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/

D4.1 – WP4 Scientific Report and Prototype Description - Y1

Work Package	WP4 – Data as a Service: Data Paths Services
Lead Author (Org)	Paula Ta Shma, Yosef Moatti (IBM)
Contributing Author(s) (Org)	Stathis Plitsos (Danaos) Guy Khazma (IBM) Pavlos Kranas (LeanXscale) Luis Tomás Bolívar (RedHat) Marta Patiño, Ainhoa Azqueta (UPM) Christos Doulkeridis, Peter Jason Gould, Dimitris Poulopoulos (UPRC)
Due Date	30.11.2018
Date	29.11.2018
Version	1.0

Dissemination Level

X	PU: Public (*on-line platform)
	PP: Restricted to other program 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

Table 1 – Versioning and contribution history

Version	Date	Author	Notes
0.1	04.07.2018	Yosef Moatti (IBM)	Skeleton
0.2	16.10.2018	Yosef Moatti (IBM)	Update of skeleton
0.3	25.10.2018	Dimitris Poulopoulos (UPRC) Yosef Moatti (IBM)	First version of data Cleaning Update of skeleton
0.4	1.11.2018	Pavlos Kranas (LXS)	Contribution to the seamless analytical framework
0.5	10 Nov 18	Marta Patiño (UPM) Ainhoa Azqueta (UPM)	Contribution to the real-time CEP
0.6	6 Nov.18	Paula Ta Shma / Yosef Moatti (IBM) Pavlos Kranas (LXS)	Contribution to section 4 Contribution to section 8
0.7	11 Nov 18	Paula Ta Shma / Yosef Moatti (IBM)	Contributions to sections 3 & 6
0.8	12 Nov 18	Yosef Moatti (IBM)	Improvements
0.9	12 Nov 18	Peter Gould (UPRC)	Contribution to section 9 + other small changes
0.10	12 Nov 18	Marta Patiño (UPM) Pavlos Kranas (LXS)	Contribution to section 10 Contributions to roadmap table, mapping with use cases on sections 7-8, description of design of section 8, and update the early prototype and next steps of 7-8.
0.11	13 Nov 18	Paula Ta Shma (IBM) Yosef Moatti (IBM)	Section 6 Contributions to sections 1 2 3 & 8 + minor fixes
0.12	14 Nov 18	Paula Ta Shma (IBM)	Contribution to section 6
0.13	14 Nov 18	Guy Khazma (IBM) Stathis Plitsos (DAN) Yosef Moatti (IBM) Dimitris Poulopoulos (UPRC)	Improvement for section 6 Improvements for section 3 Fix forgotten 10.5 & 10.6 text Section 5.1 – Changes in 5.2 till 5.6 are not clear since design figure “disappeared” in new version
0.14	14 Nov 18	Marta Patiño (UPM) Dimitris Poulopoulos (UPRC) Paula Ta Shma (IBM) Peter Gould (UPRC)	Completion of section 10 T4.1 text now fully integrated Completion of section 8 –ready for Pavlos’ review Completion of section 9
0.15	15 Nov 18	Marta Patiño (UPM)	2 text additions in 3.1 and 3.3
0.16	15 Nov	Yosef Moatti (IBM) Peter Gould (UPRC) Luis Tomas Bolivar (RHT) Yosef Moatti (IBM)	Remove all table footnotes except for first requirement table 2 contributions for section 2.2 and 4 Contribution to section 4.1 and 4.3 Requirements for T4.2 (section 6) Conclusion

0.17	18 Nov 18	Yosef Moatti (IBM) Stathis Plitsos (DAN) Paula Ta Shma (IBM)	Improvement of REQ-SDAF-05 Improvement of section 4.1 Changes in requirements at section 6.1
0.18	18 Nov 18	Dimos Kyriazis (UPRC) Yosef Moatti (IBM)	Review of the 0.17 version **Partial** fixes from Dimos' review
0.19	25 Nov 18	Yosef Moatti (IBM)	Some more of Yosef' fixes for Dimos' review. All missing fixes from partners are mentioned with a @ mark (e.g., @UPM)
0.20	26 Nov 18	Dimitris Poulopoulos (UPRC)	Fixes Dimos' comments for Data Quality section
0.22	26 Nov 18	Yosef Moatti (IBM) Pavlos Kranas (LXS) Dimitris Poulopoulos (UPRC)	Almost all of Stelios comments are now handled Almost all of Dimos' comments handled except for task specific for which I wait for answers from task leaders (Pavlos is done) Answers from LeanXcale to Dimos' remaining comments Addresses most of Dimos' comments
0.23	27 Nov 18	Yosef Moatti (IBM) Pavlos Kranas (LXS) Dimitris Poulopoulos (UPRC) Peter Gould (UPRC)	Fix Amarylis' comments Further fix Dimos' comments Futher fix Stelios' comments
0.24	28 Nov 18	Dimitris Poulopoulos (UPRC) Pavlos Kranas (LXS)	Fix in Executive Summary Added text and Grafana figure in section 6.3
0.25	29 Nov 18	Christos (UPRC)	Fig 2 modified to be consistent with Fig 5 Tentative co-authors list – last opportunity to fix! Anyone
0.26	29 Nov 18	Paula Ta Shma (IBM) Marta Patiño (UPM) Peter Gould (UPRC) Yosef Moatti (IBM) Dimitris Poulopoulos (UPRC)	Multiple improvements to IBM related text Improvement in section 1 and 8 Minor improments in section 9 Fixes for latest Dimos' review Changes in section 8 (no longer dependent on CEP)
1.0	30 Nov 18	Dimitris Poulopoulos (UPRC)	Small fix in 4.2

Table of Contents

List of tables

List of figures

1. Executive Summary

The BigDataStack project was conceived as a data centric platform, integrating approaches for Data as a Service. Its data services are covered in this work package and are naturally at the core of BigDataStack.

This first deliverable of WP4 presents the architecture and the design of these data services. Most of the below are considered also to be standalone techniques that can be used separately. However, in the context of BigDataStack, they run on top of the data-driven infrastructure management system and provide the required data services.

- Big Data Layout and data skipping (section 5) covers automated big data layout, as well as state of the art skipping techniques, in order to improve SQL analytics on rectangular data in object storage. A significant recent novel achievement includes data skipping for arbitrary SQL predicates (including AND/OR/NOT, User Defined Functions and built-in functions). This component also aims to research automatic algorithms for dynamic data layout and data skipping index creation, driven by analysis of the data properties and query history.
- The adaptable distributed storage component (section 29) which is based on the LeanXcale relational datastore and whose core achievement is to enhance its distributed storage engine to offer the following abilities which are beyond the current state-of-the-art:
    - dynamic reconfiguration of data regions to achieve optimal balance (done by fragmenting datasets and distributing them across the data nodes)
    - scaling in/out per as function of (lack of) resources availability.
    - Ensuring data consistency and transactional semantics but without causing: a) any downtime, b) any significant performance reduction, c) discarding any data modification operation
- The Data Quality mechanisms (section 8), which offers domain-agnostic data cleaning, veracity and enhancement, while also checking for data source malfunction or performance deterioration.
- The Predictive and Process Analytics component (section 9) strives, using multiple process mining algorithms, to analyse, structure and enhance process models derived from event driven data. The use of multiple algorithms on a given event log aims at maximizing the four evaluation metrics of process mining:
    - Replay fitness (ability to replay the log over the generated model)
    - Precision (not underfitting the log)
    - Generalization (not overfitting the log)
    - Simplicity
- the Complex Event Processing (CEP) (section 10) which will run on geo-distributed environments in order to process the data as close to the source as possible to avoid delays in the processing and optimize resource consumption.

The seamless data analytics framework (section 7) builds on top of the LeanXcale database and IBM Cloud Object Storage, where the latter may be enhanced by an acceleration cache. This service can present to data scientists a single logical view for a dataset distributed across heterogeneous data stores (LeanXcale database and object storage).

When combined, the aforementioned set of data services is powerful. This is what we show in section 4 where we detail two common data scenarios: a) data ingestion; b) data query. These two scenarios were chosen for the first iteration / phase of the project because of their importance. The goal is not only to show the strength of solutions built out of BigDataStack data services but also the ease with which they can be assembled. To this end, sub-section 4.2**Error! Reference source not found.** also gives a roadmap of the data services development up to M18.

2. Introduction

2.1. Relation to other deliverables

This deliverable is related with a set of other project deliverables as described below:

- D2.2 - Requirements & State of the Art Analysis II (M12). Collected requirements have been analysed to drive the design specifications of data services (top-down approach), while technical requirements from the data services have also been collected and analysed to provide input to other components of the overall architecture (bottom-up approach).
- D2.4 - Conceptual model and Reference architecture (M9) which is a high-level preview of the more detailed and advanced D4.1 which extends and details the relevant part of D2.4
- D3.1 and D5.1 – respectively WP 3 and WP 5 Scientific Reports and Prototype Descriptions (M12). Alongside D4.1 Data as a Service, D3.1 and D5.1 present the current technical status, Data-driven infrastructure management and Dimensioning Modelling and Interaction services respectively, of the BigDataStack project.

2.2. Document structure

The structure of this deliverable follows the structure of the overall Data as a Service main building block: one section is dedicated to each one of the key components / mechanisms of the envisioned data services. Since there are no real dependencies between the components, the order of these sections is not very significant. One exception is the seamless data analytics framework (section 7), which combines the capabilities of the LeanXcale data base (section 6) as well as of the Big Data layout and data skipping (section 5) and which comes after both.

Furthermore, two data analysis services are additionally offered in the context of the data services layer, the data cleaning (section 8) and the predictive and process analytics (section 9).

Prior to these sections, section 3 gives an overview of the various components / mechanisms that build up the overall data services block. It is followed by section 4 that describes two Big Data use case scenarios which demonstrate how these data services can be combined to solve real problems.

3. Solution Architecture

3.1. Vision

BigDataStack provides a complete data-driven infrastructure, (see Figure 1**Error! Reference source not found.**. The envisioned BigDataStack platform capabilities could not be realized without a full stack that can facilitate the requirements of Big Data operations and applications.

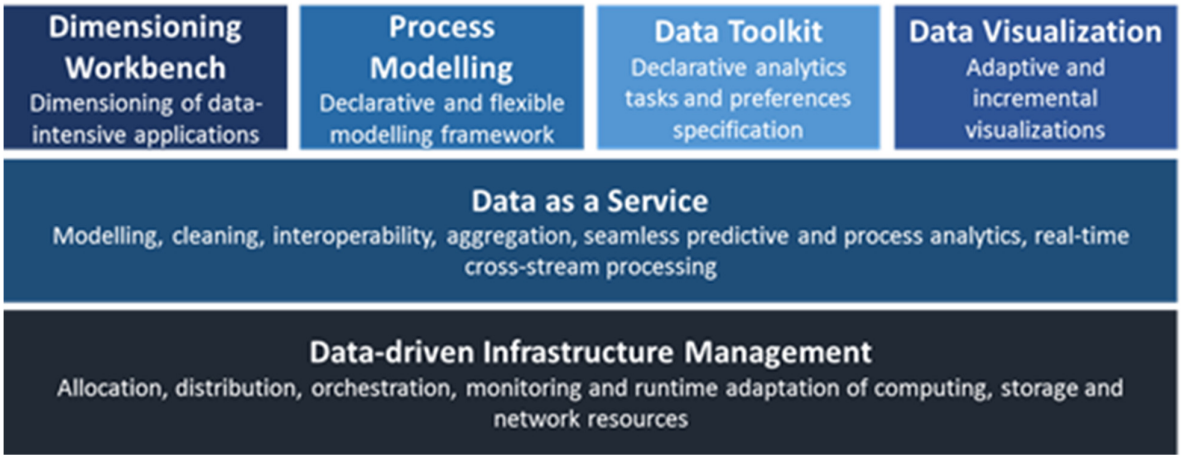


Figure 1 - BigDataStack core platform capabilities (extracted from D2.1)

The "Data as a Service" layer is the cornerstone on which the four upper platform capabilities of BigDataStack rely. This layer offers a set of services that provide the building blocks of an efficient and modern data infrastructure covering all the major phases of data life cycle and usage, i.e., data ingestion, data storage and data analytics.

Let us now present the main components / mechanisms of the Data as a Service layer along with the respective aforementioned basic capabilities:

Data Quality Assessment & Improvement is an essential part to data ingestion as it offers domain-agnostic data cleaning and enhancement services and guarantees data veracity by providing a data-source health review.

Big Data Layout and Data Skipping yields novel capabilities for SQL analytics on object storage. It will provide a pluggable data skipping engine with the ability to handle complex queries as well as User Defined Functions (UDF). These enhanced data skipping capabilities will be complemented by an online and offline data layout engine which takes into account data properties and query workload in order to enable efficient querying of data stored on object stores by maximizing the data skipping.

Distributed Storage is the first data storage approach of BigDataStack, it is based on the LeanXcale internal key value storage layer. This component will enhance the capabilities of this layer with:

- data fragmentation of the stored datasets;
- dynamic reconfiguration by splitting, merging and moving the data regions across the distributed data nodes in order to balance themselves against diverse loads (both in terms of incoming work and data load);
- the elastic re-deployment of the storage that will be able to horizontally scale in/out to adapt to lack/surplus of resources.

The second approach of BigDataStack is not Object Storage per se but rather new techniques to overcome the "data ingestion" problem that is typical of Object Storage[1]: BigDataStack presents advances both in data layout (or data partitioning) and data skipping. These advances have the potential to greatly reduce the data ingestion problem.

The Seamless Data Analytics Framework provides a novel storage component as it enables storing and querying data that lie both on transactional database (LeanXcale) and on the object store as a single logical data set.

Predictive & Process Analytics: the two main scenarios of the Predictive & Process Analytics component are the discovery of insights and the prediction of future events in the context of process flows derived from event driven data. Process mining techniques will be utilized to extract knowledge from event logs which in turn will be transformed into insights and recommendations for the user in the Process Modelling phase of the project.

Before we describe the CEP component,

Real-time Complex Event Processing (CEP) is another essential part for data ingestion as it permits to process data being ingested to yield essential information (e.g., triggering of alarms, push notifications or alerts to end-users, etc). This component gives the ability to process information on the fly before being stored. The CEP will run on distributed setups over the Wide Area Network (WAN), distributing queries so that communication overhead is minimized and also enabling early generation of alarms. The CEP will aggregate data as they are produced speeding up the analytical processing.

3.2. Platform Roles

The following table lists the BigDataStack roles (as described in deliverable D2.1) that are relevant to the Data as a Service block.

