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/

D3.1 – WP 3 Scientific Report and Prototype Description - Y1

Work Package	WP3 – Data-driven Infrastructure Management
Lead Author (Org)	Orlando Avila-García (ATOS)
Contributing Author(s) (Org)	Ismael Cuadrado-Cordero (ATOS), Bernat Quesada, Marti Sanchez, Matteo Sotil (ATOS WDL) Jean Didier Totow (UPRC), Sophia Karagiorgou (UBI), Nikos Drosos (SILO), Mauricio Fadel Argerich, Bin Cheng (NEC), Pavlos Kranas, Ricardo Jiménez-Peris, Jose Maria Zaragoza, Diego Burgos Sancho, Javier López Moratalla (LXS), Richard McCreadie (GLA), Marta Patiño, Ainhoa Azqueta (UPM), Luis Tomas Bolivar (RHT)
Internal Reviewer(s)	Yosef Moatti (IBM), Pavlos Kranas (LXS), Dimosthenis Kyriazis (UPRC)

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.

Due Date	30.11.2018
Date	5.12.2018
Version	1.0

Dissemination Level

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

Versioning and contribution history

Version	Date	Author	Notes
0.1	24.10.2018	Orlando Avila-García (ATOS)	Creation of the skeleton and first draft of Section 3 and Section 0
0.2	25.10.2018	Orlando Avila-García (ATOS)	Adding general and component-specific sections for *Experimental Plan.* Fixing numbering of component-specific subsections.
0.3	09.11.2018	Orlando Avila-García (ATOS)	Adding contributions from ATOS, UPRC, NEC, UBI in sections 6, 7, 8 and 9. Refinement of section 4.
0.4	19.11.2018	Orlando Avila-García (ATOS)	Adding contributions to requirements from LXS to sections 5, 6 and 8, and to the design to sections 6 and 8. Adding section 7 from GLA and ATOS. Adding amendments to sections 6 and 9 from NEC and UBI, respectively. Adding requirements from UPM to section 8. Adding section 5 from RHT. Refinements of section 2 and 3 by ATOS.
0.5	23.11.2018	Orlando Avila-García (ATOS)	Reorganizing changes in CEP integration at section 8 by UPM. Updating section 5 by RHT. Amendments following the internal review made by IBM: Adding Acronyms table; updating sections 6 by NEC; updating section 7 by GLA and ATOS; updating section 8 by ATOS and UPRC; updating section 9 by UBI; Adding section 3.1 (assumptions) by GLA. Adding section 1 (Introduction) from ATOS.
0.6	30.11.2018	Orlando Avila-García (ATOS)	Refinement of section 5 from RHT. Refinement of section 4.1 (experimental setting) from ATOS WDL. Amendments following the internal review made by LXS and UPRC: Updating section 3.2 (assumptions) by GLA and RHT; upgrading section 3 and 4 by ATOS; upgrading section 5 by RHT; updating section 6 by NEC; upgrading section 7 by ATOS and GLA; updating section 8 by ATOS and UPRC; updating section 9 by UBI.
0.7	3.12.2018	Orlando Avila-García (ATOS)	Final amendments by ATOS.
1.0	5.12.2018	Orlando Avila-García (ATOS)	Adding subsection 4.3 (example scenario) elaborated by GLA, IBM and ATOS.

Table of Contents

List of tables

List of figures

Acronyms

ADS	Application and Data Services
CDP	Candidate Deployment Pattern
ADW	Application Dimensioning Workbench
QoS	Quality of Service
SLA	Service-Level Agreement
SLO	Service-Level Objective
KPI	Key-Performance Indicators
VM	Virtual Machine
AWS	Amazon Web Services
GCP	Google Cloud Platform
CEP	Complex Event Processing
CI	Continuous Integration
CD	Continuous Delivery
SDN	Software-Defined Network
EKS	AWS Elastic Kubernetes Service
AWS	Amazon Web Services
RL	Reinforcement Learning
OKD	Openshift Origin Kubernetes Distribution
LbaaS	Load Balancer as a Service
CRD	Kubernetes Custom Resource Definition
OVN	Open Virtual Networking

1. Executive Summary

This is the Scientific Report and Prototype Description (Y1) for the work done in WP3 on the Data-Driven Infrastructure Management capability of BigDataStack. The document shows the plan to deliver the solution through a series of implementation and experimentation increments. It describes the high-level architecture of the solution, as well as the detailed specification of requirements, design, first prototype and experimentation plan per solution component.

2. Introduction

This deliverable presents Scientific Report and Prototype Description (Y1) for the work of WP3, which is related to the so-called Data-Driven Infrastructure Management capability of the BigDataStack platform. The document shows how the implementation of the solution is planned to be delivered following an incremental and iterative methodology, having cycles of implementation and experimentation. On one hand, the document describes the high-level assumptions and architecture of the capability, as well as detailed requirements, design and prototypes per component. On the other hand, it describes the experimental use case scenarios and plans, as well as the experimental plan per component and its mapping with the use case scenarios.

2.1. Relation to other deliverables

This document is related to the following past and immediately upcoming deliverables in the project.

- D2.4 – Conceptual model and Reference architecture (M6). The description of the high-level architecture of BigDataStack as well as the interplay and integration between the main components. The architecture of the Data-Driven Infrastructure Management as well as the design of the components have been devised to fit into such a global architecture.
- D2.2 – Requirements & State of the Art Analysis II (M11). The specification of BigDataStack requirements is centralized in this deliverable. This specification is a refinement of the first version delivered in D2.1 (M6). The architecture of the Data-Driven Infrastructure Management (DDIM) as well as the design of the components have been devised to satisfy those requirements. **Please note that for the reader's convenience, the requirements related to each one of the DDIM components have also been included (literally brought from D2.2) in the present deliverable, specifically, at subsections 5.1, 6.1, 7.1, 8.1 and 9.1.**
- D4.1 – WP4 Scientific Report and Prototype Description - Y1 (M11). The D3.1 makes references to some of the requirements and components which are designed, implemented and experimented with at WP4, while also the D4.1 references and raises requirements that are being described in the current document. In fact, the Data-Driven Infrastructure Management is meant to provide infrastructure services (Infrastructure-as-a-Service) to those components.
- D5.1 – WP5 Scientific Report and Prototype Description - Y1 (M11). The D3.1 makes references to some of the requirements and components which are designed, implemented and experimented with at WP5; this is because the tools developed at WP5 will interact with the services and resources provided by the infrastructure to implement certain functionality supporting the different BigDataStack stakeholders (see Section 3.3).

2.2. Document structure

The document is structured as follows: Section 3 describes de solution architecture of the data-driven infrastructure management capability of BigDataStack, including the assumptions made (Section 3.2), the architecture vision (Section 3.1), the related platform roles (Section 3.3) and the high-level design (see Section 3.4).

Section 0 presents the implementation and experimentation plan. Starting with the experimental setting (i.e. scenarios, Section 4.1) and plan (i.e. research questions and methodology, Section 4.2), it finalizes with the implementation and experimentation roadmaps (Section 4.3).

The final five sections are dedicated to the requirements specification, design description, use case mapping, prototype description and next steps elaboration for each of the five high-level components of the architecture: Cluster Management (Section 5), Dynamic Orchestration (Section 6), ADS Ranking & Deploy (Section 7), Triple Monitoring & QoS Evaluation (Section 8) and Information-Driven Networking (9).

3. Solution Architecture

This section describes the technical solution for the Data-driven Infrastructure Management. It firstly describes the vision of this BigDataStack platform capability (context, goal, main functions or services). Secondly, it enumerates the assumptions the work package makes about the environment that BigDataStack will be deployed within. Thirdly, it shows the platform roles engaged in the use of the capability as well as an example scenario. Finally, it describes the global design of the solution.

3.1. Vision

The envisioned BigDataStack platform represents a full stack which aims to facilitate the needs of data operations and applications (all of which tend to be data-intensive) in an optimal way. In such a stack, a layer of self-managed and self-optimizing data-driven infrastructure will be the basis for upper-level layers providing higher capabilities (see Figure 1) to BigDataStack platform roles (see Section 3.3).

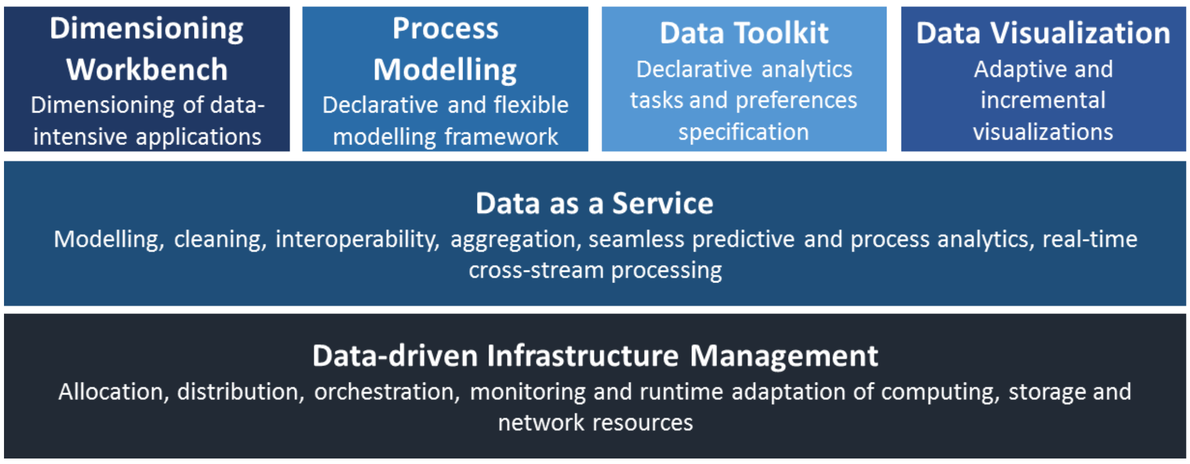

Dimensioning Workbench
Dimensioning of data-intensive applications
Process Modelling
Declarative and flexible modelling framework
Data Toolkit
Declarative analytics tasks and preferences specification
Data Visualization
Adaptive and incremental visualizations
Data as a Service
Modelling, cleaning, interoperability,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

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

These six BigDataStack core platform capabilities are envisioned to achieve the business goals or expectations from the different stakeholders. In the case of the Data-driven Infrastructure Management capability, the goal is **to provide means for efficient and optimized infrastructure operations, incorporating all aspects of data-driven management for the compute, storage and network resources.**

This capability is mainly engaged in the Operation Phase of BigDataStack (see D2.4). It is realised through different components of the BigDataStack infrastructure management system and aims at the management of the complete virtual and physical infrastructure resources, in an optimised way for data-intensive applications.

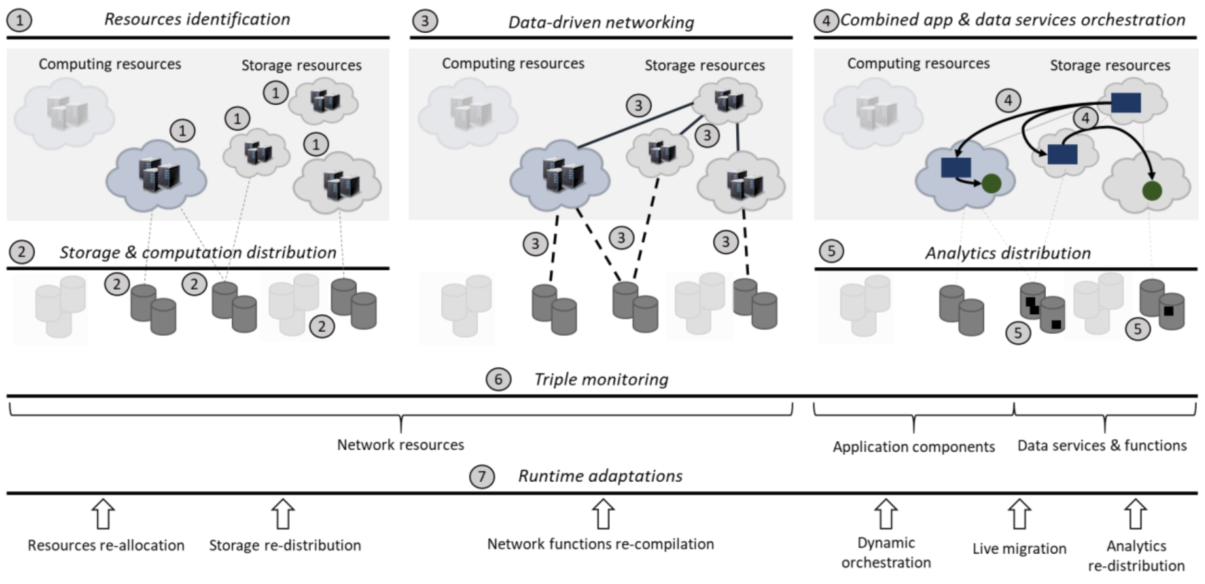


Figure 2 – The BigDataStack seven-step process (extracted from D2.4)

Figure 2 shows the seven-step process comprising the BigDataStack Operation Phase, which needs to be fully supported by the Data-driven Infrastructure Management capability (extracted from D2.4):

1. Based on benchmarking and previous deployments, compute (e.g. VMs, instances, containers) and storage (e.g. block, volumes) resources are allocated.
2. According to the allocated resources, distributed stores (e.g. databases, object stores) are deployed and the data uploaded.
3. Data-driven networking services are also deployed to facilitate the diverse networking needs between different computing and storage resources.
4. Application components and data services are deployed and orchestrated based on application and data-aware deployment patterns. A ranking and deployment function will perform optimal deployments according to those deployment patterns and the reserved computing, storage and networking resources.
5. Data analytics tasks will be distributed across the different nodes of data processing clusters, while orchestration of application components and data services is also performed.
6. Monitoring data is collected and evaluated for the resources (compute, storage and network), application components and data services and operations.
7. Runtime adaptations take place for all elements of the environment including resource re-allocation, storage and analytics re-distribution, re-compilation of network functions and re-deployment or applications and data services.

3.2. Assumptions

Before discussing the requirements for Data-driven Infrastructure Management, it is important to specify any foundational assumptions that the work package makes about the environment that BigDataStack will be deployed within. Indeed, we have so far simply referred to compute, storage and network resources that form the ‘Infrastructure’ in Data-driven Infrastructure Management. However, we need to also consider *what* we are deploying and *when* and *where* we are deploying it.

3.2.1. Containerized Applications

The overall aim of WP3 is to enable the efficient and effective deployment and management of arbitrary user applications on high-performance compute clusters. However, deploying a user application is a complex process, as an application may have hardware dependencies (e.g. it needs a GPU accelerated machine) or software dependencies (e.g. a particular operating system, libraries, or processing engine like Apache Spark). Indeed, it would be an impossible task to build a system that could accept any application without restrictions. As such, simplifying assumptions need to be made to create a general deployment solution.

Over the last few years, one solution to this problem has become popular, namely *application containers*. The core idea underpinning this technique is that the user will create containers for their application's services, which contains the compiled code and any dependencies, all in a single bundle. A container is therefore effectively self-contained, and hence can be more easily deployed. Container orchestration platforms such as Kubernetes or OpenShift then provide a standard platform for deploying containers on bare metal hardware[1]. In this case, the orchestration platform is responsible for assigning resources (e.g. CPU cores and RAM) to individual containers, where the resources of a single physical machine are shared across multiple containers. Complex user applications are often comprised of multiple containers that depend on one another, or rely on shared resources such as a central database repository. Orchestration platforms often provide the means to define groups of containers (e.g. Kubernetes Pods) that are deployed together to support these more complex types of application.

A foundational assumption of BigDataStack is that user applications/services will be either provided in containers, or the user will be making use of pre-built services from the BigDataStack library that are already containerized. The role of Data-driven Infrastructure Management is then to deploy and manage these containers such that user-defined Service-Level Objectives (SLOs) are met.

3.2.2. Hardware Environment

In general, we can consider three scenarios where a user requisitions big data infrastructure, each one comes with its own advantages and limitations. The first and simplest case is where the user already owns and administers their own cluster that they want to manage solely with BigDataStack. We refer to this as the '*dedicated cluster*' scenario. Under this scenario, BigDataStack would be deployed on and would manage the entire cluster.

The second case is where the user wants to requisition a portion of a shared cluster, where the unit of requisition is a single machine (physical or virtual). This might represent a case where users do not own their own big data infrastructure and are looking to other companies to provide that for them. Or they are part of a large company with a single cluster that is shared amongst multiple teams. We refer to this as the '*opaque cluster*' scenario. In this scenario, BigDataStack would be first deployed on top of a small portion of the external

[1] Sometimes there may be an additional virtualization layer, where the container orchestration platform itself is deployed on a virtual machine. This may occur when hardware is shared with other orchestration platforms or other non-containerized applications.

opaque cluster. New machines may then be requisitioned from the opaque cluster on-demand, possibly for an additional fee. An example of this scenario is a case were a user wants to make use of BigDataStack's proposed Data-driven Infrastructure Management capabilities over a public cloud like Amazon Web Services or Microsoft Azure.

The third case is where the user wishes to use an external provider's orchestration platform for deploying their containers (e.g. Amazon EKS). This might occur in scenarios where a user wants to leverage optimisations or additional services provided by an external provider's orchestration platform (e.g. native cross-availability zone support in Amazon Elastic Kubernetes Service or EKS) but would still like to make use of BigDataStack's data services and/or dimensioning and modelling capabilities. We refer to this as the 'opaque orchestrated cluster' scenario.

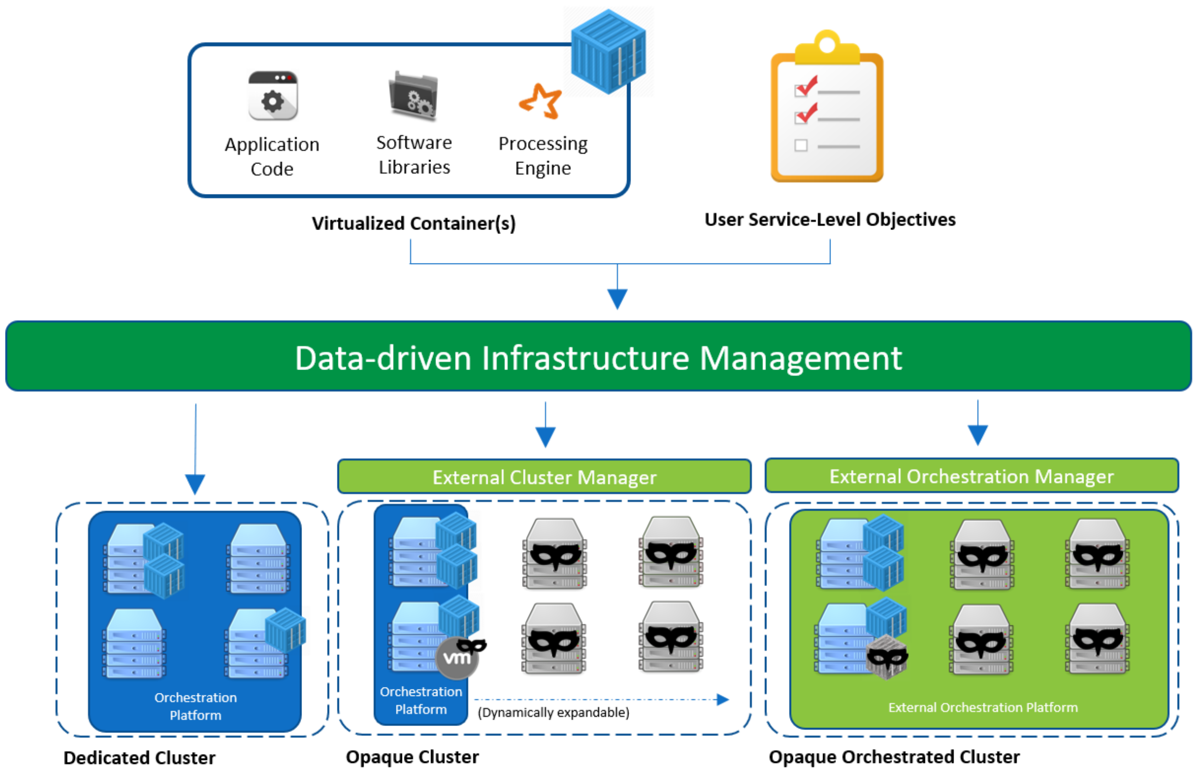


Figure 3 – Cluster Management Scenarios

We can see these three scenarios in Figure 3. The key difference between these scenarios is who controls the way that the virtualized containers are allocated to physical hardware. In the dedicated cluster scenario, that process would be controlled directly by BigDataStack (as part of step 1 of the operations phase). In the opaque cluster scenario, BigDataStack still controls the allocation of virtualized containers to physical hardware, but that hardware may be being shared with other processes (e.g. we ask for 4 CPU cores and the external cluster manager allocates us half an 8 CPU core physical machine). Finally, in an opaque orchestrated cluster, allocation to physical hardware is controlled by an external orchestration manager (BigDataStack has no control of the physical hardware).

It is important to note this distinction, as which scenarios BigDataStack targets will impact on component design. If BigDataStack targets the opaque cluster and/or the opaque orchestrated cluster scenarios, it will be usable by a wider audience, as users do not need to own or rent a dedicated cloud. On the other hand, reliance on external management software may result in containers and/or services from different user applications sharing a physical hardware, making it more difficult to estimate with confidence if Service-Level Objectives (SLOs) will be met in advance (as hardware sharing would be unpredictable and invisible to BigDataStack). Meanwhile, targeting only a dedicated cloud means that BigDataStack would have the power to avoid hardware sharing, or at least account for it, when checking SLOs (as all deployed services would be visible to BigDataStack). However, this markedly reduces the potential user-base for BigDataStack, and even if hardware sharing can be identified as a root cause for an SLO failure (which may not be possible), it is unclear what actions BigDataStack could take other than move the failing service to another machine.

For these reasons, subsequent design for Data-driven Infrastructure Management assumes that BigDataStack will support both dedicated and opaque cluster scenarios. Note that as a result, some BigDataStack functionalities may be unavailable in opaque cluster deployments (in cases where those functionalities require dedicated control of underlying hardware to be effective).

3.2.3.Application Lifecycle Support

It is important to note we envision a The Data-Driven Infrastructure Management capability (DDIM) which plays a role in two clearly differentiated stages of the software application lifecycle: (first) deployment and operations.

- <u>Deployment:</u> In the first deployment, DDIM carries out an optimal deployment configuration for the application to satisfy its Quality of Service (QoS) constraints. The decision-making process is based on the previous experience with the deployment of that application (from both benchmarking and production executions) as well as similar applications.
- <u>Operations:</u> At runtime, the DDIM continuously monitors the application as well as the underlying infrastructure to evaluate its QoS. In case it is not satisfied, the DDIM triggers a dynamic adaptation process to reconfigure the application deployment. This decision-making process is based not only on previous experience in application deployments but more importantly the experience gathered during the current execution.

3.3. Platform Roles

The following table lists the BigDataStack roles that will be interacting with the Data-driven Infrastructure Management (see the complete list of roles in Deliverable D2.2).

Id	Name	Description
ROL-01	Data Owner	They would need to move both in-motion (streaming) and at-rest data into BigDataStack data stores layer, which support both SQL and NoSQL data stores.
ROL-02	Data Scientist	They would need to deploy and operate their analytics tasks by utilizing a declarative paradigm, which includes preferences regarding the data services (such as data sores or processing) to be used, and the Quality of Service (QoS) constraints to be applied on the analytics tasks.

ROL-04	Application Engineers	They would need to experiment with different deployments for their data applications, which include analytics tasks along with other application services. They would need to benchmark the data application to come up with the optimal deployment configuration to satisfy QoS constraints.
ROL-05	Data Engineers	They would need to deploy and operate data services (such as data storage, data processing or data visualization) on the BigDataStack platform, in a way that let them satisfy the QoS constraints requested by the consuming analytics tasks.

Table 1 – BigDataStack Platform roles

3.4. Example Scenario

In this section, we provide a detailed example scenario of a data-driven application deployment and operation to illustrate how Data-driven Infrastructure Management is envisaged to function and how the application engineers and data scientists benefits from its functionalities.

An example data-driven application

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

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

Operational environment (scenario assumptions)

To make the example more concrete we consider the following assumptions:

a) The BigDataStack infrastructure is deployed on an opaque cloud provided by a public vendor, where compute resources can be requested on demand.
b) The cloud environment is relatively stable in term of performance and, for the sake of simplicity, we have a unique size of allocable server.

c) The time needed to complete the compute task is a distribution that has been learnt (e.g., a normal distribution) where the average is some (linear) function of the input data size (which may change every day) and a (decreasing) function of the number of servers allocated.
d) The distributed storage possesses an adaptation mechanism which is independent from those of the Data-Driven Infrastructure Management, which means it can be scaled in/out independently, considering decisions based on its internal metrics and handle on its own the reconfiguration of the internal data regions (see Section 5.1, REQ-CM-04).
e) The benchmarking phase (see D5.1), give hints (estimates) on the expected computation time depending on different variables, including the opaque cloud configuration, the application deployment configuration and even the day of the week/month, but with some non-negligible uncertainty.
f) The historical big data is stored in a secure datastore managed by BigDataStack, including the specification of what data needs to be processed by what service (e.g., the price modelling service need the last twelve months' sales information).
g) The Data-driven Infrastructure Management (DDIM) decision-making includes a policy to minimize the overall cost of the underlying public (opaque) cloud; this would let the system to allocate no more resources than necessary to deploy and operate applications. Therefore, application engineers do not need to explicitly ask for a minimization of cost (or specify cost as a Service-Level Objective (SLO) as this objective will be embedded in the DDIM decision-making process.
h) The Service-Level Objectives (SLOs) violations do not penalize economically the Data-driven Infrastructure Management function in favour of those setting SLOs, that is, the application engineers and data scientists. This means there is no trade-off to make between public cloud cost versus penalties. Therefore, we avoid the scenario where the DDIM decision-making compromise between the cost (in terms of public cloud) of a re-deployment and the cost of keeping on violating an SLO. This is indeed a simplification of real-world scenarios, but it is necessary to keep this example scenario concise enough as to make it an effective example.

