D4.7 Final release of the Cloud Data Analytics Service Management and Scalability Components

Version 1.0

Document Information

Contract Number	780622
Project Website	https://class-project.eu/
Contractual Deadline	M29, May 2020 (Due to COVID situation this deliverable has been submitted on M31, July 2020)
Dissemination Level	PU
Nature	DEM
Author(s)	Roi Sucasas (ATOS)
Contributor(s)	(ATOS)
Reviewer(s)	Erez Hadad (IBM); Elli Kartsakli (BSC)
Keywords	Cloud, Edge, WP3, WP4, Rotterdam, Container-as-a-Service (CaaS), Docker, Kubernetes, Openshift, container orchestration

Notices: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "780622".*

Change Log

Version	Date	Author	Description of Change
V0.1	20/07/20	Roi Sucasas (ATOS)	First draft version
V0.2	29/07/20	Roi Sucasas (ATOS)	Merged contribution from partners
V0.3	30/07/20	Erez Hadad (IBM)	Internal Revision
V1.0	31/07/20	Elli Kartsakli (BSC)	Final Version, Ready to EC revision

Table of contents

Table of contents 3
Table of Figures 5
Terms and Abbreviations 6
Executive Summary 7
1 Introduction 8
1.1 About this deliverable 8
1.2 Relation to other deliverables and work packages 8
1.3 Structure of the document 9
1.4 Glossary adopted in this document 9
2 Functional Description 11
2.1 Transparent lifecycle management of data analytic workloads in multiple Cloud and Edge clusters 12
2.2 Creation and management of connections to multiple Cloud and Edge containers orchestrators 12
2.3 Deployment and management of applications and serverless functions in Edge devices 13
2.4 Real-time QoS guarantees, SLA management and data analytics service scalability 13
2.5 Performance monitoring of data analytics workloads and infrastructures 14
3 Technical description 14
3.1 Major changes in final release 14
3.2 Baseline Technologies and dependencies 15
3.3 Architecture 17
3.3.1 Rotterdam 19
3.3.2 SLA Manager & Monitoring tools 20
3.4 Deployment Diagrams 21
3.4.1 CLASS Cloud standalone environment in Modena Data Center 23
3.5 Interfaces Provided 23
3.5.1 Rotterdam 23
3.5.2 SLA Manager 25
4 Installation and usage guides 27
4.1 Packages distribution and requirements 27
4.2 Installation 27
4.2.1 Container orchestrator 27

4.2.2 Monitoring tools 28
4.2.3 Rotterdam 28
4.2.4 SLA Manager 30
4.3 Usage 31
4.3.1 Rotterdam tasks 34
4.3.2 QoS templates 36
4.3.3 Infrastructures 36
4.3.4 Serverless functions 37
4.3.5 Usage example: deployment and scalability 38
4.3.6 Usage example: MicroK8s in Edge device 42
5 Demonstration 47
5.1 Scenario description 47
5.2 Integration with COMPSs 48
5.2.1 Demo 49
5.3 Management of multiple clusters in Edge and Cloud 55
5.3.1 Demo 56
6 Conclusion 61
References 62