1 https://www.researchgate.net/publication/317066213_Too_Big_to_Eat_Boosting_Analytics_Data_Ingestion_from_Object_Stores_with_Scoop

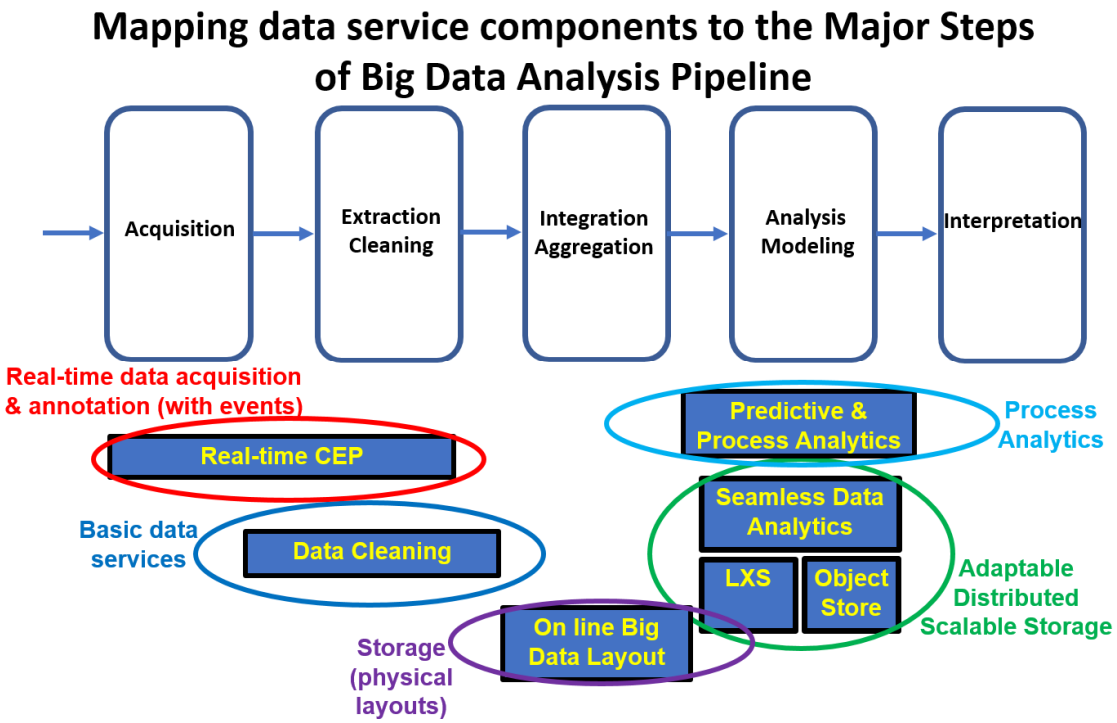
Id	Name	Description
ROL-01	Data Owner	BigDataStack offers a unified Gateway to move (streaming) data from data owners into BigDataStack data stores layer, which support both SQL and NoSQL data stores.
ROL-02	Data Scientist	Data as a Service offers to the data scientists: a) data cleaning services b) Complex Event Processing, which can be applied on the data streaming in, both before and after it passes though the unified Gateway. c) the possibility to store the data as a single logical data set on both transactional data bases and object store. These data sets can seamlessly be queried.

Table 2 – BigDataStack Platform roles

3.3. Design

The set of data capabilities offered by the Data as a Service block are in fact mostly independent. As it will be presented in the following section 4, they may be naturally used together as required from several scenarios of data usage. However, the design of each component is quite component-centric and independent of other components / mechanisms. A first exception is for the seamless analytics framework, which takes the LeanXcale database and the object store and produces a new entity built upon these two first ones, permitting a) to define rules for automatic balancing of data sets between the two basic data storage components (e.g., data older than 3 months should be moved to the object store), b) to define and use a dataset which may be spread over these two data storage components seamlessly.
A second exception is the synergy between the data cleaning component (see section 8) and the Real-time Complex Event Processing (CEP) (see section 10): indeed, since data cleaning on-line processing is stateless, CEP happens to be a nice support for data cleaning which can benefit of all the CEP advantages.

Typical Big Data processing starts from data acquisition at the edge and then goes through a pipeline as described in



where the extraction / cleaning may be performed at the edge or / and near the data store. Data as a Service offers data services that map to all these phases, but the last one (interpretation).

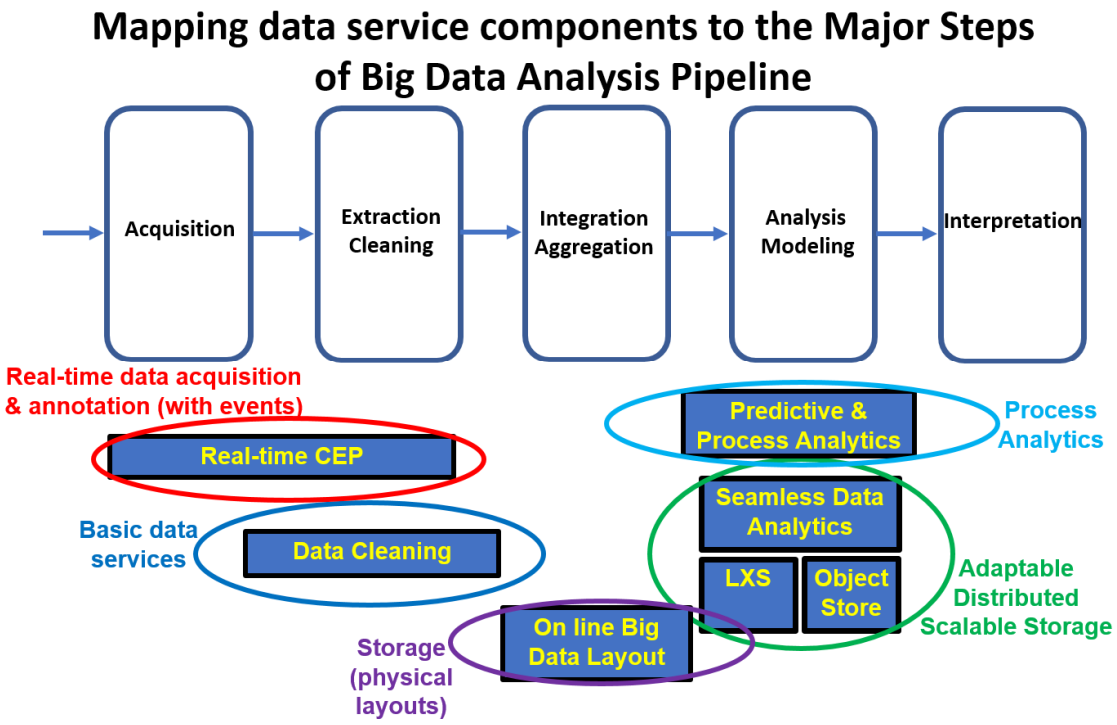


Figure 2 – Data as service components mapping to Big Data pipeline

The CEP (section 10) distributes and parallelizes the queries to run in distributed setups. The operators of queries can be User Defined Functions (UDF), such as the data cleaning. In this sense the CEP can be used as an infrastructure for parallelizing UDFs.

4. Implementation and Experimentation

4.1. Experimental Setting

This section introduces the use cases and scenarios to be supported in the incremental development of the solution.
Data as a Service is designed to fit a broad scale of Big Data analytics use cases with demanding requirements. As a first step, the aim is to develop and test the various components of the overall Data as a Service block for the ship management use case (see deliverable D2.1 section 4.1). Indeed, the two main scenarios that are built around this use case will exercise most (if not all) of the capabilities of the Data as a Service components / mechanisms.

As few general notes valid for this section and in fact this deliverable:

1. Our design and solution are agnostic of what exact object store we are using. It could be the IBM COS object store[2], as well as OpenStack Swift[3], or minio[4], or Ceph[5], etc. Therefore, in the following text we use with interchangeability "object store" or "IBM COS". The sole component which will demand a specific implementation of Object Storage is the seamless component (see section 7) which will contain the novel COS acceleration cache. This component (the acceleration cache) is specific to COS.
2. In term of distributed compute framework, we chose Apache Spark as distributed SQL engine for Big Data because of the following reasons:
    1. Among SQL engines available for Big Data (such as Presto), Apache Spark has biggest momentum (1400 contributors, Spark SQL alone has 450)
    2. Apache Spark has a very large user base
    3. ANSI SQL 2003 support: Spark SQL has the best ANSI SQL 2003 standard support among all Big Data SQL engines
    4. Apache Spark SQL supports complex and long running queries better than competition
    5. Apache Spark is best when it comes to extensibility and modularity
    6. Apache Spark beats competition in term of SQL query performance[6]

We now will present the two use case scenarios that will be dealt during the first half of the project: first of all, we'll discuss the ship management use case data ingestion path. Figure 3 – Data ingestion pathshows all the components that are involved in the ingestion path, which we will detail from the IoT data creation up to its upload in the object store. In the following description "[x]" will refer to the component with "x" label.

So, the data consists of many Internet of Things (IoT) records (approximately 125 different attributes per minute, excluding meteorological and oceanographic data) that are collected from several on-board components and sensors, as follows:

- The navigational system (latitude, longitude, wind speed and angle, speed over ground and speed through water);
- The Alarm Monitoring System (main engine related attributes such as the rotations per minute and the torque of the main shaft, fuel oil inlet temperature and pressure, exhaust gas out temperature, etc.);
- Key-Machinery components (fuel oil volume and temperature, etc).

[2] https://www.ibm.com/cloud/object-storage
[3] https://wiki.openstack.org/wiki/Swift
[4] https://www.minio.io/
[5] https://ceph.com/
[6] https://cdn2.hubspot.net/hubfs/488249/Asset%20PDFs/Benchmark_BI-on-Hadoop_Performance_Q4_2016.pdf

We refer to the use case description (see deliverable D2.1 section 4.1) for details about the ship management use case. These many IoT records are gathered by what we call the "ship management on board application" [2] which outputs every minute a new row comprising the updated value for each of the sensors.

These IoT data is fed twice to CEP components: once on ship (to CEP1 [3]) and a second time after passing the gateway (to CEP2 [5]). These two CEP components fulfil different goals.

The IoT data rows are input to the CEP1 on ship component, which has for main goal to detect malfunctioning equipment, such as damaged sensors or recording equipment. This detection is done thanks to a set of rules which, if they apply to the data prove that the IoT data is erroneous (e.g., the speed through water is zero but the rotations per minute of the main shaft is not). Upon CEP1 processing completion, the IoT data is sent to the Gateway [4]. If data inconsistency is detected by CEP1, an alarm record is created and also sent to the Gateway [4].

The gateway [4] is the external accessible point of the BigDataStack analytics system and it receives both IoT data which it forwards to CEP2 [5] and the alarms which it forwards to the Alarms component [7]. The Alarms component gathers the alarms generated by CEP1 [3] and CEP2 [5]. The initial plan is to model it through OpenShift route which enables, under the same domain (i.e., same public IP and same URL, just different path), layer 7 load balancing of incoming requests. Therefore, making easy to differentiate the queries/request having to be redirected to CEP2 or to the Alarms component. If extra requirements arise, the plan is to move to the usage of service mesh (Istio[7] in this case) and make use of the sidecar containers to handle the extra required actions and the Istio-gateway to redirect the traffic through the entry point, i.e., the Gateway. For more information about the gateway, check section 5 at Deliverable D3.1.

CEP2 [5] has much more processing resources than CEP1 [3] so that it will process the incoming IoT data in ways that do not fit CEP1. First of all, the basic data cleaning is running within CEP2. After data cleaning the IoT data undergoes additional checks and possible alarms are forwarded to the Alarms component [7].

IoT data output from CEP2 [5] is now ingested by the LeanXcale data base [8] which is a relational database that can sustain intensive operational workloads. Data is first stored into LeanXcale, which ensures consistency in terms of transactional semantics and can be scaled out accordingly to handle massive ingestions.

The seamless section 7 details how historical data slices are moved from the LeanXcale database to the object store. The data slices are pushed from the LeanXcale data base to a Kafka cluster [9], from which they are pulled by an Apache Spark[1][12] application which might be based on Kafka Connect [10] (or alternatively Spark Streaming) which will create objects out of the data slices and upload them to the object store [13].

[7] *https://istio.io/*

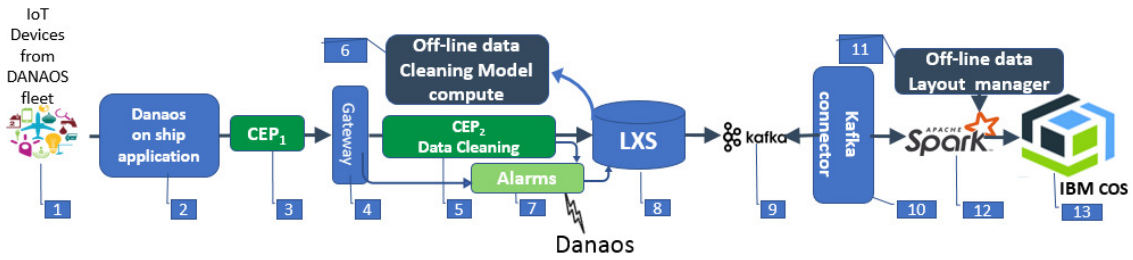


Figure 3 – Data ingestion path

The data query path (depicted in Figure 4 – Data Query Path) is based on the architecture detailed in the Seamless section 7.2 where the object store is on premise. This is the query path that we plan to implement in the first iteration / phase of the project (planned for M18).

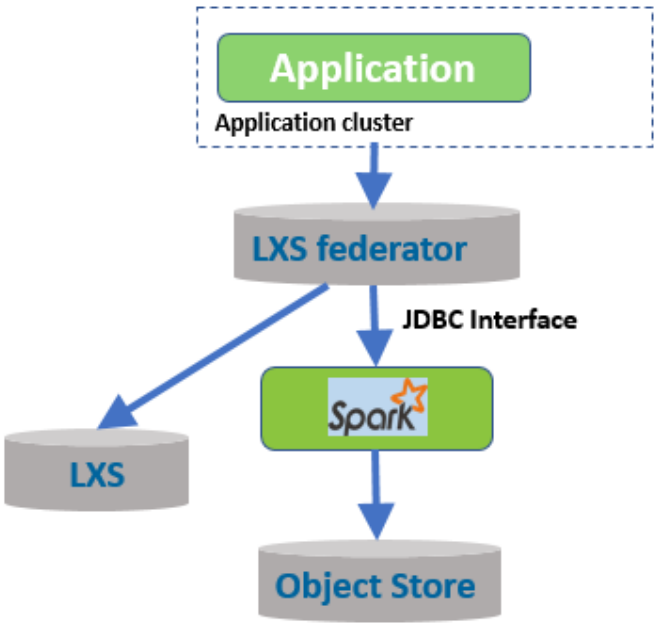


Figure 4 – Data Query Path

The seamless analytics component permits to access a data set that may be stored on both the LeanXcale data base and on object store as a single logical data set, that is without having to bother with location of data.

The entry point of the seamless component is the LXS federator which federates the two underlying data stores. The component will be based on LeanXcale internal Query Engine that has polyglot capabilities and can join data coming from different sources, exploiting the concept of data lakes. We further refer the reader to sub-section 7.2 for details on how the seamless component design and possible usage.

As depicted in Figure 5 - Process and Predictive Analytics, the entire analytics flow takes advantage of the seamless analytics component illustrated above to gather information from both the LeanXcale distributed data base and IBM's object store. The event data is processed through the Predictive and Process Analytics component (section 9) and information concerning the relationship among events is forwarded to other components, more precisely, the Process Modelling Framework part of WP5.

