm Performance & Runtime Adaptations
Playbook

Figure 27 - User Interaction Layer Sequence Diagram

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 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.

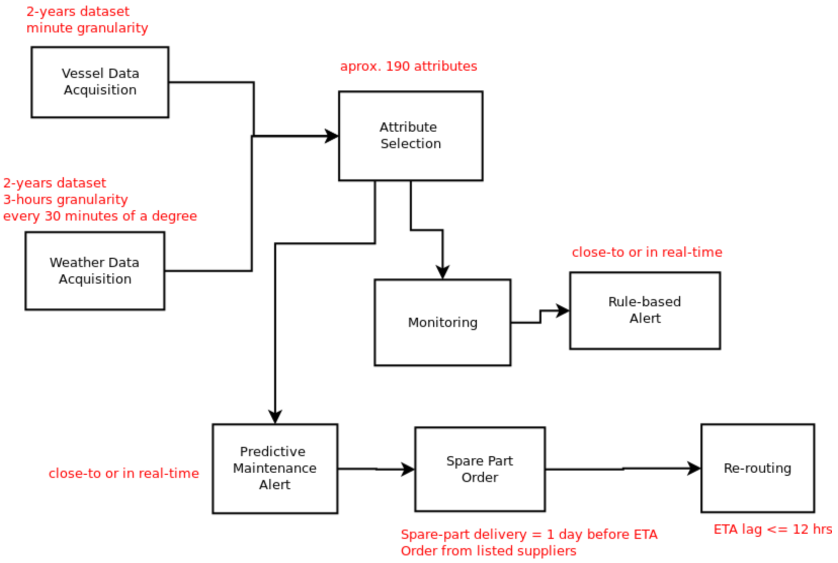


Figure 28 - Business analyst view

Data Analyst’s View

Following the outcome of the process modelling (previous view), the following figure 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.

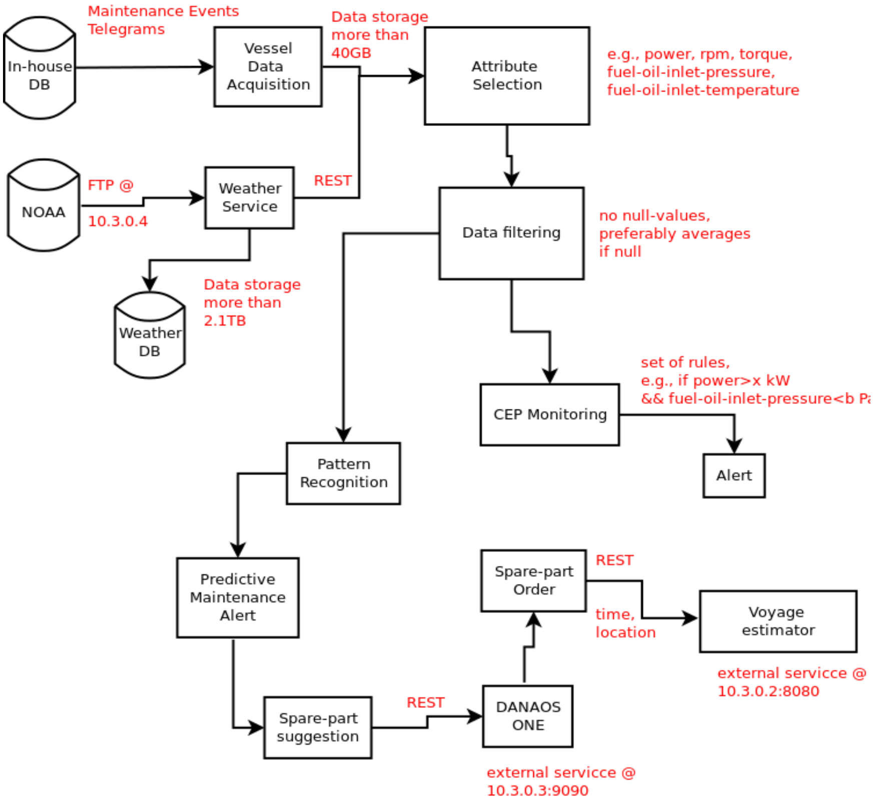

In-house DB
Maintenance Events
Telegrams
Vessel Data Acquisition
Data storage more than 40GB
Attribute Selection
e.g., power, rpm, torque, fuel-oil-inlet-pressure, fuel-oil-inlet-temperature
NOAA
FTP @
10.3.0.4
Weather Service
REST
Data filtering
no null-values, preferably averages if null
Weather DB
Data storage more than 2.1TB
set of rules, e.g., if power>x kW && fuel-oil-inlet-pressure<b P
CEP Monitoring
Alert
Pattern Recognition
Predictive Maintenance Alert
Spare-part Order
REST
time, location
Voyage estimator
external servicce @ 10.3.0.2:8080
Spare-part suggestion
REST
DANAOS ONE
external servicce @ 10.3.0.3:9090

Figure 29 - 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. 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:

- **Model Learning**: This task is concerned with the creation of a supervised model for ranking deployment patterns that can later be used on-demand.
- **First-Time Ranking of Candidate Deployment Patterns and Subsequent Deployment**: 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, and then uses that pattern to deploy the user’s application on the cloud.

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	Feature Aggregator	ADS-R Feature Aggregator
Application and Data Services Ranking	Learning to Rank	ADS-R LTRank
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
Global Decision Tracker	N/A	Global Decision Tracker

Table 4 - Short-name Component Mapping Table

Model Learning

The goal of model learning is to produce a learning to rank model that takes in a list of candidate deployment patterns and scores each based on their suitability to the user's requirements. This model is learned based on past application deployments and logging data associated to those deployments. In short, we aim to learn what makes effective deployments from the past, and then use that information to select new deployments in the future.

Model learning is triggered by the Dynamic Orchestrator component within BigDataStack, as the central control node within Realization. This trigger is anticipated to be based on the availability of new deployment logging data that we can learn from. Once triggered, the Updater sub-component of Application and Data Services Ranking (ADS-R) will request the set of past deployments and log data (containing quality of service information) from the Global Decision Tracker. This data is used internally to learn a new learning to rank model. Learning happens asynchronously. Once leaning is complete, the new model is sent to the Model sub-component of ADS-R, where it can be used in the other two tasks discussed below.