Before deployment (and prior to Data-Driven Infrastructure Management)

The application engineer uploads the application to the BigDataStack platform and specifies the main workflow of their application via the Process Modelling Toolkit. This implies that there are two phases: Phase 1) the price modelling service runs, and Phase 2) the price application service and store-front update service run concurrently.

As part of the service, it is agreed a Service Level Objective (SLO) of *end-to-end completion time* < 7 hours–to be accomplished between 9pm and 4am, as they plan to schedule the process to start at 9pm every day. Finally, preferences they may have regarding the configuration of the application deployment are also specified, for example, the container images location and basic resources allotted to them (e.g., virtual CPUs and memory). All this information is compiled in a so-called Playbook.

The Playbook is passed to the Application Dimensioning Workbench, where it is converted into multiple CDP (Candidate Deployment Pattern) Playbooks, each one describing a potential deployment configuration (what compute and memory resources to request). Each of these configurations will undergo a brief benchmarking step, where the resource usage of the application is estimated (this involves using a small sample of data to test the system). For the purposes of this example, we assume that a per-service virtual CPU usage, memory usage, data input/output bandwidth and time-to-complete is estimated. It is the resultant set of CDP Playbooks with benchmarking information that is passed to Data-driven Infrastructure Management to optimize its decision-making models.

Application deployment (initial)

The first function of the Data-driven Infrastructure Management is resource estimation. As discussed later in Section 7, the ADS Ranking component takes the set of CDP Playbooks resulting from the benchmarking activities and selects the most suitable one by comparing the benchmarking estimates with the user SLOs and application deployment preferences. In our example, for each CDP Playbook, this process involves analysing each of the two application phases to determine what resources they need to run efficiently. This means using the benchmarking data in conjunction with historical information from past similar deployments to model factors such as expected virtual CPU and memory usage per-service (under average and peak conditions), affinity for scale-out vs. scale up, and monetary cost (machine cost of the cloud provider in this example). This way, each CDP Playbook is assigned a score, indicating how well it is predicted to perform. ADS Ranking then filters out any CDP Playbooks that are predicted to fail the user's SLOs (i.e. the aggregate time for both phases exceeds the completion time limit set by the user) and selects the highest scoring CDP Playbook of those remaining.

The next step is for the Deployment service (see Section 7) to take the selected CDP Playbook and requests the resources specified within. This petition is fulfilled (the deployment operations enacted) by the underlying Cluster Management service (see Section 5), which in this case will map the requested resources into a series of machine allocation demands that are sent to the public cloud provider for the application services. After this, data-driven networking is configured, enabling the secure information transfer between the machine holding the object store and the machines needing to perform the compute, as well as hooking in monitoring functionality to facilitate data transfer optimisation.

Application operations (re-deployment)

Once the application has been successfully deployed, the Dynamic Orchestrator (see Section 6) enters scene as responsible for managing run-time adaptation of the application deployment and networking configurations. The first action that the Dynamic Orchestrator performs is to request the Triple Monitoring and QoS Evaluation (see Section 8) of the new deployment. As a result, metrics regarding the completion time of the application services start to be collected as well as evaluated against the SLO originally set by the application engineer: *end-to-end completion time* < 7 hours.

After a certain period, the Dynamic Orchestrator observes that the estimated completion time (measured by the rate at which 'data row processed' events are received) for Phase 1 has changed from 4 hours to 6 hours. As a result, end-to-end completion time is now estimated at 8 hours (an estimated completion time of 5am), constituting an SLO failure state.

Thus, the Dynamic Orchestrator will try to fix this automatically via run-time adaptation within the platform. The exact operation that the Dynamic Orchestrator will employ depends on the log data from the running application and to what extent the application supports run-time changes. For this example, we will assume that the log data indicates that the compute nodes are maximising CPU usage, and hence we might fix the issue by increasing the amount of compute resources available. Furthermore, we assume that the application is run-time scalable, i.e. it supports changing at run-time the degree of parallelism (e.g., an Apache Flink or Apache Storm application). As such the Dynamic Orchestrator will send a request to the ADS Ranking component requesting service re-deployment, where both the target completion time and CPU capacity are updated to reflect the state of the running service.

The ADS Ranking will then re-score all of the CDP Playbooks that it has available for the user's application in light of these new requirements. The difference with the previous deployment is that by including a target CPU capacity, CDP Playbooks with increased CPU resources will be favoured. Moreover, by re-estimating the completion time and matching against the new target completion time for Phase 1, un-suitable configurations can be discarded. Note that this may also result in changes in the way that the Phase 2 services are deployed (to increase speed of Phase 2 processing in order to save total time). At the end of this re-scoring, a new CDP Playbook will be selected.

This new CDP Playbook will then be sent to ADS Deployment, which will compare the current deployment against the new deployment and operationalize the needed changes. These changes might involve requesting another machine to increase parallelism of the service, which would be allocated by the Cluster Management component and would be subject to the same data and network configuration as during the initial deployment (e.g. to assure that the needed communication ports are open). This might involve physically starting a new copy of the target service on the additional machine (if the application is self-configuring). Otherwise, if the application integrates a process management platform like Apache Flink, then this would involve starting a management daemon on the new machine, effectively adding the machine to the 'Flink Cluster' for the application, which will seamlessly scaling computation up to use the newly available resources.

Concluding remarks

To conclude this example, let's assume that this deployment correction increases throughput of Phase 1 sufficiently to catch up to the target completion time set by the user; if it did not, then other run-time adaptations may be attempted and/or the user informed.

3.5. Design

The conceptual view of the Data-driven Infrastructure Management capability shows the main high-level functions as well as the data flows among them (see Figure 4).

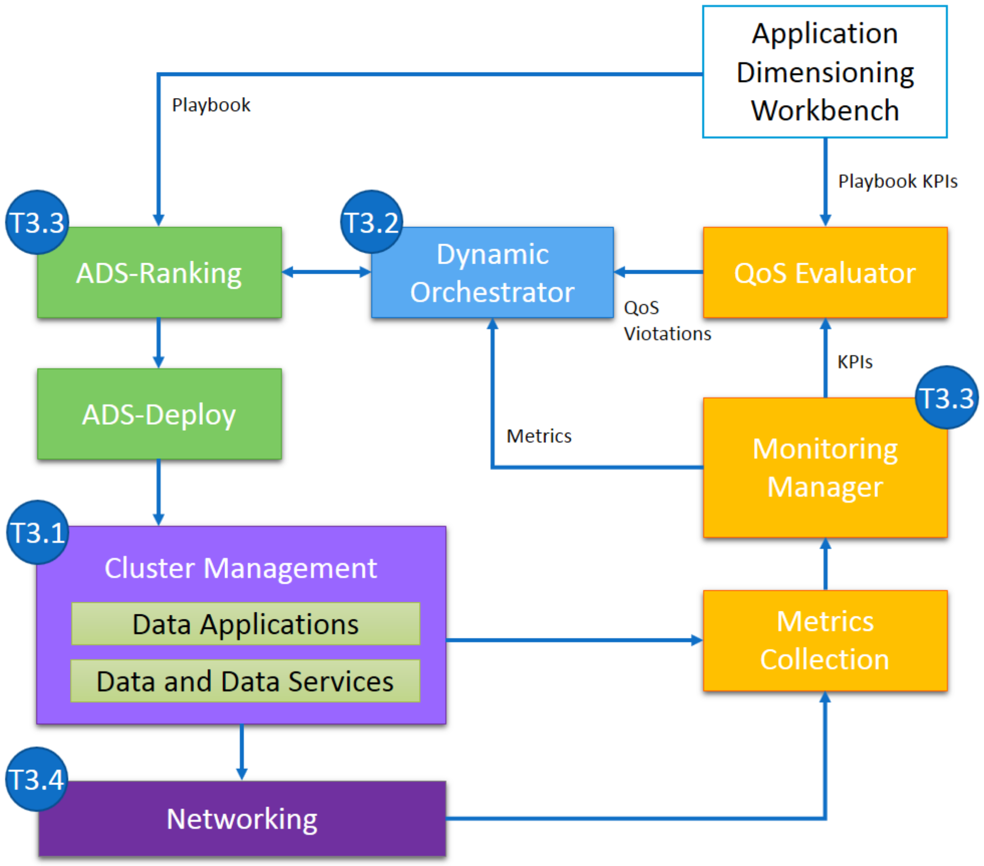


Figure 4 – Data-driven Infrastructure Management capability - conceptual view.

These functions are organized in (realized by) five solution building blocks (components), which corresponds to the five tasks within WP3:

1. Cluster Management (WP3-T3.1): Infrastructure services providing cluster computing and data storage resources and is responsible for deploying the BigDataStack platform and associated container orchestration platform on physical hardware. For opaque clusters, this also involves dynamic scaling of the orchestrated hardware on-demand. The component also provides an API to support (re-)configuration actions or resources requests by the ADS Deployment. This provides functionality for BigDataStack operations – Steps 1 and 2 (see Section 3.1).
2. Dynamic Orchestration (WP3-T3.2): Runtime adaptation service in charge of resource re-allocation, storage and analytics re-distribution, re-compilation of network functions and re-deployment or applications and data services. This provides functionality for BigDataStack operations – Steps 5 and 7 (see Section 3.1).
3. ADS Ranking & ADS Deployment (WP3-T3.3). Self-optimized deployment service for application components and data services, which are orchestrated following resource, application and data-aware deployment patterns. This provides functionality for

BigDataStack operations – Steps 4, 5 and 7 (see Section 3.1).

4. Triple Monitoring and QoS Evaluation (WP3-T3.5). It consists of the resource clusters, data and application-level metrics collectors, the monitoring manager (which also gathers database related metrics) and the QoS evaluator, which evaluates Service-Level Objectives (SLOs) over those metrics. This provides functionality for BigDataStack operations – Step 6 (see Section 3.1).
5. Networking (WP3-T3.4). Data-driven networking services satisfy the diverse networking needs among computing and storage resources as well as application components and data services. It plays a critical role in the optimal (self-) management of the infrastructure to satisfy QoS. It is important to note that this component not only collects networking metrics, but it also applies the (re-) configuration actions over networking resources requested by the ADS Deployment. It provides functionality for the BigDataStack operations – Step 3 and 7 (see Section 3.1).

Figure 5 describes the capability from a lower-level logical perspective, including the main components, their interfaces and dependencies. Dependencies from/to external components are also shown; specifically, the need to have access to the Decision Tracker and to provide service to the Application Dimensioning Workbench.

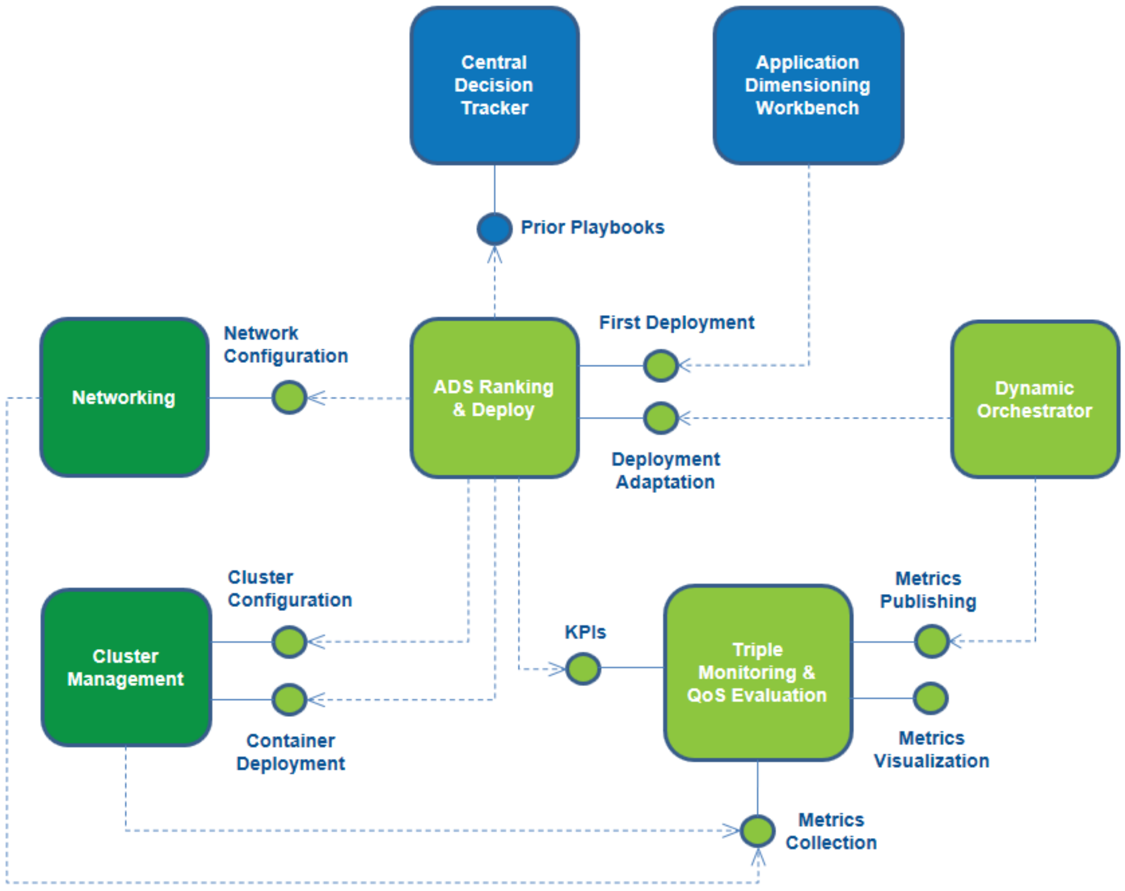


Figure 5 – Data-driven Infrastructure Management capability - components view.

Figure 6 shows the sequence of activities and the interplay between the different components to give support to the BigDataStack seven-step process (operations time).

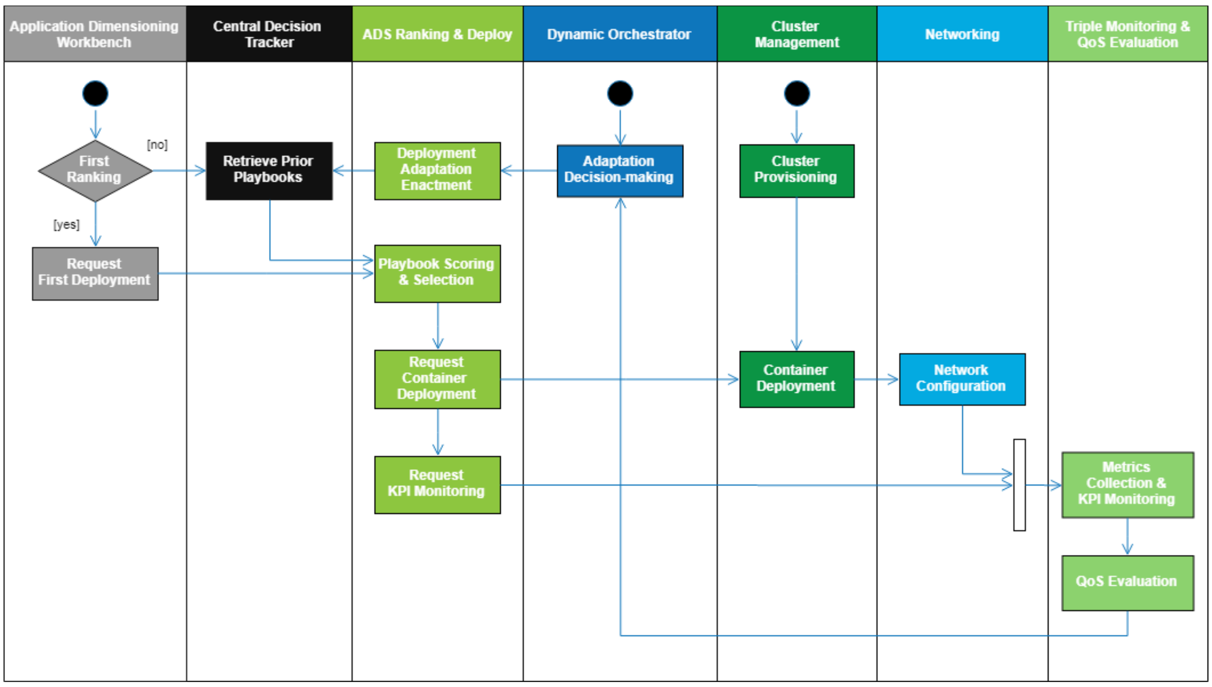


Figure 6 – Data-driven Infrastructure Management capability - activity view.

4. Implementation and Experimentation

This section introduces the experimental scenarios and the methodological approach WP3 is taking to answer important questions and validate certain hypothesis to develop the Data-Driven Infrastructure Management capability. The section explains the three prototypes (to be released at M12, M15 and M18) which are meant to address three different scenarios. They will let WP3 to verify and validate the solution before facing the full integration with WP4 and WP5 components and the full deployment of the use cases implementations (WP6).

4.1. Experimental Setting

The BigDataStack use case we have chosen to evaluate and validate the Data-Driven Infrastructure Management prototypes in the upcoming 7 months is the so-called *Connected Consumer (CC): Multi-sided market ecosystem*, provided by ATOS WORLDLINE-EROSKI[2]. Some of the highlights of the use case are (please refer to D2.1 for the full description):

- EROSKI, one of the largest distribution companies in Spain with more than 35.000 workers, is collaborating with ATOS in the definition and test of a use-case related to the grocery business.
- EROSKI needs data insights to better understand how to create and offer added-value services to their consumers.
- The use case objective is to predict both which products and which promotions are more likely to be interesting for the customers at the right time. In this way, EROSKI can adapt the most appropriate message (i.e. product and/or promotion) for each customer and send it at the right time and through the most appropriate channel, thus increasing the ROI of their marketing activities.

4.1.1. Scenario 1: Inference without Data Access (M12)

An application engineer wants to deploy a recommendation model implemented by a data scientist. This recommendation system will provide product recommendations for customers visiting the EROSKI's e-commerce web site. Customer events in such a site will continuously feed the system to improve the recommendation model.

- The analytics application is made of two services (see Figure 7):
  - Normalization, which receives customer events and updates the Customer Preferences table with the customer activity. This table is then used as input in the Inference process.
  - Inference, takes the up-to-date Customer Preferences table and compute Product Recommendations table, which contains the list of products recommended per user.
- These application services contain state (i.e. Customer Preferences and Product Recommendations tables). Therefore, they cannot scale horizontally unless we provide a persistent storage. In M12 we won't integrate with a datastore, so we flush

2 https://www.eroski.es/

the data to an already made, in-memory, distributed cache, so that the application services can become stateless and therefore horizontally scalable.

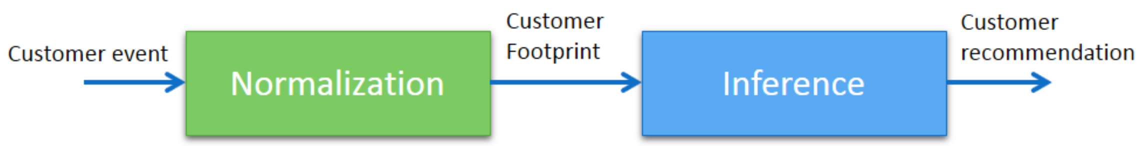


Figure 7 – Experimental scenario 1: Inference without data access (M12) - data flow view.

<u>Requirements and constrains:</u>

- A cache (in-memory) service is required to be deployed alongside the Normalization and Inference application services to store Customer Preferences and Product Recommendations tables and hence let them scale out (horizontally).
- The Inference is based on cross-selling by "collaborative filtering." The algorithm used will be one of those already implemented in the NumPy library for Spark.
- Different experiments on the performance of the recommendation system will be accomplished, including the evaluation of latency for the Normalization service and throughput for the Inference service.
- Different experiments executing the Inference process in batches of different sizes.
- The Inference will be executed on a Spark engine, which will be bundled and deployed together with the recommendation algorithm in a single container (stand-alone deployment). The single-node Spark configuration seek to serve as a first step to deploy Spark operations: In scenarios 2 and 3 the configuration will pass to be a more realistic multi-node cluster.

<u>Customer events</u>

The analytics application which is the subject of the scenarios is meant to provide service to EROSKI's e-commerce web site, specifically, product recommendations to customers. The analytics application service computes recommendations and the web application uses those recommendations to decide which products to show to the customer visiting the web; for each product it gives the option to view the detail of the product, add the product to the car, or discard that product so that it is not shown again as a recommendation to that customer.

Al these customer actions are captured as events and notified to the Normalization service which registers them in the Customer Preferences table, which in turn serves as input to the Inference service to update the Product Recommendations table. The definition of those events is the following:

- <u>Recommendation shown</u> (attributes: customer id, id recommendation, list of product ids). It will be used to discard a recommended product if the client has not shown interest in it (has not displayed it and has not added it to the car) after being shown as a recommendation a certain (configurable) number of times.

- Product added to the cart (attributes: id client, id recommendation, id product). We will give more weight to the recommendation of this product for this client.
- Product displayed (attributes: id client, id recommendation, id product). We will give more weight to the recommendation of this product for this client (but less than if you add it to the car).
- Product discarded (attributes: id client, id recommendation, id product). Directly this product will be eliminated from the list of product recommendations for the given customers.

Deployment

Both services are expected to the containerized and deployed on Kubernetes as a single pod. This means that the scaling of the services will the carried out together, that is, increasing or decreasing the number of replicas at the pod level and not at the container level (i.e. scaling in and out).

The other action that can be carried out to dynamically adapt the deployment is to change the number of vCPUs per container (i.e. scaling up and down).

Quality of service

In different settings, the data scientist will need both processes to run with varying constraints of response time. Moreover, the throughout will be also an important consideration for the application engineer.

Other application-specific metrics (e.g., precision of the prediction, the success rate of the product recommendation) are not considered in this scenario.

4.1.2. Scenario 2: Inference with Data Access (M15)

Scenario 1 at M12 is enhanced by considering the persistence of both Customer Preferences and Product Recommendations tables in a data store, LeanXcale database.

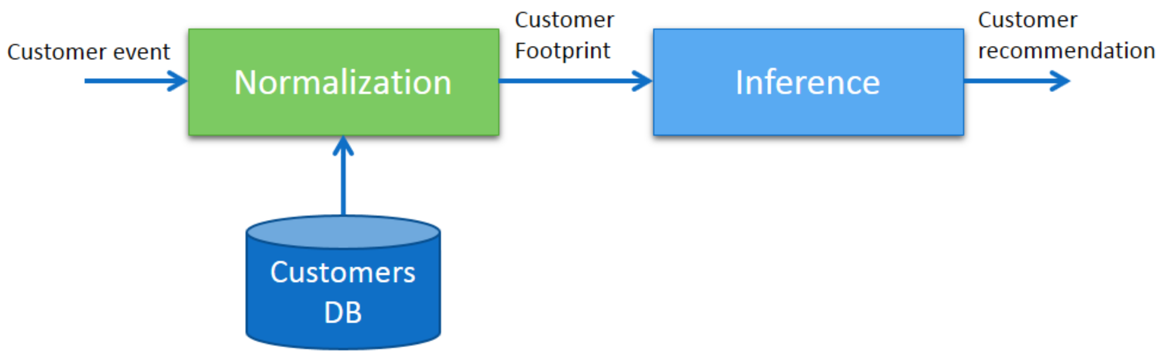


Figure 8 – Experimental scenario 2: Inference with data access (M15) - data flow view.

Requirements and constrains (refine scenario 1):

- A cache (in-memory) service is required to be deployed alongside the Normalization and Inference application services to store Customer Preferences and Product Recommendations tables and hence let them scale out (horizontally).

- The cache (in-memory) service permanently store Customer Preferences and Product Recommendations tables in a LeanXcale database every time there is write operation.
- The Inference is based on "customer habits" by "individual behavioural analytics."
    - Instead of producing the whole table for all the customers in every run, the inference process updates just the product recommendations for the customer/s whole events are received in a given time window.
    - The algorithm used will be one of those already implemented in the NumPy library for Spark.
- The Inference will be executed on a multi-node Spark cluster, so there is a need to come up with its optimal deployment (e.g., number of nodes, flavour of VMs, etc.).
- Different experiments executing the Inference as streaming analytics in micro-batches and real-time (i.e., with the arrival of every single event) will be accomplished.

Deployment

The application components are deployed in Kubernetes in the same way as in Scenario 1. For this scenario, the LeanXcale data base is expected to be deployed and operated as a WP4 prototype. This means it deployment is not part of this scenario, which focuses on the integration between W3 and WP4 regarding the storage layer and the impact on the analytics application layer.

Quality of service

Like in the previous scenario, different experimental settings with different QoS targeting low response time and high throughput will be run.

At least one application-specific metric (e.g., precision of the prediction or the success rate of the product recommendation) will be considered in this scenario.

4.1.3. Scenario 3: Integration with WP5 and WP4

Scenario 2 at M15 is enhanced by considering the integration with components and services from WP4 and WP5. For example:

- As the data cleansing pre-processing is a service from WP4, a process to clean and enrich the events before being submitted to the Normalization will be integrated. Note this process will be implemented through the Real-time Complex Event Processing (CEP) resulting from T4.6 (see D4.1).
- As the application process modelling tool is a component from WP5, the Normalization + Inference process will be declaratively modelled in the tool and the automated deployment from that tool validated. Note this tool will be implemented over Node-RED at T5.2 (see D5.1).