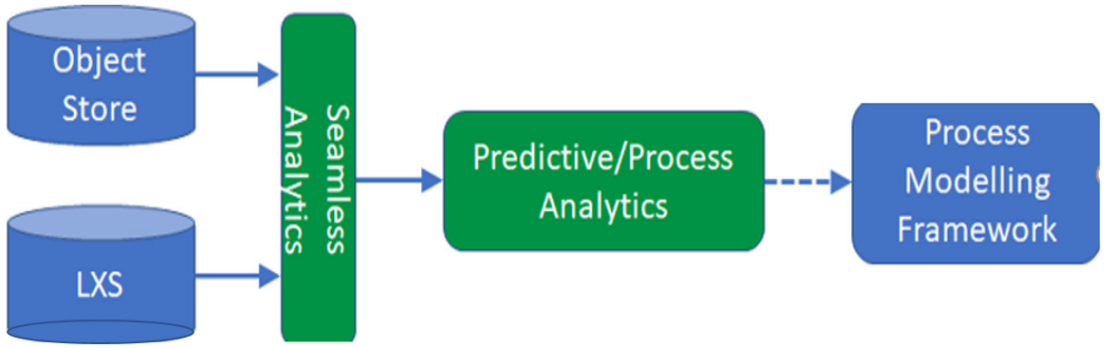


Figure 5 - Process and Predictive Analytics

4.2. Implementation Roadmap

At the current stage of the project, initial prototypes have already been developed for several components. The following table section gives a roadmap for the various Data as a Service components that will need to be integrated for the mid-project integrated prototype.

Table 3 – Implementation Roadmap

	M12	M14	M16	M18
On ship data simulator	First version of real time simulator of ship management data	Tentatively final version of simulator		
CEP1	First version	Integrated with data simulator includes alarm generation	Full integration with gateway	Fully working
Gateway		First integration with simulated data	Gateway dispatches stream (IoT) data	Gateway dispatches stream (IoT)

				data and alarms
Deployment work	First integration with data simulator	Updated deployment of OpenShift cluster (improved performance)	Fully working	
CEP2		First version not including cleaning	CEP2 includes alarms and CEP2 -> LeanXcale DB complete,	Fully working
Data Cleaning	First standalone version	Main path working	Fully working and integrated	
LeanXcale	Static deployment of LeanXcale data base and integration with Monitoring subcomponent	Deployment of LeanXcale data base using Kubernetes that will allow to automatically deploy the datastore from the WP3 tools. LeanXcale data base Monitoring Proxy will be released	*Adaptable Storage Driver* sub-component will be released to enable run-time reconfiguration	
LeanXcale database to Kafka	Definition of data schema for the ship management use case. Schema extension to include metadata needed for data movement from LeanXcale data base to IBM OS	Definition of input expected from the IBM OS, LeanXcale data base will be able to send data to Kafka in the agreed format	LeanXcale data base will send only historical data to IBM object store with respect to data consistency, and will not send modified operational data	
Kafka to object store			main path for data move from LeanXcale data base to object store (no	Fully working

			transactional behaviour)	
Spark & skip technology		Installed and connected to OS	Offline data partitioning working	Fully working
Data Skipping and Advanced data Layout	Definition of geospatial data skipping index.	Mapping of geospatial UDF functions to geospatial data skipping index.	Geospatial data layout of ship management data in object storage.	Demonstration of data skipping and layout for geospatial queries on ship management data in object storage.
Seamless federator	Seamless federator will provide access only to LeanXcale data base (via the JBBC). No federation will IBM OS will take place	Seamless federator integrated with Spark thru Thrift - Query main path thru federator is working. Federator do updates on LeanXcale data base, can retrieve data either from LeanXcale data base or IBM OS, but not from both	First prototype of Seamless federator integrated and working against LeanXcale data base and Spark / OS. Experimentation on the performance and resource consumption	Seamless federator fully integrated and working against LeanXcale data base and Spark/OS
Test of JDBC/Spark if	Spark Thrift server tested			Fully working
Call-back OS -> LeanXcale data base		Definition of the protocol to be used for notifying the LeanXcale data base upon a successful ingestion of a data slice	Integrated	Fully working
Predictive and Process Analytics	First working stand–alone version	Connection to Process Modelling Framework with generically made event log (First Recommendations)	Connection with Global Event Tracker – Integrated	

5. Big Data Layout and Data Skipping

5.1. Requirements Specification

We refer the reader to deliverable D2.1 section 8.10 (Big Data Layout) for: a) a description of the problem tackled by this module and b) a comprehensive review of the best practices.

In order for this section to be self-contained, we reproduce here the basics of "data skipping" and "data layout":

Data Skipping is a technique to minimize the data that has to be read from object store to Compute Store. This technique utilizes metadata to track information about objects and their dataset columns which can then be used for data skipping i.e. to show that an object is not relevant to a query and therefore does not need to be accessed from storage or sent on the network from Object Storage to Spark. To make the Data Skipping technology efficient, we index the metadata, so that during query execution, objects that are irrelevant to the query can be quickly filtered out from the list of objects to be retrieved for the query processing. This technique applies to all data formats and avoids touching irrelevant objects altogether (see IBM presentation[8] at the Spark Summit).

To get good Data Skipping one typically needs to pay attention to Data Layout. Data layout refers to all details regarding the storage of the data including object size, format, Hive style partitioning, and data partitioning, i.e. the assignment of data records to objects. We focus now on data partitioning. For any given query, we would like the records which satisfy the query to be grouped together in a small set of objects, so that the remaining objects can be skipped. In general, we need to partition the data so that it gives as much as possible data skipping for an incoming stream of queries (i.e. a workload), not just a single query. Note that the various queries may have conflicting requirements. Moreover, the workload changes over time, as does the data. In 2017, IBM Research independently developed the notion of *k*-d tree partitioning which uses query history to choose the partitioning columns and dataset medians as cutting points for partitioning. In parallel, a paper was published by an MIT team which used a similar approach and similarly applied it to Apache Spark for data skipping[9]. The work in this paper went beyond previous work by providing an adaptive approach to repartition datasets on the fly according to a cost model. This is a cutting-edge research area which is also promising in terms of its applicability to analytics on real world big datasets.

This section undertakes further research in this area as well as apply it to a commercial setting. The following tables present the requirements that Big Data Layout and Data Skipping module should satisfy. Please refer to the footnotes for further clarifications.

[8] https://databricks.com/session/using-pluggable-apache-spark-sql-filters-to-help-gridpocket-users-keep-up-with-the-jones-and-save-the-planet

[9] A. Shanbhag, A. Jindal, S. Madden, J. Quiane, and A. J. Elmore, "A robust partitioning scheme for ad-hoc query workloads," SoCC, 2017.

Table 4 - requirement REQ-BDL-01 for Big Data Layout

	Id[10]	Level of detail[11]	Type[12]	Actor[13]	Priority[14]
	REQ-BDL-01	Software	FUNC	Developer	MAN
Name	Support data skipping for arbitrary query predicates				
Description	The query predicate could comprise UDFs and AND/OR/NOT. Example UDFs could be geospatial or temporal functions.				
Additional Information	This functionality is important for the ship management use case, which requires geospatial UDFs.				

Table 5 - requirement REQ-BDL-02 for Big Data Layout

	Id	Level of detail	Type	Actor	Priority
	REQ-BDL-02	Software	FUNC	Developer	MAN
Name	Support a truly pluggable architecture for data skipping				
Description	The goal of this requirement is to enable the addition of new data skipping index types without changing the core data skipping library. This is needed for requirement REQ-BDL-01 since supporting new UDFs may require new index types.				
Additional Information	External users can also exploit this capability				

Table 6 - requirement REQ-BDL-03 for Big Data Layout

	Id	Level of detail	Type	Actor	Priority
	REQ-BDL-03	Software	FUNC	Developer	MAN
Name	Enable layout change for (part of) a dataset				

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

[11]**Level of detail**: Following the use of ISO/IEC/IEEE 29148:2011 (see section 2.1 Methodology), we use the following levels: Stakeholder, System and Software (i.e., technology details).

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

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

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

Description	There is a strong relationship between how a dataset is laid out in the object store and the performance of data skipping against this data set. Moreover, this performance may be also very dependent on the queries. Hence the need to adapt the layout, not only for future data but also for heavily queried data already in object store.
Additional Information	

Table 7 - requirement REQ-BDL-04 for Big Data Layout

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-BDL-04	Software	FUNC	Developer	MAN
Name	Enable on-line data layout				
Description	Layout is critical for the data skipping performance. As of now data is stored as is and possibly laid out again offline. The need is to upload dataset chunks with the best-known layout as data is ingested.				
Additional Information					

5.2. Design

Our Data Skipping module allows skipping over objects which are not relevant to a query by using metadata which summarizes that data in one or more columns for the object. Currently we store this metadata in Elastic Search. Examples of data skipping metadata include storing minimum and maximum values for numeric types, or bounding boxes for geospatial data. Our code intercepts the SQL query execution flow in the phase where an object listing is created and pruned and adds an additional phase which further filters the list of objects by skipping over objects irrelevant to the query. See figure 6 which shows the regular Spark SQL query flow on the left, and our modified flow on the right.

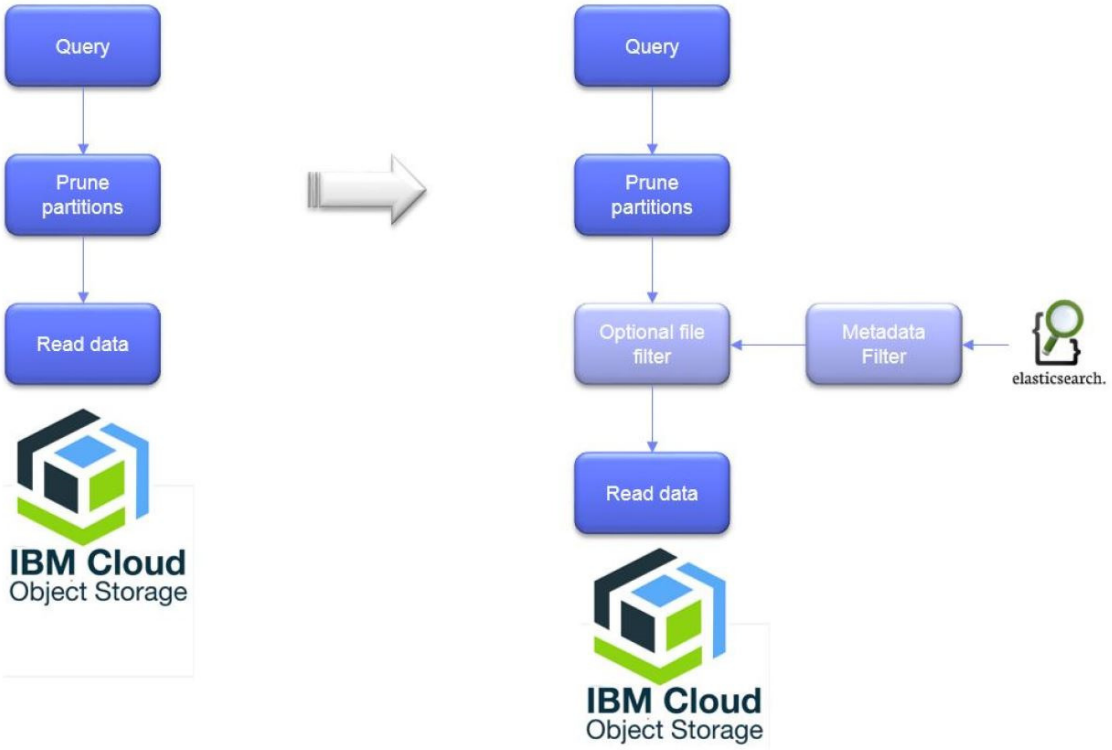


Figure 6 - Spark SQL Query Execution Flow

Our data skipping library code works with Apache Spark SQL but does not require any changes to core Spark. To do this we use the "Extra Optimizations" API of Catalyst, Spark's optimizer. We add a special optimization rule which scans the logical plan tree and replaces the default *InMemoryFileIndex* with our enhanced *IndexedCatalog,* which extends the *InMemoryFileIndex* but further filters objects using Elastic Search as described above.

The Data Skipping module allows creating data skipping indexes on datasets residing in the Object Storage, and subsequently exploiting these indexes to skip over objects irrelevant to a given SQL query. We made novel and significant extensions to the Data Skipping component by:

1. Allowing users to define new data skipping indexes, without changing the core data skipping library;
2. Supporting data skipping for arbitrary query predicates including UDFs and AND/OR/NOT operators.

These two capacities are important and novel for different Big Data applications and operations, such as the ship management use case.

The purpose of the Data Layout Manager component is to organize/partition the rows of a dataset as objects in the Object Storage in such a way that improves analytics performance, by reducing the number of bytes sent from the Object Storage to an analytics framework, such as the Apache Spark (the main KPI targeted by this component). This component is essentially a Spark application which works in two phases:

1. The first phase analyses the dataset and builds a *k*-d-tree

2. The second phase uses partitions the dataset into objects according to the *k*-d-tree

In our current implementation, the user specifies explicit commands to control the data layout, by specifying the columns to use for layout and their relative priorities. In future, we plan to collect query history and data statistics for specific use cases, and based on this, to automatically recommend a data layout and associated choice of data skipping indexes for that use case to enhance the effectiveness of data skipping.

Data Layout can be performed on a dataset, which already exists in the Object Storage – we call this Offline Data Layout. Alternatively, Data Layout can be performed dynamically while data is ingested into Object Storage. We call this Online Data Layout.

There are two main flows in which Data Skipping and Data Layout participate. In the ingestion flow, Online/Offline Data Layout is used to organize the dataset rows into objects, either on the fly or after the fact. Moreover, the Data Skipping module is used to create data skipping indexes which can be used by subsequent analytics. The parameters such as which columns to use for layout and skipping are currently provided by the user. In future, there will be a query logging component which records the query history, as well as a data logging component which records dataset properties and their change over time as new data are ingested. This information can then be used to recommend the above parameters.

Figure 7 – Data Ingestion flowdepicts the various possibilities for the ingestion flow. Note that data is ingested from the LeanXcale data base via Kafka using a connector and uploaded to Object Storage.

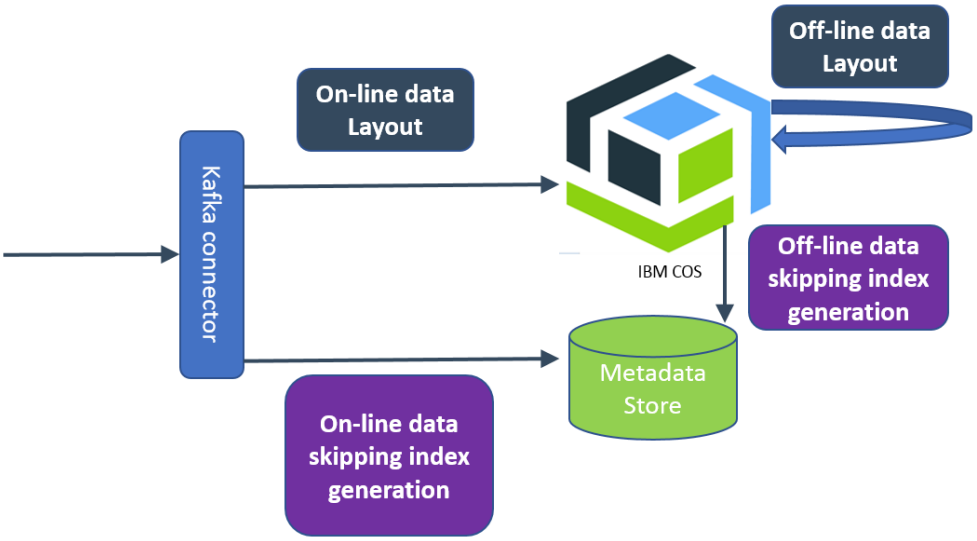


Figure 7 – Data Ingestion flow

In the analytics flow, (see Figure 8 – Data Analytics flow) given a query, the metadata store (in which the generated meta-data is stored) is consulted in order to skip over irrelevant objects whenever possible. Note that here a given SQL query is submitted by the LeanXcale data base and reaches Spark via a JDBC interface.