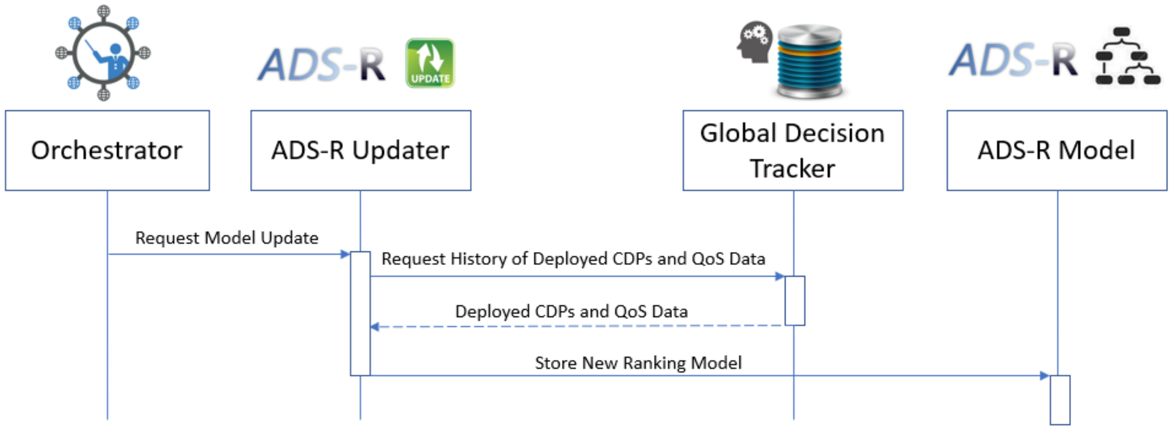


Figure 30 - Interaction Diagram for Model Leaning

First-Time Ranking of Candidate Deployment Patterns and Subsequent Deployment

The second 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 was 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 component, or more specifically the Feature Aggregator 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 Learning to Rank sub-component, which uses a pre-trained 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 Selector sub-component, which selects the best one. This selected CDP is then sent to the Application and Data Services Deployment component for physical deployment. 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 for persistence.

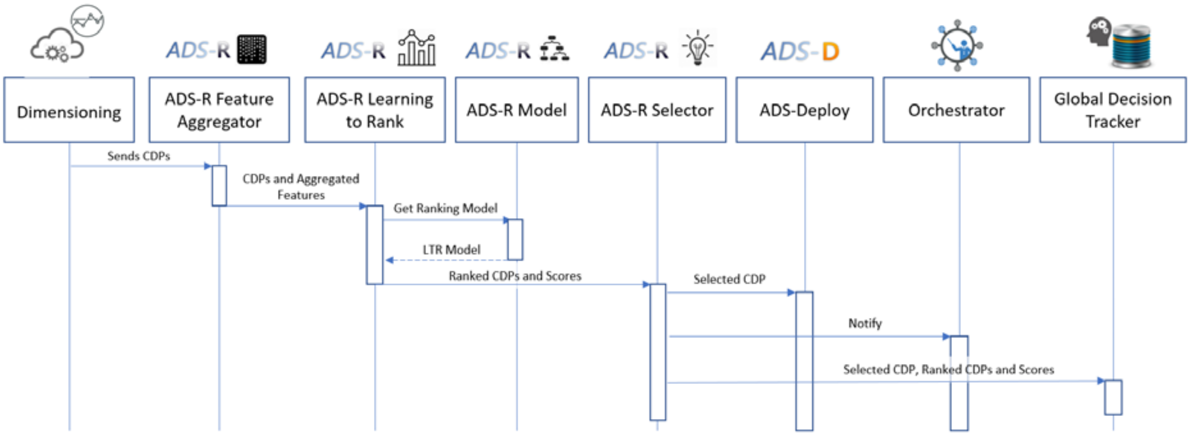


Figure 31 - Interaction Diagram for First-Time Ranking

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 32).

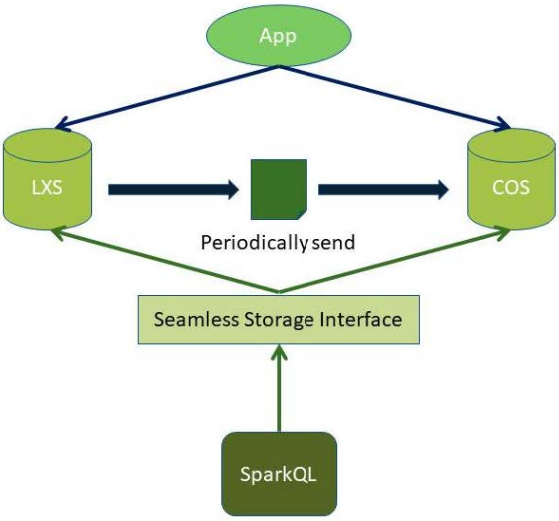


Figure 32 - Architecture of data stores

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 data base capable of storing numerous terabytes of data, but lacks transactional processing and does not offer SQL capabilities. As a result, data will be firstly inserted in LeanXcale (LXS), which ensures transactional properties. However, after a period of time (i.e. one year or several months), data will be periodically send to the Cloud Object Store (COS) which can offer analytical processing over historical data.

On top of the datastores lies the Seamless Storage Interface (SSI) which provides a seamless way for executing queries over a distributed dataset held by different datastores that provide different interfaces. The connectivity for information retrieval can be facilitated via the use of SparkQL, as the SSI has been already integrated with the latter. As a result, the end-user can write SparkQL queries and let the SSI decide where the data is located and obtain the result. However, the execution of the queries directly to each data store is also permitted. As a result, we have the following scenarios:

Direct access the LXS

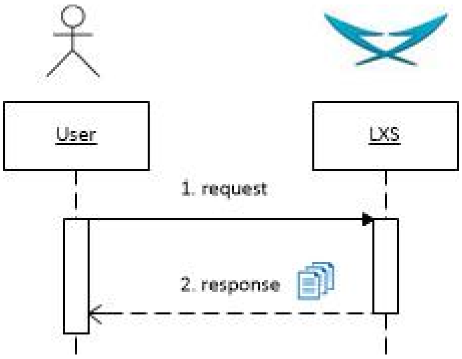


Figure 33 - Direct access the LXS

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 COS

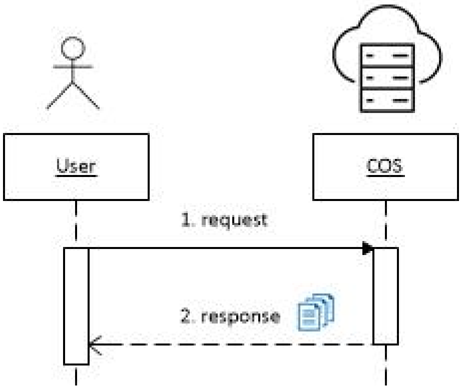


Figure 34 - Direct access the COS

User executes a query, 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 SparkQL query