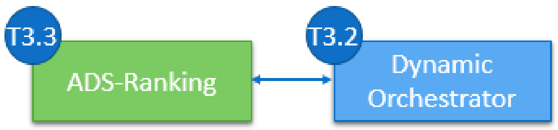
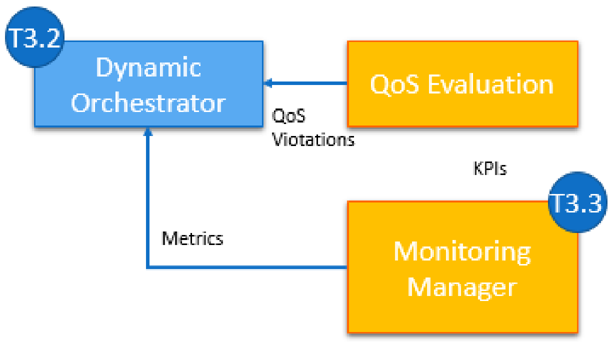
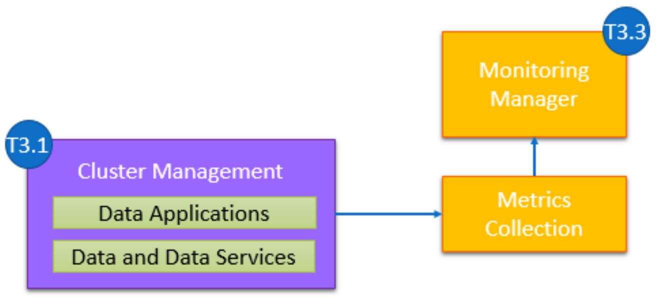
The final decision about the set of integrations to be tested at M18 will need to wait until M15, when WP3 prototypes and scenarios 1 and 2 are validated, and WP4 and WP5 have finalized to validate their corresponding prototypes and scenarios. The idea is to select those individually-validated WP3, WP4 and/or WP5 components and features to be evaluated and validated in an integrated end-to-end BigDataStack use case scenario at M18.

4.2. Experimental Plan

This section explains the experimental design, including success criteria (KPIs to evaluate or hypothesis to validate) at the global (capability) level. It also describes the methodology.

4.2.1. Research questions

The following questions to be addressed in the experimental scenarios have been formulated in terms of the Data-Driven Infrastructure Management's conceptual model (see Figure 4).

A	1. How the dynamic orchestrator and ranking & (re-) deployment components share responsibilities? 2. How do they work together to provide a coherent and optimal runtime adaptation behaviour?	
B	3. How is the interplay between Dynamic orchestrator and QoS Evaluation? 4. What information does the former need for the latter? When and how (through which mechanism)? 5. Does it need violation events or violation metrics (e.g., number of violations in a given period)?	 T3.2 Dynamic Orchestrator QoS Evaluation QoS Violations KPIs T3.3 Metrics Monitoring Manager
C	1. What application-level metrics do we need to monitor and how? 2. Can we prepare the monitoring system to collect any application-level metrics related to QoS attributes of interest to engineers and data scientists?	 T3.3 Monitoring Manager T3.1 Cluster Management Data Applications Data and Data Services Metrics Collection

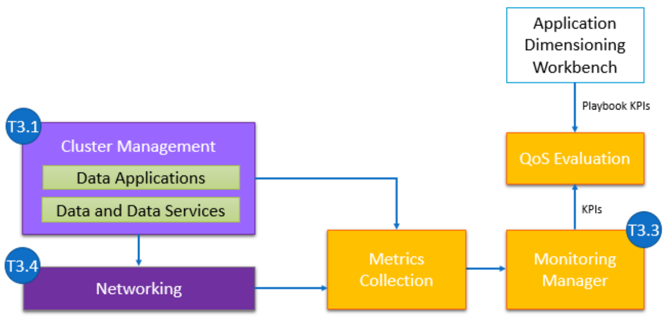
D	1. How does the QoS evaluation component get to know what QoS attributes or KPIs to evaluate? 2. How does it determine the relationship between them and the metrics collected by the Monitoring System? 3. Do we need to have a predefined catalogue of QoS attributes or KPI specifications to be shared among all actors (sort of QoS ontology)?	 Application Dimensioning Workbench Playbook KPIs T3.1 Cluster Management Data Applications Data and Data Services QoS Evaluation KPIs T3.3 T3.4 Networking Metrics Collection Monitoring Manager

Table 2 – Research questions.

4.2.2. Research method

We will be developing prototypes in the upcoming seven months to validate certain hypothesis on answers to the above-mentioned questions. These prototypes will be delivered at M12, M15 and M18, each of them representing milestones in the WP3 research and innovation plan.

The (experimental) evaluation scenarios have been extracted from the following BigDataStack use case: *Connected Consumer (CC): Multi-sided market ecosystem*. Along with the use case provider ATOS WORLDLINE-EROSKI, we have identified realistic requirements and constraints to deploy different functionality of their application by using the different prototypes of the components of the Data-driven Infrastructure Management solution.

The three different scenarios (see section 4.1) focus on different aspects of the solution to answer different questions (see section 4.2.1). Nevertheless, at M18 it is expected we showcase the interplay between all BigDataStack capabilities within one or more integrated end-to-end scenarios from the BigDataStack use cases.

4.3. Implementation Roadmap

Table 3 summarizes the experimentation (evaluation and validation) plan for the Data-driven Infrastructure Management capability for the upcoming seven months:

	M8	M15	M18
Milestone	Prototyping	Validation	Implementation
Objective	The consortium uses the UBI-provided OpenStack-based	The consortium starts deploying services in the (cloud native)	ALL WPs are obligated to use the (cloud native) WP3-provided

	computing infrastructure to deploy and run virtual machines	WP3-provided Kubernetes-based computing infrastructure to deploy and run containers by WP3	Kubernetes-based computing infrastructure, when technically possible.
Success criteria	The different partners deploy and test BigDataStack services directly on virtual machines. Ideally, using containers as unit of deployment.	ALL WP3 services are deployed and running on Kubernetes to test the platform.	N/A
	WP3 tests its tools for cluster management (Openshift), monitoring and dynamic adaptation	Partners can deploy their BigDataStack services on Kubernetes. At least, two-three components of WP4/WP5 are deployed.	N/A
		Prototypes at M12 and M15 successfully pass experimentation scenarios 1 and 2.	Prototypes at M18 and M15 successfully passes experimentation scenario 3.

Table 3 - Data-driven Infrastructure Management capability experimentation phases.

Table 4 summarizes the Data-driven Infrastructure Management capability implementation roadmap for the upcoming seven months:

	M12	**M15**	**M18**
Scenario	1	2	3
Cluster Management	OpenStack integration Gateway	OpenStack integration Operators Gateway	OpenStack integration Cluster performance improvements Operators Gateway
Dynamic Orchestrator	Agent Interpreter	Agent Interpreter	Agent Interpreter
Ranking & Deployment	ADS-Ranking ADS-Deploy	ADS-Ranking ADS-Deploy	ADS-Ranking ADS-Deploy
Triple Monitoring & QoS Evaluation	SLALite Prometheus Graphana Cluster metrics	SLALite Prometheus Graphana Cluster metrics	SLALite Prometheus Graphana Cluster metrics

	Data metrics	Data metrics Networking metrics Application metrics	Data metrics Networking metrics Application metrics
Information-driven Networking	N/A	Native Kubernetes Networking & Policies Enforcement Calico Istio	Native Kubernetes Networking & Policies Enforcement Calico Istio

Table 4 - Data-driven Infrastructure Management capability implementation plan.

5. Cluster Management

The cluster management component is in charge of both deploying the BigDataStack components as requested, as well as to keep its status overtime. This will not only include the containers but the related services and even the OpenShift Origen Kubernetes Distribution (OKD) cluster itself. In addition, it is in charge to adapt the current deployments to the new preferred status requested by the above layers, for example to increase the size of the cluster, or scale up/down a given application.

5.1. Requirements specification

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.2 and literally included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.2, and that they are included in here for the reader's convenience.

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-CM-01	System	FUNC	Developer	MAN
Name	Support OpenShift installation on OpenStack VMs				
Description	Include the needed steps on the OpenShift installer to handle OpenShift cluster installation on top of OpenStack resources, i.e, VMs, networks, volumes, etc.				
Additional Information	This needs to be done in the 'upstream' way so that it is supported also after the project lifecycle. It entails modification to different repositories, not only the openshift/installer (https://github.com/openshift/installer) but also other related such as: • cluster-network-operator[3] • cluster-api-provider-openstack[4] • gophercloud[5]				

Table 5 - Support OpenShift installation on OpenStack VMs (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-CM-02	System	PERF	Developer	MAN
Name	Avoid double encapsulation of network packages				
Description	Integrate Kuryr on the OpenShift installer to avoid the double encapsulation problem due to using 2 different overlays (OpenStack SDN and OpenShift SDN on top). Kuryr enables containers running on top of OpenStack VMs to use the same SDN as the VMs itself, i.e., the OpenStack SDN. Thus, avoiding				

[3] https://github.com/openshift/cluster-network-operator
[4] https://github.com/kubernetes-sigs/cluster-api-provider-openstack
[5] https://github.com/gophercloud/gophercloud

	the double encapsulation and enabling a remarkable throughput gain, needed for handling the data at the BigDataStack components.
Additional Information	Similarly, to REQ-CM-01, this needs to be done in the 'upstream' way so that it is supported after the project. It entails modifications to the same repositories plus the addition of a kuryr operator that will handle the kuryr related operational actions,

Table 6 - Avoid double encapsulation of network packages (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-CM-03	System	ENV	Developer	DES
Name	Spark operator				
Description	This operator will be responsible for handling the spark cluster, not only its installation but also the scaling actions. It will offer an API to the spark management through the OpenShift API.				
Additional Information	This is related to the dynamic orchestrator, as the optimization actions could be then simply triggered through standard OpenShift API commands (e.g., modifying the information at the associated spark ConfigMap)				

Table 7 - Spark Operator (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-CM-04	System	ENV	Developer	DES
Name	Accept requests to allocate additional resources to one of the storage layer components				
Description	The Adaptable Distributed Storage component can be scaled in/out independently, considering decisions based on its internal metrics and handle on its own the reconfiguration of the internal data regions. Due to this, it is necessary from the Cluster Management to provide a mechanism that allows the storage layer to request for additional resources or the release of already provided ones.				
Additional Information	This is closely related to requirement REQ-ADS-04 "Be able to request additional resources from the infrastructure layer," described in D4.1.				

Table 8 - Accept requests to allocate additional resources to the storage layer (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-CM-05	System	ENV	Developer	OPT
Name	Force the storage layer to release some of its available resources				
Description	The cluster management might identify that the overall BigDataStack platform is running out of available resources. To ensure the execution of crucial components, it might decide to reduce some of the already allocated resources for some services, for the benefits of others. Due to this, it should be able to request the release of the storage resources and wait for its				

	proper response. The storage should be able to reject such requests, in cases that could lead to data loss.
Additional Information	This is close related with requirement REQ-ADS-05 "Being able to release resources and adapt if resources are deallocated from the infrastructure," as described in more details in D4.1.

Table 9 - Force the storage layer to release some of its available resources (system requirement).

5.2. Design

To make BigDataStack components widely available and mainly focus on its functionality, we have selected OpenShift as our cluster management engine over which we will build our functionality. OpenShift is based on Kubernetes with extra options for DevOps, as well as for their life-cycle management such as image build automation, deployment automation or Continuous-Integration/Continuous-Delivery (CI/CD). Having Kubernetes at the core, provides all the pods (i.e., group of containers with a single IP) orchestration functionality needed to ensure pods scheduling, replica set management, load balancing, etc.

By using OpenShift at the core of our cluster management, we focus on the following points at the cluster management layer to better support BigDataStack operations:

- OpenStack integration
- Cluster performance improvements
- Operators
- Gateway

Figure 9 show an overview of the existing components and their interactions. The upper layers (such as ADS Ranking & Deploy component, or the Triple Monitoring) will communicate with the cluster management through the OpenShift API. As the figure highlights, this API is extended by creating different operators that expose the functionality of their respective components, enabling actions such as scaling a Spark cluster or extend the OpenShift cluster. Note in the later, that will also entail different subsequent actions that are handled by the operators. For instance, upon an OpenShift cluster scale up, the operator will need to call OpenStack to create the needed resources (in this case VMs and Volumes), and then it will need to configure them: install the required Openshift components, as well as include the monitorization components to account for the new resources.

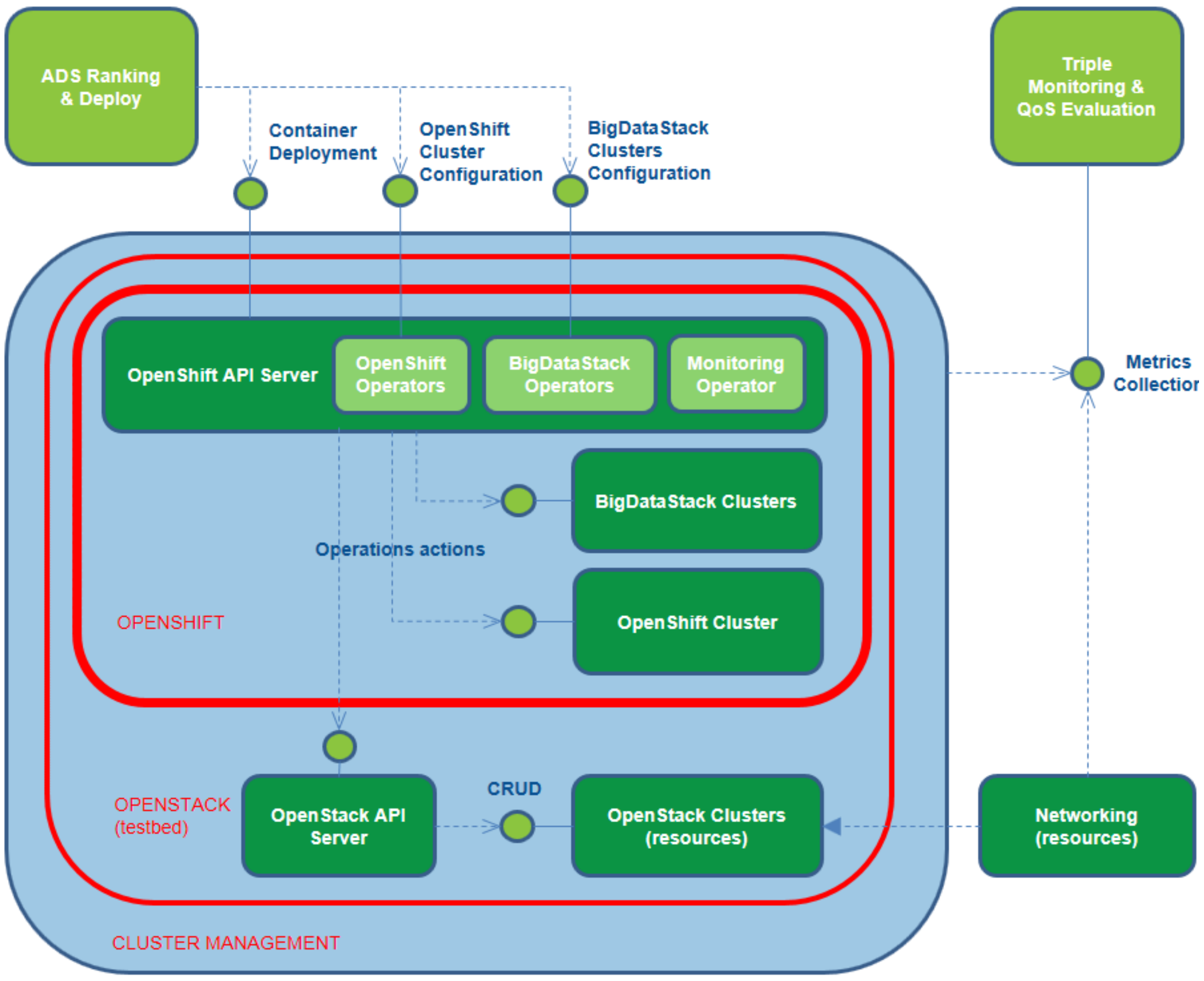


Figure 9 – Cluster Management - components view.

5.2.1. OpenStack integration

OpenShift clusters can be installed on top of different infrastructures, e.g., directly on physical servers or on top of VMs in a private (OpenStack) or public cloud (Amazon). Currently, the OpenShift installer[6] supports the installation and management of clusters on top of physical servers or on top of VMs (on AWS).

As expected, the best performance can be obtained when it is running directly on *bare metal* servers. However, to make BigDataStack functionality to a larger group, and due to the wide use of OpenStack for private clouds, we target the integration of OpenStack into the OpenShift installer as part of the BigDataStack contributions. This means that we need to make the installer able to manage OpenStack resources, such as VMs or Volumes. Not only for the initial installation, but also for the management operations such as cluster scaling up/down or node failures. Note this support will be integrated as part of the installer as well as part of the cluster management operators (see next subsection 5.2.3).

[6] https://github.com/openshift/installer

5.2.2. Cluster performance improvements

Due to the BigDataStack requirements, not only related to fast data processing but also speeding up communications between the different components running on top of OpenShift, there is a need for performance improvements into the network data plane. Simply installing OpenShift/Kubernetes on top of OpenStack VMs means that, on the one hand you have the OpenStack network overlay (to manage the traffic between the VMs), and on the other hand the OpenShift SDN (e.g., openshift-sdn). This leads to the so-called "double encapsulation problem" which impose severe performance degradation on the network throughput (besides the added complexity on network management and debugging upon failures). To avoid this problem Red Hat has been working on an OpenStack project named Kuryr[7] that enables the usage of OpenStack Software-Defined Networks (SDNs) at the OpenShift cluster running on top of the VMs, therefore avoiding the double encapsulation problem. As a result, we plan on also integrating Kuryr on the OpenShift installer as well as creating an operator for its management (see Figure 10).

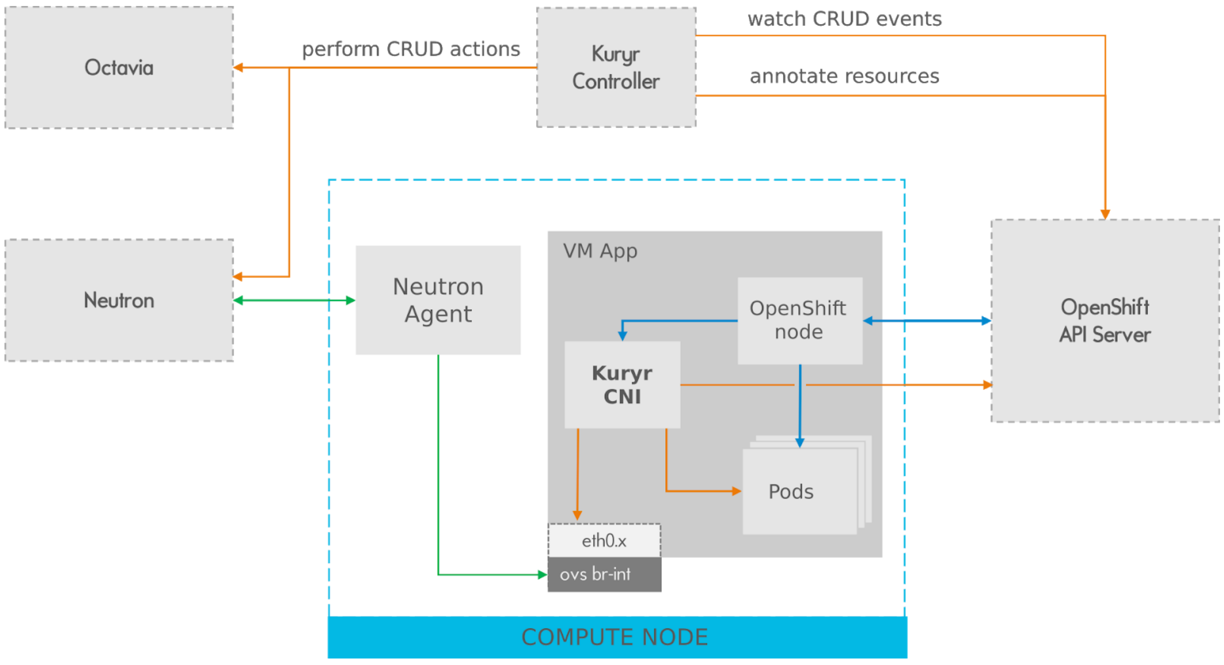


Figure 10 – Red Hat Kuryr's architecture to avoid the "double encapsulation problem."[7]

This however also imposes certain requirement on the OpenStack side. The next components need to be installed and or have specific configuration:

- Octavia (LoadBalancer as a Service) component need to be installed, and with it, its dependencies such as Barbican
- Neutron needs to be configured with Trunk ports support. Depending on the used ml2 driver, the configuration can be slightly different. For instance, it is out of the box if OVN is being used, but if ML2/OVS is being used, it needs to be enabled, and the

[7] https://docs.openstack.org/kuryr-kubernetes/latest/

openvswitch driver needs to be set to enforce security group policies on the containers.

- Depending on the installed, Heat is also needed to create a stack containing all the OpenShift related resources, i.e., VMs, Volumes, Networks, LbaaS, …
- And of course, the user quota needs to be adapted to the container deployments scale, i.e., it will not be enough with just a few ports as each container will be using a neutron port. Thus, some of the resources quota need to be increased by an order of magnitude (depending on the side of the OpenShift deployment)

5.2.3. Operators

Operators are a relatively new concept for packaging, deploying and managing Kubernetes/OpenShift applications. In this context, an OpenShift application is defined as an application (set of containers, configmaps, CRDs, services, etc.) that is both deployed on OpenShift and managed by the OpenShift API.

As an example applied to the BigDataStack, we plan to work on a Spark operator which manages both the installation of the Spark cluster on top of OpenShift, as well as its operations over time. This cluster can then be managed through OpenShift commands, for instance scaling it up or down by simply modifying the associated Custom Resource Definitions (CRDs) where the Spark cluster is defined. Thus, this could be used by the dynamic orchestrator to easily trigger the optimizations on the deployment.

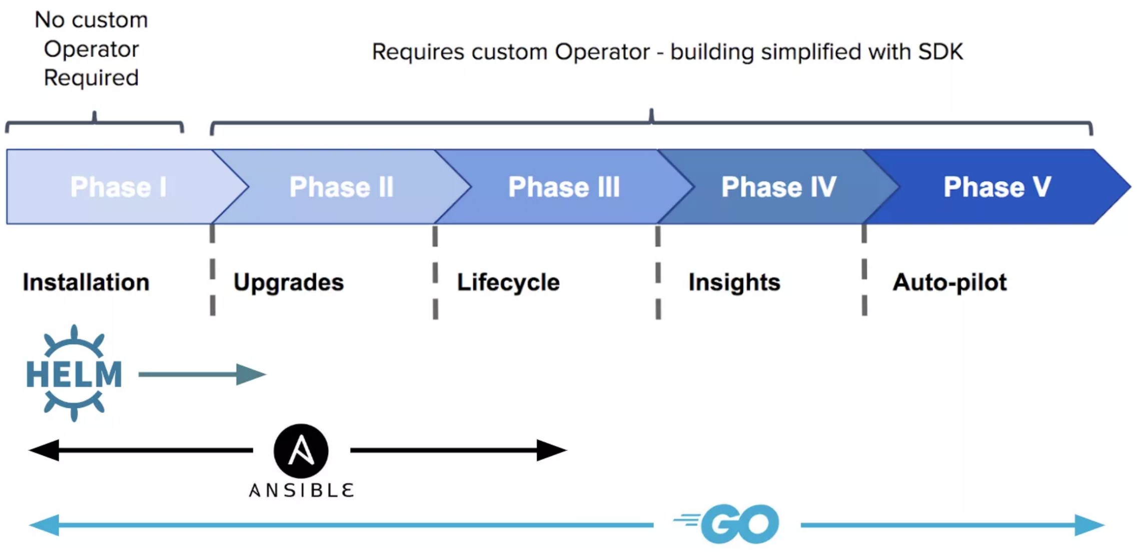


Figure 11 – Kubernetes operators – conceptual view.

It is common to think of Operators as the runtime that manages an application/service on OpenShift. It leads us one step closer to manage the cluster in a declarative way, i.e., it watches over your OpenShift environment and ensures that the state of your cluster or applications conforms with what you requested. More advanced operators are designed for handling applications upgrades seamlessly or even perform complete cluster scaling operations upon resources shortage or failures.

As highlighted on Figure 9, the operators API is offered through the own OpenShift API, therefore having well-defined primitives, and enabling an easy integration with other components. The operators need to take care of the actions that needs to be triggered when the CRD resource is modified, to achieve the new specified status.

5.2.4. Gateway implementation

The gateway for the BigDataStack engine can also be implemented as part of OpenShift, in 2 different ways depending on the final requirements:

- By using OpenShift routes: Route is a way to expose OpenShift services by giving it an externally reachable hostname, like www.example.com. It has the option to perform the routing based on paths, i.e., we can use it to redirect some queries to the CEP component (i.e., www.example.com/cep/…) and others to the Alarm component (i.e., www.example.com/alarms/...)
- By using Istio service mesh: A service mesh is a network of microservices that enables applications and the interactions among them. It offers functionality like load-balancing, fine grain traffic control, access control, logging, tracing, etc., through *sidecards* containers associated to the applications pods. One offered functionality is Istio-Gateways which controls the exposure of services at the edge of the mesh. This could be used to tie gateways to specific virtual services that can perform the extra required actions that the gateway may require besides redirecting the traffic to the desired endpoint.

5.3. Early Prototype

Initial support for OpenStack has been included into the OpenShift installer to handle the creation of OpenStack resources.