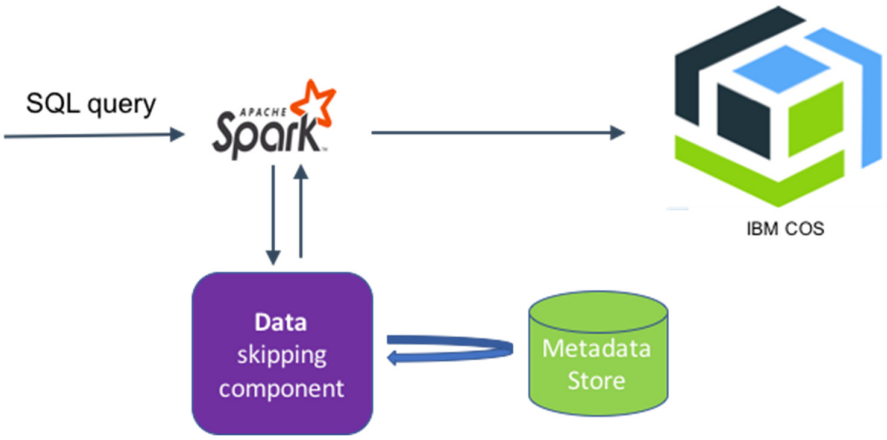


Figure 8 – Data Analytics flow

5.3. Early Prototype

IBM Data Skipping and Data Layout modules have been extended in several significant ways for the benefit of BigDataStack.

Both the Data Skipping and Data Layout code was extended to accurately measure bytes read from COS to Apache Spark. It is important to measure this accurately since it is the main KPI. This is done using the Spark Measure library from CERN[15]. Bytes read from COS is measured when creating data skipping indexes, and also when running queries against COS. Moreover, when running the Data Layout Manager, we measure both the bytes read number and the bytes written to COS number. Work measuring bytes written to COS accurately is currently in progress since it requires a change to the underlying Stocator[16] driver. Note that previously we were only able to measure the total number of objects skipped by a query, and to aggregate the total sum of bytes skipped in those objects. However, these statistics are not helpful when data is stored in formats such as Parquet or ORC, because when accessing these formats Spark typically accesses only parts of the objects, for example, it only reads those columns accessed by the query. Our new method is able to measure the number of bytes actually read. We authored a blog on existing best practices for Big Data Layout for Spark SQL analytics on data in IBM Cloud Object Storage.[17]

The Data Skipping module was extended to support arbitrary query predicates, including UDFs (user defined functions) and AND/OR/NOT. This is particularly challenging, because UDFs can be arbitrary functions about which the Spark optimizer knows very little. This work will be applied to geospatial UDFs, which are important for the ship management use case. See the next section to understand why this is needed. In addition, the Data Skipping code now supports a truly pluggable architecture, where new data skipping index types can be added without changing the core data skipping library. For example, we added a new

[15] https://github.com/LucaCanali/sparkMeasure
[16] https://github.com/CODAIT/stocator
[17] How to Layout Big Data in IBM Cloud Object Storage for Spark SQL https://www.ibm.com/blogs/bluemix/2018/06/big-data-layout/

geospatial data skipping index type using this capability, which works well in conjunction with geospatial UDFs. External users can also exploit this capability. More technical details describing this work will appear in future.

The Data Layout Manager has undergone a complete refactoring. It has also been updated to handle string data types as well as numeric data types. String data types behave differently from numeric data types since in most cases they are queried using '=' or 'IN' predicates and not using inequality predicates. Therefore, our *k*-d-tree approach which splits the data according to the median value may not be ideal in this case. The *k*-d-tree approach was extended to support arbitrary cut nodes with unlimited number of children which enables each cut node to implement its own logic to decide how to split the data. We are experimenting with various alternatives of cut nodes, some of which take into account the number of distinct values in a column, and the number of appearances of each value. A key challenge is to do this efficiently. We also support sampling the input dataset when running the Data Layout Manager, which is important to achieve good performance.

5.4. Use Case Mapping

The ship management use case requires efficiently querying vessel (ship) trajectories and vessel engine data. The ship management use case also requires joining this data with weather data, to enable queries which combine both data sources. As described in section 7, this data will be stored across both the LeanXcale data base and Object Storage, where the bulk of the data stored long term will be in Object Storage. Therefore, an efficient method of organizing the data in Object Storage and querying it is required.

Many of the queries required by the ship management use case are geospatial in nature. For example, the data scientists analysing the ship datasets might need to check whether there is a correlation between engine failures and a certain geospatial region. Moreover, joining the vessel data with weather data is geospatial in nature, because one needs to combine information about a ship with weather data from a similar location (and time).

In order to query geospatial data effectively, geospatial library functions are needed, which take into account the curvature of the Earth as well as handling various coordinate representations. Such libraries are often not built in to a Big Data engine, but can be added dynamically, as is the case for Spark SQL. Once such a library is added, its functions can be registered as UDFs (user defined functions), and queries can refer to them.

Optimizers such as Spark's Catalyst optimizer are typically unable to make intelligent decisions regarding UDFs since they know nothing about them. Specifically, for queries involving Big Data on Object Storage, essential optimizations are to choose an effective layout of the data and to enable skipping as much data as possible when it is not relevant to a query. This is critical, but challenging, when UDFs are involved. For example, consider the following query, where the ST_distance function computes the spatial distance between two points:

```
// Registration of Spatial Functions, "spark" is the sparkSession:
SqlGeometry.registerAll(spark)
// We assume df, is the original dataframe
// and that it does *not * contain a column named "location"
val locationDf = df.withColumn("location",toPointEG($"lat",$"long"))

val  targetLat =  someValue
val targetLng = someValue
val  targetRadius =  someValue

// We create a temp view named VESSEL_DATA
locationDf.createOrReplaceTempView("VESSEL_DATA")

val query = "select vessel_id
FROM VESSEL_DATA
where
  ST_Distance(location,ST_WKTToSQL('POINT(targetLat , targetLng )'))
  <
  targetRadius"

// and run the query:
spark.sql(query)
```

Data layout and skipping are essential here to reduce the amount of data read from the Object Storage and shipped across the network to the area surrounding Barcelona. However, without any knowledge of the ST_distance UDF in the Spark Catalyst optimizer this is not possible.

In order to cater for the ship management use case, we will exploit the data layout and data skipping modules to handle cases where arbitrary query predicates including UDFs are involved. Moreover, we handle the case where data is multi-dimensional and data layout and skipping needs to take multiple dimensions (geospatial, time, and additional dimensions) into account.

Note that UDFs can be useful for many use cases and are not specific to geospatial scenarios. Moreover, the requirement to handle multi-dimensional data applies to most use cases.

5.5. Experimental Plan

The main KPI which indicates successful data layout and data skipping is the number of bytes sent across the network from the Object Storage cluster to the Analytics cluster (in our case Apache Spark). A successful application of Data Skipping and Data Layout techniques will significantly reduce this KPI.

An additional KPI which is also of some interest is the number of REST requests issued from Apache Spark to Object Storage. Each REST request incurs some overhead, so a smaller number is better.

In order to characterize the ship management workload, the following information is considered:

1. dataset schema
2. historical dataset
3. Live updates to the dataset while the project is underway
4. any existing data layout choices
5. historical SQL query workload
6. live SQL query workload – queries issued while the project is underway

The above can be used to provide a benchmark for the ship management use case. For this benchmark we can run the given SQL query workload against the supplied dataset, using any chosen data layout and data skipping techniques, or without them, and measure/compare the above KPIs.

5.6. Next Steps

Next steps include applying our data skipping and layout technology to the ship management use case. This requires some focus on geospatial analytics and the use of external geospatial libraries.

- Workload/benchmark:
  - Ingest the ship management dataset and the weather dataset to the Object Storage
  - Collect ship management SQL queries as benchmark
- Identify a suitable library with geospatial functions for use with Spark
- Apply data skipping for arbitrary query predicates to the ship management use case:
  - Handle AND/OR/NOT
  - Handle UDFs
- Enable the Data Layout Manager to employ UDF functions
- Apply the DataLayoutManager to the datasets above, and generate Data Skipping metadata
- Complete work on accurately measuring bytes written to COS using the Data Layout Manager
- Apply the workload/benchmark and compare with/without Data Layout and Data Skipping.

6. Adaptable Distributed Storage

6.1. Requirements Specification

The adaptable distributed storage is a fully distributed storage layer relying on the data nodes of the LeanXcale relational datastore, and its main purpose is to be able to dynamically reconfigure both its resources and its data fragments in order to serve concurrently diverse workloads. For this goal, this component should be able to split the existing datasets in

different fragments, move them across the data nodes in order to reduce the resource consumption in nodes that are over-consuming the available resources and under-performing, and request additional resources from the infrastructure, in order to scale out appropriately. As it has strong dependencies with the components of WP3, several requirements have been identified that both concern its internal functionality, as well as the interactions with these aforementioned components.

The following tables present the initial list of those requirements, as identified during the first iteration of this deliverable, and they are categorized both as mandatory for the delivery of the prototype, while others can be considered optional at this phase of the project.

Table 8 – requirement REQ-ADS-01 for Adaptable Distributed Storage

	Id	Level of detail	Type	Actor	Priority
	REQ-ADS-01	System	DATA	Developer	MAN
Name	Being able to fragment a dataset and move the data fragments across different nodes.				
Description	The adaptable distributed storage should be able to split a dataset into different regions, and move these regions to different data nodes, in order to adapt in case of increased load (both in terms of user workload or data load) so as to achieve efficient consumption, based on the provided resources.				
Additional Information	When a movement (move, split, join) of a data fragment occurs, the storage must not suffer from a down-time. On the contrary, it must remain operational with minimum overhead on the overall performance.				

Table 9 - requirement REQ-ADS-02 for Adaptable Distributed Storage

	Id	Level of detail	Type	Actor	Priority
	REQ-ADS-02	System	ENV	Developer	MAN
Name	Identify data nodes that are overprovisioning.				
Description	The adaptable storage must be able to identify data nodes that are overprovisioning their available resources and send internal alerts to trigger a dynamic reconfiguration of the deployment of the data fragments.				
Additional Information					

Table 10 - requirement REQ-ADS-03 for Adaptable Distributed Storage

	Id	Level of detail	Type	Actor	Priority
	REQ-ADS-03	System	FUNC	Developer	DES

Name	Solve the non-linear resource allocation problem to suggest alternative deployment of the data fragments.
Description	According to the available resources for the deployment of the data nodes and the stored data set, along with its split points that define data fragments, there is a non-linear resource allocation problem for the optimal deployment of the data fragments.
Additional Information	As a non-linear, the solution of the resource allocation problem requires exponential time to be solved, which is not acceptable for run-time requirements. The provided solution should take into account possible acceptable solutions that can solve the problem and improve the resource consumption, under a minimum time interval.

Table 11 - requirement REQ-ADS-04 for Adaptable Distributed Storage

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADS-04	System	ENV	Developer	DES
Name	Be able to request additional resources from the infrastructure layer.				
Description	In case of overprovisioning of the resources, the adaptable distributed storage should be able to request additional resources from the infrastructure of BigDataStack.				
Additional Information	As noted in REQ-ADS-02, the adaptable storage must identify data nodes that are overprovisioning, and using REQ-ADS-03, it can suggest different distribution of the data fragments. However, there might be cases that this is not possible due to the overprovision of the whole system, and in such case, a horizontal scale out must take place. The adaptable storage should request additional resources, and grant them, if they are available. The communication should be as follows: 1. The adaptable storage requests an additional node with the specific requirements for resources. 2. The infrastructure responds if it can allocate additional resources for the storage. 3. The infrastructure informs the storage that the additional resources are now available. This requirement also includes the need from the adaptable storage to inform the infrastructure that it can release resources that are not needed.				

Table 12 - requirement REQ-ADS-05 for Adaptable Distributed Storage

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADS-05	System	ENV	Developer	OPT
Name	Being able to release resources and adapt if resources are deallocated from the infrastructure.				

Description	There might be cases where the whole infrastructure is overprovisioning there are no more resources to be allocated to tasks. Then, the infrastructure might decide to reduce the overall resources of specific components, in favour of others that might execute some critical operations, or they have biggest priority at that point. The adaptable storage engine should be listening to the infrastructure for such cases and adapt accordingly.
Additional Information	Once the adaptable distributed storage receives a request to release some of its nodes, then it should inform if it is capable of doing so: releasing some the data nodes, might result to not have the required amount of storage available for the dataset. In such cases, it should be responding that this is not permitted, as this would lead to data loss. In case that this is permitted, then it should re-distribute its data load, and inform the infrastructure that the node is ready to be released.

Table 13 - requirement REQ-ADS-06 for Adaptable Distributed Storage

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADS-06	System	ENV	Developer	DES
Name	Inform the re-deployment component regarding reconfigurations of the data fragments.				
Description	As it is up to the storage itself to decide its optimal configuration of its data load, the re-deployment component cannot be aware of possible reconfigurations, that might affect the overall deployment of an application. Therefore, the storage should inform the re-deployment component about these actions.				
Additional Information	A message should be sent just before the re-configuration takes place, along with the setup, so that the re-deployment component can be notified and not take into account possible outliner monitoring information coming from this subcomponent. During this time, the re-deployment component should not modify any deployments that rely on the data set that is being re-configured. When the reconfiguration is finished, the adaptable storage should notify the redeployment component again, in order for the latter to start looking on the new monitoring information and decide upon possible redeployment of existed applications as well.				

Table 14 - requirement REQ-ADS-07 for Adaptable Distributed Storage

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADS-07	System	ENV	Developer	MAN

Name	Re-establish connectivity with the monitoring subcomponent when a horizontal scaling action takes place
Description	The adaptable storage engine exports its monitoring data to a specific place where the Prometheus, part of the monitoring subcomponent of BigDataStack can periodically pull and gather this information. Prometheus can be configured on where to pull this information upon its initialization. However, in cases of a runtime redeployment that takes place after a horizontal scaling action, information regarding the newly deployed nodes should also reach the monitoring component.
Additional Information	There should be a monitoring proxy of the adaptable storage that will take the responsibility to send monitoring information to the target component. This proxy should encapsulate the details of the underlying deployment. It should gather all information of the data nodes, reconfigure itself to take into account newly deployed data nodes, and send everything to the Prometheus.