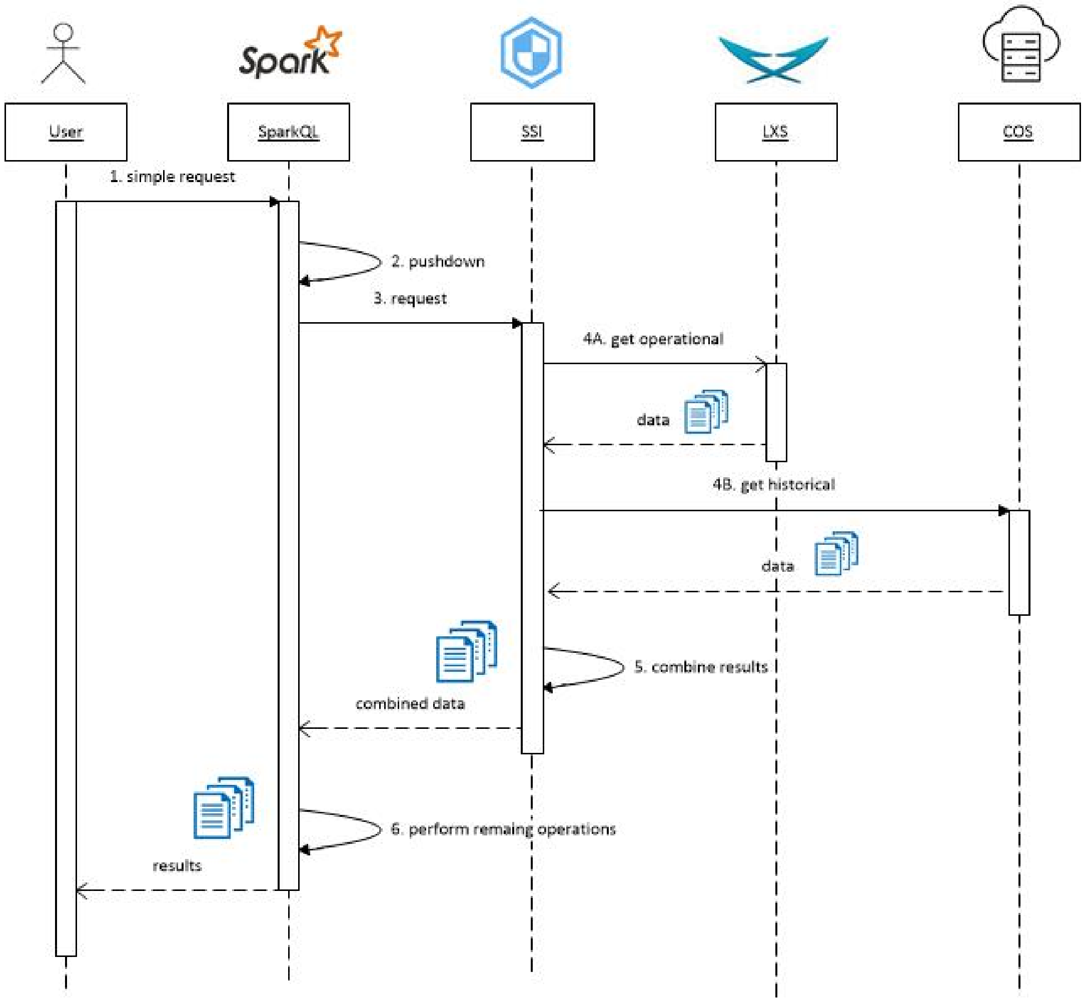


Figure 35 - Request data using a simple SparkQL query

User sends a request for data retrieval writing a SparkQL query. The SSI is already integrated with SparkQL and provides a set of SQL capabilities. According to the query to be executed, SparkQL can *pushdown* some operations that can 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 datastore's internal language and request the data from both of them. At the end, it aggregates the results and returns the data back to SparkQL. Finally the latter, executes the operations that could not be *pushed down* due to the incapability of SSI, and returns the final results to the user. Notice that steps 4A and 4B might be in parallel according to the implementation of SSI.