Table of Figures

Figure 1 – CLASS architecture & WPs 8
Figure 2 – Cloud computing platform in the context of the CLASS architecture 17
Figure 3 – Rotterdam and internal components 19
Figure 4 – SLA Manager and monitoring tools components 21
Figure 5 – Data Analytics Service Management and Scalability Cloud Infrastructure 22
Figure 6 – Data Analytics Service Management and Scalability Multi Cloud and Edge Infrastructure 22
Figure 7 – Modena Data Center deployment 23
Figure 8 – OKD Web interface – Rotterdam deployment 29
Figure 9 – OKD Web interface – Rotterdam 29
Figure 10 – OKD Web interface – SLA Manager 31
Figure 11 – OKD Web interface - Rotterdam and SLA Manager running in “default” namespace 31
Figure 12 – Rotterdam Swagger REST API – Tasks methods 32
Figure 13 – Rotterdam REST API – Infrastructure and QoS methods 32
Figure 14 – OKD GUI – empty “class” project / namespace 38
Figure 15 – OKD GUI – “default” project / namespace with Rotterdam and the SLA Manager 38
Figure 16 – Rotterdam (swagger) REST API – QoS template definition 39
Figure 17 – Rotterdam (swagger) REST API - Task definition 39
Figure 18 – Rotterdam (swagger) REST API – Task deployment result / response 40
Figure 19 – OKD GUI – nginx server deployment in “class”namespace 40
Figure 20 – OKD GUI - nginx server deployed and ready 41
Figure 21 – nginx server application 41
Figure 22 – SLA Manager REST API – SLA 42
Figure 23 – OKD GUI - nginx server’s instances are halved after SLA violation 42
Figure 24 – Rotterdam (swagger) REST API – Infrastructure creation 43
Figure 25 – Rotterdam (swagger) REST API – Infrastructure creation response 43
Figure 26 – Rotterdam (swagger) REST API – list of managed infrastructures 44
Figure 27 – Rotterdam (swagger) REST API – MicroK8s deployment 44
Figure 28 OKD GUI – Rotterdam logs (MicroK8s deployment) 45
Figure 29 – Rotterdam (swagger) REST API – task deployment 45
Figure 30 – Rotterdam (swagger) REST API –response of task deployment 46
Figure 31 – nginx server application running in the Edge device 46
Figure 32 – Edge device console 46
Figure 33 – Openshift (Modena Data Center) and Edge device used in the demos 48
Figure 34 – Integration with COMPSs master application 49
Figure 35 – Management of multiple clusters and applications 55

Terms and Abbreviations

Acronym	Definition
D	Deliverable
WP	Work Package
M	Month
MS	Milestones
QoS	Quality of Service
IoT	Internet of Things
SLA	Service Level Agreement
K8s	Kubernetes[1]
MicroK8s[2]	Micro Kubernetes
COMPSs	Component Superscalar framework (from BSC)
OKD	Openshift Kubernetes distribution
UCs	Use Cases

[1] Kubernetes is an open-source container-orchestration system for automating application deployment, scaling, and management: https://kubernetes.io/

[2] Kubernetes version for IoT, Edge devices, workstations etc. https://microk8s.io/

Executive Summary

This deliverable (D4.7) accompanies the final release of the "Cloud Data Analytics Service Management and Scalability" components for the CLASS project. This includes the results of WP4 tasks, "*4.2. Data Analytics Service Management*" and "*4.3. Data Analytics Service Scalability*", done between months M16-M29, and the ATOS contributions to "*WP3 Edge Computing*" in task "*3.2. Develop, experiment and evaluate edge platform agent for analytics*".

As both deliverables D4.3 ("*Final release of the Cloud Data Analytics Service Management components*") and D4.5 ("*Final release of the Cloud Data Analytics Service Scalability components*"), as planned according to the CLASS DOW [1], are very related to each other, it has been decided to merge[3] their content into this new deliverable, thus providing a more complete picture of the envisioned Cloud computing environment.

The scope of the current deliverable comprehends:

- A functional and technical description of the final release of the cloud (and edge) standalone environment, including the requirements, detailed designs and scientific findings
- the installation manuals and technical documentation of the different software components found in the overall final prototype

[3] This change has been communicated to the Project Officer and is in the process of being formally approved through an amendment.

1 Introduction

1.1 About this deliverable

The main goal of CLASS is to deliver a platform to support the development of big data analytics in Edge and Cloud for Smart Cities and Connected Cars Use Cases, including the provision of QoS guarantees. In this context, one of the main objectives is to provide a cloud computing environment to allow developers to focus on tasks and not on cloud computing specific details.

Following the requirements, specifications and results presented in MS1 and MS2 deliverables, D*4.1* [2], D*4.2* [3] and D*4.4* [4], the purpose of this deliverable is to present the work done in the third phase ("*Two-stage Data-in-motion Real-time Analysis (M16-M29)*") of the project, which includes the release of a cloud environment coordinated with the edge layer for data analytics service management and scalability. It also merges the envisaged content of deliverables D4.3 and D4.5 into one single document.

1.2 Relation to other deliverables and work packages

Figure 1 shows the relationship between the CLASS WPs. This document includes not only the work done in WP4, but also the contributions to WP3 in the context of the cloud data analytics service management and scalability set of tools developed during M16-M29. During the previous period, the focus has been placed on the cloud environment part. On one side, during this period we improved the work done in this area, and on the other side, we also developed new features that cover WP3 activities, such as task 3.2, "*Develop, experiment and evaluate edge platform agent for analytics*".