Table 15 - requirement REQ-ADS-08 for Adaptable Distributed Storage

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADS-08	System	ENV	Developer	MAN
Name	Enable a deployment of the data node component using Kubernetes				
Description	As the infrastructure of BigDataStack uses Kubernetes for deploying the various application/platform components, the adaptable distributed engine must be able to deploy and configure additional data nodes via this technology.				
Additional Information					

6.2. Design

Figure 9 - Adaptable Distributed Storage architectural designdepicts the main architectural pillars of the adaptable distributable storage, along with the main components of the project that interacts. The adaptable distributable storage consists of the adaptable storage driver, the reconfiguration engine and the elastic manager. It mainly interacts with the components of the infrastructure management system, from which it relies to deploy the data nodes of the storage, requests the allocation of additional resources, or gets informed to force the release of already available resources, pulls the monitoring information of the storage and informs the re-deployment component regarding new configurations. Moreover, the figure depicts some internal build-in components of the LeanXcale relational data store that are involved in the functionalities of the adaptable storage. Finally, it is important to note that LeanXcale relies on its own distributed key value store (KiVi) which is used to persistently

store data. KiVi consists of a metadata node, which contains all meta-information that describes the operational status of the storage, and various instances of data nodes. The key value store offers its own API for data access, allowing not only simple get/put operations, but also complex SQL-alike queries and aggregation operations. The metadata node of KiVi is part of the configuration component of LeanXcale, while on the other hand, each KiVi data node co-exists with an instance of the Query Engine, in order for the latter to exploit the locality of the data in each node. As a result, the LeanXcale datanodes consist of both a KiVi data instance, which contains a fragment of the dataset, along with a Query Engine instance. The adaptable storage will manage the scalability of the these data nodes.

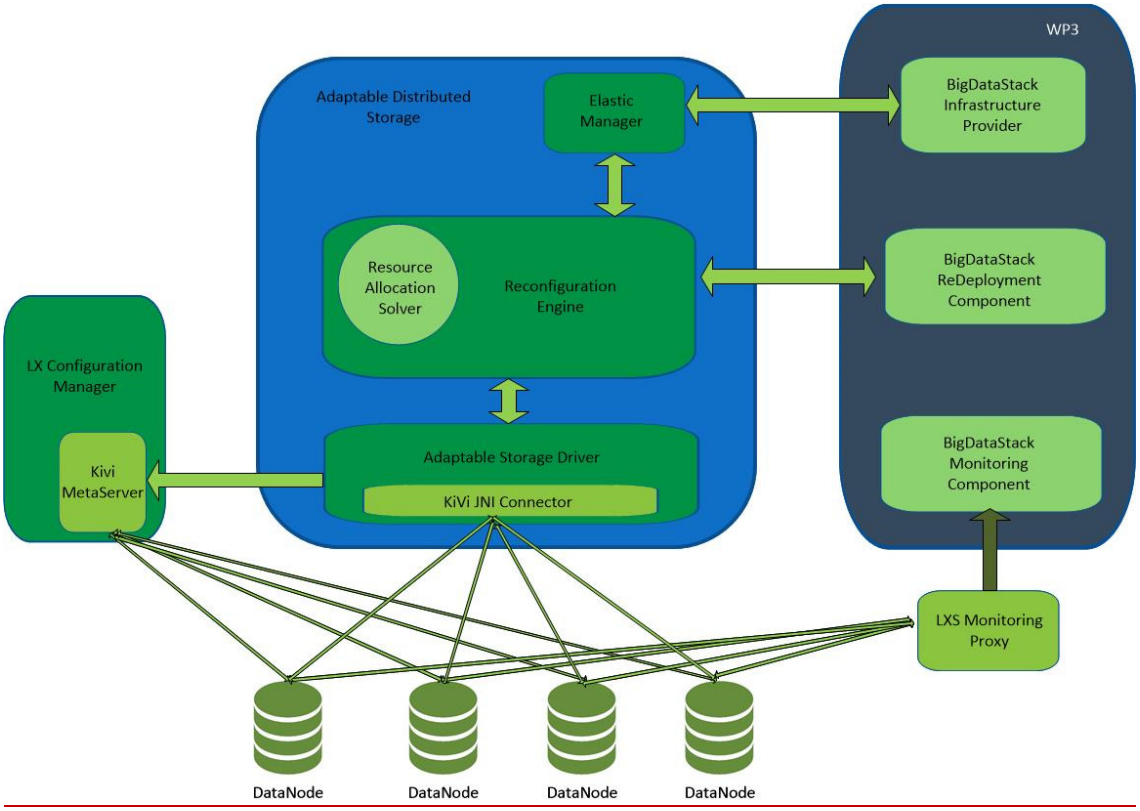


Figure 9 - Adaptable Distributed Storage architectural design

The main purpose of the adaptable distributed storage is to efficiently manage the cluster of the underlying data nodes. The information regarding the nodes configuration is managed by the LeanXcale internal Configuration manager, which contains the metadata server of its internal storage engine. The tools that provide the basic pillars of the adaptable storage, which are the data fragmentation of the datasets, their split to different fragments, their join and movement across different data nodes, are included in the Adaptable Storage Driver. The latter provides utility methods to the upper layers that implement the business logic of this component, thus dynamically reconfigure the deployment of the data fragment, by splitting/merging/moving the existed datasets among the data nodes of the LeanXcale during the runtime and provide elasticity capabilities by scaling in/out the available resources of the storage when necessary. Its implementation is written in Java, and therefore, internally makes

use of JNI[18] calls to allow this subcomponent to call the required bindings which are written in C and handle internally the details of the data movement.

The reconfiguration engine implements the business logic regarding the steps and decisions that should be taken in order to maintain the efficient consumption of the available resources. It can understand hot spots, meaning data nodes that are overprovisioning their available resources. When it identifies such situations, and taken into account the overall available resources, it will ask its internal Resource Allocation Solver which is responsible for solving the corresponding non-linear problem. The latter suggests different configurations of the data fragments, and if this is possible, the Reconfiguration Engine executes all necessary steps.

In case that the whole system is overprovisioning, and no alternative configurations can balance the system according to the available resources, the Reconfiguration Engine will request additional resources for the system from the Elastic Manager. The latter will ask from the tools of WP3, additional resources, and will reply back when these resources are available (or if the request was rejected), while on the other hand, will deploy the data node software elements to the new allocated resources. When this process completes, the Reconfiguration Engine consults again the Resource Allocation Solver in order to obtain the updated configurations, and once a new configuration is received, it drives a process to dynamically redistribute the fragments of the dataset, following the same flow as already described. It is important to note that the Elastic Manager can also receive requests from the infrastructure component of WP3, in order to release resources, dealing with cases that the whole BigDataStack platform is lacking resources and other tasks are more demanding or crucial. The Elastic Manager will reply if the storage can release some of its resources (i.e. the available storage should be bigger than the overall size of the datasets, otherwise it would lead to data loss), and if yes, it will force the Reconfiguration Engine to redistribute the load.

Finally, the LeanXcale monitoring proxy can receive monitoring information from all the underlying data nodes, taken into account scenarios when a re-deployment takes place which leads to a horizontal scalability action. As additional data nodes are being deployed, the monitoring proxy takes into account all the connectivity details with the new nodes and will serve as the central point for pulling data from, by the monitoring subcomponent of WP3. During a dynamic reconfiguration, the Engine will inform the deployment component of WP3 that such a process is taking place to prohibit any reconfiguration till it completes.

6.3. Early Prototype

The goal for M12 is to provide an early prototype deployed in the BigDataStack, so that it can be used by other components, in order to deliver a functional platform at this early phase of the project. Due to this, main focus will be given to enable a deployment of the data nodes of the storage, providing only static deployments at this phase. Moreover, the integration with the monitoring subcomponent will take place, so that it can receive monitoring information

[18] https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html

for the storage to be consumed by the relevant components. The following figure shows the dashboard showing monitoring information, after an initial integration with this component.

Figure 10 - Grafana dashboard with monitoring information

6.4. Use Case Mapping

The ship management use case will be used as the main demonstrator to highlight the advantages and innovations of the tools developed. As described in the corresponding section, the ship fleet produces IoT data from numerous sensors deployed on each of its vessel. This data is pre-processed by different installations of CEP subsystems, cleaned and finally they arrive to the relational data store that has the responsibility to store them to the underlying *Adaptable Distributed Storage* component. The continuous data ingestion (hundreds of IoT data produced per minute by each of the vessels) might impose requirements for a significant amount of storage size, in order for this component to be able to store all the data. Moreover, the size of the required storage will be constantly increased, as new data are being ingested per minute. Due to this, the *Adaptable Distributed Storage* component should be automatically scaled out, by requesting additional data nodes from the infrastructure, when needed during the runtime. The provision of additional data nodes will cause its re-configuration engine to split, move and finally balance the ship management dataset among the available resources. This allows to dynamically increase data loads.

Moreover, as described in the use case, data ingested to LeanXcale relational datastore will be periodically (e.g., after 3 months) transferred to the IBM object store, so that it can be later used for analytical queries over historical data. As mentioned in more details in the following section, IBM Object Store eventually imports this data and informs the LeanXcale data base about the successful ingestion. The LeanXcale data base can then safely discard this data from the storage As a result, the LeanXcale data base will periodically perform a *vacuum* process, which is resource consuming (in terms of memory and computation usage) but which frees storage resource. This might lead to under spending of the available resources, which are not

needed at that time, so the *Adaptable Distributed Storage* can request from the infrastructure to de-allocate the reserved resources.

6.5. Experimental Plan

An initial plan for experimentation taking into account three release iterations during the first phase of the project can be summarized as follows:

1. A data movement operation should not downgrade the performance of the system for more than a specific threshold.
2. The overall time to balance the system (reconfigure the data fragments) after an additional node is allocated should not exceed a maximum time interval.
3. The responsiveness of the Resource Allocation Solver should be tested to validate if it increases linearly as function of the input size.

6.6. Next Steps

After having a deployment of the adaptable storage engine, and its integration with the monitoring subsystem of BigDataStack, the main focus will be given on its deployment using Kubernetes. This will allow for the platform to dynamically deploy this component, and the ability for the latter to be scaled in/out automatically. As mentioned in the corresponding subsection 8 (Triple Monitoring & QoS Evaluation) of D3.1, the current solution can be integrated with monitoring systems like Prometheus providing configuration information regarding static deployments. Therefore, in order to support a scenario when the dynamic load balancing and elastic redeployments take place, the LeanXcale data base monitoring proxy will be released, it will gather all available monitoring information of the data nodes and provide this to the BigDataStack platform. As a result, this subcomponent will act as the main point to pull the metrics so that they can be available to the platform. Finally, the data fragmentation of the datasets will be released by M16-M18, along with all the functionalities that will allow for the reconfiguration of the data fragments. These functionalities that are related with the REQ-ADS-01 requirement are the main pillars of the adaptable storage engine and will be used by the latter in all scenarios regarding reconfiguration and scalability of its resources.

During the second phase of the project, the plan is to allow the dynamic reconfiguration of the data fragments and the redistribution of the data load after recognizing that a data node is overspending its resources. Towards this direction, the necessary mechanisms that will allow the storage engine to identify hot spots using the monitoring information internally will be released. Moreover, a basic solver of the resource allocation problem will be implemented that will allow the engine to decide at run-time for an improved deployment of the data fragments on top of the already allocated resources. This will allow the adaptable storage engine to be able to reconfigure its datasets dynamically, being able to adapt to diverse workloads, both in terms of storage and of resource requirements. Upon reconfiguration, the re-deployment component will be notified when such actions take place. Regarding the elasticity and the ability of the storage to horizontally scale in/out at runtime, the focus will be also given in the second phase of the project. When a horizontal scaling action takes place, the system has to dynamically adapt, and redistribute its datasets to the new provided resources. In order to do so, all necessary tools coming from WP3 which are planned for M18

will be functional, as the elastic scalability of the resources would require from infrastructure to support this.

7. Seamless Data Analytics Framework

7.1. Requirements Specification

Table 16 – requirement REQ-SDAF-01 for Seamless Data Analytics

	Id	Level of detail	Type	Actor	Priority
	REQ-SDAF-01	Software	FUNC	Developer	MAN
Name	Provide access to data stores via a single and common interface.				
Description	BigDataStack includes two different data stores: the LeanXcale relational data store and IBM object store. The dataset can be fragmented and distributed over the two data stores (historic data being moved to object store). However, the application should be kept unaware of these internal data transfers. The application needs a common interface to submit queries, without having to specify where the data is stored.				
Additional Information	A federation mechanism is required that will encapsulate the process of data retrieval from the two data stores. The LeanXcale access point will act as the federator between the relational and the Object Storage. The LeanXcale data base already provides a common JDBC interface for data connectivity. The federator will receive the query and execute it in both data stores. For the object store, the access would be via Spark SQL, which also provides a JDBC interface. The federator will take into consideration the operations that can be supported in order to push down the operations accordingly. Regarding the relational store, all operations will be pushed down to the store. At the very end, the federator will merge the results and return back the result set. It shouldn't count data that appears in both data stores twice.				

Table 17 - requirement REQ-SDAF-02 for Seamless Data Analytics

	Id	Level of detail	Type	Actor	Priority
	REQ-SDAF-02	System	DATA	Developer	MAN
Name	Move historical data from the relational data store to the object store.				
Description	Data ingested by the use cases will be stored into the relational datastore, as they are operational, in order to ensure data consistency in terms of ACID properties. After a configurable period of time, called the *freshness window* (which depends on the data set), the data becomes outdated and is no longer used by operational workloads. However, this historical data is still				

	valuable and can be exploited by Big Data analytics algorithms. This data should be moved from the LeanXcale data base to the IBM object store.
Additional Information	The LeanXcale data base provides a mechanism that allows to periodically produce a dumb snapshot of the modified data. This information will be transformed accordingly and will be pushed to an Apache Kafka queue. A Kafka based connector will, periodically pull this information and import the historical data to the object store.

Table 18 - requirement REQ-SDAF-03 for Seamless Data Analytics

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-SDAF-03	Software	DATA	Developer	MAN
Name	Inform the LeanXcale data store when data are imported to the object store.				
Description	When data are pushed to the Apache Kafka queue, the LeanXcale data base can drop them. However, due to the asynchronous design, the LeanXcale data base cannot know when the data has been made available to the object store. As a result, the object store must inform the LeanXcale data base regarding the successful insertion of the data, so that the LeanXcale data base can safely drop these data.				
Additional Information	One possible solution to deal with this requirement will be the introduction of marking the data to be transferred to the object store by additional timestamps. Data that is being flushed and exported to the Kafka queue can be marked that way, so that later on, the object store can inform the LeanXcale data base that this bunch of data has been successfully imported. By doing so, the *federator* component can push down operations accordingly, and only request specific data from the underlying data stores. Data that are known to the LeanXcale data base that has been previously uploaded to the object store, will not be retrieved by the *federator* and can be safely discarded by the *vacuum* process of the LeanXcale data base.				

Table 19 - requirement REQ-SDAF-04 for Seamless Data Analytics

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-SDAF-04	Software	DATA	Developer	OPT
Name	Optimize query execution				
Description	The federator receives a query and executes it into the different stores. The federator will be based on the LeanXcale query engine. The latter provides a query optimizer, which allows it to examine the different execution plans that can be produced in order to execute a query. However, it has been implemented to evaluate plans to be executed locally. It should be extended in order to take into consideration the operations that can be pushed down to the object store, and whether or not it is worth for an operator to be				

	pushed down, according to the response time of the execution from Spark SQL, the amount of data that will be retrieved to the federator etc.
Additional Information	

Table 20 - requirement REQ-SDAF-05 for Seamless Data Analytics

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-SDAF-05	Software	DATA	Developer	OPT
Name	Optimize access to Object Storage.				
Description	In order to perform analytics efficiently on Object Storage, a client-side caching/acceleration layer is needed. This is critical for a hybrid cloud scenario, where some of the customer data is on premise (potentially the LeanXcale data base and Spark) and some is in the cloud (potentially IBM COS). In such a scenario, when performing analytics, data needs to move from COS to Spark across the WAN, therefore minimizing the amount of data movement when part of the data is retrieved multiple times is of utmost importance. A similar scenario involves multi-cloud, where a dataset may be distributed among more than one cloud, also requiring data transfer across the WAN for the purposes of analytics.				
Additional Information	This complements data skipping and data layout techniques to further reduce the KPI measuring the number of bytes sent from Object Storage to Spark.				