Request data using combined query on SparkQL

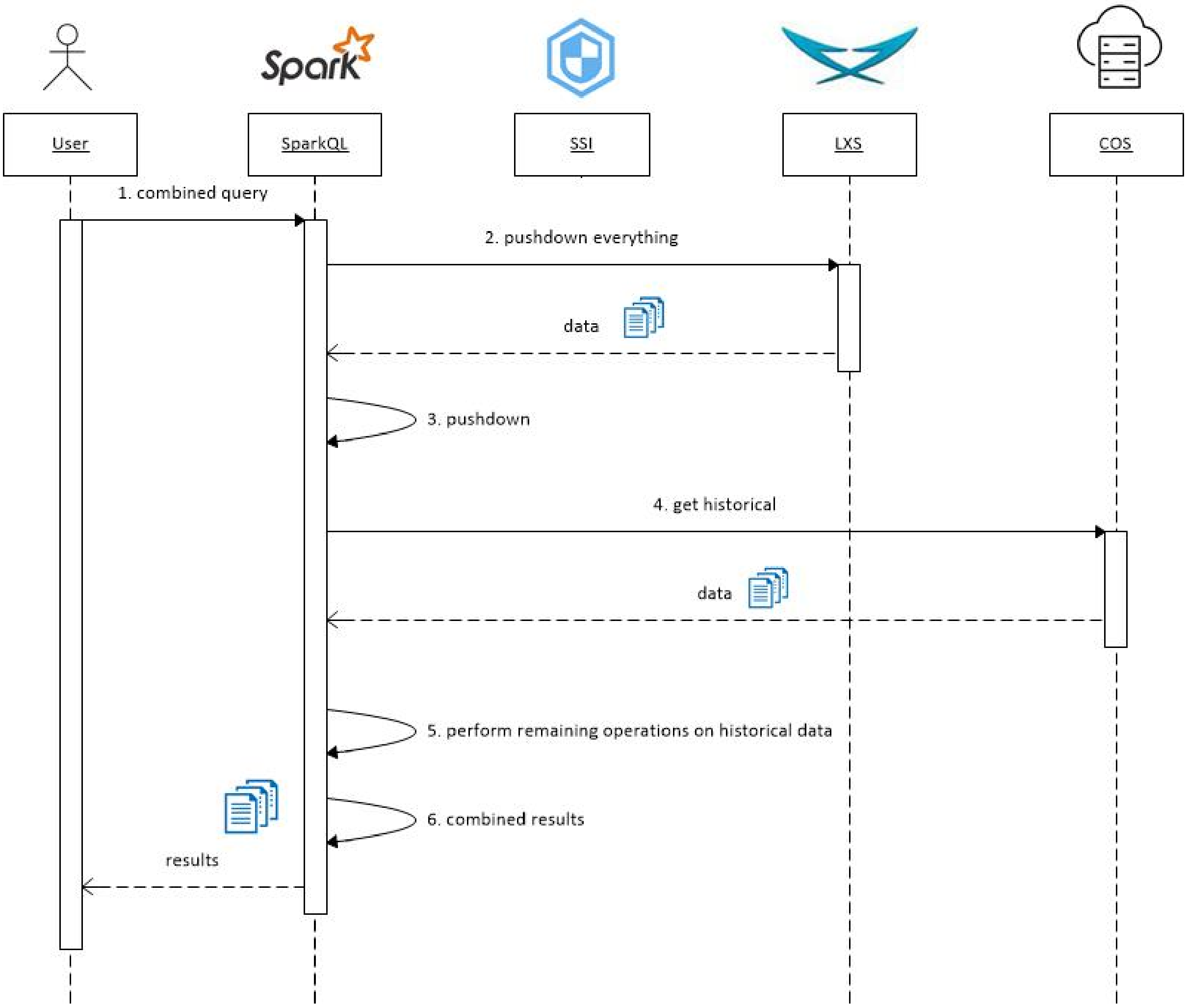


Figure 36 - Request data using combined query on SparkQL

As LXS provides a JDBC implementation, it is fully compliant with the SQL standard. Moreover, SparkQL provides JDBC connectivity. As a result, SparkQL can *pushdown* all operations it supports to a relational database in order to improve performance. This makes possible a slightly different use case where the user can write a SparkQL query combining data coming from two different datastores. SparkQL will *pushdown* all operations that concerns LXS and retrieve the data from it much faster. It will also *pushdown* as many operations as supported by COS. It will then perform the remaining operations that could still have to be executed and finally will combine the results and will return the response. This might lead to an improved performance, as a part of the operations will be executed at the datastore level, and data might not need to be transmitted up to SparkQL to be evaluated against the remaining operations. The main disadvantage of this scenario is that the user must be aware of data distribution across the two data stores in order to write the appropriate query.

Insert data to BigDataStack

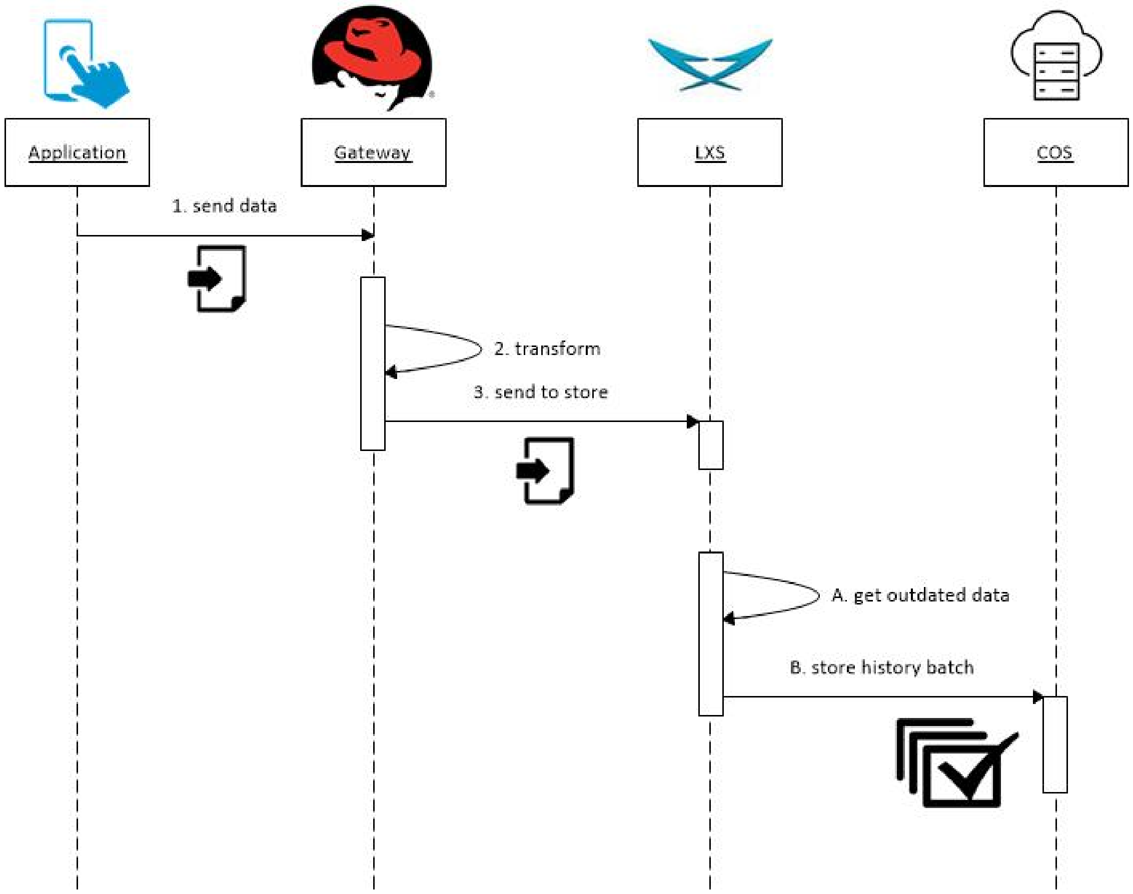


Figure 37 - Inserting data

An integrated application produces data that wants to store in the BigDataStack platform. The data are being sent to the *Gateway*, which is the entry point for 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 collects all data that has been inserted from more than a constant period of time, which are now considered historical and hence are not needed to be stored to a relational database, and as such sends them to the COS for further analytical processing.