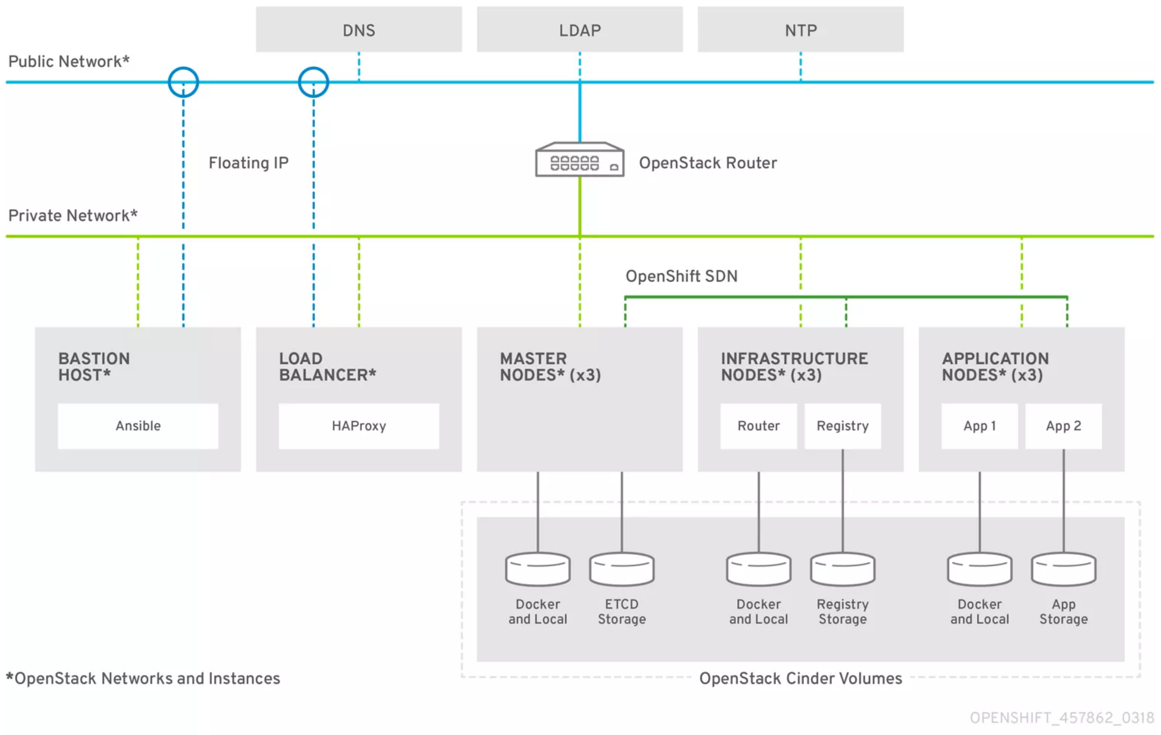


Figure 12 – Best practices for deploying OpenShift on top of OpenStack.

This support extends the OpenShift installer to create OpenStack VMs and later install the packages, configuration files, keys, services, etc., needed to install and configure the OpenShift cluster on top of them. It includes the basic operators and prepares the system for the new ones to be created as part of the BigDataStack project.

Figure 12 shows the best practices (configuration) for deploying OpenShift on top of OpenStack. As it can be seen it includes several OpenStack resources types: Networks, LoadBalancers, VMs, Volumes, etc.

The next table shows the minimum number of each OpenStack resource type that are needed for a minimal installation of OpenShift on top of OpenStack:

OpenStack Resources	Requirements
Virtual Machines	- 1 VM for the Bastion that triggers the installation - 3 VMs for the OpenShift/Kubernetes masters (it can be reduced to 1 if no HA is needed) - 3 VMs for the Infrastructure Nodes (this is needed for routes and registry functionalities) - 3 VMs for the App nodes (where users' applications are run)
Volumes	- 1 Volume per VM (master, infra, app) - No need to have a volume for the master - This is a soft requirement, as the VMs ephemeral disk can be used too, but this may lead to data lost upon VM deletion
LoadBalancers	- 1 needed in front of the master nodes for HA - 1 needed in front of the infra nodes (for defining routes) - 1 will be created for each new OpenShift service
Networks	- Public net/subnet - VMs net/subnet - Pods net/subnet - Services net/subnet - +1 extra net/subnet for each OpenShift project (to provide network isolation between projects)
Floating IPs	- 1 for the LoadBalancer in front of the masters - 1 for the LoadBalancer in front of the infra - 1 for each service of LoadBalancer type
Routers	- 1 for connecting every network
Security groups	- There are a few security groups needed for the installation, plus a few more for the usage. So, recommended limits here is over 100s, e.g.: 500

OpenStack Resources	Requirements
Ports	- One port is needed for each VM, plus one per pod. As before, it could be set to 500us

Table 10 – Cluster management - minimal installation of OpenShift on top of OpenStack.

5.4. Use Case Mapping

N/A, cluster management will be used by all use cases.

5.5. Experimental Plan

The experimental plan will first focus on individual components as well as testing automation. The initial actions points are:

- Automate OpenShift installation testing to ensure no performance regressions.
- Automate Operators testing to ensure their functionality is not broken by new updates/additions.
- Ensure BigDataStack operators are agnostic of the infrastructure. This means they are simply applications running on top of OpenShift and should not be aware of where the OpenShift cluster is running, e.g., on top of bare metal servers, or OpenStack VMs or any Public Cloud (AWS, Azure, …). Note some platform dependent operators, such as kuryr-SDN or other infra related ones, should be aware of the infrastructure as they are the ones in charge of configuring it.
- Initial integration testing of the different components relying on the cluster manager and their operators.

5.6. Next Steps

The plan is to continue with the OpenStack integration into the OpenShift installer, with special focus on Kuryr integration to avoid double encapsulation and therefore creating an OpenShift installation with improved network performance.

In addition, we will focus on Openshift operators development to easy enable the different BigDataStack features on the OpenShift cluster. Among others: Kuryr-SDN operator and Spark operator.

6. Dynamic Orchestration

The Dynamic Orchestrator will provide more flexibility and enhanced performance for applications that utilize the BigDataStack. The application's performance and compliance with its requirements will be monitored during runtime and when a requirement violation exists, the Dynamic Orchestrator will change the application's deployment in order to comply with all requirements.

6.1. Requirements specification

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.2 and literally included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.2, and that they are included in here for the reader's convenience.

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-01	Stakeholder	FUNC	Developer	MAN
Name	Correction of Requirements or SLOs Violations				
Description	When an application or service is running, the orchestrator shall detect the violation of an application requirement or service level objective (SLO) and send a signal to the ADS-ranker to trigger a change in the deployment to try to satisfy the requirements or SLOs.				
Additional Information	N/A				

Table 11 - Correction of Requirements and SLOs Violations (stakeholder requirement).

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-02	Stakeholder	FUNC	Developer	MAN
Name	Decision Efficiency				
Description	When the violation of a requirement has been detected, the orchestrator shall be able to decide what modification to the deployment (e.g. change the number of replicas or the number of vCPUs) has the highest probability of improving the requirements or SLOs satisfaction, as long as any change is possible (i.e. all resources are at its full capacity due to limits).				
Additional Information	N/A				

Table 12 - Decision Efficiency (stakeholder requirement).

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-03	System	FUNC	Developer	MAN
Name	Resources Limits				

Description	The orchestrator shall be able to receive a trigger from the ADS-Ranker when a deployment parameter, such as the number of replicas, the number of vCPUs or the assigned cluster memory, cannot be further increased or decreased (i.e. this resource has reached its maximum or minimum possible value) and use this information in its own decisions.
Additional Information	The complete list of deployment parameters to be taken into account still needs to be determined.

Table 13 - Resources Limits (stakeholder requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-DO-04	Stakeholder	FUNC	Developer	DES
Name	Orchestration for Improvements				
Description	When an application or service is running, the orchestrator shall detect changes in the system status or inputs (e.g. less new events per minute) and trigger a change in the deployment that results in lower costs (e.g. to use less replicas) without compromising the application functioning.				
Additional Information	N/A				

Table 14 - Orchestration for Improvements (stakeholder requirement).

6.2. Design

The Dynamic Orchestrator observes the application performance through the monitoring of runtime system metrics or Key Performance Indicators (KPIs) and Service-Level Objective (SLO) violations, which are received from the Triple Monitoring and QoS Evaluation component. When a requirement or SLO violation is detected, the Dynamic Orchestrator decides how the current deployment should be modified and sends a trigger (along with deployment change recommendations) to the ADS Ranker to perform the redeployment.

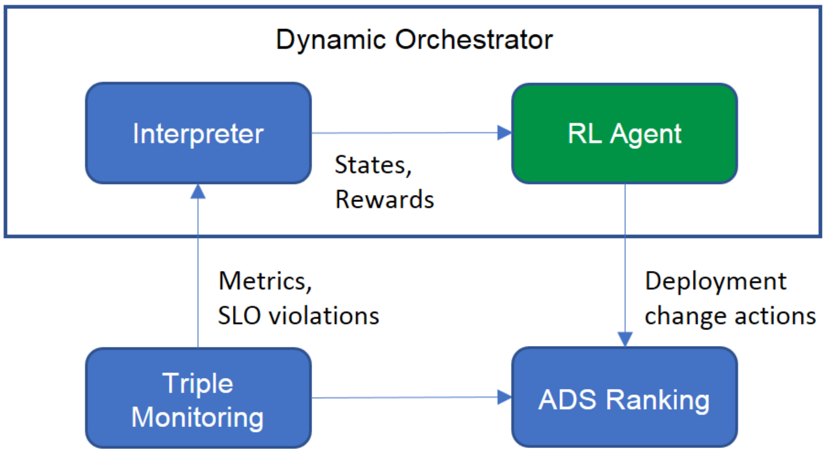


Figure 13 – Dynamic Orchestrator – conceptual diagram.

To implement the Dynamic Orchestrator's logic, we propose a Reinforcement Learning (RL) approach that learns about the application's performance during runtime and learns, based on its own experience, what kind of changes should be performed to satisfy requirements and SLOs. The design of the Dynamic Orchestrator has been completed to provide the following overall functionality (see Figure 14):

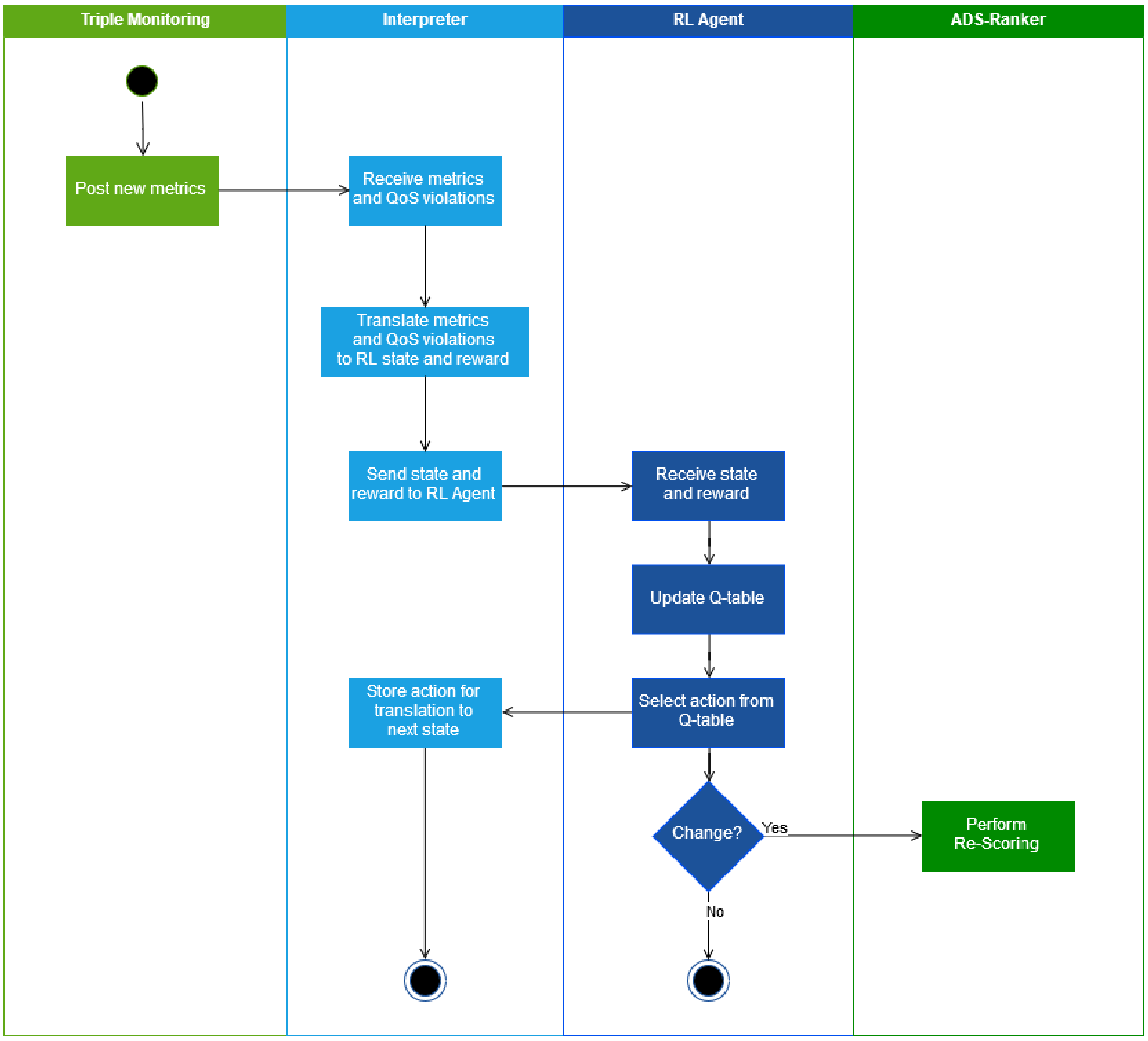

Triple Monitoring
Interpreter
RL Agent
ADS-Ranker
Post new metrics
Receive metrics and QoS violations
Translate metrics and QoS violations to RL state and reward
Send state and reward to RL Agent
Receive state and reward
Update Q-table
Select action from Q-table
Store action for translation to next state
Change?
Yes
No
Perform Re-Scoring

Figure 14 – ADS Ranking Interaction Diagram

1. The Triple Monitoring informs the Interpreter about the current system metrics and the SLO violations.
2. The Interpreter, converts these metrics and SLO violations in states and rewards.
    a. The states represent the system status in a discrete space.
    b. The rewards indicate the Reinforcement Learning Agent if an executed action was "good" or "bad" in terms of requirements and SLOs compliance (e.g. if the requirements and SLO violations disappeared after the execution of an action).
3. The Interpreter sends the current state of the system to the RL (Reinforcement Learning) Agent and according to this, the RL Agent selects an action that should be executed by the ADS-Ranker.

   a. The actions are type of changes in the deployment such as change the number of replicas, change the number of vCPUs or change the vRAM assigned (note: these are just some of the changes that are being considered, the full list of deployment changes still needs to be determined).
   b. One of the actions is to keep the current deployment.
4. Once an action has been executed, the interpreter receives the new metrics and SLO violations, calculates the reward and sends it to the RL Agent.
5. The RL Agent updates its state-action ranking (Q-table in RL terminology).

In addition, the Interpreter updates the current state which will then be observed by the RL Agent to take the next action.

6.2.1.Adaptable Distributed Storage interplay

The Adaptable Distributed Storage (as described in D4.1) will not rely on the Dynamic Orchestrator or the Ranking & Deployment to scale in/out its resources; rather, because of the larger number of metrics available internally, it integrates its own Elasticity Manager subcomponent that is responsible for taking this kind of decisions for the storage layer. As a result, the storage can be re-configured automatically, moving data regions across its current nodes and scale in or out to be adapted under diverse workloads. As these redeployments are being triggered separately, the Dynamic Orchestrator should be aware of those, and postpone any redeployment action on the application level until the reconfiguration of the storage is finished, and the system is balanced.

Therefore, the Dynamic Orchestrator needs to consider there is a second dynamic adaptation mechanism acting at the storage layer level. This second adaptation component (i.e., Elasticity Manager) will inform the Dynamic Orchestrator component regarding reconfigurations of the data storage layer; in fact, this has been specified as a requirement imposed on the Adaptable Distributed Storage (see REQ-ADS-06 in D4.1) by the Dynamic Orchestrator. More specifically, the Adaptable Distributed Storage will notify information regarding pending redeployments of the storage, when the process of data reconfiguration starts and finishes, along with the current deployment of this layer.

This information helps the Dynamic Orchestrator to determine the best course of action in this scenario: to wait until the data layer reconfiguration finishes and to re-evaluate the performance afterwards, or to perform a quick change that will deliver a faster improvement until the data layer changes take effect.

6.3. Early Prototype

An early prototype of the Dynamic Orchestrator has been developed using a Reinforcement Learning (RL) based approach. We have chosen RL because it gives the orchestrator the ability to learn from its own experience. This means that the orchestrator will learn from the application and the deployment itself during the application runtime, making it more flexible than any predefined rule-based approach.

The dynamic orchestration problem has been analysed and framed as a RL problem, by defining the states-action configurations as well as the reward function. The states are determined by the current metrics of the system, and the actions are referred to the type of change needed in the deployment to improve the current performance. Because of the several metrics and actions involved, the Reinforcement Learning agent will have to deal with a large action-state. To solve this problem, we are looking into preselecting the most relevant metrics for the application and discretizing its values in order to reduce the states. A similar preselection step is being considered to reduce the actions space. To have a more in-depth view of our planned approach, see Section 6.6.

In addition, we have implemented an early prototype of the Dynamic Orchestrator using Q-learning on Python. In this implementation, we use metrics from the applications' inputs and the CPU to define the RL state and three actions: to increase or decrease the number of replicas or to keep the current deployment. We have tested the approach using a simulated environment and application, in which the Dynamic Orchestrator performed better than a rule-based implementation.

6.4. Use Case Mapping

The BigDataStack use case that has been chosen to validate the dynamic orchestrator of BigDataStack is the Connected Consumer: Multi-sided market ecosystem, provided by ATOS WORLDLINE-EROSKI.

Most people do their groceries after 6 p.m. since this is the time they finish work and head home. This creates a peak time for cashiers at supermarkets, in which even though more checkout points are opened, queues of clients arise. For the Connected Consumer system, which is shared between all physical locations, this means a peak in recommendation requests that must be served in a timely manner to avoid further delays in customer service. To support the higher throughput required, the application deployment needs to change. For this use case, the Dynamic Orchestrator workflow is as follows:

1. The Triple Monitoring component informs the Interpreter about the current metrics, with the increase in the rate of incoming events received by the system.
2. The Interpreter translates this into a RL state and sends a message to the Dynamic Orchestrator.
3. The Dynamic Orchestrator observes the change in the state and decides the best change in the deployment to be performed.
4. The Dynamic Orchestrator sends a message to the ADS-Ranker that will perform the change.
5. The Triple Monitoring informs the Interpreter about the new metrics status and the Interpreter updates the RL state and calculates the reward generated by the change in the deployment; this reward will be positive if the change improved the throughput of the system and negative otherwise.
6. The interpreter sends the new state and the reward to the Dynamic Orchestrator.
7. The Dynamic Orchestrator observes the reward and the new state of the system.
8. The Dynamic Orchestrator updates its internal action-state ranking according to the received reward.

9. If the state has come back to normal, i.e. the application throughput keeps up with the incoming events rate, the Dynamic Orchestrator will take no further action until the state is changed again.

It is important to note that the Dynamic Orchestrator will manage competing workloads of different applications by taking into account the resource availability in its decisions, this means that, for example, if the memory assigned to an application is already at its maximal value, then the Dynamic Orchestrator will not consider increasing the memory size as an action.

6.5. Experimental Plan

To evaluate the performance of the Dynamic Orchestrator, we plan to test its behaviour throughout the use case described above in section 6.4. The tests will be performed simulating a single as well as multiple workloads. Specifically, we will evaluate the performance by monitoring the following aspects:

- Time to adapt: Dynamic Orchestrator should detect the need to change and decide the change rapidly, to provide the highest SLO/Requirements satisfaction rate possible
    - KPIs: SLO/Requirements satisfaction rate, time measured from when SLO/Requirement violation started to when decision to change deployment is taken.
- Change effect: the change decided by the Dynamic Orchestrator should improve the current situation and, if possible, satisfy all the SLO/Requirements
    - KPIs: SLO/Requirements satisfaction rate, SLO/Requirements metrics before and after change was executed.
- Time to learn to adapt: Reinforcement Learning starts choosing actions randomly until it properly learns about its environment and how the actions taken affect it, in this setting it is very important to observe how long does it take for the algorithm to acquire enough knowledge to be able to adapt properly.
    - KPIs: time (probably in RL steps) from boot up to when SLO/Requirements satisfaction becomes stable

6.6. Next Steps

In the future, we plan to further develop the logic for the Dynamic Orchestrator. In particular, there are two challenging aspects:

- Several metrics can be used to define the state. The states space must be limited in order to ensure the RL agent learns about its environment in few steps so the application can offer a high requirements and SLO satisfaction rate since its initiated. The first step to do this is to discretize the metric values properly, creating meaningful bins in which the metric values will be accommodated. The second and maybe even more important step, is to limit the number of metrics to be taken into account, if properly chosen, the right action for every state will be quickly learnt, otherwise, the RL agent will not have enough information to take decisions or will take too long to learn what to do in each state.

- Several deployment changes can be performed by the Dynamic Orchestrator and in different way. Each deployment change will be an action and in the same way that a large states space can negatively affect the learning, a large actions state will compromise the reinforcement learning logic performance. To address this problem, it is important to define what actions are most meaningful for each deployment and in which way they should be implemented, i.e. by giving a determined value for a parameter or by just asking the ADS-ranker to increase/decrease it. In addition, the available actions will vary according to the characteristics of the deployment, e.g. is the deployment in a private or public cloud?

In addition, we plan to do more tests to evaluate the Dynamic Orchestrator's performance and robustness to different runtime situations and with different applications.

7. ADS Ranking & Deploy

The role of the ranking and deployment module of Big Data Stack is to decide how to deploy the user's application and then operationalize that deployment via a container orchestration platform (e.g. Kubernetes). Ranking and deployment is part of the application deployment back-bone that enables a user to get their application running on a hardware cluster. Prior to ranking and deployment, the user will have defined in a conceptual manner what their application is comprised of and how the different services within that application interact. This conceptual definition will have been expanded into multiple candidate deployment pattern (CDP) playbooks representing different ways that the application/services can be mapped onto compute resources for deployment. Finally, these CDP Playbooks will have been benchmarked, providing estimated resource usage and quality of service information for each. Ranking and deployment takes these CDP Playbooks and associated benchmarking information as input.

As its name suggests, ranking and deployment is split into two distinct components, namely: ADS (Application and Data Services) Ranking and ADS (Application and Data Services) Deployment. ADS Ranking is responsible for taking the different CDP Playbooks and associated benchmarking information, and deciding which CDP Playbook is the most suitable based on the user requirements and preferences. This has two uses within BigDataStack, namely: to determine what compute resources to request for a user's application when first deploying it (see the BigDataStack operations phase Step 1); and to re-estimate compute resource needs in cases where a current deployment is predicted to miss one or more Service-Level Objectives (see the BigDataStack operations phase Step 7). Meanwhile, ADS Deployment is responsible for taking the selected CDP Playbook and using the configuration information contained within, to operationalize deployment of the user's application on the cloud infrastructure (see the BigDataStack operations phase Step 4).

7.1. Requirements specification

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.2 and literally included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.2, and that they are included in here for the reader's convenience.

This section contains the requirements for both the ADS Ranking and ADS Deployment components. For reference, ADS Ranking requirements are denoted REQ-ADSR-XX, while ADS Deployment requirements are denoted REQ-ADSD-XX.

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-01	System	FUNC	Application Dimensioning Workbench	MAN
Name	Ingest Candidate Deployment Playbooks and Benchmarking Information				
Description	The Application Dimensioning Workbench sends a series of candidate deployment patterns (CDP) playbooks and benchmarking information to the ADS Ranking component. ADS Ranking needs to collect all these				

	patterns for subsequent scoring/ranking based on the user requirements and preferences.
Additional Information	Ingestion occurs via a common publisher/subscriber platform (RabbitMQ).

Table 15 - Ingest Candidate Deployment Playbooks and Benchmarking Information (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-02	System	FUNC	Developer	MAN
Name	Deployment Suitability Feature Extraction				
Description	Once a series of candidate deployment pattern playbooks and associated benchmarking information has been received, the next step is to determine how each pattern is predicted to perform based on the benchmarking information. In effect, this involves defining a series of functions that relate individual or groups of user requirements to the predicted performances produced by benchmarking. The output of this step is a vector representation for each CDP playbook, representing how that playbook is predicted to fair under different user requirements.				
Additional Information	Features produced here are dependent on the capabilities of the benchmarking system and the amount of information the user provides in terms of requirements and preferences.				

Table 16 - Deployment Suitability Feature Extraction (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-03	System	FUNC	Developer	MAN
Name	CDP Playbook Scoring (Heuristic)				
Description	Given a vector representation for a CDP Playbook, we next need to map this vector into a single score, representing how suitable that playbook will be overall (such that we can compare different CDP Playbooks). This involves combining the different elements within the vector (that each represent some aspect of pattern suitability, such as cost, or predicted compute wastage). The first version of this will use a hand-tuned linear combination.				
Additional Information	N/A				

Table 17 - CDP Playbook Scoring (Heuristic) (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-04	System	FUNC	Developer	DES
Name	CDP Playbook Scoring (Supervised)				
Description	Given a vector representation for a CDP Playbook, we next need to map this vector into a single score, representing how suitable that playbook will be				

	overall (such that we can compare different CDP Playbooks). This involves combining the different elements within the vector (that each represent some aspect of pattern suitability, such as cost, or predicted compute wastage). The second version of this will learn how to combine the elements based on logging information from past deployments. Models may be non-linear in nature.
Additional Information	Depends on REQ-ADSR-06.