7.2. Design

Figure 11 - Federation of the two data stores in the scope of the Seamless Data Analytical Frameworkshows the main architectural subcomponents of the Seamless Data Analytical Framework, from an application perspective. This framework can be considered as a black box from an application point of view, which includes two distinct data stores of a different type and purpose: LeanXcale relational data store that can serve write-intensive operational workloads, ensuring transactional semantics, and IBM object store that can execute heavily analytical queries on huge datasets requiring petabytes of storage. Data can either exist on the LeanXcale data base, or the object store, or co-exist in both. As a result, data can be fragmented across the data stores, however, from an application point of view this must be totally encapsulated by the framework itself.

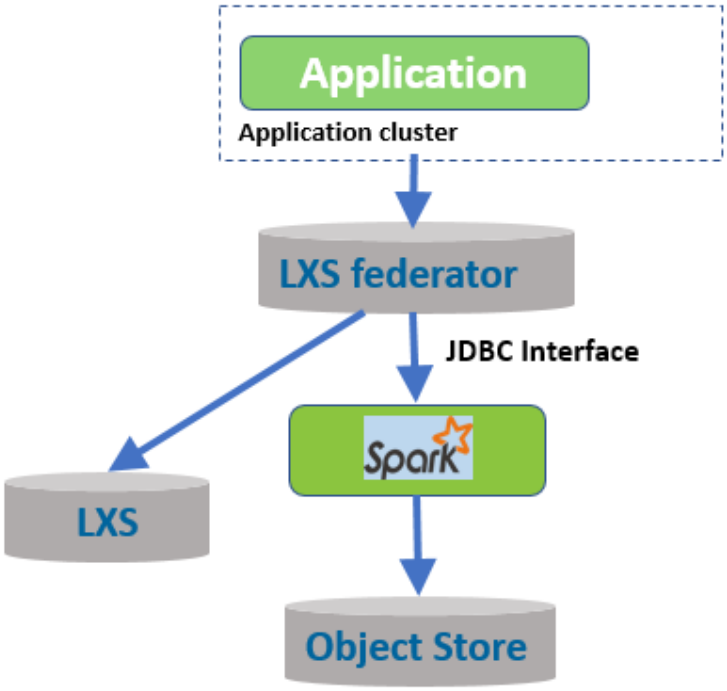


Figure 11 - Federation of the two data stores in the scope of the Seamless Data Analytical Framework

The federation between the two data stores will be based on LeanXcale internal Query Engine that has polyglot capabilities and can join data coming from different sources, exploiting the concept of data lakes. Having LeanXcale Query Engine (QE in the following) as the *federator* of the framework, it will address the REQ-SDAF-01 requirement: Data connectivity with the *federator* will be made via the JDBC implementation, thus providing a common way for all use cases and platform components to commonly access both stores in all cases. Regarding the data modification operations, they will be directly forwarded to LeanXcale relational data store to be executed. However, data retrieval operations have to involve both stores, as data can be stored to any of those (or both). In order to retrieve data from the LeanXcale data base, the QE gets the requests from JDBC and executes the query in its distributed storage. Exploiting its capabilities for intra-query parallelism and the ability of its key-value storage to accept push downs and execute locally various operations, the query engine splits the execution and distributes it in parallel, by pushing down the majority of operations, and then, it merges the intermediate results and returns them back to the user. The LeanXcale QE can push down simple selections and aggregation operators. Constructing the results for the latter take into account the following: the sum/count operations can accumulate the intermediate results, the max operation will require a logical operation to keep the maximum value across the nodes per granular row, and the average operation can be split to sum divided by count. Therefore, the LeanXcale Query Engine can be distributed across various nodes and uses a random instance to coordinate the distributed execution. In the case of the IBM object store, the latter is integrated with Spark, which can be used on top to provide additional query capabilities. Moreover, Spark provides a JDBC interface to enable data connectivity. Thus, by exploiting the polyglot capabilities of the LeanXcale data base, its Query

Engine can be used to push down operations both to the LeanXcale data base and to Spark, via the JDBC, and merge the intermediate results as previously described.

Figure 12 - LeanXcale data base to IBM Object Storage pipeline for historical datashows the pipeline designed to move historical data from the LeanXcale relational data store to the IBM object store. The LeanXcale data base periodically exports a dump of the latest updates, which now fall outside the freshness window (depicted in the diagram as cold data slices). It transforms it to the common agreed format and push this information to a Kafka queue. A Kafka based connector, on the other hand, periodically pulls information from the queue, and injects this data into the object store. As a result, eventually the data exported by the LeanXcale data base is now available in the latter. After the successful ingestion of the historical data, the LeanXcale data base needs to be informed that the data has been persisted in the object store, so that it can be discarded from the relational database.

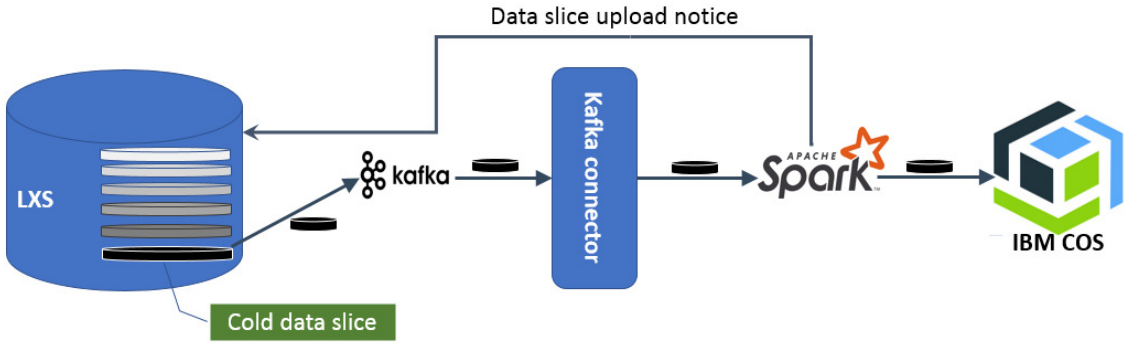


Figure 12 - LeanXcale data base to IBM Object Storage pipeline for historical data

7.2.1. Object Storage Acceleration Layer

The COS Acceleration Layer is a client-side gateway to IBM COS, based on flash storage. Applications access this layer in the same way they would access COS itself, via an S3 API. When an application retrieves data, the layer is responsible for reading data from a remote instance of COS and caching the data locally in flash storage, while also returning the data to the application. The layer evicts data from the cache according to some cache replacement algorithm. Figure 13 - COS Acceleration Layerdepicts the proposed architecture of this feature.

The API to access this layer is at the Object Storage level, therefore this layer is general purpose and all applications that require Object Storage can be supported. However, in the case of BigDataStack, the data stored in Object Storage will typically be rectangular data, and the workload will typically be SQL queries over that data. For this scenario, more information is known about the workload which can potentially provide hints to the caching layer to pre-fetch data which is about to be read, retain in cache data which is likely to be read again in the near future, or discard data from cache which is unlikely to be read in the near future.

This feature complements the data skipping and data layout technologies described in section 5. Moreover, we expect high synergy between these topics. For example, data skipping metadata could be extended to contain information about which objects are accessed most frequently, which would help make caching decisions. In addition, for the purposes of data

skipping we currently intercept Spark SQL queries at the place where the list of objects to be retrieved is assembled and filtered. At this point we may have an opportunity to prefetch data to the caching layer if desired. Moreover, if data skipping metadata is itself stored in COS, then the caching layer could prioritize its caching above other data in COS, and this would enable efficient access to metadata for data skipping. Finally, when the Data Layout Manager is used, this can potentially enhance cache efficiency, since the object layout will more closely match the queries used – i.e. the rows retrieved by a query will likely be spread across less objects.

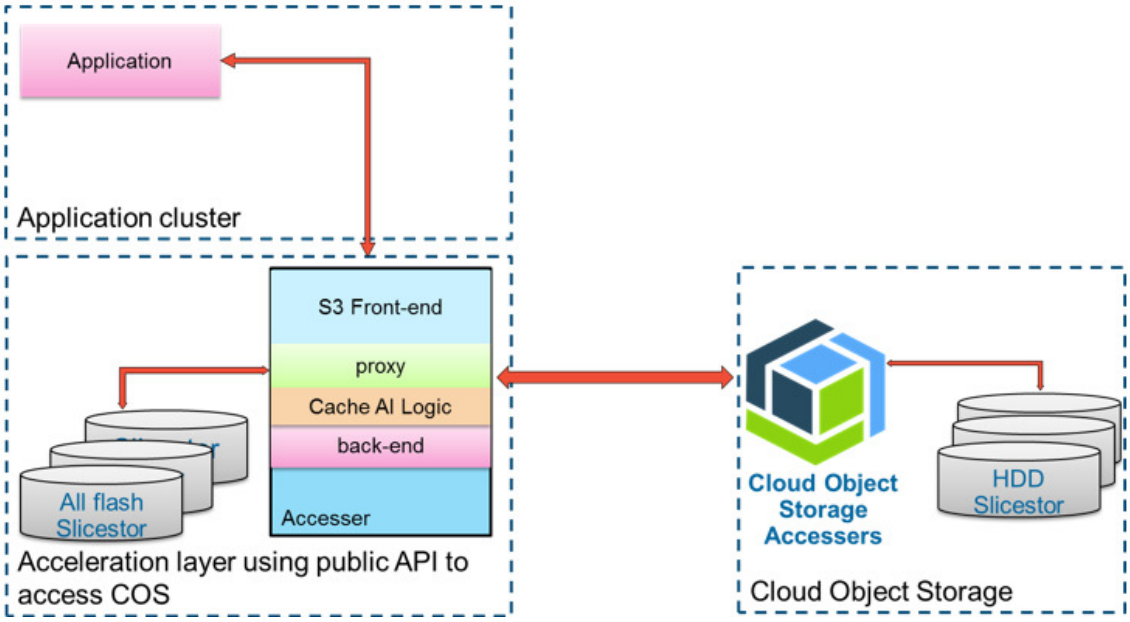


Figure 13 - COS Acceleration Layer

7.3. Early Prototype

As an early prototype in M12, the main goal is to provide a functional prototype, so that all components that need to retrieve data from both stores, should be able to do it. Towards this direction, the deployment of both datastores will be ready, along with an early implementation of the data federator. The latter provides a JDBC implementation, which is a well-known standard that enables data connection with a database system. As a result, all components that will need to retrieve data will have to use the JDBC driver provided by the LeanXcale data base. The federator should be able to retrieve data that resides in two different stores. However, in M12, data would be retrieved only from LeanXcale relational datastores. In parallel, a study towards the ability to connect to Spark via JDBC and its capabilities in terms of supported operations will be conducted, so that its results can be used during the integration of the *federator* with IBM object store at the latter phase. Finally, the definition of the data schema to be used for the ship management use case will be released, along with its necessary extensions to include metadata needed for the movement of the historical data from the LeanXcale data base to IBM object store.

7.4. Use Case Mapping

The *Seamless Data Analytical Framework* is based on two main pillars: The *Federator* mechanism which lies on top of the two involved data stores and it is the central point of access to the data of the platform, and the deployed *pipeline* that is being used for transferring historical information from the LeanXcale relational data store to IBM object store. Regarding the latter, the ship management use case, as described in the previous section, will be mainly used to validate its functionality. This use case produces IoT data coming from various sensors deployed on its fleet, which are ingested in the store as a per-minute basis. This information can be considered historical after a certain point in time, and will be transferred to the object store, through this pipeline.

Regarding the *federator,* as the main access point to the data stored in the platform, it will be used by all components of BigDataStack that require the storing or retrieval of data: the integral components of the platform (i.e. data toolkit, and other WP5 tools, the cleaning process and the CEP engine of Data as a Service block etc.) and the use case applications. When it comes to update operations, the *federator* will push down these operations directly to the LeanXcale relational data store that ensures transactional semantics. When it comes to read operations, it will push down the queries both the LeanXcale data store and IBM object Store and will merge the intermediate results, filtering out possible records that co-exist in both stores.

7.5. Experimental Plan

The plan for experimentation can be summarized as follows:

1. The overhead of the federator in terms of latency should be zero, when requesting data that are stored in the LeanXcale data store only.
2. The overhead of the federator in terms of latency should be zero, when requesting data that are stored in the object store only.
3. The overhead of the federator in terms of latency should be less than defined value when joining data coming from both data stores.
4. The responsiveness of the framework should be the same when executing heavy analytical queries, while at the same time, the LeanXcale data store exports the historical data and pushes them to the Apache Kafka Queue.
5. The responsiveness of the framework should be the same when the LeanXcale data store exports the historical data under heavy operational workloads coming from the IoT data ingestion that happens in parallel.
6. The framework can handle Big Data analytics, when the result is of a significant size.

7.6. Next Steps

During M14, the federator should be able to execute queries both in the LeanXcale data store and in IBM object store, as its integration with the Spark should have been finalized. By this, data modification operations will be directly forwarded to the relational store, while data retrieval operations will be available either from the LeanXcale data store or from the object store. However, merging the intermediate results will be still in progress, and planned to be delivered in M16. At this phase, the first prototype of the federator should have been released, and the experimentation is planned to take place in order to measure the performance and resource consumption of the prototype, so as to further improve its implementation.

Moreover, apart from the federator, the data pipeline will be implemented. Until M14 the definition of the input expected by IBM will have been decided and LeanXcale is planned to be able to export the dumps in the Kafka queue, with respect to the predefined format. Until M16, the data transfer of historical data from the LeanXcale data store to the IBM Object Storage should be working, and IBM should be able to retrieve this information from the queue and import it to the object store. Finally, the definition of the notification mechanisms to be developed so that the LeanXcale data store can be aware when data are available to the object store, so that could safely discard them, will be defined by M14. At M16 this mechanism should be functional, so that there can be a fully working prototype of the Seamless Data Analytical Framework by M18, taken into account that data might exist either on the LeanXcale data store, or in the IBM object store, or co-exist in both.

8. Data Quality Assessment & Improvement

8.1. Requirements Specification

The following tables present the requirements that Data Quality Assessment & Improvement module should satisfy.

Table 21 - requirement REQ-DQAI-01 for Data Quality Assessment & Improvement

	Id	Level of detail	Type	Actor	Priority
	REQ-DQAI-01	Software	DATA	Developer	MAN
Name	Infer data schema				
Description	The Data Quality Assessment and Improvement module should be able to infer a data schema for a given dataset. The data schema should describe the name of each field, its type (e.g., integer, floating number, string, etc.) and its presence (mandatory or optional).				
Additional Information	The schema will be stored in a sharable format (e.g. JSON document) and the system should be able to recall it and compare a new dataset against it, to discover lurking anomalies.				