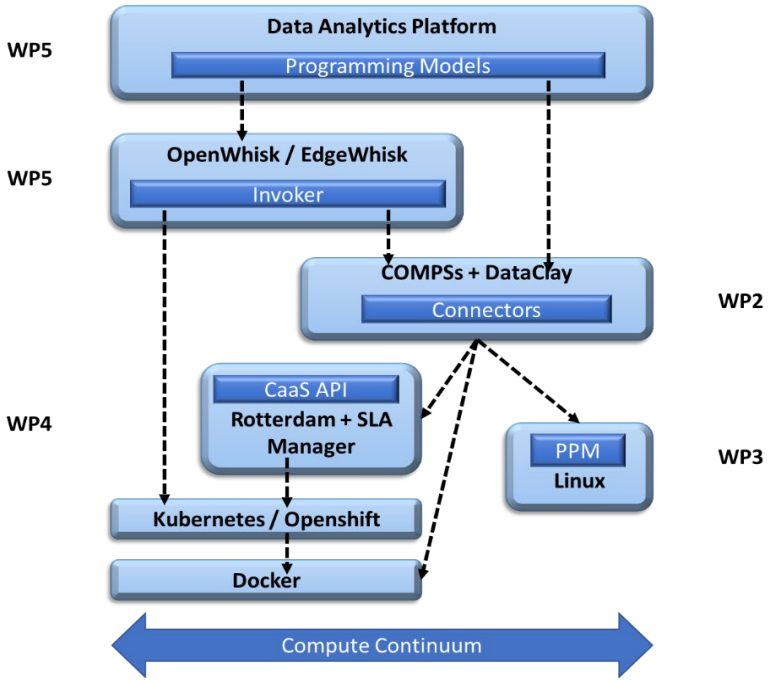


Figure 1 – CLASS architecture & WPs

Finally, the results of this deliverable will be part of the validation deliverables from WP3 (*"Validation of the CLASS edge computing subsystem"*) and WP4 (*"Validation of the Cloud Data Analytics Service Management and Scalability components"*) to be presented on M36.

1.3 Structure of the document

This document is structured as follows:

- Section 1 contains the introduction, the glossary of terms used in this document, and the description of the document's structure.
- Section 2 presents the functional description of the released Cloud and Edge environment platform.
- Section 3 describes the main changes included in this final release and the technical aspects of this platform, including the architecture of the different platform elements, the exposed interfaces and the execution environment.
- Section 4 presents the installation and usage guides, including the links to the code repositories and demo videos.
- In section 5, two demonstration scenarios of the Cloud and Edge environment platform are presented.
- And finally, section 6 presents the conclusion.

1.4 Glossary adopted in this document

This section contains the list of terms used in this deliverable in order to clarify its meaning to the readers:

- **Rotterdam** is the *Cloud Data Analytics Service Management and Scalability* component responsible for the deployment and management of the tasks running on the Cloud and Edge containers orchestrators it manages.
- A **Rotterdam Task** is an application or a long-running service that runs on containers orchestrators managed by the Cloud and Edge environment system. It encapsulates all the elements and properties needed to run the application in a container orchestrator. In the case of Kubernetes and Openshift[4], a Rotterdam Task is composed by the *Deployment*, *Services* and *Pods* entities needed to be defined and created before running the correspondent containers in the platform. Apart from this information, a Rotterdam task also defines the following properties:
    - The name and URL of the containerized application
    - The cluster identifier (where to run the application)
    - The dock identifier (in which namespace of the cluster the container orchestrator will run the application)
    - Number of initial instances or replicas
    - QoS requirements and policies (e.g., scaling out the application if there is a violation)

[4] Openshift Community Distribution (OKD) of Kubernetes optimized for continuous application development and multi-tenant deployment: https://www.okd.io/

```
{
   "name": "redis-app",
   "dock": "class",
   "qos": {
      "name": "NoMissedDeadlines",
      "description": "scale out task if missed deadlines > 0"},
   "replicas": 5,
   "containers": [{
      "image": "redis",
      "ports": [{
         "containerPort": 6379,
         "hostPort": 6379,
         "protocol": "tcp"}]}]}
```