Table 18 - CDP Playbook Scoring (Supervised) (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-05	System	FUNC	Developer	MAN
Name	CDP Playbook Selection				
Description	Once all candidate deployment patterns have been scored, the final step is to select one of those patterns to pass to ADS Deployment. In many cases this will simply involve selecting the highest scoring pattern. However, the user may have the option to select an alternative configuration at this stage.				
Additional Information	N/A				

Table 19 - CDP Playbook Selection (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-06	System	FUNC	Developer	DES
Name	Supervised Model Training				
Description	To support REQ-ADSR-04, a supervised scoring model is needed. To react to changes in the deployment environment over time, this model needs to be frequently updated based on new information from current deployments. This model needs to be trained based on logging data being collected by the Triple Monitoring Framework.				
Additional Information	Requires logging information produced by the Triple Monitoring Framework and stored in the Central Decision Tracker.				

Table 20 - Supervised Model Training (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-07	System	FUNC	Developer	MAN
Name	CDP Playbook Re-Scoring				
Description	It is envisaged that in (rare) scenarios, an ongoing application deployment will fail to meet the user's quality of service requirements. This might occur due to assumptions on data input volumes being violated for instance. In this case, we may not be able to solve this issue without fully redeploying the user application with different resources. To support such re-				

	deployment activities, ADS Ranking supports a re-scoring function, where a previous set of CDP playbooks for a user's application can be re-scored based on updated preferences provided by the Big Data Stack Orchestrator, as well as live data about how the previous deployment performed (and failed).
Additional Information	N/A

Table 21 - CDP Playbook Re-Scoring (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-01	Stakeholder	FUNC	Application developers	MAN
Name	Performance Measurability				
Description	Each environment should be measurable according to a set of characteristics, that is, Key Performance Indicators (KPIs).				
Additional Information	The KPIs considered must include: • vCPUs • Memory				

Table 22 – Performance Measurability (stakeholder requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-02	Stakeholder	FUNC	System	MAN
Name	Standards-based Playbook				
Description	The description of the environments and deployments (i.e., playbooks) will follow a standard specification language				
Additional Information	N/A				

Table 23 - Standards-based Playbook (stakeholder requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-03	System	FUNC	System	MAN
Name	Standard deployment information				
Description	When communicating with other components, as described in Section 7.2, these components will use the playbook standard defined in REQ-RD-02.				
Additional Information	N/A				

Table 24 - Standard deployment information (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-04	System	FUNC	System	MAN

Name	Application Scoring System
Description	The ranking system evaluates each environment's deployment, which keeps track of the most suitable configuration for each application. When trying a deployment configuration for a new application, this ranking will be used to select the most suitable one.
Additional Information	The evaluation will be done following the measurements defined in REQ-RD-01.

Table 25 – Application Scoring System (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-05	System	FUNC	Cluster management component	MAN
Name	Compatibility with Kubernetes				
Description	Since the technology used to run and orchestrate the applications is based in Kubernetes (OKD[8]). Thus, the ADS-Deployment component is required to be compatible with Kubernetes.				
Additional Information	The ADS-Deploy component should translate from the playbook standard defined in REQ-RD-01 into Kubernetes primitives.				

Table 26 - Compatibility with Kubernetes (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-06	System	FUNC		MAN
Name	Synchronous communication				
Description	The communication with and within both components should be done through an API REST.				
Additional Information	N/A				

Table 27 - Synchronous Communication (system requirement).

7.2. Design

In this section we summarize the main architectural design for the Ranking and Deployment module. More precisely, we first introduce the two components and how they interact with other components around them in the larger Big Data Stack platform, and secondly, we define the internal activity flow for ADS Ranking and ADS Deployment.

The high-level architecture of the Ranking and Deployment module is illustrated in Figure 15. The two main components within the Ranking and Deployment module are coloured in green, while other components that the module depends on or are dependent upon the module are coloured in blue. When an application is sent for deployment, it first passes through the Application Dimensioning Workbench for CDP (Candidate Deployment Pattern) Playbook

[8] OKD - https://www.okd.io/

creation and benchmarking. The resultant CDP Playbooks and benchmarking information are sent to ADS Ranking via an interface (First-Time Ranking Interface).

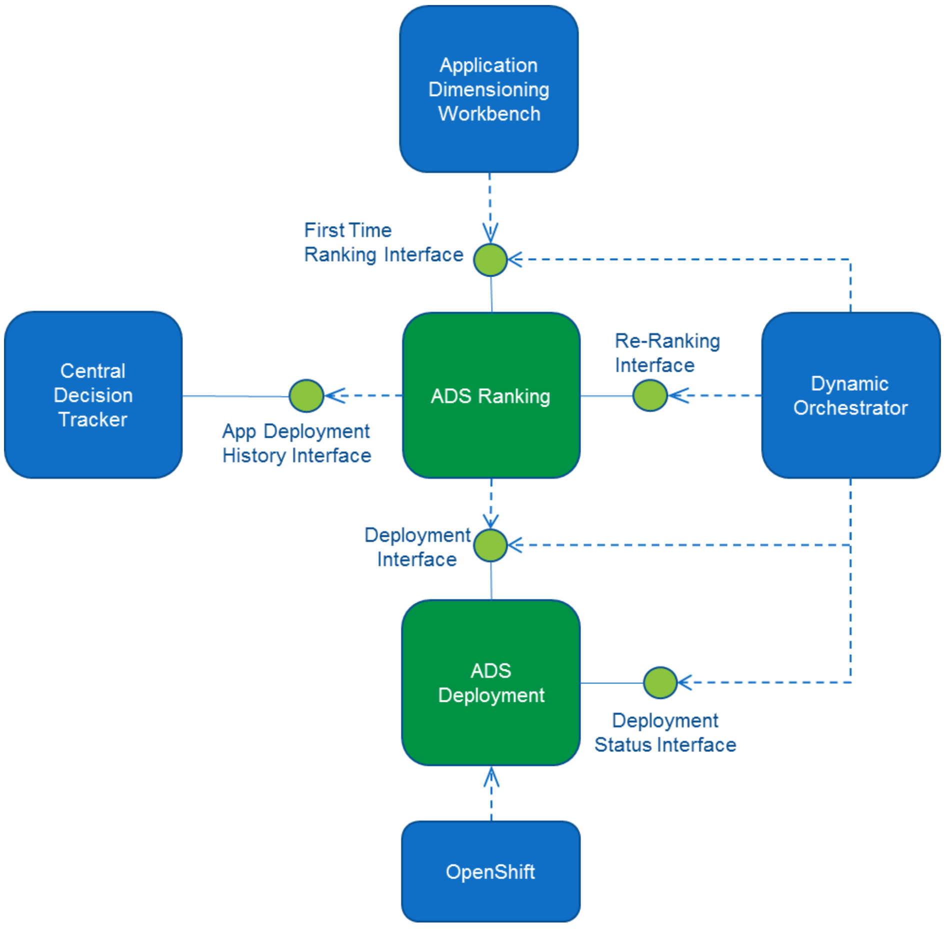

Application Dimensioning Workbench
First Time Ranking Interface
Central Decision Tracker
ADS Ranking
Re-Ranking Interface
Dynamic Orchestrator
App Deployment History Interface
Deployment Interface
ADS Deployment
Deployment Status Interface
OpenShift

Figure 15 – Ranking and Deployment Module Architecture.

ADS Ranking uses information within each CDP Playbook along with historical data obtained from the Central Decision Tracker to score and then select one CDP Playbook for deployment. This is CDP Playbook is sent via the Deployment Interface to ADS Deployment, where it is used to request resources for the user's application, using the Cluster Management component (See Section 5). This sequence of actions is what we refer to as 'first-time deployment' and represents the case where a user comes with a new application to deploy. It is also worth noting that a separate component, referred to as the Dynamic Orchestrator (not to be confused with the container orchestration platform) monitors first-time deployment via the different interfaces, as it is responsible for triggering actions based on deployment state (e.g. if we cannot find a suitable deployment). Additionally, due to REQ-ADSR-07, the Dynamic Orchestrator may request the re-scoring of CDP Playbooks directly in the case where application failure post-deployment is detected. This is achieved via a bespoke Re-Ranking interface provided by the ADS Ranking component.

The ADS Ranking component has two main 'modes' of operation, which directly relate to the first-time deployment and re-scoring actions discussed above.

Figure 16 provides an overview of the interactions involved in these two operation modes. Under the first-time deployment action, the Application Dimensioning Workbench sends CDP Playbooks and benchmarking information to ADS Ranking. ADS Ranking then ingests those

CDP Playbooks (REQ-ADSR-01), reformatting them into a form that can be easily processed. Next, each CDP Playbook is transformed into a feature vector by CDP Playbook Feature Extraction (see REQ-ADSR-02). These features represent the predicted performance of the user's application under expected average and peak load given the suggested compute resources.[9] Once a CDP Playbook has been vectorised, it is next subject to scoring based on the user's requirements and preferences. The scoring function may be unsupervised (REQ-ADSR-03) or may use a machine learned model trained on previous application deployments (REQ-ADSR-04). Finally, the scored CDP Playbooks are subject to selection, where the most suitable one will be picked as the template for application deployment (REQ-ADSR-05). The selected CDP Playbook is then passed to ADS Deployment to operationalize the physical deployment.

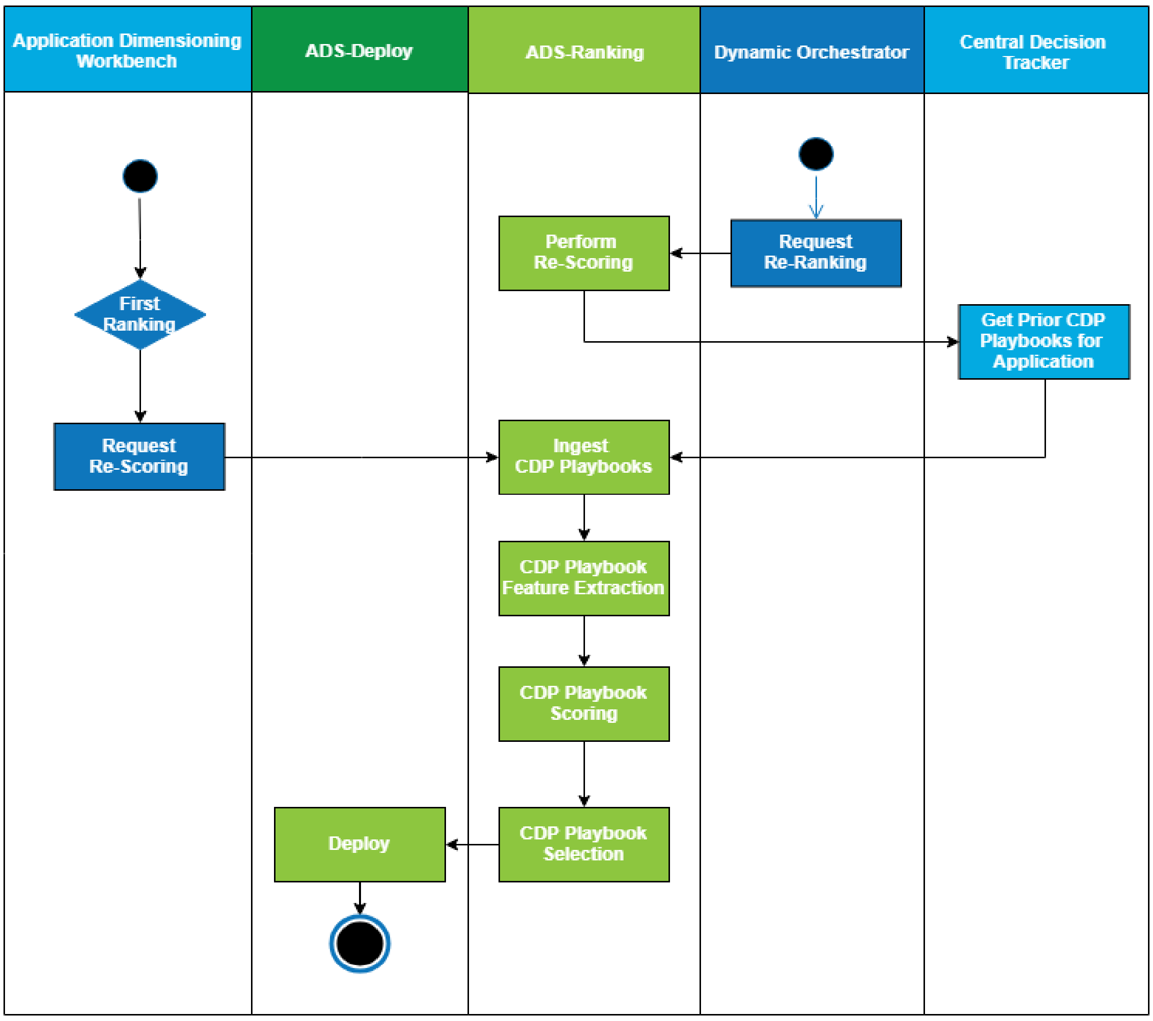


Figure 16 – ADS Ranking Interaction Diagram

[9] Note it is anticipated that these predicted performances will be subject to some degree of error. For example, benchmarking statistics (over a small data sample) may not reflect true deployment performance. Moreover, in opaque cluster scenarios (see Section 3.2), services from other users may share hardware with our user's application, potentially causing performance degradation that could not be predicted during benchmarking. It is expected that such error may be handled by including compute resource 'head-room' when estimating resources, and if that is not sufficient then corrections will be possible post-deployment via run-time adaptations by the Dynamic Orchestrator.

The second mode of operation is re-ranking. This is based on requirement REQ-ADSR-07, i.e. in cases where a prior deployment has been deemed as no longer suitable, we need to select a new CDP Playbook. In this mode, the Dynamic Orchestrator triggers the re-ranking process, providing updated application preferences based on the reason-of-failure for the prior deployment. ADS Ranking will retrieve the full set of CDP Playbooks for the user's application (from the Central Decision Store) and then perform scoring and selection in a similar manner to the first-time deployment mode.

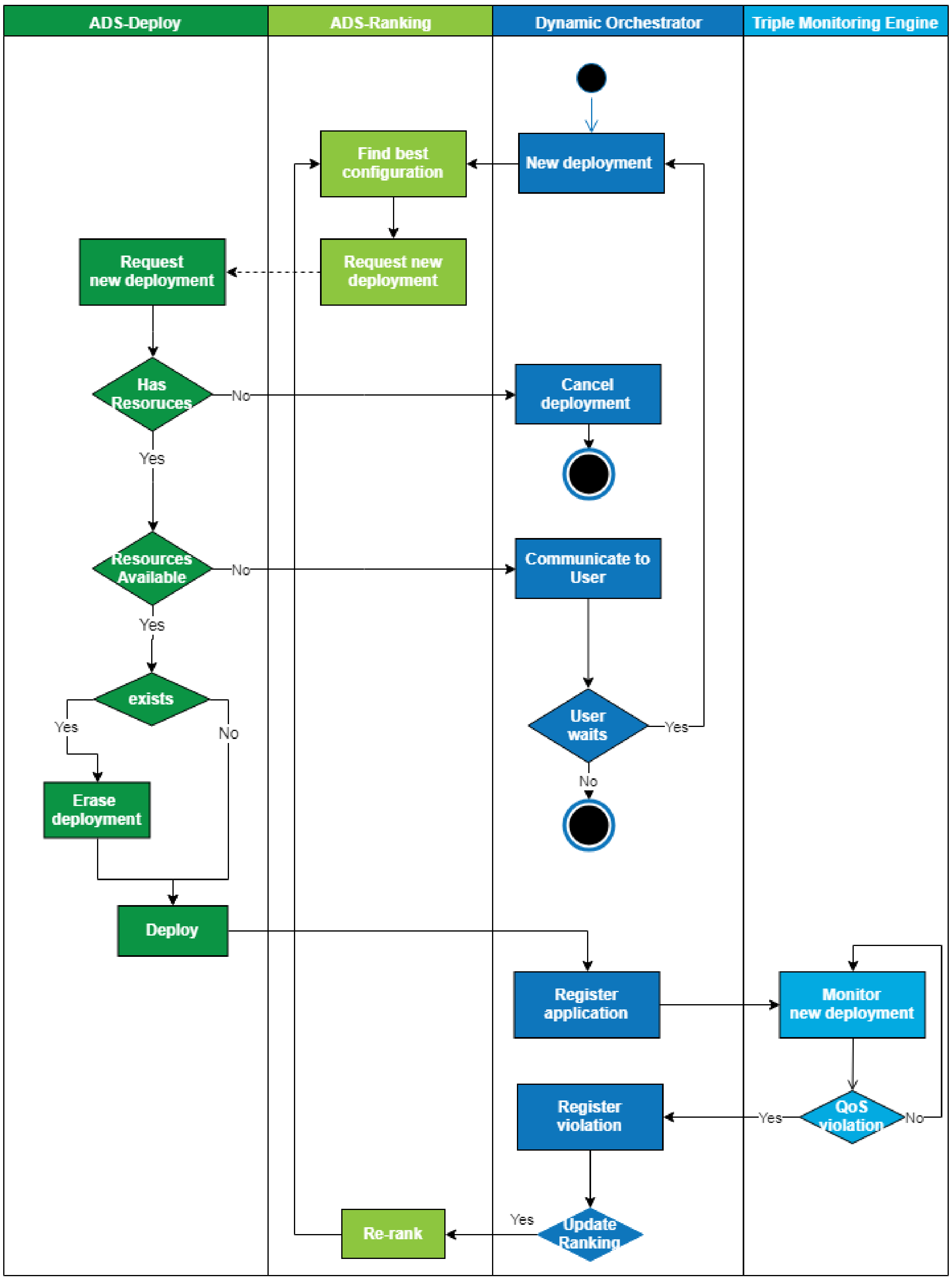


Figure 17 – Interaction diagram of the ADS Deployment and Ranking design

Figure 17 shows the interaction between ADS-Deploy and the other components in the Big Data Stack platform. The request for a new deployment by the ADS-Ranking component starts a new thread to process a CDP Playbook. The ADS Ranking component communicates this deployment request to ADS Deployment, which interprets the CDP Playbook and proceeds with the deployment of the application. There are three possible outcomes to this deployment process:

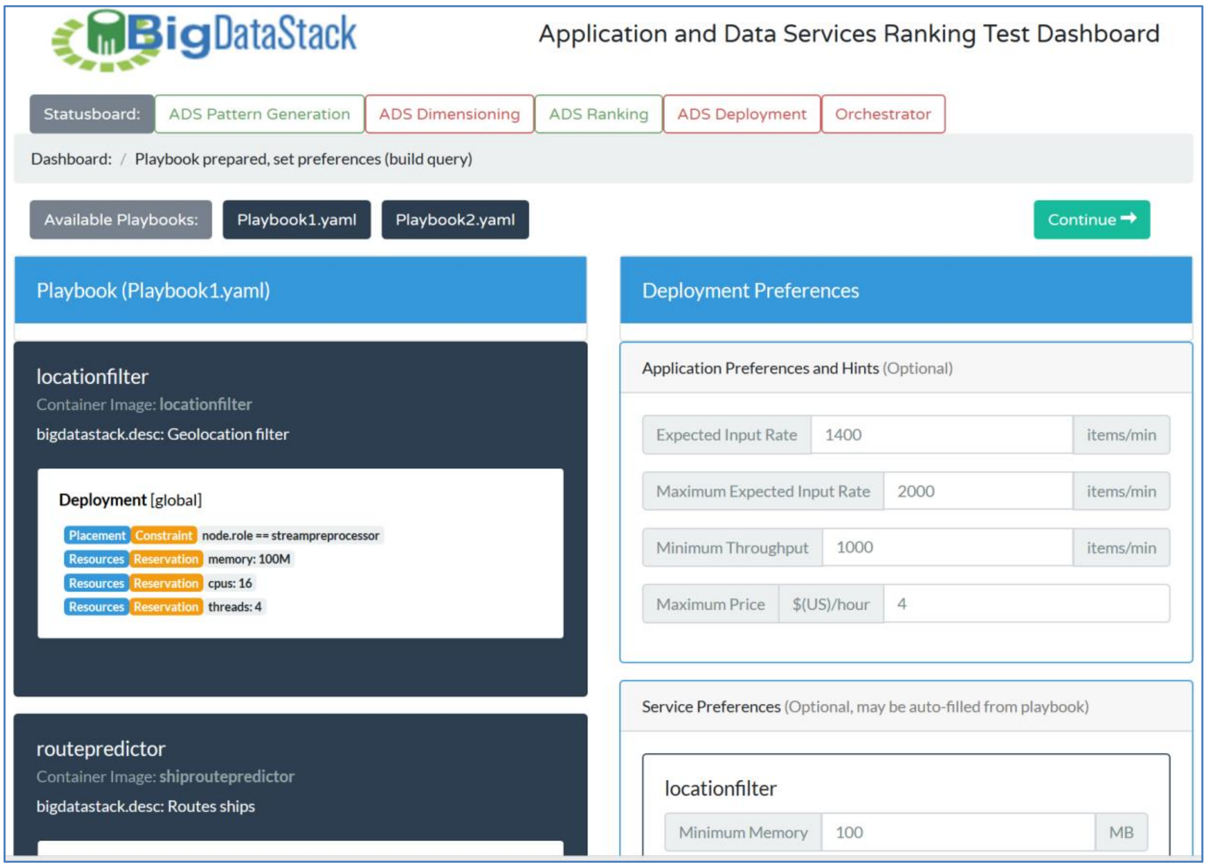
1. The requested compute resources are available: In this case, ADS deployment sets up the environment, deploying and starting the user service. Once the environment has been fully set-up, the ADS Deployment component communicates this deployment to the Dynamic Orchestrator.
2. The cluster has the available compute resources but some of those resources are busy: In this case, the environment allocation will be scheduled for when the cluster has sufficient free compute resources. ADS Deployment component will communicate the scheduled state of the user application to the Dynamic Orchestrator, which can opt to either leave the application to wait for resources to become available or otherwise cancel the deployment process (e.g. at the behest of the user).
3. The cluster does not have sufficient available resources: In rare scenarios, it may be the case where the estimated compute resources needed to run the user's application are greater than can be provided by the cluster. This might happen if using a small dedicated cluster for instance, or in cases where a resource usage cap is in place. In this case, deployment is cancelled, and ADS Deployment component will communicate this to the Dynamic Orchestrator, which can alert the user.

Once a user's application has reached a running state (i.e. the resources were immediately available or became available while the application was scheduled), the Dynamic Orchestrator calls the Triple Monitoring Engine and QoS Evaluation (TME), described in Section 8, for monitoring of the deployment. The TME evaluates periodically if the deployment agrees with an agreed minimum QoS between the system and the user. Every time that the minimum QoS is not respected, this data is communicated to the Dynamic Orchestrator. This may trigger re-scoring by ADS Ranking (among other possible run-time adaptations). In this case ADS ranking proceeds to re-evaluate the list of available CDP Playbooks, based on to their performance, ultimately leading to the user's application being re-deployed with a new set of compute resources.

7.3. Early Prototype

At M11 the first Tier 0 implementation of the ADS Ranking component has been developed and tested. This version provides CDP Playbook ingestion (REQ-ADSR-01), a basic level[10] of Deployment Suitability Feature Extraction (REQ-ADSR-02), CDP Playbook Scoring (Heuristic) (REQ-ADSR-03). This component is deployable via an Apache Spark cluster. At M11 development of the ADS Deployment component has not started.

[10] Subsequent Tiers will include more advanced feature extraction capabilities as new benchmarking data becomes available from advances made from the development of the application dimensioning workbench.


Application and Data Services Ranking Test Dashboard

Statusboard: | ADS Pattern Generation | ADS Dimensioning | ADS Ranking | ADS Deployment | Orchestrator

Dashboard: / Playbook prepared, set preferences (build query)

Available Playbooks: Playbook1.yaml | Playbook2.yaml | Continue

Playbook (Playbook1.yaml)

locationfilter
Container Image: locationfilter
bigdatastack.desc: Geolocation filter

Deployment [global]
Placement Constraint node.role == streampreprocessor
Resources Reservation memory: 100M
Resources Reservation cpus: 16
Resources Reservation threads: 4

routepredictor
Container Image: shiproutepredictor
bigdatastack.desc: Routes ships

Deployment Preferences

Application Preferences and Hints (Optional)

Expected Input Rate	1400	items/min
Maximum Expected Input Rate	2000	items/min
Minimum Throughput	1000	items/min
Maximum Price	$(US)/hour	4

Service Preferences (Optional, may be auto-filled from playbook)

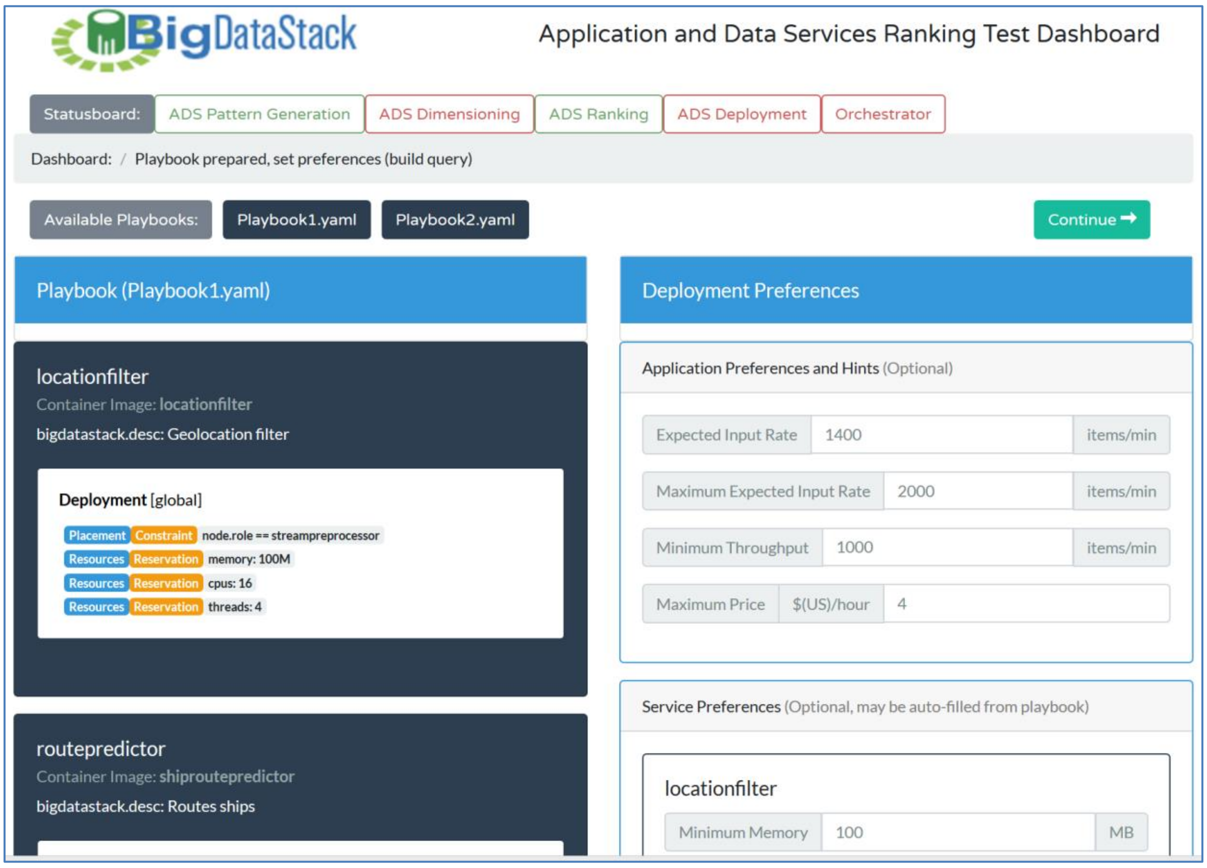
locationfilter

Minimum Memory	100	MB

Figure 18 – ADS Test System, Playbook View.