Inserting streaming data to BigDataStack

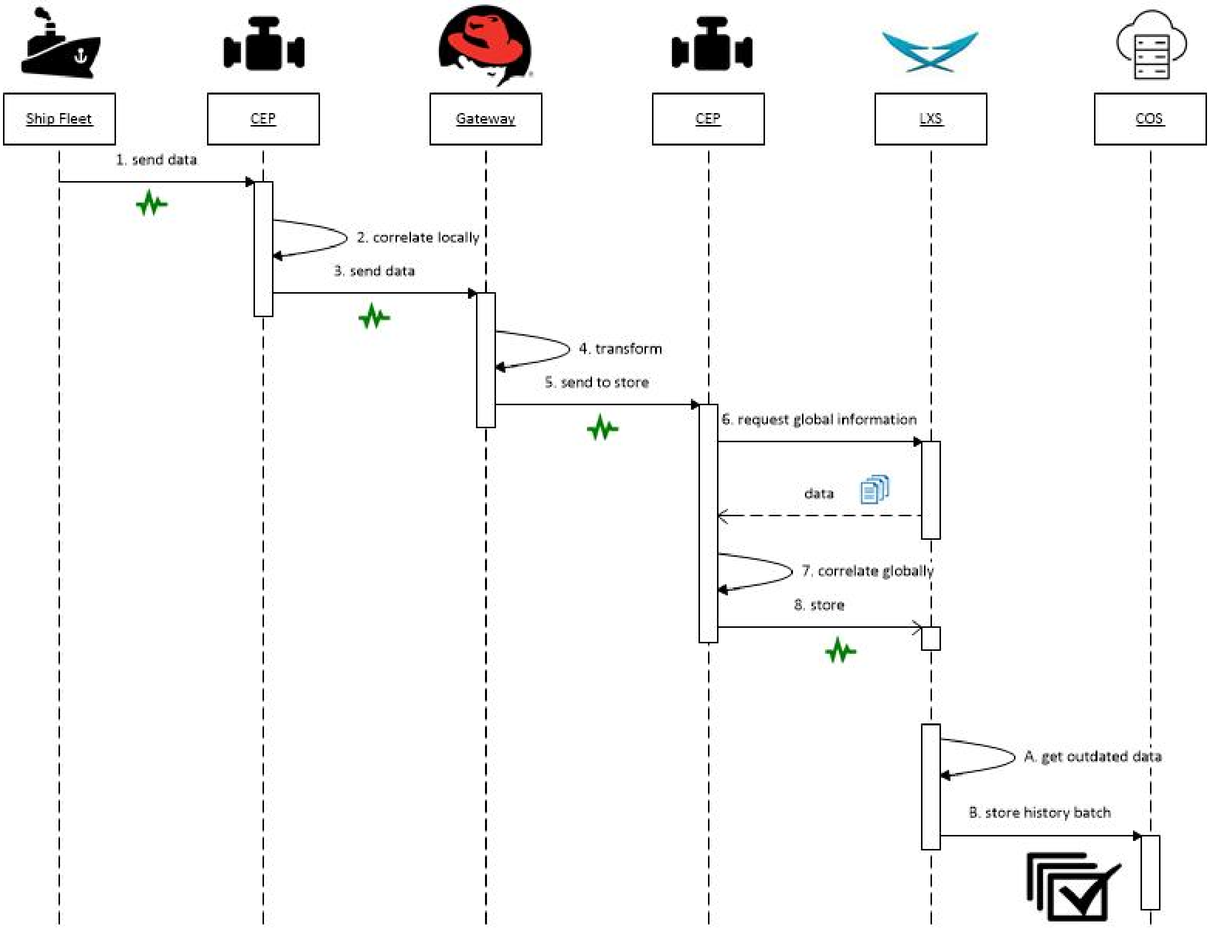


Figure 38 - Inserting streaming data

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 in order for the latter to correlate them locally and identify possible threats, producing alerts. Then, it sends the data to the platform´s Gateway who is responsible of transforming the data to the platform's internal format. A CEP is also installed inside the platform and receives data from the Gateway in order to correlate them globally and identify potential opportunities, combining data coming from all the fleet. The global correlation involves a query to 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. The latter, as explained before, periodically collects all data that have been inserted from more than a constant period of time.

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, i.e. 1) the interactions required to perform the actual monitoring of a running application and 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:

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 plugins. In the case of the deployment of an exporter or a Netdata plugin 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.