Rotterdam task example: definition of a redis application with QoS requirements

- A **Rotterdam Dock** is a logical workspace for Rotterdam tasks, used to abstract away the resources and elements of the underlying infrastructure to be shared among a set of tasks. In Kubernetes and Openshift, it is called a namespace or project.
- A **Rotterdam Infrastructure** defines a container orchestrator, its location, and the properties needed to access it. Rotterdam supports the following K8s distributions: Kubernetes, Openshift and MicroK8s.
- A **COMPSs[5] Workflow** is an application composed by a set of data analytics functions that can be distributed and executed as asynchronous parallel operations.
- **COMPSs Tasks** (or workflow tasks) are the analytics functions which are part of a workflow.
- **Kubernetes (Openshift) Pods** are the smallest deployable units managed in Kubernetes and Openshift. Usually, one Pod corresponds to one docker container.
- A **Kubernetes (Openshift) Deployment** is a Kubernetes element that describes an application's lifecycle. Usually, deployments are composed by one or more pods.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: redis-deployment
 labels:
   app: redis
spec:
 replicas: 3
 selector:
   matchLabels:
     app: redis
 template:
   metadata:
```

[5] COMP Superscalar (COMPSs) is a framework for the development and execution of applications in distributed infrastructures : https://github.com/class-euproject/compss