In addition, to facilitate the testing of the Ranking and Deployment functionality, a test system was created. This test system was developed to provide a way for BigDataStack developers to analyse the inner workings of application deployment, in general, and the ADS Ranking functionality, in particular. This is needed in part because the Ranking and Deployment functionalities are largely invisible to the BigDataStack users, and hence a separate test system is needed to observe its function.

More precisely, the test system is designed to isolate the Ranking and Deployment functionality from the rest of the BigDataStack platform such that it can be tested separately. It is designed to simulate and allow the user to customise the output of the Application Dimensioning Workbench, such that ADS Ranking can be tested under different experimental scenarios. The test system is comprised of four main screens. First, illustrated in Figure 18At M11 the first Tier 0 implementation of the ADS Ranking component has been developed and tested. This version provides CDP Playbook ingestion (REQ-ADSR-01), a basic level of Deployment Suitability Feature Extraction (REQ-ADSR-02), CDP Playbook Scoring (Heuristic) (REQ-ADSR-03). This component is deployable via an Apache Spark cluster. At M11 development of the ADS Deployment component has not started.



Dashboard: / Playbook prepared, set preferences (build query)

Playbook (Playbook1.yaml)

locationfilter
Container Image: locationfilter
bigdatastack.desc: Geolocation filter

Deployment [global]
Placement Constraint node.role == streampreprocessor
Resources Reservation memory: 100M
Resources Reservation cpus: 16
Resources Reservation threads: 4

routepredictor
Container Image: shiproutepredictor
bigdatastack.desc: Routes ships

Deployment Preferences

Application Preferences and Hints (Optional)

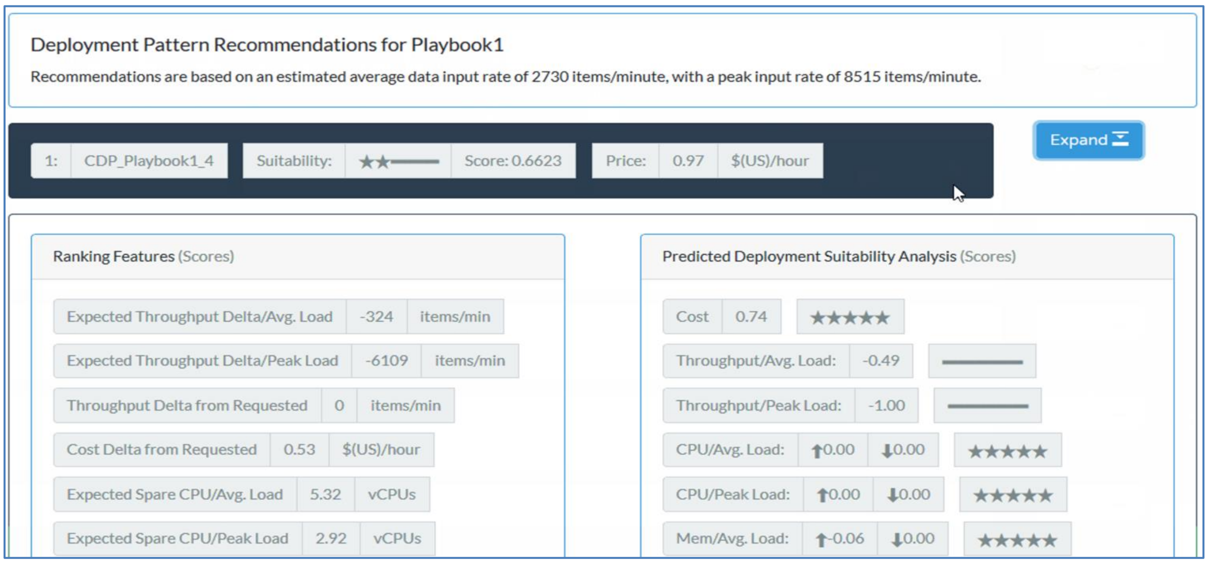
Expected Input Rate	1400	items/min
Maximum Expected Input Rate	2000	items/min
Minimum Throughput	1000	items/min
Maximum Price	$(US)/hour	4

Service Preferences (Optional, may be auto-filled from playbook)

locationfilter

Minimum Memory	100	MB

, is the Playbook view. This allows the user to load pre-defined application Playbooks, which can then be customised (using the controls on in the right pane) to create a test scenario. The second view is the simulated dimensioning view, which is responsible for configuring the output of the application benchmarking (more information on this can be found in deliverable 5.1). This is important as benchmarking will be imperfect, hence we need a means to model benchmarking accuracy, as that will in turn impact the performance downstream in ADS Ranking. The third view is the ADS progress view. This view is simply to allow the tester to monitor the time taken for dimensioning, ranking and deployment.



Deployment Pattern Recommendations for Playbook1
Recommendations are based on an estimated average data input rate of 2730 items/minute, with a peak input rate of 8515 items/minute.

1: CDP_Playbook1_4 | Suitability: ★★ | Score: 0.6623 | Price: 0.97 $(US)/hour

Ranking Features (Scores)

Expected Throughput Delta/Avg. Load	-324	items/min
Expected Throughput Delta/Peak Load	-6109	items/min
Throughput Delta from Requested	0	items/min
Cost Delta from Requested	0.53	$(US)/hour
Expected Spare CPU/Avg. Load	5.32	vCPUs
Expected Spare CPU/Peak Load	2.92	vCPUs

Predicted Deployment Suitability Analysis (Scores)

Cost	0.74		★★★★★
Throughput/Avg. Load:	-0.49		
Throughput/Peak Load:	-1.00		
CPU/Avg. Load:	↑0.00	↓0.00	★★★★★
CPU/Peak Load:	↑0.00	↓0.00	★★★★★
Mem/Avg. Load:	↑-0.06	↓0.00	★★★★★

Figure 19 – ADS Test System, Ranking View.

The fourth view is the ranking view, illustrated in Figure 19. The ranking view provides statistical information about how each CDP Playbook was scored in terms of suitability with respect to the user's requirements and preferences. For example, this may include a comparison of how the expected cost of the deployment relates to the amount the user is willing to pay; or comparing the expected CPU usage of the user's services against the hardware being requested. This information allows the developers of the ADS Ranking component better understand where deployments may fail, and hence how to improve the CDP Playbook scoring algorithm (REQ-ADSR-03 and REQ-ADSR-04).

7.4. Use Case Mapping

ADS Ranking and ADS Deployment components are not explicitly linked to any particular use-case, as it forms part of the underlying pipeline for application deployment, which is required for all use-cases in the project. Indeed, the identification of suitable CDP Playbooks for a user's application is critical, such that sufficient resources to support that application are identified and requested during deployment. For example, for the ATOS-WORDLINE-EROSKI scenarios (SCE-CC-01 and SCE-CC-02), large amounts of processing capacity is needed for learning recommendation models for the connected consumer. It is the role of ADS Ranking to identify the compute resources (represented by a CDP Playbook) that will enable this learning in a fast and efficient manner. Meanwhile, the ADS Deploy component is responsible for the requesting resources for required services on the target cluster. As such, ADS Ranking and ADS Deployment can be considered an implicit requirement for all user scenarios (SCE-RSM-01, SCE-RSM-01, SCE-CC-01, SCE-CC-02, SCE-IMB-01 and SCE-IMB-02).

7.5. Experimental Plan

The development of ADS Ranking is a primarily research-orientated activity, as how to effectively score user application deployments based on benchmarking information, hardware availability and user requirements/preferences is an open problem. Meanwhile, the development of ADS Deployment is primarily an engineering activity, building on top of existing software deployment platforms such as Kubernetes and OpenShift. As such, we focus primarily on the experimental evaluation of ADS Ranking in this section. In Section **¡Error! No se encuentra el origen de la referencia.** we provide a short summary highlighting the challenges of identifying deployment configurations for a user's application. Section 0 introduces our proposal for adapting state-of-the-art learning to rank approaches for this task. Finally, in Section 0 we discuss how we plan to evaluate the effectiveness of ADS Ranking.

7.5.1. Background and Related Work

Currently, companies that wish to deploy large-scale applications onto cloud infrastructure tend to follow one or two strategies. First, if they are developing/have developed their application in-house and have the technical expertise, the developers or other IT experts within the company will be responsible for identifying suitable hardware for that application, as well as subsequent deployment and maintenance on the cloud. On the other hand, if local expertise is not available, companies may make use of external "cloud consultants" (e.g. https://cloudspectator.com/) to handle the benchmarking and optimization of their workloads for a fee. BigDataStack aims to innovate in this space by providing the benchmarking and optimization of cloud deployments automatically, eliminating the need for experts to manually test and monitor user applications. ADS Ranking forms a critical link in achieving this, as it provides the means to automatically relate benchmarking information about an application to the user's requirements and preferences, which previously required expert knowledge.

To our knowledge, there has been no prior works that examine how to tackle automated identification of suitable cloud deployment configurations for an arbitrary user application to-date. However, there has been prior research into the benchmarking tools previously employed by the experts that that provide valuable insights into how to tackle this problem. For example, *CloudSuite* [7] and *DCBench* [15] are benchmark suites for scale-out cloud services. What is important to learn from these tools is how each application defines different deployment targets and constraints. For example, for a video streaming service, the degree of video buffering that the user sees is what matters, while for a Web host, low response latency is critical. Moreover, the targets and constraints of an application may change based on the user scenario. For instance, for an application, training a complex machine learned model [14], the user may only care about time-to-completion if the model is needed to be ready for the next day's processes. Meanwhile, in less time critical settings, minimizing cost may be the main requirement [8]. ADS Ranking aims to provide a solution to identifying suitable compute resources for a user's application that will satisfy such goals automatically, where these goals are specified by the user (as requirements and preferences).

However, this is challenging for two reasons. First, given the diverse and continually expanding number of quality of service targets/constraints that a user might desire, modelling each individually is impractical, necessitating a more general solution. Second, modern application workloads are complex in that they can span multiple inter-dependant functions or services with very different performance profiles. This requires model that can capture the interactions between those functions/services under different loading scenarios, to avoid application processing bottlenecks derived from a single function or service receiving insufficient resources.

Learning to rank techniques are machine learned algorithms, which take as input a set of features describing items, and learns how to effectively combine of those features to create a score for each of those items indicating how related they are to a separate feature vector (commonly referred to as the 'query') [10]. The most common application of learning to rank techniques are Web search engines, where the items to be scored are Web documents and the separate feature vector is the user's search query (hence why we refer to this separate feature vector as the query, although it is important to note that the separate feature vector can be used to represent anything). The goal of learning to rank is to find the feature combination (referred to as a model) which results in the most effective ranking for a set of items given a query. In the next section we summarize how we propose to adapt learning to rank techniques to tackle the challenges of identifying the most effective CDP Playbook for a user's application.

7.5.2.Learning to Rank CDP Playbooks

To tackle the challenges of identifying the most effective deployment configuration for a user's application, we propose to extend current state-of-the-art learning to rank approaches. The core idea underpinning this is that CDP Playbooks can be seen as our items to rank, where we can represent each CDP Playbook by aggregating: 1) information provided by the user about their application/services; 2) the suggested compute resources contained within the CDP Playbook; and 3) the benchmarking data obtained from the application dimensioning workbench. Our 'query' is then the user's defined requirements and preferences. The goal of learning to rank in this case is to learn a model that effectively combines all the aggregated CDP Playbook information into a single score that represents how suitable that CDP Playbook is based on the user's requirements and preferences. In effect, this model aims to learn how value the features of an CDP given different sets of requirements and preferences. This model can be used to score all CDP Playbooks generated for the user application, where the highest scoring is likely the best to use to deploy the user's application.

However, adapting learning to rank techniques into this different domain brings with it three additional challenges. First, we need to define an effective series of features using which we can represent each CDP Playbook. As noted above, to achieve this, we have three potential sources of evidence, in the form of application/service information provided by the user, compute resource information contained in the CDP Playbook itself and benchmarking statistics provided by the application dimensioning workbench. This CDP Playbook evidence needs to be aggregated and normalized to form meaningful features. Second, as noted in our discussion of the challenges of identifying hardware configurations that meet deployment targets and constraints, there are a large and diverse set of user requirements and preferences. Hence, we need an effective approach to map these requirements and preferences into a generic feature vector (our 'query'). Finally, to learn an effective learning to rank model, a large training dataset is needed. As no-one has attempted to use machine learning approaches for this task to-date; such a training dataset does not exist. As such, we will need to develop such a dataset during the project.

These are the three main research challenges that will be investigated in Y2 of the BigDataStack project within Task 3.3.

7.5.3.Evaluation Methodology and Metrics

The ADS Ranking and ADS Deployment components are planned to be initially tested as part of the evaluation scenarios planned at M12 (Inference without Data Access, see Section 4.1.1**¡Error! No se encuentra el origen de la referencia.**) and M15 (Inference with Data Access, see Section 4.1.2). Under these evaluation scenarios a single user application will be deployed by the BigDataStack platform and instrumented under variable load conditions. These initial tests will be performed in a dedicated cloud scenario, without significant competition for cluster resources. Note that we anticipate that this testing will include only unsupervised versions ADS Ranking, as the core research needed to facilitate future learning to rank based versions will be being performed in parallel.

- <u>Evaluation Setting:</u> For each evaluation scenario, an application Playbook will have been created by an up-stream process that describes the application services that are to be deployed. This Playbook will be ingested by the Pattern Generation component, which will produce a series of candidate deployment pattern playbooks (CDP Playbooks). Each playbook will have been subject to benchmarking by the application dimensioning workbench. Each CDP Playbook will be processed by ADS Ranking, resulting in the generation of a suitability score for each. The effectiveness of ADS Ranking will be evaluated based on how many of the top-scored CDP Playbooks would have been suitable based on the user's requirements. To create a ground truth for whether each CDP Playbook was in fact suitable, the user's application will be physically deployed using that CDP Playbook as its deployment configuration. A CDP Playbook is considered suitable if it passes all of the user's requirements during its run-time. The ground truth may additionally include multiple suitability 'grades' based on to what extent the CDP Playbook also met the user's preferences.

- Metrics: To evaluate effectiveness of a ranking of CDP Playbooks deployed by the ADS Ranking component, we will use standard ranking metrics from the information retrieval literature. In particular, we will report:
  - *Precision@1*: This evaluates whether the top ranked CDP was suitable for the user's application
  - *Mean Average Precision (MAP)*: Average precision (at a particular rank) is the proportion of suitable CDP Playbooks down to that rank. MAP is average precision calculated at the maximum rank over multiple application deployments. [13]
  - *NDCG@5*: Discounted Cumulative Gain (DCG) is a measure the usefulness, or *gain*, of an item based on its position in a ranking. Total gain is accumulated starting from the top of the result and moving downwards to a set rank (@N). Gain of each result is discounted at lower ranks and can incorporate (suitability) grades. NDCG is DCG normalized across (in our case) different application deployments to account for some deployments being easier to find suitable patterns for than others. [9]

7.6. Next Steps

It is currently envisaged that there will be three further releases of the ADS Ranking component and two releases of the ADS Deployment component during BigDataStack, integrating more advanced functionality:

- ADS Ranking
  - *Tier 1*: This second version of the ADS Ranking component will integrate directly with the first iteration of the application dimensioning workbench to obtain benchmarking features. This version will also include the first implementation of the re-ranking functionality (REQ-ADSR-07).
  - *Tier 2*: This version will transition from using current heuristic scoring of CDP Patterns to the first version of our proposed learning to rank approach.
  - *Tier 3*: This version will include our second iteration of the learning to rank approach with support for reinforcement learning from live data collected by the Triple Monitoring Framework.
- ADS Deploy
  - *Tier 1*: This Version of the ADS Deploy component will see to integrate with the existing technologies (REQ-ADSD-05) for containerization and provide a working prototype, able to deploy an environment from a given Playbook CDP.
  - *Tier 2*: This version of the ADS Deploy will implement all the synchronization functionalities (REQ-ADSD-06) needed for the interaction with the component.

This implies that it will be integrated with the ADS Ranking and Dynamic Orchestrator components.

8. Triple Monitoring & QoS Evaluation

The triple monitoring component collects and stores several metrics on performance at an application, data service and resource cluster level. These metrics are used to dynamically adapt the environment and ensure the best QoS (Quality of Service) to the user. When a user requests a service from BigDataStack, a minimum QoS is agreed between the user and the system. At runtime, certain metrics or Key Performance Indicators (KPI) are collected by the Triple Monitoring Engine and evaluated against the agreed Service-Level Objectives (SLOs) by the QoS Evaluator.

8.1. Requirements specification

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.2 and literally included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.2, and that they are included in here for the reader's convenience.

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-01	Stakeholder	FUNC	Developer	MAN
Name	Regular recording of QoS metrics				
Description	When a user's application is deployed, the Triple Monitoring Framework monitors that application, tracking statistical information about its operation and associated QoS data, including network, data storage, virtualization layers, etc. This data is needed to support the learning of ranking models by ADS-Ranking service (part of Application and Service Deployment; see REQ-ADSR-03) and regularly saved in a centralised data store for later access.				
Additional Information	Input: - Candidate Deployment Pattern (application identifier from this is the primary key for saving monitoring data for an application) Output: - Deployment QoS Snapshot (monitoring/QoS data, every few mins) Service Dependencies: - Centralised Data Store (Storage Service) - This is implemented over Prometheus[11] as the monitoring collector.				

Table 28 - Regular recording of deployment QoS information (stakeholder requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-02	Stakeholder	FUNC	Developer	MAN
Name	QoS violation alert				

[11] Prometheus. https://prometheus.io/

Description	If the system does not respect the agreed QoS, an alert is raised.
Additional Information	This alert is used internally to evaluate the performance of an environment, relating to REQ-RD-004.

Table 29 - QoS violation notification (stakeholder requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-03	Stakeholder	FUNC	Developer	DES
Name	QoS violation monitoring				
Description	QoS violations are also monitored and shown to the user/admin.				
Additional Information	N/A				

Table 30 - QoS violation monitoring (stakeholder requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-04	System	FUNC	Developer	MAN
Name	Metrics pusher				
Description	The metric pusher retrieves KPI data, clean them and ingest them into the monitoring collector (*Prometheus)*.				
Additional Information	The metrics pusher is used when the exporter approach is impossible to apply. This solution will be very useful for getting application specific metrics (it's not approved yet).				

Table 31 - Metrics pusher (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-05	System	FUNC	Developer	DES
Name	API REST for accessing the collected monitoring metrics				
Description	The metrics are accessible through an API REST.				
Additional Information	This component translates client's requests to Prometheus request compatible. Grafana[12] will be used for visualization.				

Table 32 - Monitoring metrics API REST (system requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-06	Software	FUNC	Developer	MAN
Name	Pub/Sub Mechanism for Metrics				

[12] Grafana. https://grafana.com/

Description	This component queries the metrics repository periodically and publishes this information through a publisher/subscriber mechanism. Each client sends subscription requests to the system.
Additional Information	The monitoring metrics getter is implemented on RabbitMQ[13]

Table 33 - Monitoring metrics getter (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-07	Software	FUNC	Developer	DES
Name	Spark compatible				
Description	The triple monitoring engine monitors the performance of Apache Spark[14], which is used in the BigDataStack project as an analytics engine for Big Data, thus needs to be compatible with this technology.				
Additional Information	Monitoring Spark is done using Spark measure project, which can be embedded in spark application allowing the collection of some metrics after each SQL execution. Those metrics are sent to *push gateway* to be exported to Prometheus.				

Table 34 - Spark compatibility (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-08	Software	FUNC	Developer	DES
Name	LeanXcale compatibility				
Description	LeanXcale database[15] already uses Prometheus for its monitoring subsystem. However, the integration is relied on static deployments. Thus, it should be extended to consider re-deployments in cases when an elasticity action takes places which leads to a scale in/out of the resources. In these scenarios, LeanXcale should reconfigure its integration with the existing Prometheus deployment on the run-time and provide monitoring information for the new nodes				
Additional Information	N/A				

Table 35 - LeanXcale compatibility (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-09	Software	FUNC	Developer	DES
Name	OKD compatibility				

[13] RabbitMQ. https://www.rabbitmq.com/
[14] Apache Spark. https://spark.apache.org/
[15] LeanXcale. https://www.leanxcale.com/

Description	The Triple Monitoring engine monitors the performance of Openshift OKD[16], which is the baseline technology used in the orchestration of containers. Therefore, the triple monitoring engine needs to be compatible with this technology.
Additional Information	N/A

Table 36 - OKD compatibility (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-10	Software	FUNC	Developer	DES
Name	CEP compatibility				
Description	The triple monitoring engine monitors the performance of CEP, which is used in the BigDataStack project as a streaming engine for processing data in real-time. Therefore, the triple monitoring engine needs to be compatible with this technology.				
Additional Information	The CEP exposes several monitoring metrics that are exported to Prometheus.				

Table 37 - CEP compatibility (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-11	Software	FUNC	Developer	DES
Name	Minio compatibility				
Description	The triple monitoring engine monitors the performance of Minio[17], which is used for object storage in the system. Therefore, the triple monitoring engine needs to be compatible with this technology.				
Additional Information	N/A				

Table 38 - Minio compatibility (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-12	Software	FUNC	Developer	DES
Name	OpenStack Networking Services compatibility				
Description	The Triple Monitoring engine monitors the performance of the internal network connecting the different containers inside an application. BigDataStack uses the OpenStack networking services for managing this network communications, so the triple monitoring engine needs to be compatible with this technology.				

[16] Openshift OKD (Origin Kubernetes Distribution). https://www.okd.io/
[17] Minio Private Cloud Storage- https://www.minio.io/

Additional Information	N/A

Table 39 - OpenStack Networking Services compatibility (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-13	Software	FUNC	Developer	MAN
Name	Persistently store the monitoring metrics				
Description	The triple monitoring engine should use a database for persistently storing monitoring metrics and is connected to Prometheus by http.				
Additional Information	This database is based on influxDB24.				

Table 40 - Monitoring database (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-14	Software	FUNC	Developer	ENH
Name	Spark Monitoring Pushgateway				
Description	This component is used to gather metrics from Spark and ingest them into the metrics collector.				
Additional Information	The connection between this component and the applications use http.				

Table 41 - Monitoring Pushgateway (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-16	Software	FUNC	Developer	ENH
Name	Metrics visualization				
Description	The metrics must be shown to the end-user via a graphical interface. Grafana is used for metrics' visualization.				
Additional Information	Grafana[18] is configured for receiving metrics from two sources (Prometheus, InfluxDB).				

Table 42 - Metrics visualization (software requirement).

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TM-17	System	FUNC	Dynamic Orchestrator	ENH
Name	Asynchronous rich notification of SLA violations				
Description	SLA violations should be notified by means of a publish/subscribe mechanism, together with the metrics (KPIs) upon which the SLA imposes restrictions.				

[18] Grafana - https://grafana.com/

Additional Information	The main consumer of the SLA violations notifications is the Dynamic Orchestrator.

Table 43 - Metrics visualization (software requirement).

8.2. Design

Figure 20 describes the high-level architecture of the Triple Monitoring Engine and QoS Evaluator components, following the requirements defined in Section 8.1. As it is shown, the metrics collector is in a central place, receiving information from the compatible technologies (REQ-TM-7 to REQ-TM-12) and feeding both the metrics getter and, subsequently, the pub/sub mechanism and the database, which is accessed by Grafana. Finally, the QoS evaluator accesses the metrics collector through the API to read and evaluate the KPIs.