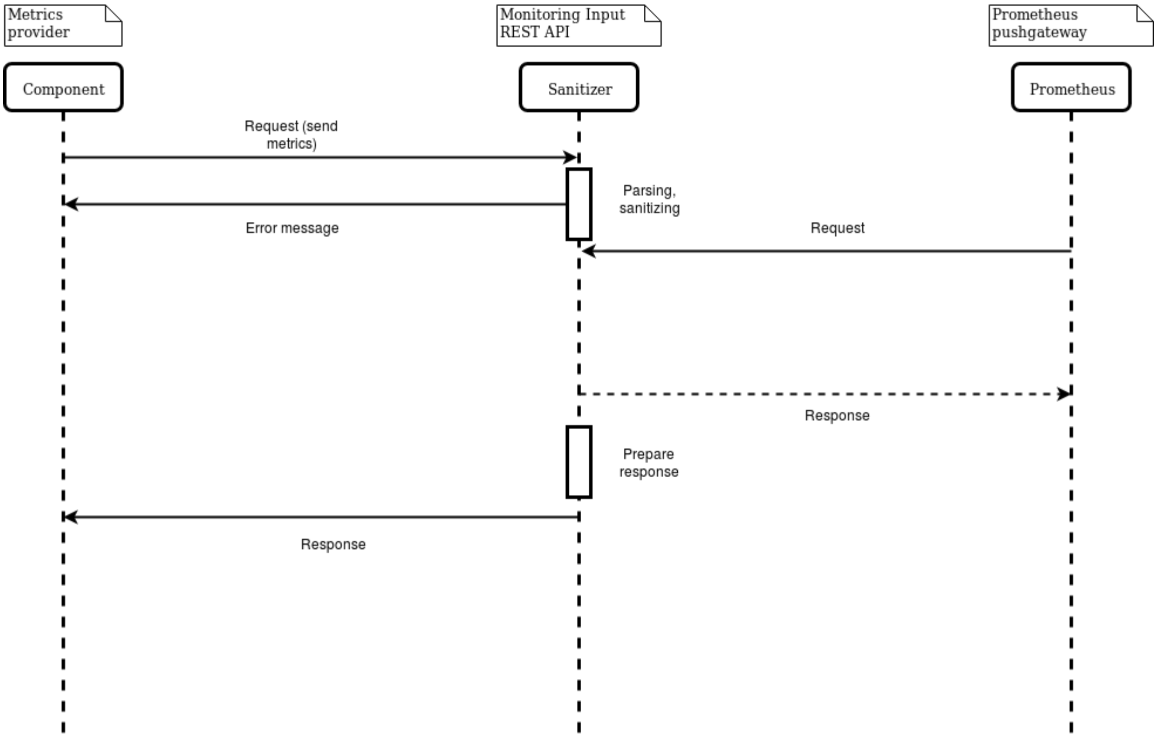


Figure 39 - Prometheus exporters

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 performed by the amount of time specified in the Prometheus's configuration.

The triple 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 Prometheus query and return a result. The following sequence describes this process.

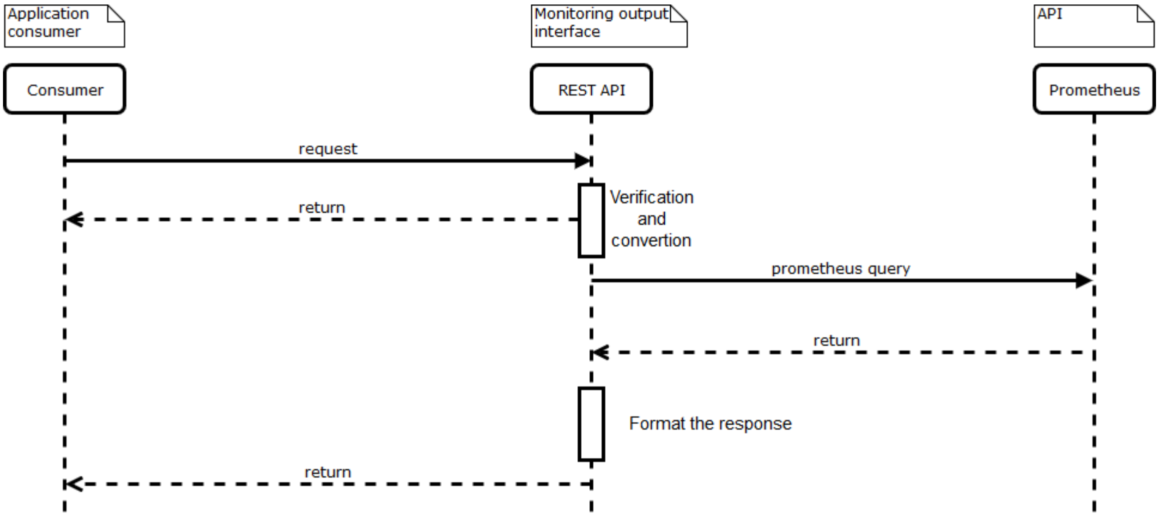


Figure 40 - Prometheus REST API

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

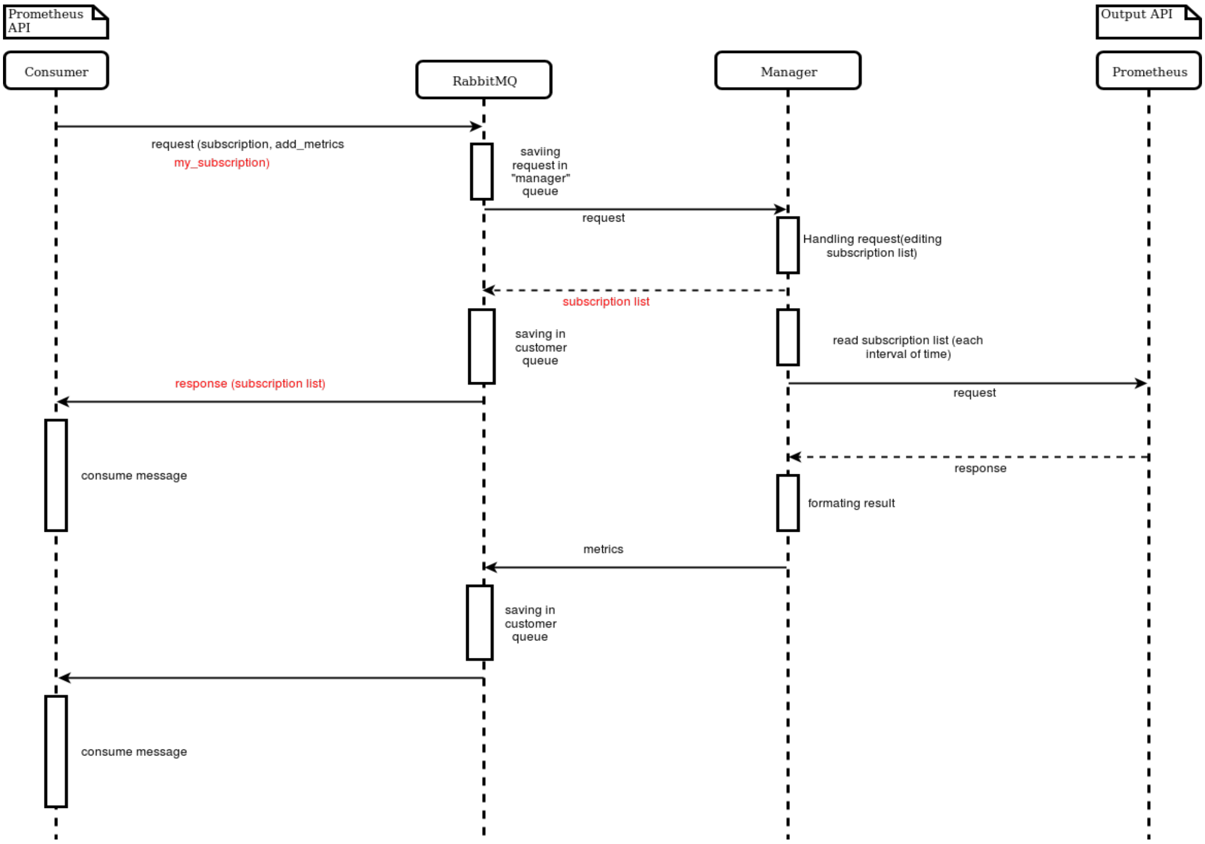


Figure 41 - Publish/subscription mechanism

An application that needs steaming data can through this component subscribe and receives metrics in real-time. Three types different of requests are available. The first request type is the “subscription”, the consumer after having created his queue, its 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 add this last in the subscription list. A confirmation message is then returned to the consumer. Each interval of time (defined base of the minimum scrape interval in Prometheus's configuration) the manager reads the subscription list, get metrics from Prometheus and publishes them in the concerned consumer's 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 last request type is "my_subscription", the consumer send its name and queue name. The manager returns the corresponding subscription list.