```
    labels:
      app: redis
  spec:
    containers:
    - name: redis
      image: redis:latest
      ports:
      - containerPort: 6379
```

Deployment example: definition of a redis application

- A **Kubernetes (Openshift) Service** is an abstraction which defines a set of pods and a policy to access these pods. In other words, services are used to expose applications running on Kubernetes or Openshift.
- **Modena Data Center** is the location of the main Cloud testbed enviroment used in WP4 and UCs.

2 Functional Description

The final version of the standalone Cloud and Edge environment components presents the improvements and changes made to the platform features presented in the first release, and described in deliverables *D4.1* [2], *D4.2* [3] and *D4.4* [4]. This release also presents the implementation of new infrastructure and service management features for the data analytics service developers. As it was stated in these previous deliverables, the main objectives of the Cloud Data Analytics Service Management and Scalability system can be summarized as follows:

- To provide a transparent deployment and lifecycle management of data analytics resources, including the logical and physical orchestration of Cloud and Edge resources and the applications running on them.
- To assess the fulfilment of data analytics services' QoS requirements by creating, managing and evaluating SLAs, including the implementation of a dynamic adaptation and orchestration mechanism of Cloud and Edge resources to ensure the QoS of these analytics services.

These two objectives are completely fulfilled with the improvement of the features delivered in the first release, and with the new features implemented during this third project phase:

- Transparent lifecycle management of data analytic workloads in multiple Cloud and Edge clusters
- Creation and management of connections to multiple Cloud and Edge containers orchestrators (e.g. K8s, MicroK8s and Openshift)
- Deployment and management of applications and serverless functions in Edge devices
- Real-time QoS guarantees, SLA management and data analytics service scalability
- Performance monitoring of data analytics workloads and infrastructures

This section provides a full description of all these functionalities.

2.1 Transparent lifecycle management of data analytic workloads in multiple Cloud and Edge clusters

One of the main objectives of the Cloud Data Analytics Service Management and Scalability set of tools is the **simplified deploymet and lifecycle management of data analytics tasks in multiple containerized orchestrators** located in different places, **in Cloud and Edge**. As it was described in the first release deliverable (*D4.2* and *D4.4*), this set of tools was already able to deploy and manage these kinds of applications in one cloud infrastructure. This has been improved for this final release by enabling the deployment and management of these applications in more than one orchestrator at a time. Thus, this new release supports the management and monitoring of multiple applications and workflows running on multiple containerized orchestrators at the same time, allowing users to select the cluster or location that better fits their needs together with other properties.

In order to launch a data analytics workflow previously packed as a containerized application (i.e. a docker image), a user needs to define the following properties:

- The identifier of the container orchestrator located in the cloud or in an Edge device. If not defined the system will use the default cluster, which should be the main orchestrator deployed in the Cloud.
- The containerized application URL and the initial number of instances / replicas. In the case of COMPSs workflows this number corresponds to the number of workers required by the master.
- The QoS constraints defined for this application or workflow, which are used later to generate the SLAs. These QoS requirements can be defined in a simple JSON format or by using an identifier of a previously defined QoS template.

Internal complexities are transparent to final users. This means that these users don't have to take care of the management and configuration of the orchestrators, clusters and servers elements.

2.2 Creation and management of connections to multiple Cloud and Edge containers orchestrators

By default, the Cloud Data Analytics Service Management and Scalability system manages one Cloud infrastructure (i.e., Openshift cluster located in Modena Data Center). This default Cloud environment is the one presented in the first release deliverables and the one that will be used by the Use Cases in the final evaluation. This feature has been improved during the last year to make it possible for final users or service providers the deployment of their applications in different locations. The same way Kubernetes Cluster Federations do, this new feature gives them the possibility to create new connections to other cloud and edge infrastructures by defining the locations of existing containers orchestrators and the way to access them. This way they have more choices to deploy their applications according to their needs.

The final version of the Cloud and Edge environment system supports the deployment and management of data analytics workflows in the following containers orchestrators:

- Kubernetes
- Openshift
- Micro Kubernetes (MicroK8s)