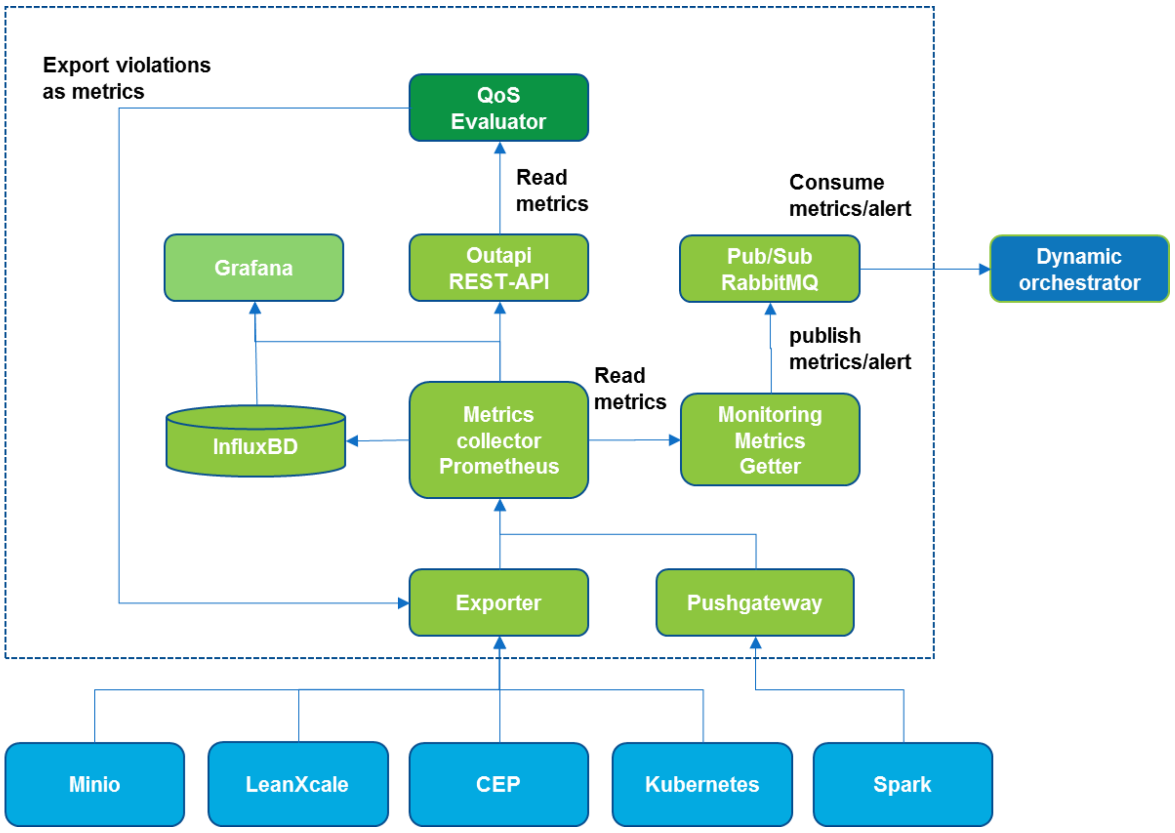


Figure 20 – Triple Monitoring Engine & QoS Evaluation – conceptual view.

Figure 20 shows the interaction between the triple monitoring engine (Prometheus for KPI monitoring and RabbitMQ for pub/sub). As it is shown, each component starts its own thread. First, Prometheus focuses solely on periodically retrieving measures for the KPIs and showing them to the user and administrators of the system. Second, RabbitMQ is used for managing the messages between components. Subscribers, such as the QoS Evaluator, register in the system and are notified every time that there has been a change in the data evaluated by Prometheus. The monitoring metrics getter directly queries Prometheus for obtaining these data and puts them to RabbitMQ.

The QoS Evaluator is designed synchronously react to Prometheus updates. Thus, it performs a periodical check on Prometheus' metrics and, for each value, the component compares it to the agreed QoS. If the minimum agreed QoS (minimum expected performance for the KPIs) has been violated, the QoS Evaluator simultaneously raises an alert to the system and communicates this violation to Prometheus, so violations are also monitored as a KPI.

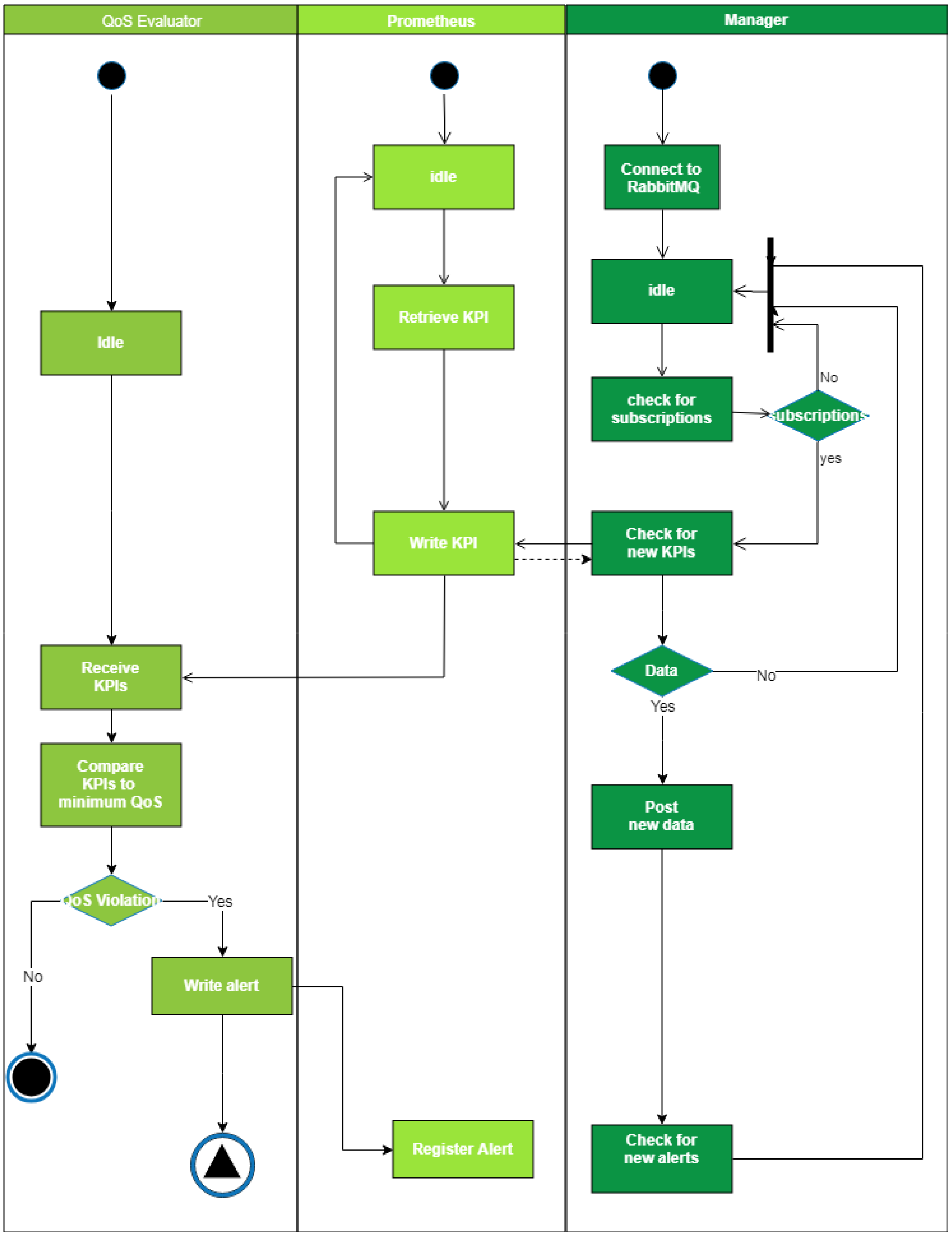


Figure 21 – Interaction between monitoring and QoS Evaluator components.

Figure 21 shows the interaction between the Triple Monitoring and QoS, and the rest of the system. As described before, the Triple Monitoring Engine can provide information on the performance of the system, compared against the expected QoS. This comparison, carried out in the QoS Evaluator component, triggers and alert every time that the minimum agreed QoS is not respected. This alert is intercepted by the ADS Deploy component, which decides if it is necessary to re-deploy the environment.

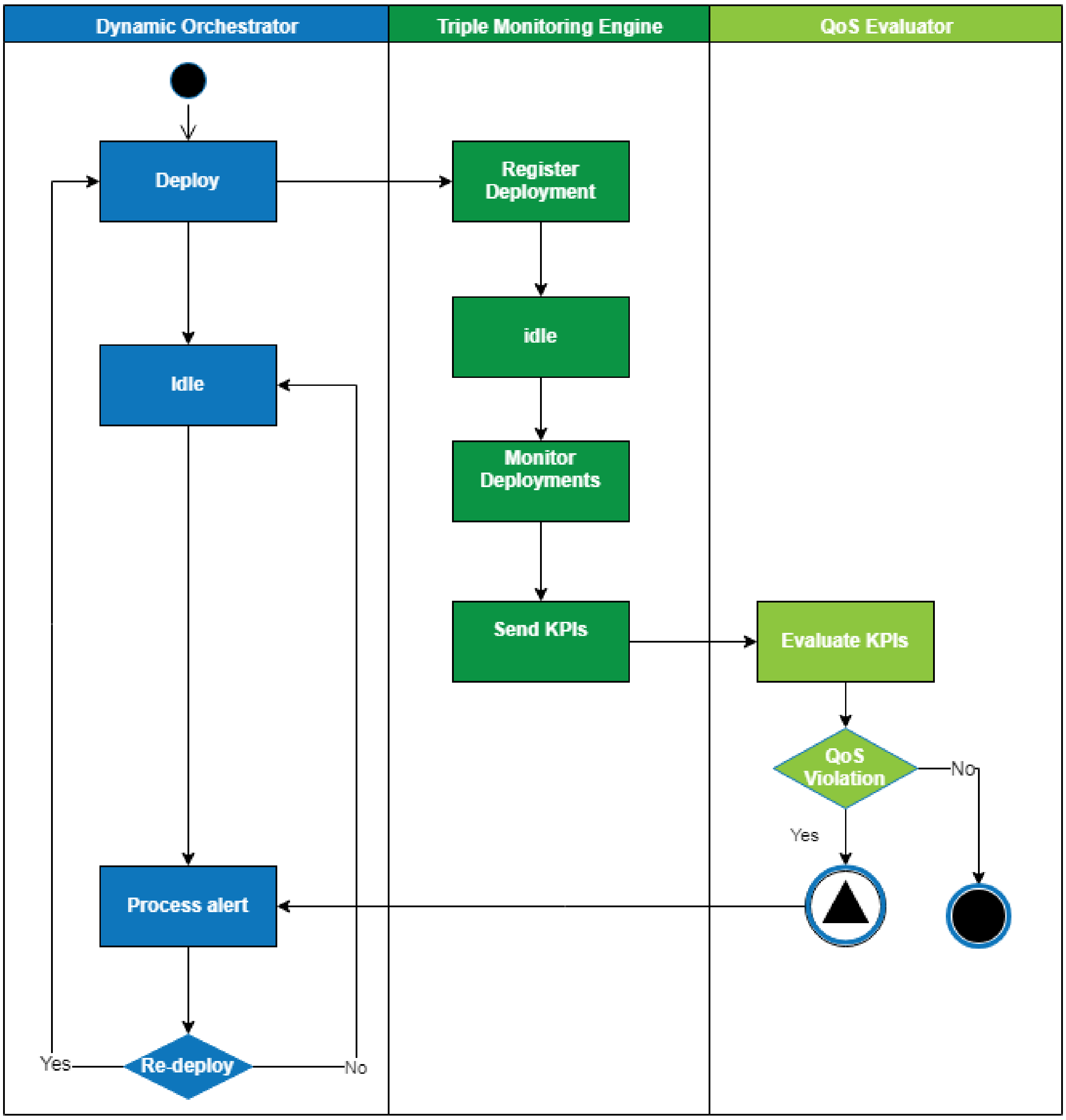


Figure 22 – Interaction between Triple Monitoring Engine, QoS Evaluator and ADS Deploy components.

8.2.1. Integration details: LeanXcale

LeanXcale provides monitoring information for its two major components (REQ-TM-8): the data nodes of its key-value distributed storage and the instances of its Query Engine. It is important to note that each deployed *data node* contains both an instance of the storage along with an instance of the Query Engine, so that the latter can exploit data locality. The query engine is written in Java and provides monitoring information using the *dropwizard* framework[19]. The advantage of the latter is that it can additionally provide statistical information on a monitoring metric, like mean time, mean time between a period, the histogram of the metric etc. Dropwizard can be used with a JMX[20] (Java Management Extensions) plugin which publishes the metrics as managed beans via the JMX. Other metrics are also published and are available directly via the JMX, while the use of the latter allows to take advantage of Java's built-in monitoring information which is available for every java virtual machine (i.e. number of threads, memory usage, garbage collection statistics, etc). Additionally, the usage of JMX to publish monitoring information makes the integration with Prometheus to rely on the JMX-Exporter. Query engine's monitoring information can be grouped by specific categories (version, network, logger performance, query executions, general information etc.).

Apart from the Query Engine, a data node also contains the key value storage nodes themselves, which are part of the adaptable distributed storage. The latter is written in C and can provide monitoring of low level information. Data node's built-in functionality exposes statistics and monitoring information in the standard output. Due to this, there has been implemented a routine that periodically takes this output as its input and transforms it to JSON files, known as metrics. Then, Prometheus can be configured to pull these files from the predefined location to load this monitoring information. However, this requires a static configuration and would need to restart Prometheus system, each time the storage is being scaled in/out and new nodes are added/removed. Due to this, an LXS monitoring proxy will be provided, that will be configured with Prometheus, and take the responsibility to collect the monitoring information of all data nodes at run-time, considering possible redeployments (for more information about the design of the Adaptable Distributed Storage see D4.1).

8.2.2. Integration details: CEP

The CEP provides monitoring information of the queries (REQ-TM-10). The UPM CEP exports all metrics through a component, the Metric Server. Figure 23 shows the metric server in the CEP, all CEP components (Instance Managers and Orchestrator) send their metrics using a push-based socket protocol to the Metric Server (black arrows). The Metric Server opens a java socket for receiving metrics from the other components and makes available these metrics to Prometheus through an HTTP server. Prometheus is configured with a new job targeting the Metric Server to pull the CEP metrics (green arrow).

The CEP leverages Prometheus aggregation functions to send raw metric values minimizing the overhead of metric processing in the components.

The list of available metrics is the following:

- Orchestrator
    - Is Active: *binary* value. 1 means the Orchestrator is running
    - Active Instance Managers: *int* value. Number of running Instance Managers.
    - Registered Queries: *int* value. Number of registered queries.
    - Deployed Queries: *int* value. Number of deployed queries.
- Instance Manager
    - Is Active: *binary* value. 1 means the Instance Manager is running

[19] https://metrics.dropwizard.io/4.0.0/
[20] https://en.wikipedia.org/wiki/Java_Management_Extensions

- CPU Load: *double* value. Percentage of CPU used by the Instance Manager Java process.
- Events received: *long* value. Counter of the events received by the IM
- Events sent: *long* value. Counter of the events sent by the IM.
- Results: *long* value. Counter of the result events sent to clients.
- Sub-query Instance counter: *long* value. Counter of the events processed by a sub-query instance.
- Operator counter: *long* value. Counter of the events processed by an operator.
- Operator latency: *double* value. Latency of the operator processing time.

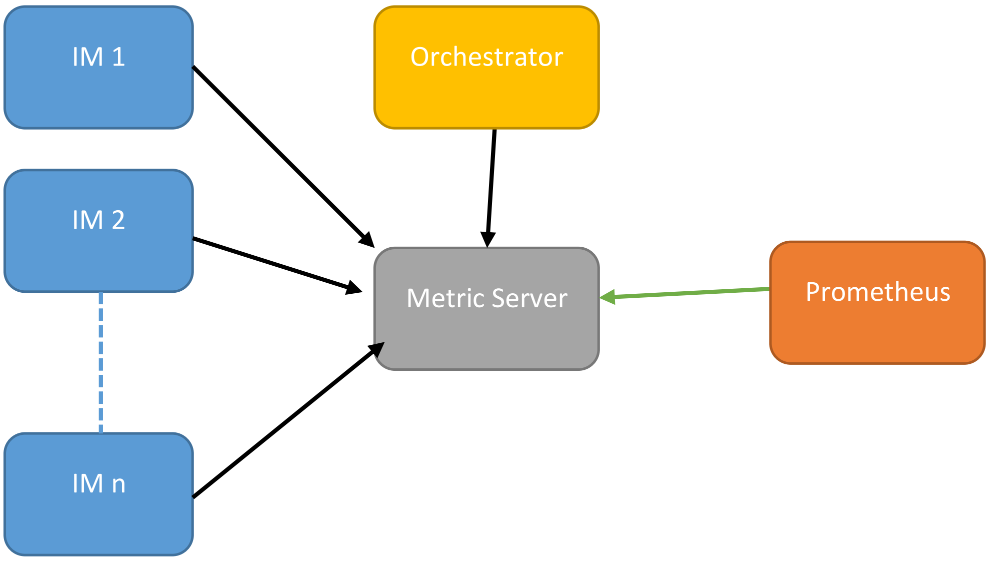


Figure 23 – Prometheus-CEP integration – conceptual view.

8.2.3. Integration details: Spark

Apache Spark is involved in BigDataStack for big data layout and data skipping as described in the deliverable 4.1 section 6. Thus, IBM and NEC are interested by metrics generated by Apache for they components for improvement, prototyping and adaptation reasons. BigDataStack will use SparkMeasure[21] project which needs to be imported in a Spark driver (spark application). For each SQL query executed, a set of metrics will be gathered then send to the triple monitoring engine via the *pushgateway* as described in the Figure 9. Spark Measure establishes connection with different spark executors created for executing SQL query then collects all metrics produced. Since those values concern a single SQL execution but they are coming from different executors, Spark Measure aggregates them (sum, max) before being dispose to the end user. The approach used until now in BigDataStack is to format those metrics for being compatible with Prometheus naming standard.

[21] Spark Measure, https://github.com/LucaCanali/sparkMeasure

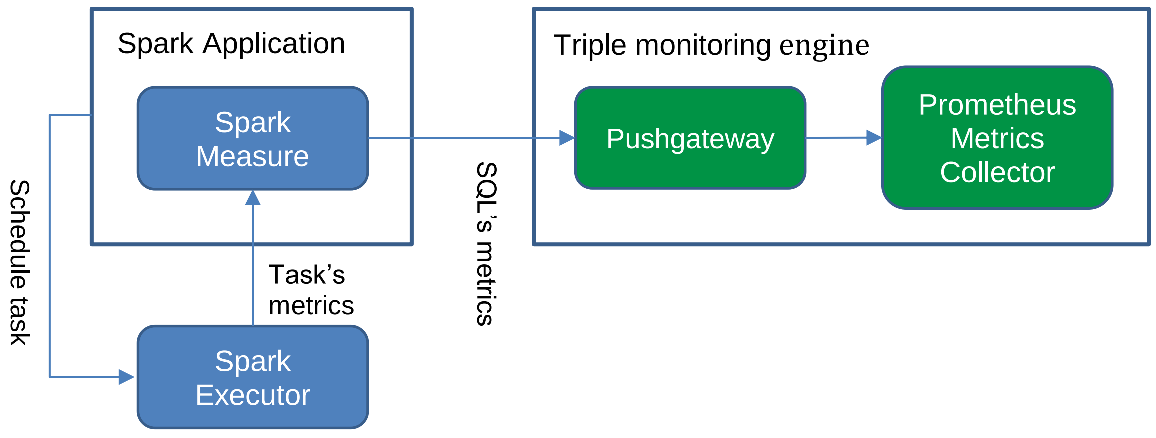


Figure 24 – Prometheus-Spark integration – conceptual view.

The list of available metrics generated by this library is the following:

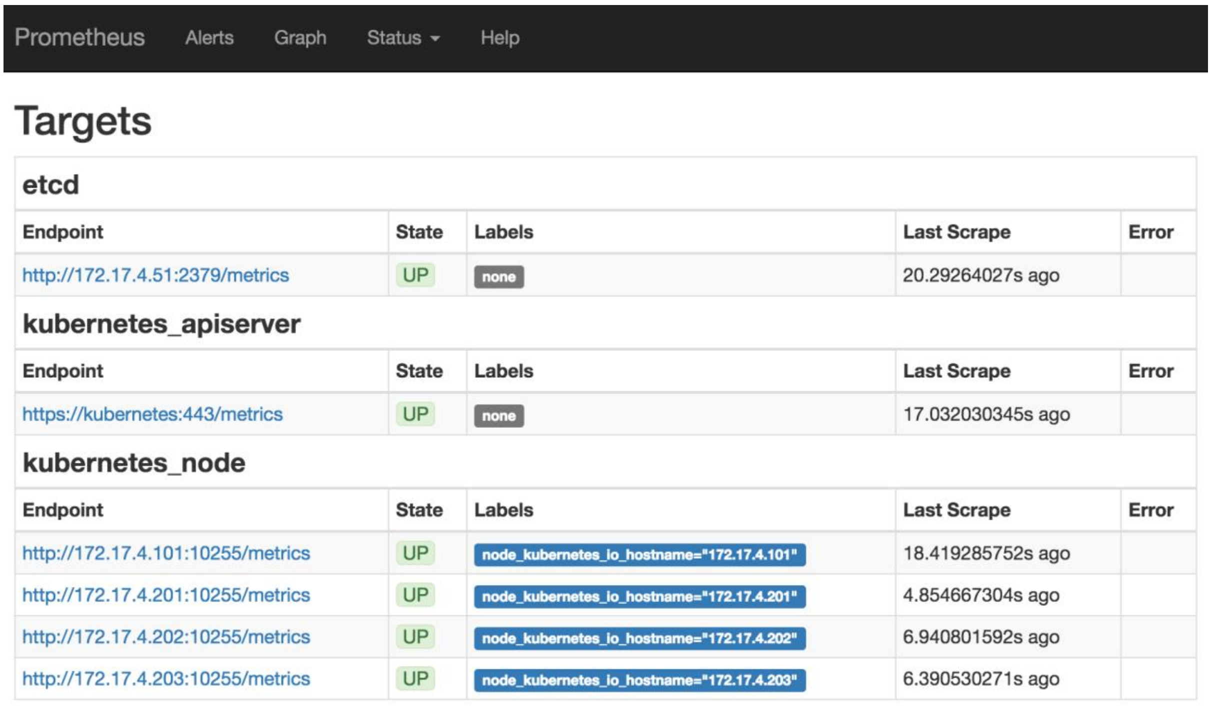
- *numStages*
- *numTasks*
- *elapsedTime*
- *stageDuration*
- *lexecutorRunTime*
- *executorCpuTime*
- *executorDeserializeTime*
- *executorDeserializeCpuTime*
- *resultSerializationTime*
- *jvmGCTime*
- *shuffleFetchWaitTime*
- *shuffleWriteTime*
- *resultSize*
- *numUpdatedBlockStatuses*
- *diskBytesSpilled*
- *memoryBytesSpilled*
- *peakExecutionMemory*
- *recordsRead*
- *bytesRead*
- *recordsWritten*
- *bytesWritten*
- *shuffleTotalBytesRead*
- *shuffleTotalBlocksFetched*
- *shuffleLocalBlocksFetched,*
- *huffleRemoteBlocksFetched*
- *shuffleBytesWritten*
- *shuffleRecordsWritten.*

8.3. Early Prototype

In an early prototype, a Prometheus deployment has been integrated with the QoS Evaluator. The communication between both is done directly, bypassing RabbitMQ, through Prometheus' API REST. Prometheus has been configured to evaluate the following low-level criteria:

- Ingestion of metrics from the three levels:
    - Resource clusters: Kubernetes / OKD
    - Data services: LeanXcale, Minio, Spark, CEP
    - Application services

The QoS Evaluator has also been deployed and integrated with the system, along with a minimum QoS based on the value of the "spark response time" variable. If the value of this KPI goes over certain Service-Level Objective or SLO (threshold or setpoint), then the QoS is considered to have been violated and the system registers it. This information is accessible through Prometheus and Graphana.