Table 22 - requirement REQ-DQAI-02 for Data Quality Assessment & Improvement

	Id	Level of detail	Type	Actor	Priority
	REQ-DQAI-02	Software	DATA	Developer	OPT
Name	Create schema environments				
Description	The Data Quality Assessment and Improvement module should be able to different schema environments, for example for training and serving datasets.				
Additional Information	This way the user should be able to feed the system slightly different datasets for training, validation, testing and serving purposes.				

Table 23 - requirement REQ-DQAI-03 for Data Quality Assessment & Improvement

	Id	Level of detail	Type	Actor	Priority
	REQ-DQAI-03	Software	DATA	Developer	MAN
Name	Check data anomalies				
Description	The Data Quality Assessment and Improvement module should be able to compare a given dataset to a restored data schema and produce a data anomaly assessment, e.g., discovering missing columns or wrong data types.				
Additional Information	The final analysis will be stored in a sharable format (e.g. JSON document) and the system should be able to recall it and present it to the user to act.				

Table 24 - requirement REQ-DQAI-04 for Data Quality Assessment & Improvement

	Id	Level of detail	Type	Actor	Priority
	REQ-DQAI-04	Software	DATA	Developer	MAN
Name	Check data skew and drift				
Description	The Data Quality Assessment and Improvement module should be able to detect skew between training and serving data, as well as drift between training datasets in different model versions.				
Additional Information	The final analysis will be stored in a sharable format (e.g. JSON document) and the system should be able to recall it and present it to the user to act.				

Table 25 - requirement REQ-DQAI-05 for Data Quality Assessment & Improvement

	Id	Level of detail	Type	Actor	Priority
	REQ-DQAI-05	Software	DATA	Developer	DES
Name	Detect data source deterioration				

Description	The Data Quality Assessment and Improvement module should be able to detect if a data source, e.g., an IoT sensor, is not malfunctioning and always emits corrupted data.
Additional Information	The final analysis will be stored in a sharable format (e.g. JSON document) and the system should be able to recall it and present it to the user to act.

Table 26 - requirement REQ-DQAI-06 for Data Quality Assessment & Improvement

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-DQAI-06	Software	DATA	Developer	MAN
Name	Detect unexpected range of values				
Description	The Data Quality Assessment and Improvement module should be able to detect unexpected range of values in any field of the dataset, given a specific context, i.e., the values in neighbouring columns.				
Additional Information	The final analysis will be stored in a sharable format (e.g. JSON document) and the system should be able to recall it and present it to the user to act.				

8.2. Design

The design of the data cleaning module consists of two phases: i) data schema inference, ii) data veracity. In **Error! Reference source not found.**, an overview of the whole process is depicted.

During the first phase, we compare every new dataset to a previously inferred data schema and look for structure or statistical anomalies. More precisely, given a training dataset, we compute common statistics that permits us to infer a schema prototype for this kind of dataset. For example, if we want to train a machine learning model, we assume that the first training dataset is drawn from a distribution that truthfully represents the statistical population. In the case of ship management use case, a carefully curated dataset, that is attentively cleaned for the purposes of training a model like predictive maintenance, can also be used as a training prototype or template for the data assessment and improvement models.

Using this statistical analysis and by employing conservative heuristics, we infer the schema of this dataset. Then, if we want to retrain a machine learning model that was built upon this dataset, or query it for predictions, the new training or serving dataset that is used is compared to the inferred schema and checked for anomalies. This way we can also check for skew or drift between the training and serving dataset distributions, and act accordingly. Thus, if the new training dataset derives from a very different statistical distribution, the system should raise an alert to the user. The data analyst should be also notified if a serving dataset, in production, is drawn from a different distribution, causing erroneous predictions.

In the second phase, we take advantage of the recent developments in artificial neural networks (ANN) and deep learning (DL), to extract latent features that correlate different fields, and identify possible defects in their measurements.

By passing each measurement through a deep ANN, we get a distributed representation of each concept, e.g., engine temperature or valve pressure from the vessels of the ship management use case, and their relationship, which is a much more comprehensible notion for the machine. This process helps us gain knowledge about the correlation between fields and diagnose irregularities in the input data. The output of the neural network is a scoring value, that measures the likelihood of those measurements occurring together. Figure 15 - Data veracity ANN architecture presents a high-level design of this neural network.

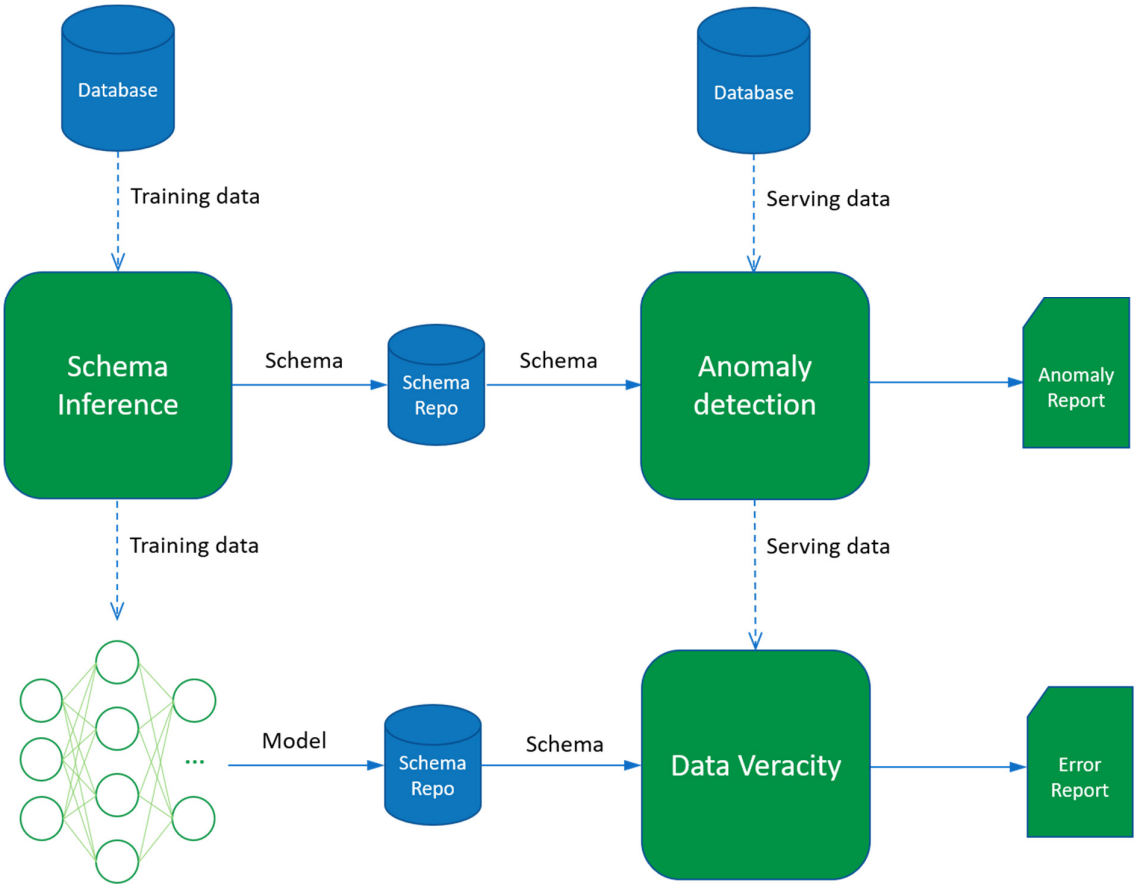


Figure 14 - Data Cleaning Overview

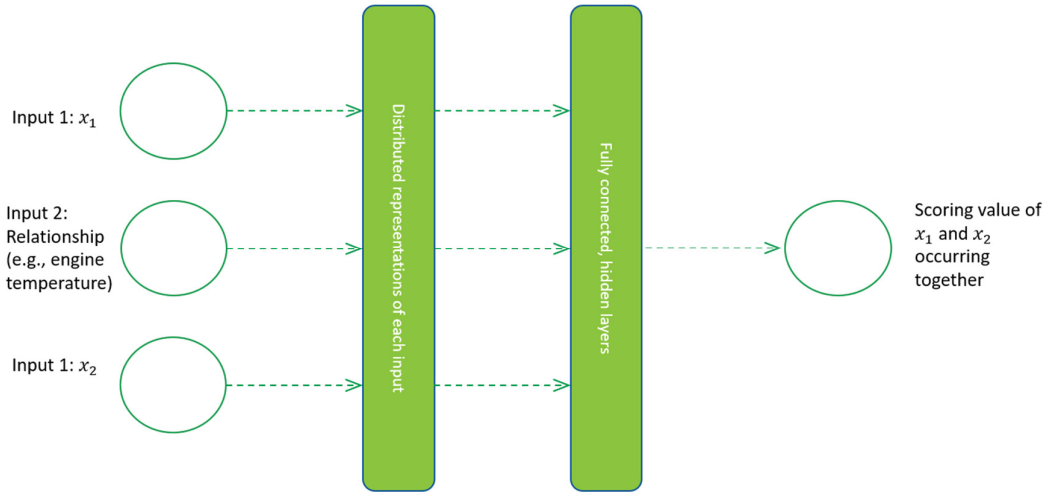


Figure 15 - Data veracity ANN architecture

8.3. Early Prototype

In the first version of the data cleaning module we focus on the schema inference part, and all the utilities that make this sub-component work effortlessly while integrated with the whole system. For example, making the results (descriptive statistics, data schema, anomaly report, etc.) available to every other system component, through an agreed transfer protocol (RESTful APIs, JSON files, etc).

Figure 16 - Dataset schema exampledepicts how an inferred data schema is represented in JSON file format, while Figure 17 - JSON anomaly report document exampledepicts an anomaly report represented also as a JSON file format. In the schema example we see a feature named “payment type”, from the connected consumer use case and some computed characteristics of this feature, such as valid values or presence (optional or mandatory). In the report we can see that there are some missing values, and moreover, some values are not of the expected type (e.g., FLOAT).

The flow consists of reading the dataset, infer the schema and store it back to the database. On serving time, the system retrieves the previously inferred schema, compares it to a new dataset and prints out an anomaly report, which it also stores to the database. Every file that is produced during this process should be available to the user, through explicitly developed APIs, or through the datastore.

Baseline technologies that will be used include Deep Learning frameworks, mainly TensorFlow[19] and PyTorch[20], as well as several extensions of these, such as TensorFlow Data Validation and Serving[21]. Other technologies, such as Luigi[22], can be used to build the necessary pipeline.

[19] https://www.tensorflow.org/
[20] https://pytorch.org/
[21] https://www.tensorflow.org/tfx/
[22] https://github.com/spotify/luigi

```
{
   "stringDomain":[
      { },
      {
         "name":"payment_type",
         "value":[
            "Cash",
            "Credit Card",
            "Dispute",
            "No Charge",
            "Pcard",
            "Unknown"
         ]
      }
   ],
   "feature":[
      { },
      { },
      { },
      { },
      { },
      { },
      {
         "valueCount":{
            "max":"1",
            "min":"1"
         },
         "type":"BYTES",
         "name":"payment_type",
         "domain":"payment_type",
         "presence":{
            "minFraction":1.0,
            "minCount":"1"
         }
      },
      { },
      { },
      { },
      { },
      { }
   ]
}
```


Figure 16 - Dataset schema example

```
"payment_type":{
   "reason":[
      {
         "type":"FEATURE_TYPE_LOW_FRACTION_PRESENT",
         "shortDescription":"Column dropped",
         "description":"The feature was present in fewer examples than expected."
      },
      {
         "type":"UNKNOWN_TYPE",
         "shortDescription":"Expected data of type: BYTES but got FLOAT",
         "description":""
      }
   ],
   "shortDescription":"Column dropped",
   "description":"The feature was present in fewer examples than expected.",
   "severity":"ERROR"
},
```


Figure 17 - JSON anomaly report document example

8.4. Use Case Mapping

In the scope of the Data as a Service block, the ship management and connected consumer use cases will be used as the two main demonstrators to highlight the functionalities of the Data Assessment and Improvement module. The ship fleet produces IoT data from numerous sensors deployed on each of its vessels, while in the connected consumer use case, a recommender system for a retail customer is being built.

In the case of the ship management use case, the data is pre-processed by different installations of CEP subsystems (see subsection 4.1), which contain a simple, rule-based cleaning procedure (e.g., removing null values), and then, the Data Assessment and Improvement module can act as a second layer, automated and domain-agnostic data quality estimation. This operation could discover anomalies in data, allowing users to act and pinpoint deterioration in the functionality of different IoT sensors.

In the case of the connected consumer use case scenario, the data analyst could discover anomalies or errors in the datasets used to train the recommendation system model(s), which could affect performance and accuracy. Moreover, detecting skew between training and serving data, as well as drift in training data in different days will be critical in locating bugs in production or during the re-training process, on both scenarios.

8.5. Experimental Plan

An initial plan for experimentation can be summarized as follows:

1. The schema inference submodule should correctly predict the data schema of a given dataset
2. The data analyst can validate, enhance or expand the inferred data schema
3. The Data Assessment & Improvement module should correctly detect the anomalies in a corrupted version of the previously given dataset
4. The Data Assessment & Improvement module should facilitate the application of common cleaning tasks, like removing null values
5. The system comprises a report and presents it to the data analyst to act
6. Development and testing dataset will be used to assess the performance of the models, as the data gathered by the system accumulate

8.6. Next Steps

During M12, the Data Assessment and Improvement module should be able to correctly infer a dataset schema and detect the anomalies lurking in a corrupted version of this dataset. It should also detect skew between training and serving data, as well as drift between training sets of different time periods.

By M14 the module should be able to pinpoint possible errors in the value range of any field in the dataset. This work is the focus of the second phase of the module's development, namely the data veracity phase.

By M16 the module should be fully functional and integrated to the system.

9. Predictive & Process Analytics

The Predictive & Process Analytics component is used in BigDataStack to: (a) perform process discovery from raw, event driven, data capturing activities stored as event logs, and (b) recommend the next step when designing a new process model in the process modelling phase, based on the analysis of historical data.

Its input is typically an event log generated by different processes executed in BigDataStack. The basic functionality provided by this component is to perform process mining and analytics techniques on such input event logs. The application of such techniques is useful for different tasks. First, for process discovery, in the case that the underlying process that generated the event log is unknown. In case the process is known, the process analytics component is able

to perform conformance checking. Finally, as soon as a process model has been identified from the event log, it can be exploited for recommendation purposes (i.e., recommend the next step in a process based on historical data captured in the log) or prediction of the next step in a process that is executed.

9.1. Requirements Specification

Table 27 - requirement REQ-RD-01 for Predictive & Process Analytics

	Id	Level of detail	Type	Actor	Priority
	REQ-RD-01	Stakeholder	FUNC	Developer	ENH
Name	Global event tracker connection				
Description	A connection to the Global Event Tracker (GET) is needed for the Predictive & Process Analytics component.				
Additional Information	The information stored in GET is crucial to the implementation of this module.				

Table 28- requirement REQ-RD-02 for Predictive & Process Analytics

	Id	Level of detail	Type	Actor	Priority
	REQ-RD-02	Stakeholder	FUNC	Developer	ENH
Name	Connection to the Process Modelling Framework				
Description	A connection between this component and the Process Modelling Framework needs to be established, so information can be sent and received.				
Additional Information	The recommendations made by this component will be in real time, as the Business Analyst – Data engineer is modelling the process.				