Users must only define the identifier of the cluster or orchestrator, in order to deploy their applications in a specific place. They do not need to care about the internal complexities required to deploy and manage applications in different clusters.

2.3 Deployment and management of applications and serverless functions in Edge devices

Apart from connecting and managing multiple orchestrators, one new feature presented in this final release is the ability to install, at runtime, new MicroK8s instances in "empty" Edge and Cloud devices, and to manage them. First, users must define a connection to a remote device. This device requires to have no container orchestrator installed in it. Then, users can use this connection to deploy in this remote device a MicroK8s instance.

MicroK8s is a lightweight version of Kubernetes, very easy to install, which is made for Edge, IoT and developer workstations. It supports Windows, MacOS and a wide range of Linux distributions, and it comes together with plugins for Prometheus (monitoring) and Knative (serverless functions).

Furthermore, the Cloud Data Analytics Service Management and Scalability system supports not only containerized applications, but also serverless functions, or in other words, users can also use this platform to execute functions in Edge devices with Kubeless[6] deployed on MicroK8s. When deploying a MicroK8s in an Edge device, the user can include the Kubeless functionality to enable the execution of functions. Later, the user can deploy there functions the same way a user can deploy tasks in the available orchestrators or locations.

2.4 Real-time QoS guarantees, SLA management and data analytics service scalability

Users and service providers can specify a set of QoS requirements or constraints for their data analytics applications. These QoS requirements can include, for example, the maximum number of missed deadlines for workflows tasks, the latency of a specific application, or any other metrics collected by the monitoring tools used by the system. Users can also define the actions they want to execute in the case the system detects that these QoS requirements are not met. All these specifications are defined at deployment time, before launching a workflow or application.

[6] Kubeless is a Kubernetes-native serverless framework for deploying and managing serverless functions: https://kubeless.io/

These requirements and actions definitions are then transformed into SLAs (and their guarantees), so they can be monitored by the platform. The platform checks during runtime that these guarantees are met by constantly gathering and evaluating the metrics associated to them.

In case the SLA management and monitoring parts of the system detect that one or more guarantees are not met, they will generate violations. These violations are processed later by other components of the system to take the required actions. These actions offer a dynamic adaptation and orchestration of Cloud and Edge resources according to the actions defined by the user, like the automatic scalability of the applications that caused the SLA violations. These actions are also transparent to the user.

2.5 Performance monitoring of data analytics workloads and infrastructures

The platform relies on a set of monitoring tools responsible for gathering the metrics defined in the SLAs. These tools are integrated with the Cloud Data Analytics Service Management and Scalability system, and include the following applications:

- Prometheus
- Prometheus Pushgateway
- Grafana

The Prometheus instance deployed in the main Cloud infrastructure (i.e., Openshift cluster from Modena Data Center) collects metrics from the cloud infrastructure and the applications running on it. It also connects to the Prometheus Pushgateway, which is used by other applications to expose custom metrics. Grafana[7] is used to visualize the metrics collected by Prometheus.

Edge devices with MicroK8s also make use of their Prometheus instances. These Prometheus instances can be connected to the main platform.

3 Technical description

This section describes the final architecture of the Cloud platform environment and its components, including the changes with respect to the previous release version described in deliverables *D4.2* and *D4.4*.

3.1 Major changes in final release

The Cloud Data Analytics Service Management and Scalability components from final release present some major changes with respect to the architecture and technical characteristics defined and presented in previous release. These major changes include the following:

- Added support for multiple infrastructures / containers orchestrators (described in section 2.1 and 2.2).

[7] https://grafana.com/

- New Edge layer added to the platform. As described in section 2.3, the system now supports the deployment and management of MicroK8s and Kubeless in Edge devices. This includes the applications and serverless functions running on them.
- Improvement and extension of the CaaS API Gateway. New methods have been added to provide external users with new functionalities.
- The SLA Manager can read from multiple sources (monitoring tools). This module was updated to get metrics from more than one source (e.g. Prometheus) at the same time.
- The SLA Manager REST API was also updated to access the new functionalities implemented in this project phase.
- Rotterdam application has been rewritten in Golang. First version was written in Clojure, which requires the JVM[8] to run. Thus, the new dockerized version requires less resources and space to run.