Targets

etcd

Endpoint	State	Labels	Last Scrape	Error
http://172.17.4.51:2379/metrics	UP	none	20.29264027s ago	

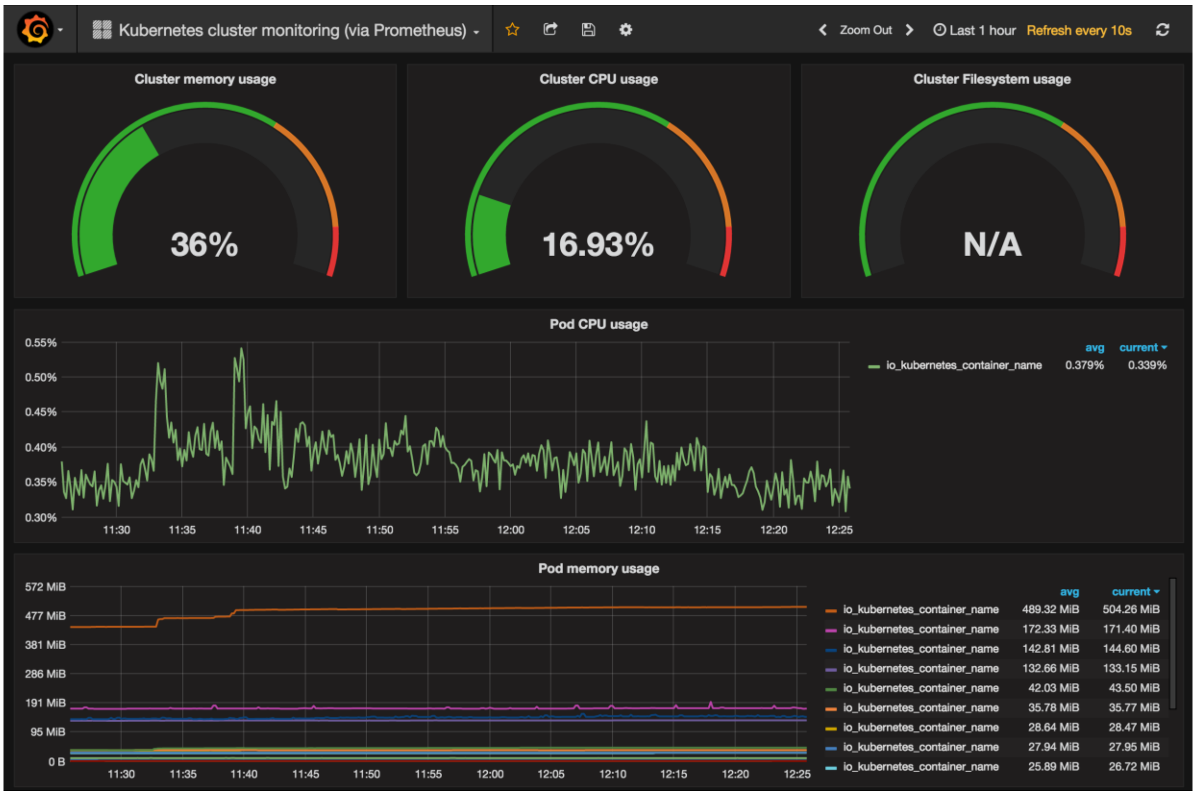
kubernetes_apiserver

Endpoint	State	Labels	Last Scrape	Error
https://kubernetes:443/metrics	UP	none	17.032030345s ago	

kubernetes_node

Endpoint	State	Labels	Last Scrape	Error
http://172.17.4.101:10255/metrics	UP	node_kubernetes_io_hostname="172.17.4.101"	18.419285752s ago	
http://172.17.4.201:10255/metrics	UP	node_kubernetes_io_hostname="172.17.4.201"	4.854667304s ago	
http://172.17.4.202:10255/metrics	UP	node_kubernetes_io_hostname="172.17.4.202"	6.940801592s ago	
http://172.17.4.203:10255/metrics	UP	node_kubernetes_io_hostname="172.17.4.203"	6.390530271s ago	

Figure 25 – Configuration of the Kubernetes monitoring in Prometheus.



	avg	current
io_kubernetes_container_name	0.379%	0.339%

	avg	current
io_kubernetes_container_name	489.32 MiB	504.26 MiB
io_kubernetes_container_name	172.33 MiB	171.40 MiB
io_kubernetes_container_name	142.81 MiB	144.60 MiB
io_kubernetes_container_name	132.66 MiB	133.15 MiB
io_kubernetes_container_name	42.03 MiB	43.50 MiB
io_kubernetes_container_name	35.78 MiB	35.77 MiB
io_kubernetes_container_name	28.64 MiB	28.47 MiB
io_kubernetes_container_name	27.94 MiB	27.95 MiB
io_kubernetes_container_name	25.89 MiB	26.72 MiB

Figure 26 – Kubernetes metrics visualization in Grafana.

In CEP early prototype, a module has been implemented to export metrics to Prometheus. Moreover, Grafana has been integrated in the CEP deployment to create a Dashboard and to check that metrics are being as expected. Different kind of dashboards can be created to observe different metrics obtained directly from Prometheus and to create aggregated metrics from the previous ones.

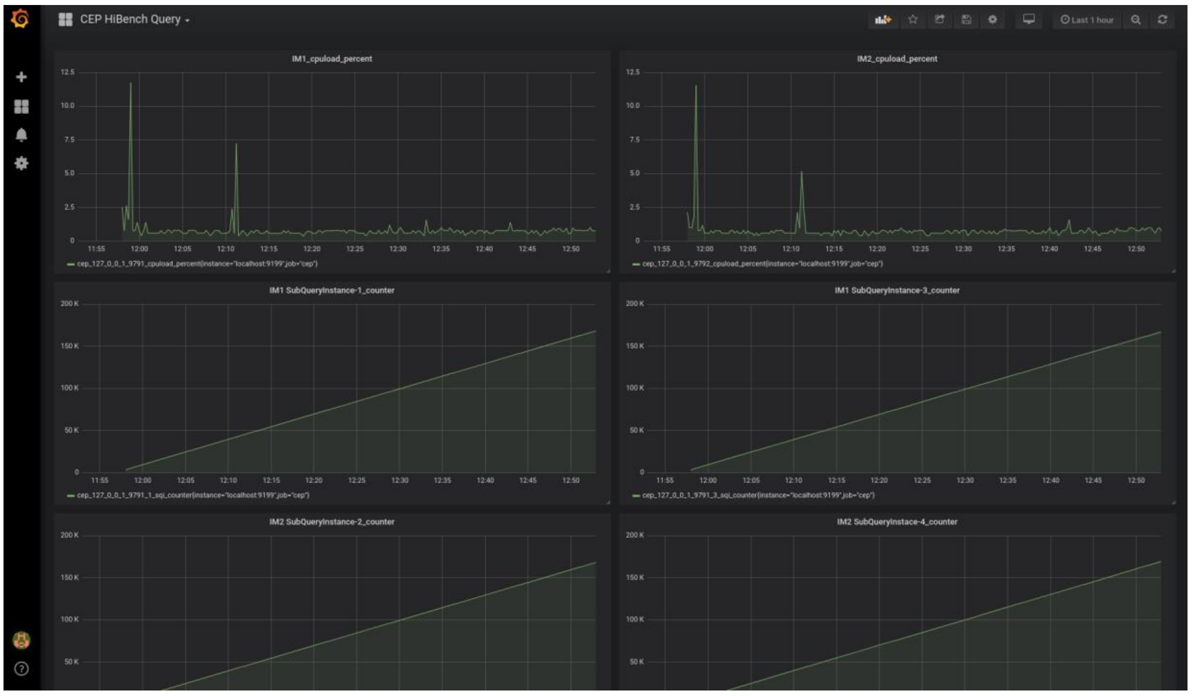

CEP HiBench Query
Last 1 hour
IM1_cpuload_percent
IM2_cpuload_percent
IM1 SubQueryInstance-1_counter
IM1 SubQueryInstance-3_counter
IM2 SubQueryInstance-2_counter
IM2 SubQueryInstace-4_counter

Figure 27 – CEP components performance visualization with Grafana.

Figure 27 shows how these metrics are visualized in Graphana. There are two instance managers, IM1 and IM2. The two graphs on the top show the CPU consumption evolution of each instance manager. Both instance managers run two subqueries (SubQueryInstance-1 and SubQueryInstance-3 in IM1 and *SubQueryInstance*-2 and *SubQueryInstance*-4 in IM2). The evolution of the throughput of each subquery is shown in the other graphs.

Next steps for this first early prototype are the deployment of the CEP platform in the testbed and the integration with the Prometheus and Graphana process that will be running in the same Big Data Stack testbed.

8.4. Use Case Mapping

The Triple Monitoring and QoS Evaluation components will be monitoring and evaluating certain key-performance indicators (KPIs) which need to be kept above a certain Service-Level Objective (SLO) specified by the data scientist of the application engineer.

In the case of the ATOS WORLDLINE-EROSKI use case, and following the experimental scenarios described at Section 4.1, the first experiments will evaluate and enforce two KPIs: the *response time* and *throughput* of the analytics services comprising a recommender system. While the Triple Monitoring collects metrics at different levels of the BigDataStack platform (application, data services and resource cluster) to compute the KPIs, the QoS Evaluation analyses the evolution of those KPIs to determine whether the SLOs are being met; If this is not the case, it notifies the QoS violations to the Dynamic Orchestrator.

8.5. Experimental Plan

As discussed, the Triple Monitoring Engine and QoS Evaluator components assess the performance of a particular CDP Playbook in the system. These components support the ADS-Ranking in choosing the most optimal CDP Playbook to ensure performance.

During the experimental plan it will be demonstrated that metrics at different levels (system and application level) can be dynamically monitored and evaluated as part of the same Quality of Service KPI.

8.5.1. Background and Related Work

The performance of a Cloud service has already been addressed in bibliography. For example, de Vaulx et al. [4] developed a model for the performance of the Cloud at an application level (Quality of Service, availability, reliability, etc.). This is consistent with the services offered by most Cloud providers, which ensure the user a minimum availability time during the lease. On the other hand, the cloud provider is interested in optimizing the performance and utilization of the data centre at a system level [5], (manageability, fault tolerance, energy consumption, etc.).

These two levels of evaluation are opposed (the user pushes for a better QoS, while the provider requires a more efficient use of resources), and there is not a standard approach to unifying the concerns of both. A compromise between the parts is usually the approach, economical (the provider makes a worse use of resources and the price to the user is increased), moral [6], etc. However, to the best of our knowledge, there is not a centralized approach to ensure the performance of metrics at both levels (application and system level) simultaneously.

8.5.2. Evaluation Methodology and Metrics

The testbed used for this experimental plan will be the same as described in Section 4, keeping consistency with the rest of components. Data will be retrieved from real-life scenarios provided by the Use Case partners, which are also described in Section 4. The metrics retrieved and evaluated will be related with the UCs, while exposing the capabilities of the system to evaluate simultaneously characteristics at the two levels.

From the UCs we obtain two main metrics: throughput and response time. The former evaluates the environment metrics (platform level), while the latter measures the behaviour of the application towards the user (application level).

The metrics considered at a platform level are:

- *Disk usage* (%): The percentage of disk which is being used.
- *Memory usage* (%): The percentage of memory which is being used.
- *CPU usage*: Distribution of load in the CPU.

The metrics considered at the application level are:

- *Availability* (% of time): Percentage of time which the application is down.
- *CPU*: Minimum computational power claimed by the application.

These metrics will be weighted according to the application needs. The weights of each metric's value will be determined in future iterations.

During the experimental evaluation a set of experiments will be run, where the data will be unchanged, and the virtual infrastructure is adapted. These experiments target to evaluate the response of the triple monitoring engine and QoS evaluator under different situation (under-provision of resources, near-optimal and over-provision of resources). The experiment will evaluate the number of QoS violations per unit of time, namely:

- *Platform metrics violations*: Violations involving the system's performance.
- *Application metrics violation*: Violations involving the application's performance.
- *Mixed metrics violation*: Violations involving any/all the above metrics.

The experiments will start on a system with solely 1 container, an extreme case of under provision. In each new experiment, the number of containers will be increased by a certain amount (to be determined in future iterations by observation of the data and expected computational requirements). As the number of containers increases, the violations in the system will go from the application level (under provision of resources) to the platform level (over provision of resources). During these experiments, it is expected that the near-optimal configuration will be reached. The set of experiments will stop once it is proven that adding new containers does not increase the performance of the system.

8.6. Next Steps

The next steps regarding the Triple Monitoring and QoS Evaluator components are many-fold:

- Integrating the communication with RabbitMQ: While RabbitMQ has already been deployed, it has yet to be used to deliver messages between the QoS Evaluator and the Triple Monitoring Engine. In the next phase, it will be forced the communication in a pub/sub manner, using RabbitMQ.

- Complete integration with the rest of technologies
- Integrating Prometheus with a prototype application: Once an application is deployed, Prometheus will be integrated with it to measure real-life KPIs.
- Choose application metrics
- Running experimentation

9. Information-Driven Networking

The Information-Driven Networking mechanisms provide a set of functionalities for traffic engineering and network management by taking into consideration inter- and intra-knowledge and requirements of data intensive operations and applications. These requirements concern services prioritization, time criticality constraints and security aspects and are controlled by means of labels and selectors to enforce specific network policies. This is achieved by defining through rules the connections that are allowed or not allowed to specific services or specific nodes in the BigDataStack cluster.

The outcome of the Information-Driven Networking mechanisms will be to translate these requirements into networking primitives that achieve the desired dissemination, regulatory compliance and sharing of the information in the BigDataStack environment.

9.1. Requirements specification

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.2 and literally included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.2, and that they are included in here for the reader's convenience.

	Id	Level of detail	Type	Actor	Priority
	REQ-IN-01	Software	FUNC	ROL-02	MAN
Name	Information-Driven Networking based on type of data				
Description	The Information-Driven Networking mechanisms enforce a set of policies by specifying the rules of how two or more components can communicate (send/receive data) with each other according to the available resources.				
Additional Information	A different policy is enforced based on different incoming data requirements, following the type of processing requirements (stream, micro-batch, batch) and the type of data (structured, semi-structured, unstructured).				

Table 44 - Network Policies based on type of data (software requirement).

	Id	Level of detail	Type	Actor	Priority
	REQ-IN-02	Software	FUNC	ROL-02	MAN
Name	Information-Driven Networking based on application requirements				
Description	The Information-Driven Networking mechanisms enforce a set of policies by specifying the rules of how to handle applications with different requirements according to the available resources. For instance, an application with analytics requiring real-time data processing may impose time-critical constraints on the handling, operation and transformation of data.				

	To support online analytics and decision making in time-critical conditions specific network policies need to be applied to deliver the results within predefined time constraints.
Additional Information	The Data Scientist can set an “allow/deny access” policy regarding the set of applications and their requirements (real-time, close to real-time needs) accessing the backend services of the BigDataStack environment to prioritize/isolate the set of ingress/egress workloads that are enabled/dis-based on their IP & Port in order to achieve efficient services interaction.

Table 45 - Network policies based on application (software requirement).

9.2. Design

Through the Information-Driven Networking tool the Data Scientist declares her intend to be realized by the underlying system to translate either the data or the application requirements into specific networking primitives that achieve the desired Service-Level Objective (SLO). This objective may refer to various kinds of traffic – streams, batches and micro batches – get the isolation/priority of availability and bandwidth that are needed to serve the network users effectively. With the convergence of all data and services in the same network, the Information-Driven Networking will manage traffic according to the network utilisation, the applications requirements and the communication latency without compromising the functionality of the network. Using policy statements, either the Network Administrators or the Data Scientists can specify which kinds of service need to be given priority, at what times and on what part of their IP based protocol.

As all the mandatory building blocks of BigDataStack are containerized, a pod representing the basic building block in Kubernetes, encapsulates an application container (or multiple containers). Therefore, a set of labels and selectors need to be defined to assign key/value pairs to pods and set up the expressions that combine these labels to identify the traffic from/to individual containers, virtual machines and hosts that it needs to be handled before it is routed/delivered to its destination. Then, the network policy definition includes a pod selector and the rules that apply to all the pods that meet the selector criteria. These rules are applicable to egress and ingress resources establishing connections to the pods, refer to labels with specific IPs or IP ranges and can permit or restrict communication to specific ports or allow/deny access to/from specific namespaces. For instance, there may be various namespaces serving different needs such as client and UIs services/applications. To configure network policies enforcement specific services (frontend, backend) need to be exposed to specific namespaces (client, UIs). In the following, we present an example of controlling ingress traffic by giving an indicative network policy definition.

```
kubectl create -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

  name: access-nginx

  namespace: sample-policy-demo
```

```
spec:
  podSelector:
    matchLabels:
      run: nginx
  ingress:
    - from:
      - podSelector:
          matchLabels: {}
EOF
```

Table 46 – An indicative network policy definition for ingress traffic.

To address the challenges of a specific application, its requirements and the respective policies enforcement, a set of mechanisms operating at the services layer are expected to set up the appropriate attributes to understand the virtual hosts, URLs and other HTTP headers. This functionality implements the policy enforcement endpoint inside the pod as sidecar container in the same network namespace. This approach is highly flexible and HTTP aware and facilitates to apply policies in support of **operational goals**, such as service routing, retries, circuit-breaking, etc.

Containers networking is realised by Networking as a Service (through Neutron in OpenStack) and easily deployed containers (through Magnum either as Virtual Machines or Physics Machines). The idea is to bridge networking functionalities supported by Neutron for Containers Use Cases using abstraction mechanisms (exploiting functionalities of Kuryr[22], presented in section 5). The outcome is to deliver Neutron networking and services to Docker containerised services.

The Information-Driven Networking mechanisms also operate at the network layer. The latter gives the advantage of being universal. Our focus is to address the challenges arising from the diverse **data types** (i.e., stream, micro-batch, batch) to enforce policies to DNS, storage services (i.e., scalable storage of LeanXscale, Object Store, etc.), real-time streaming, and a plethora of other services that do not use HTTP. This functionality implements the policy at the host node outside the network namespace of the guest pods. The workloads in the BigDataStack environment can communicate without IP encapsulation or network address translation for bare metal performance, which enables easier troubleshooting, and better interoperability. In settings that require an overlay, the Information-Driven Networking mechanisms will work with tunnelling. This approach is universal, highly efficient, and isolated from the pods and facilitates to apply policies in support of **security and data privacy goals**. In the following, we present an example of controlling communications to HTTP GET requests by giving an indicative network policy definition which consists of three policy objects.

```
# Restricting customer’s communications to HTTP GET requests.
kind: SampleNetworkPolicy
```

[22] Kuryr. https://wiki.openstack.org/wiki/Kuryr

```
metadata:
  name: customer_app
spec:
  selector: app == 'customer_app'
  ingress:
    - action: Allow
      http:
        methods: ["GET"]
  egress:
    - action: Allow
```

```
# The customer_app is the consumer of this service. Restricting incoming connections to customer_app.
kind: SampleNetworkPolicy
metadata:
  name: summary
spec:
  selector: app == 'summary'
  ingress:
    - action: Allow
      source:
        serviceAccounts:
          names: ["customer_app"]
  egress:
    - action: Allow
----------------------------------------------------------------------------------------
# Restricting access to LXS. Only the summary microservice has direct access to LXS data base.
kind: SampleNetworkPolicy
metadata:
  name: LXS_db
spec:
  selector: app == 'LXS_db'
  ingress:
    - action: Allow
      source:
        serviceAccounts:
```

```
        names: ["summary"]
  egress:
    - action: Allow
```

Table 47 – An indicative network policy definition for controlling HTTP GET requests.

In the following figure, we present the high-level functionalities of the Information-Driven Networking tool in a UML Components diagram.

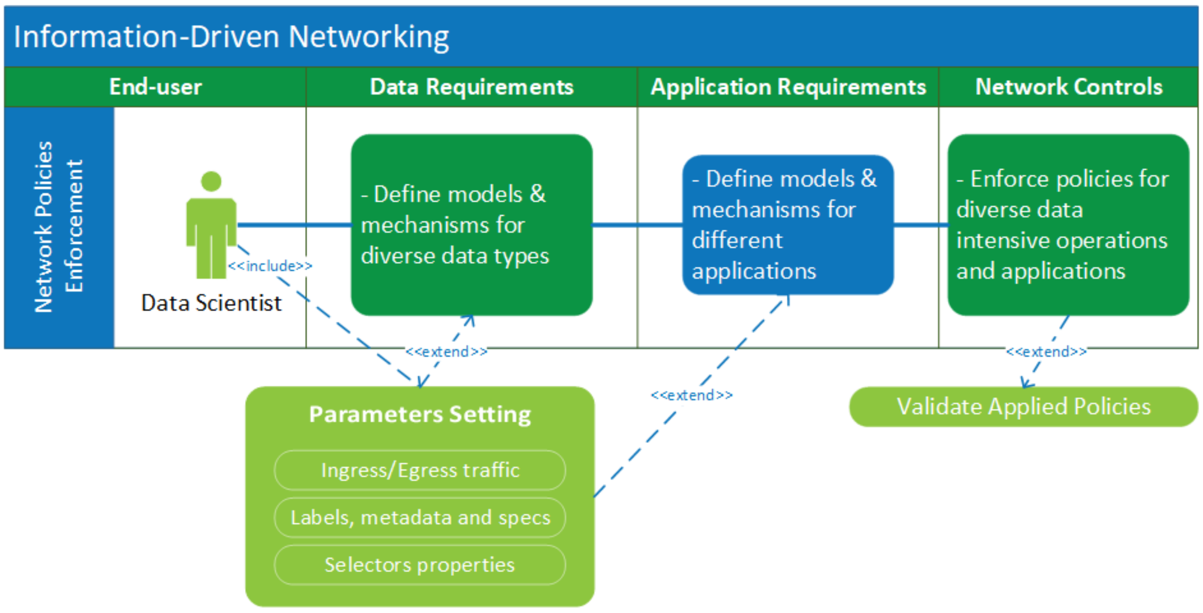

Information-Driven Networking
End-user
Data Requirements
Application Requirements
Network Controls
Network Policies Enforcement
Data Scientist
<<include>>
- Define models & mechanisms for diverse data types
- Define models & mechanisms for different applications
- Enforce policies for diverse data intensive operations and applications
<<extend>>
<<extend>>
<<extend>>
Parameters Setting
Ingress/Egress traffic
Labels, metadata and specs
Selectors properties
Validate Applied Policies

Figure 28 – Information-Driven Networking UML.

9.3. Early Prototype

The Information-Driven Networking is not currently delivering an early prototype as it starts at M13. However, a thorough review of Software Defined Networking (SDN) mechanisms in cloud environments has been made to start the development before M13 and make available an early version of the Information-Driven Networking in M18.

9.4. Use Case Mapping

The **Data Scientist** uses the **Information-Driven Networking tool**, to define metadata and means of communication to apply tailored controls to data intensive operations and applications related with analytic tasks according to specific requirements, by also including:

- The identification of the end-to-end application objectives in terms of specifying KPIs and criteria for optimal networking management and engineering;
- The definition of the constraints arising from the type of data to be processed (data transfer, liveness, readiness among services) and the requirements of the application (time criticality, security, privacy);
- The validation of the applied network controls by evaluating that the policies have been correctly enforced and that resources are distributed among consumers as requested.

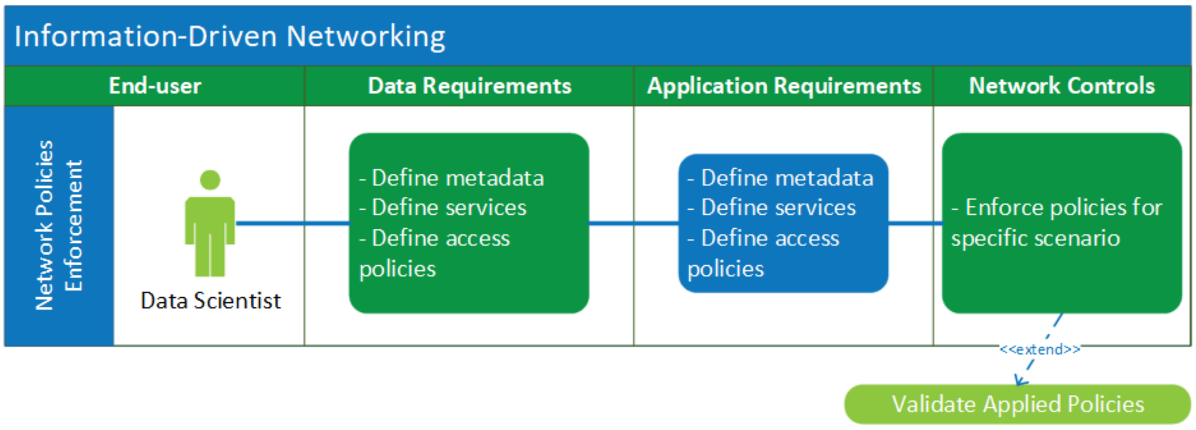


Figure 29 – Mapping of Information-Driven Networking tool with BDS Use Cases.

9.5. Experimental Plan

The Experimental Plan regarding the Information-Driven Networking includes to firstly set up the services and resources interacting within the BDS cluster and then conduct a set of scenarios including simple and more complex policies at network and application level. The simple policies include a set of isolation rules allowing/restricting access to specific pods and then we will proceed with more complex policies which may include the management of frontend and backend services.

This plan includes diverse configuration in *yaml* files with different *policyTypes* (i.e. Ingress, *Egress*) and *PodSelector* (*matchLabels* in/out app) to validate when specific policy type is enforced (i.e., allowed/restricted).

9.6. Next Steps

In the proceeding time period, we will work on with the early development of the Information-Driven Networking mechanisms and their experimentation with variable scenarios by enforcing different policies in diverse application requirements and for multiple constraints imposed by the data types. The use cases and the involvement of the Data Scientists will facilitate to address the main functionalities and deployment considerations of this tool coupled with the expressed requirements.

10. References

[1] Network Policies in Kubernetes. Available Online: https://kubernetes.io/docs/concepts/services-networking/network-policies/

[2] Project Calico. Available Online: https://www.projectcalico.org/

[3] Istio. Available Online: https://istio.io/

[4] de Vaulx, Frederic J., Eric D. Simmon, and Robert B. Bohn (2018). "Cloud computing service metrics description." Special Publication (NIST SP)-500-307. 2018.

[5] William Voorsluys, James Broberg, Srikumar Venugopal, Rajkumar Buyya, Martin Gilje Jaatun, Gansen Zhao, Chunming Rong (2009). "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation", Cloud Computing, Springer Berlin Heidelberg, 2009, P 254-265

[6] D. Guyon, A. Orgerie, C. Morin and D. Agarwal (2017). "How Much Energy Can Green HPC Cloud Users Save?" in 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), St. Petersburg, 2017, pp. 416-420.

[7] Gulisano, V., Jimenez-Peris, R., Patino-Martinez, M., Soriente, C., & Valduriez, P. (2012). "Streamcloud: An elastic and scalable data streaming system." IEEE Transactions on Parallel and Distributed Systems, pp. 2351-2365.

[8] H. Rui et al. (2014). "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications." Future Generation Computer Systems, pp. 82-98.

[9] Kalervo and Jaana. (2002). "Cumulated gain-based evaluation of IR techniques." ACM Transactions on Information Systems (TOIS), pp. 422--446.

[10] L. Tie-Yan. (2009). "Learning to rank for information retrieval." Foundations and Trends in Information Retrieval, pp. 225-331.

[11] M. Ferdman et al. (2012). "Clearing the clouds: a study of emerging scale-out workloads on modern hardware." ACM SIGPLAN Notices, pp. 37-48. ACM.

[12] Raschke, R. (2010). "Process-based view of agility: The value contribution of IT and the effects on process outcomes." International Journal of Accounting Information Systems, 11(4), pp. 297-313.

[13] Salton and McGill. (1986). "Introduction to modern information retrieval." McGraw-Hill, Inc.

[14] Sergey and Christian. (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint.

[15] Z. Jia et al. (2013). "Characterizing data analysis workloads in data centers." IEEE International Symposium on Workload Characterization (IISWC), pp. 66-76. IEEE.