Table 29- requirement REQ-RD-03 for Predictive & Process Analytics

	Id	Level of detail	Type	Actor	Priority
	REQ-RD-03	Stakeholder	FUNC	Developer	ENH
Name	Data pre-processing				

Description	The data ingested by this component needs to be in an eXtensible Event Stream[23] (XES) file format. A tool is created, depending on the format of the Global Event Tracker.
Additional Information	The XES standard defines a grammar for a tag-based language whose aim is to provide designers of information systems with a unified and extensible methodology for capturing systems behaviours by means of event logs and event streams is defined in the XES standard. An XML Schema describing the structure of an XES event log/stream and an XML Schema describing the structure of an extension of such a log/stream are included in this standard. Moreover, a basic collection of so-called XES extension prototypes that provide semantics to certain attributes as recorded in the event log/stream is included in this standard.

Table 30- requirement REQ-RD-04 for Predictive & Process Analytics

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-RD-04	Stakeholder	FUNC	Developer	ENH
Name	ProM framework				
Description	ProM is an extensible framework that supports a wide variety of process mining techniques in the form of plug-ins.				
Additional Information	The process mining techniques used will be utilized to derive metrics of the event log, to create the semantics needed between events for the recommendation process.				

9.2. Design

Figure 18 - Design of Process Analytic componentdepicts the high-level design of the Process Analytics component, focusing mainly on its constituent sub-components and their inputs/outputs. As already mentioned, the basic input for this component is an event log. Since this log may come in different formats, depending on the way data is captured and, on the application-specific logging mechanism, the first step is to transform the log in a format suitable for further analysis. This format is pre-specified and defined based on the input requirements of the next sub-component in the processing flow.

[23] http://www.xes-standard.org/

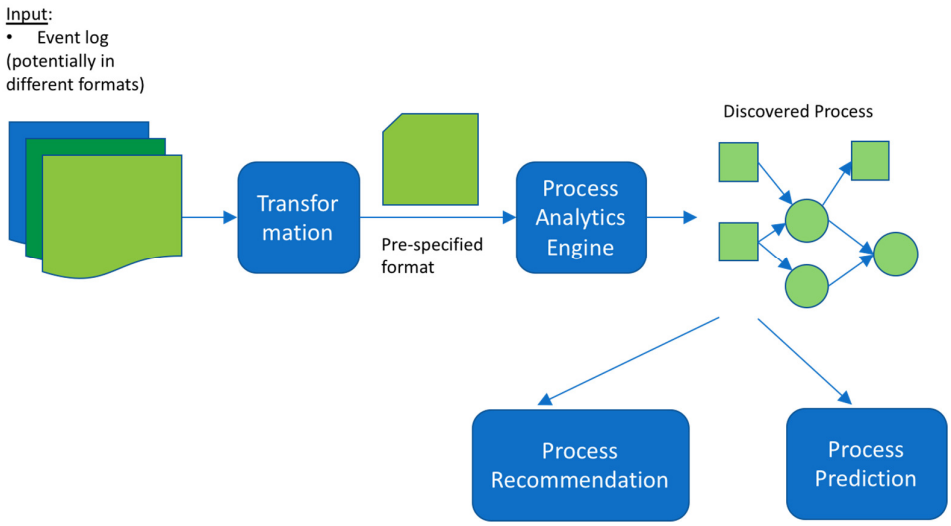


Figure 18 - Design of Process Analytic component

After this transformation, the process analytics engine takes as input the transformed event log and performs the main step of analysis. As a result, a process model is generated capturing the steps of the underlying process and the possible paths between steps, which can typically be represented using a graph-like structure. Additionally, it is possible for the graph to have edges annotated with weights, which correspond to the frequency that a specific transition between steps (an edge) has been observed. Put differently, in a weighted graph representation, the weight corresponds to transition probabilities between steps of the process.

The output of the process analytics engine can be exploited during the execution of a process, in order to perform prediction of the next step. Furthermore, the same output can be exploited during process modelling, by providing recommendations to the process designer/modeler, based on the discovered patterns of process execution.

9.3. Early Prototype

At the time of this writing, the transformation module has been implemented, and it provides the functionality to transform input event logs to the desired pre-specified format, XES. The XES standard defines a grammar for a tag-based language whose aim is to provide designers of information systems with a unified and extensible methodology for capturing systems behaviours by means of event logs. Obviously, further extensions of this module may be necessary in the future, in order to accommodate new input formats. However, this is fairly straightforward to implement.
The process analytics engine is the cornerstone module of Process Analytics providing key functionalities. In the early prototype, the process analytics engine is implemented using the ProM framework[24], which is a state-of-the-art prototype system for process analytics. Functionalities of the aforementioned framework, in the form of algorithms, are used during the discovery phase to derive sane process models and capture statistical measures that may

[24] http://www.promtools.org/

be used for the recommendation of a next step during the creation of a process in the Process Modelling Framework.
In addition, the early prototype also contains a custom event log generator that has been implemented. This generator produces synthetic logs with varying amount of noise, which is quite useful to test and fine-tune the process analytics engine using controlled inputs.

9.4. Use Case Mapping

In the case of Predictive and Process Analytics the Use Cases benefit, at the moment, primarily in the Process Modelling phase of the project, as no viable event driven data source has been presented. Considering that all the steps of a process executed in project are documented in the Global Event tracker, trough process mining the model of a process can be discovered and in combination with the mechanism proposed here provide insights and recommendations, to the user, concerning the process flow. Following is an example using the process described in Figure 18 - Design of Process Analytic component

The event log used for this example process was generically made as the Global Event Tracker is not up and running at this point. The event log created in XES format is comprised of the bellow events, shown here as output of the Command Line Interface of the ProM framework:

{Streaming Ship Data =7, CEP2 + Data Cleaning Module 1=1, LXS=5, IBM COS=3, Kafka=4, CEP2 + Data Cleaning Module 2=2, Spark=6, CEP1=0}

The number that each event is equal to is a randomly generated identification for the process. Through the algorithms used we get information concerning the frequency of each step's succession to another one, the dependency of one step to another, the amount of noise in the input file etc. Through the analysis of these metrics a recommendation system will be put into place for the Process Modelling phase of the project.

9.5. Experimental Plan

The two main application scenarios of the Predictive & Process Analytics component are recommendation and prediction in the context of processes. The experimental plan is going to verify and validate both the qualitative dimension as well as the performance dimension.
In terms of quality, typical measures, such as precision, recall, and F-measure, are going to be used. Alternative options will also be considered, such as blind testing using human users (process modelers), by capturing their feedback with respect to the produced recommendations.
In terms of performance, the execution time for processing the event log and producing recommendations is going to be reported for different input parameters (event log size, level of noise, complexity of the underlying process, etc.).

9.6. Next Steps

The next steps and planned activities of Predictive & Process Analytics component towards M18 are the following:

- Have a prototype implementation that (a) comprises all designed modules, (b) works end-to-end, and (c) provides first promising results with respect to the quality of recommendations. Therefore, the major focus until M18 is going to be on recommending the next step in a Process. The prediction of future steps in a process is going to be researched after M18.
- Perform empirical evaluation on the quality of recommendations using both synthetic and real-world event logs, in order to verify the quality and accuracy of produced results.

10. Real-time Complex Event Processing

10.1. Requirements Specification

The real-time Complex Event Processing (CEP) deals with processing of events as they are produced (before they are stored). CEP processes each event independently (e.g., filtering those that do not match a predicate, adding more information to events) of grouping them on windows (e.g., calculate average speed in the last hour). CEP runs on a single node or on top of a distributed system. CEP is able to run distributed and parallel queries over data streams. CEP can be deployed at the edge to run the queries as close as possible to the data. For instance, it can run at vessels to detect anomalous conditions and accelerate decisions before sending the information to a central on shore office.

Table 31- requirement REQ-CEP-01 for CEP

	Id	Level of detail	Type	Actor	Priority
	REQ-CEP-01	System	FUNC	Developer	MAN
Name	Manage data from different sources to generate alarms if required.				
Description	The CEP will process data on the fly coming from sensors. Each sensor sends events each minute. CEP will analyse the data according to a set of rules and generate alarms.				
Additional Information	The processing will be both stateless and over windows of time and number of events.				

Table 32 - requirement REQ-CEP-02 for CEP

	Id	Level of detail	Type	Actor	Priority
	REQ-CEP-02	System	FUNC	Developer	MAN
Name	Send alarms and data from each node of the distributed environment to the data centre.				
Description	Once metrics have been analysed, the CEP will send the alarms and the data to a central location (data centre).				
Additional Information	The CEP will run on nodes of a geographically distributed environment.				

Table 33 - requirement REQ-CEP-03 for CEP

	Id	Level of detail	Type	Actor	Priority
	REQ-CEP-03	System	PERF	Developer	MAN
Name	Data from distributed nodes is aggregate at a central location.				
Description	Further processing over remote data will be done at a central location.				
Additional Information	The CEP processing will scale to tens streams coming from different remote sources.				

Table 34 - requirement REQ-CEP-04 for CEP

	Id	Level of detail	Type	Actor	Priority
	REQ-CEP-04	Stakeholder	PERF	Developer	ENH
Name	Store data on the data store				
Description	The CEP will store the data at the rate is being produced.				
Additional Information	Both CEP and LX will run at the same location.				

10.2. Design

Figure 19 - CEP Componentsshows the main components of CEP. CEP is a parallel and distributed data streaming engine. That is, the execution of queries can be distributed among different nodes in a cluster. Streaming queries can also run in parallel in a single node.

Clients of CEP (Client, Sender App in the figure) connect and submit queries to CEP using a driver, the JCEPC driver. Applications consuming the results (Client, Receiver App in the figure) of the queries also use the JCEPC driver to receive the events.

The Orchestrator is in charge of managing the CEP components and deploying and monitoring queries. The Orchestrator stores meta-information about the system (e.g., query deployment, number of nodes…) in Zookeeper, which is used as a reliable registry. The Instance Managers (IM) are the components in charge of query execution. The number of IMs is configured at deployment time. IMs can be added and removed at runtime without stopping the query processing. The Orchestrator automatically partitions queries into subqueries to distribute them among IMs. Each IM can run several subqueries. Clients can also define the distribution and parallelism of the queries using the JCEPC driver.

CEP provides a set of performance metrics (i.e., throughput, latency, CPU and memory usage) that are handled by the metric server.

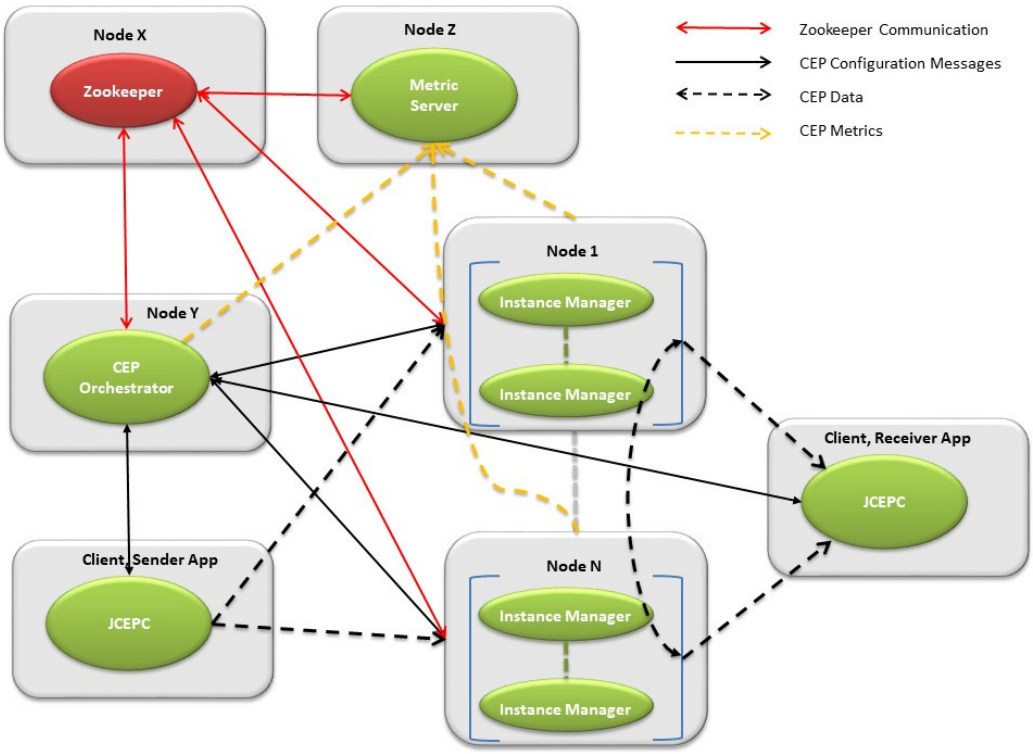


Figure 19 - CEP Components

10.3. Early Prototype

The goal for M12 is to provide a functional prototype, so that events can be processed as they are produced in remote locations. The prototype will run the queries close to where the data is produced. CEP will be able to process thousands of events per second in remote deployments. The queries will be both stateless and stateful. CEP will provide a Java driver for client applications to run queries on CEP.

10.4. Use Case Mapping

In the scope of WP4, the ship management use case will be used as the main demonstrator to highlight the advantages of CEP. The sensors (up to thousand) on the vessels produce data every minute that currently is sent to a central office every three hours to be analysed off-line. CEP will process the data as it is produced triggering alarms as soon as possible. This use case requires both stateless queries (e.g., checking the current value of a sensor) and stateful queries (e.g., triggering an alarm is the average fuel consumption during the last half an hour was higher than a given value). The data will also be set to the central office for further analysis and correlation with historical data, including basic data cleaning. This data cleaning is done for each individual record/event. For instance, replacing null values, changing the format of date and time.

10.5. Experimental Plan

The performance evaluation for the CEP component will consider the following scenarios:

- Performance evaluation with stateless queries over data coming from sensors and generating alarms. Thousands of events can be generated and processed per minute.
- Performance evaluation with stateful queries over windows of time (2 minutes) coming from thousands of sensors.
- Running the CEP on hardware with different resources.
- Performance evaluation of operators storing data on LeanXcale.

The benchmarking of the CEP will use ship management data provided and synthetic data.

10.6. Next Steps

The next planned activities for CEP consists of completing the implementation of a full functional CEP running on local cluster and execute performance evaluation experiments. In parallel, the implementation of the CEP capabilities for running federated queries on WAN environment with heterogeneous processing nodes will start. The deployment will take into account this fact in order to improve performance of queries and avoid whenever possible the communication overhead in these environments.

11. Conclusions

Almost one year after the start of the project, the Data as a Service block presents a fine set

of data services which can be mapped to the major phases of Big Data processing. At this point we defined two scenarios which reflect representative requirements of one of the project use cases. The next seven months till the half of the project (M18) will be critical in the sense the that they should permit to verify that all these services not only work fine by themselves but can also be used together to solve real industrial problems and be applied to additional scenarios of BigDataStack.

Past M18, building on the initial integration of the tasks, the goal will be twofold: first, to develop the more advanced capabilities of the tasks, secondly to get fully integrated with the other major building blocks of BigDataStack overall architecture.

[1] https://spark.apache.org/