Quality of Service (QoS) Evaluation

QoS properties (parameters) to be evaluated by the QoS Evaluation component should correspond to the kind of KPIs coming from the Application Dimensioning Workbench within the Playbooks.

- An example of KPI is the "throughput."
- There should be a trivial mapping between Playbooks' KPIs and the "terms" of SLA "agreements".

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

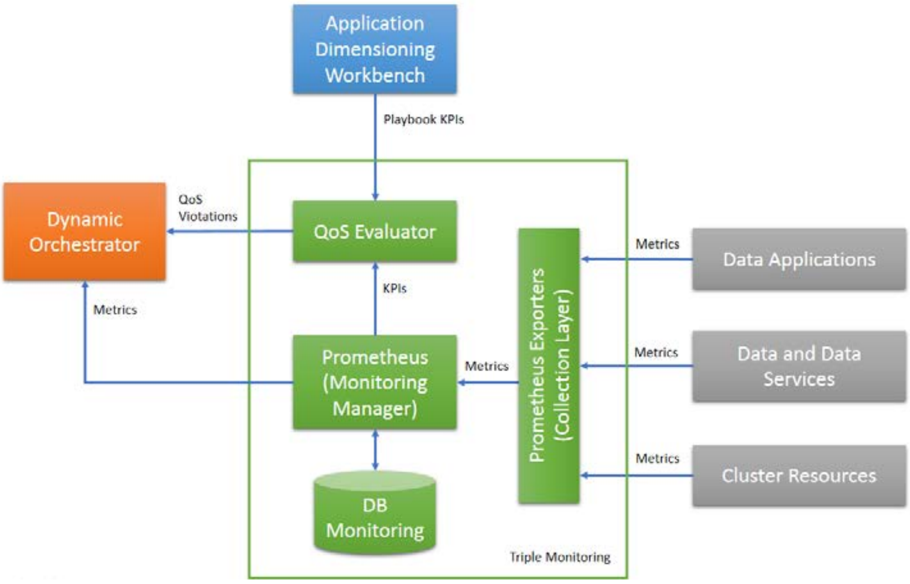


Figure 42 - QoS Evaluation component

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 QoS metrics from Prometheus.
- The Evaluator calls the Notifier when SLO violation is detected.
- Notifier calls Dynamic Orchestration passing a message describing the violation and possibly any applicable penalty.

Dynamic Orchestration communicates with the ADS (Application and Data Services) Ranking to trigger the dynamic adaptation (re-configuration) of the application or data service deployment pattern.

Adapting at Runtime

If a user's application is identified or predicted to have some deficiency with respect to the quality of service, 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.

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 above), 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).

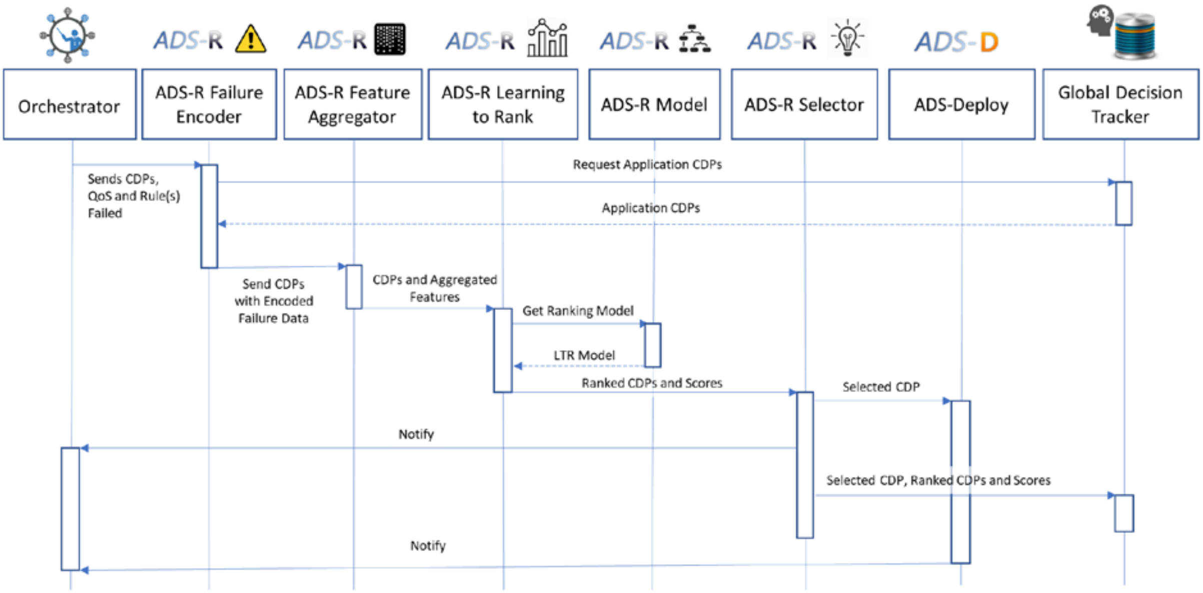


Figure 43 - Interaction Diagram for CDP Re-Ranking

8. Conclusions

This document summarizes the initial version of the BigDataStack architecture, including the offerings / capabilities of the overall environment to different stakeholders as well as the architecture of the complete environment. Additional information is provided on a component level, while the report also captures the main interactions between key components of the architecture.

This report serves as a basis for the detailed design specifications of the individual components of the architecture (to be performed on work package level and reported in the relevant deliverables of the work packages) and drives the activities of the project towards the overall vision and objectives. An updated version of the architecture will be released in M18 (June 2019) of the project following the outcomes of the initial integrated prototypes and the obtained experimentation and validation results.

9. References

[1] Docker compose, https://docs.docker.com/compose/compose-file/

[2] OpenShift Origin Kubernetes Distribution, https://www.openshift.com/

[3] [1] J. Duncan, J. Osborne "Chapter 1. Getting to know OpenShift" in "OpenShift in Action". Manning Publications Co. ISBN 978-1-6172-9483-9, 2018

[4] 2 RedHat, "How Deployments Work", https://docs.okd.io/latest/dev_guide/deployments/how_deployments_work.html

[5] 3 Larry Carvalho Matthew Marden, "The Business Value of Red Hat OpenShift", https://cdn2.hubspot.net/hubfs/4305976/s3-files/idc-business-value-of-openshift.pdf

[6] 5 RedHat, "What's New in Red Hat OpenShift Origin 3.10 OpenShift Commons Briefing", https://blog.openshift.com/wp-content/uploads/Whats-New-in-Origin-3.10.pdf

[7] [3x] Geard, https://openshift.github.io/geard/

[8] [3y] Project Atomic, https://www.projectatomic.io/

[9] 4 RedHat, "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

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

[11] BigDataStack (July 2018). "D2.1 – State of the art and Requirements analysis - I"

[12] V. Gulisano, R. Jiménez-Peris, M. Patiño-Martínez, C. Soriente, P. Valduriez, "StreamCloud: An Elastic and Scalable Data Streaming System", IEEE Trans. Parallel Distrib. Syst. 23(12): 2351-2365, 2012

[13] Apache Storm, http://storm.apache.org/

[14] Apache Flink, http://flink.apache.org

[15] Apache Spark, https://spark.apache.org/streaming

[16] Paula Ta-Shma, "How to Layout Big Data in IBM Cloud Object Storage for Spark SQL", https://www.ibm.com/blogs/bluemix/2018/06/big-data-layout/

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

[18] Workflow Patterns Initiative http://www.workflowpatterns.com/R

[19] IOStack H2020 project, http://iostack.eu

[20] Node-RED, https://nodered.org/

[21] Data-Driven Documents, https://d3js.org/

[22] Flow-Based Programming for JavaScript - NoFlo, https://noflojs.org/

Appendix 1 – DANAOS dataset structure and description

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),

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

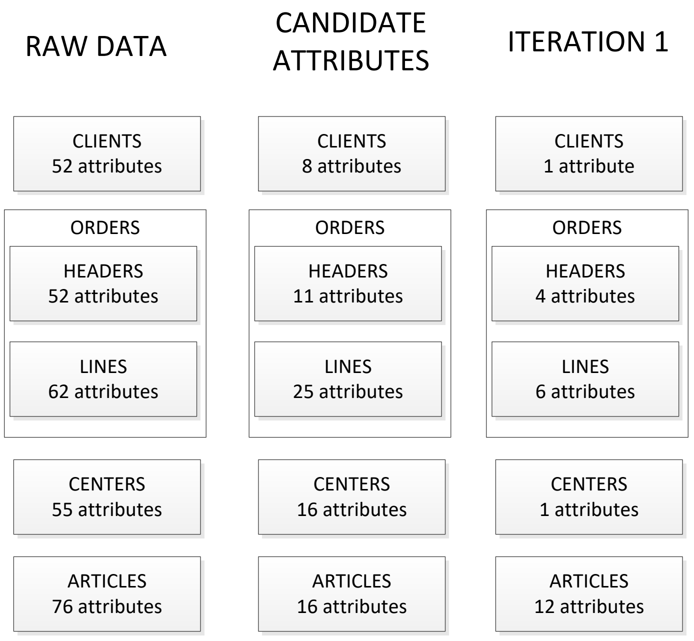
cause_of_damage: Short description for cause of damage

Appendix 2 – ATOS WORLDLINE 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.



RAW DATA	CANDIDATE ATTRIBUTES	ITERATION 1
CLIENTS 52 attributes	CLIENTS 8 attributes	CLIENTS 1 attribute
ORDERS	ORDERS	ORDERS
HEADERS 52 attributes	HEADERS 11 attributes	HEADERS 4 attributes
LINES 62 attributes	LINES 25 attributes	LINES 6 attributes
CENTERS 55 attributes	CENTERS 16 attributes	CENTERS 1 attributes
ARTICLES 76 attributes	ARTICLES 16 attributes	ARTICLES 12 attributes

Figure 44 - Dataset structure and description

CLIENTS table structure (8 over 52 attributes)

ID_CLIENTE: Client id,

COD_TARJ_TRAVEL: Travel card id,

FLG_EROSKI_CLUB: Flag to check if the client belongs to Eroski Club or not,

FLG_CLIENTE_ORO: Flag to check if the client is gold or not,

FECHA_ALTA_ORO: Date when the client became gold,

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,

COD_MISION_COMPRA: Purchase reason (weekly purchase, holidays ...).

HEADERS structure (11 over 52 attributes) – FROM TICKETS

ID_CLIENTE: Client id,

COD_LOC: Store's localization id,

DIA: Ticket's date,

COD_CAJA: Till's id,

NUM_TICKET: Ticket number (id),

HORA_EMISION: Timestamp of tickets emission,

COD_F_PAGO_DET -> M_FORMA_PAGO: Type of payment procedure,

HORA_INI_FAC: Invoice start time,

HORA_FIN_FAC: Invoice end time,

COD_TIPO_VENTA: Type of sale,

COD_TIPO_CENTRO: Type of center.

LINES structure (25 over 62 attributes) – FROM TICKETS

ID_CLIENTE: Client id,

COD_LOC: Store's localization id,

COD_ART: Article's id,

DIA: Day,

COD_CAJA: Till id,

NUM_TICKET: Ticket number (id),

NUM_LINEA: Line number (id),

HORA_EMISION: Timestamp of tickets emission,

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,

COD_N1_HIST: Area,

COD_N2_HIST: Section,

COD_N3_HIST: Category,

COD_N4_HIST: Subcategory,

COD_N5_HIST: Segment,

ANO_OFERTA: Offer's year,

COD_OFERTA_COD_EAN: Reference EAN's code.

CENTERS structure (16 over 55 attributes)

COD_LOC: Store's localization id,

COD_PROVIN: Province id,

DESC_LOC: Center's description,

DESC_PROVIN: Province's name,

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,

ARTICLES structure (16 over 78 attributes)

COD_ART: Article's id,

DESC_ART: Article's description,

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