- The JSON definitions used to create and launch applications and COMPSs workflows in Rotterdam have been simplified. Old JSON files are still valid.

3.2 Baseline Technologies and dependencies

The following baseline technologies are used within this component:

Name	Description	Version
SLA Manager [5]	The SLA Manager is a framework that manages service-level agreements between service providers and consumers. It is being developed by ATOS under an open source license (**Apache License 2.0**), and it has been used in another H2020 project, mF2C [6]. https://github.com/mF2C/SlaManagement	-
Prometheus	Prometheus is an open source monitoring system, written in Golang, and released with **Apache License 2.0**. This tool can be integrated with container orchestrators like Openshift and Kubernetes, and it can get metrics from the infrastructures and applications. https://prometheus.io/	-
Pushgateway	Prometheus Pushgateway is an intermediary application that connects Prometheus with custom applications. It can be used by these applications to push custom metrics to Prometheus. It is also written in Golang and released with **Apache License 2.0**. https://github.com/prometheus/pushgateway	-
Openshift Origin / OKD	Openshift Origin / OKD is a distribution of Kubernetes, licensed under **Apache License 2.0**. This platform adds a set of layers to	**3.10**

[8] Java Virtual Machine: https://docs.oracle.com/javase/10/vm/java-virtual-machine-technology-overview.htm

	"vanilla" Kubernetes, enhancing the security and management features. https://www.okd.io/	
Kubernetes	Kubernetes is an open source container orchestrator platform, written in Golang, and released with **Apache License 2.0**. Originally designed by Google, Kubernetes is now maintained by the CNCF[9] (Cloud Native Computing Foundation). https://kubernetes.io/	-
MicroK8s	MicroK8s is a lightweight distribution of Kubernetes that can be installed in Edge devices and small VMs.	**1.17**
Knative	Knative is a serverless framework that runs on Kubernetes and MicroK8s. Knatives makes it possible to run serverless functions. https://knative.dev/	-
Kubeless	Kubeless is another serverless framework that runs on Kubernetes and MicroK8s. https://kubeless.io/	-
Grafana	Grafana is a data visualization and monitoring tool used to show data from external sources, like Prometheus. https://grafana.com/	-

The SLA Manager has been adapted to be used in the context of CLASS. New functionalities and capabilities have been added to the SLA Manager during the project period. This application is part of the Cloud Data Analytics Service Management and Scalability components.

Grafana, Prometheus, and Prometheus Pushgateway are part of the monitoring tools module used by the SLA Manager to get the applications and infrastructures metrics needed to evaluate the SLAs and QoS defined by users for their applications. As it was described in previous deliverables, these tools are, in principle, installed and integrated in the main Cloud environment (i.e. Data Modena Center). In this deliverable they will be described separately as they now are also responsible for gathering metrics from Edge devices.

Finally, Openshift and Kubernetes are the orchestrators used in the cloud environment, meanwhile MicroK8s is the container orchestrator used in the Edge. Knative and Kubeless are two serverless framework used to deploy and manage serverless functions. They are also part of the Edge environment.

[9] https://www.cncf.io/

3.3 Architecture

The final version of the Cloud (and Edge) standalone environment for Data Analytics Service Management and Scalability is composed by three layers:

- The management and monitoring layer, which includes ***Rotterdam*** and the ***SLA Manager***. This layer is on top of the infrastructures layer (container-orchestration systems) and on top of the monitoring tools deployed in the Cloud and Edge.
- At the bottom of the diagram (Figure 2), the architecture presents the Cloud infrastructure layer, responsible for providing the tools needed by the Cloud to run containerized analytics applications.
- The Edge infrastructure layer is composed by a set of Edge devices responsible for providing the tools needed to deploy and run containerized applications and serverless functions.

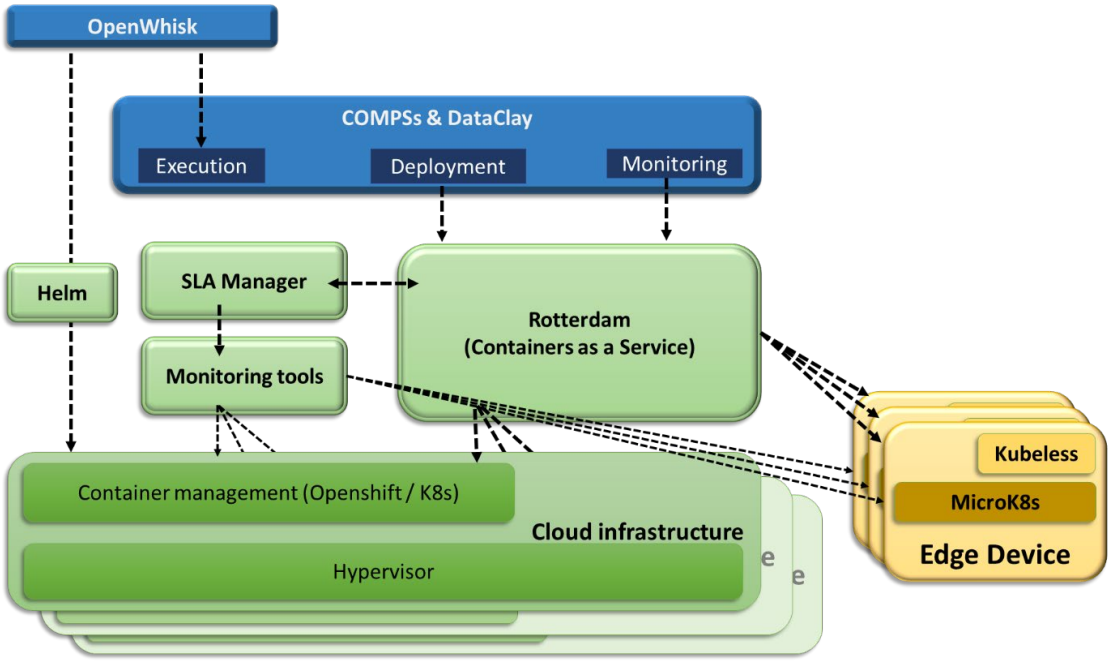


Figure 2 – Cloud computing platform in the context of the CLASS architecture

In the context of CLASS, Rotterdam is used by COMPSs & Dataclay system to run data analytics workflows in the Cloud. This COMPSs module uses the REST API provided by Rotterdam to launch and manage these workflows. On the other side, Openwhisk[10] makes use of the Cloud infrastructure layer to run there the correspondent data analytics functions.

Management and monitoring layer

This layer is composed by the following tools: Rotterdam, the SLA Manager, and a set of monitoring tools. These are the tools responsible for the deployment, management, monitoring and QoS enforcement of the containerized applications running in Cloud

[10] Apache OpenWhisk is an open source, distributed Serverless platform that executes functions in response to events at any scale: https://openwhisk.apache.org/

and Edge platforms. Apart from these containerized applications, serverless functions can also be managed by Rotterdam.

On one side, these tools hide all the complexity of container management systems to final users, by enabling a simplified deployment and lifecycle management of these data analytics workflows. The tool responsible for this task is Rotterdam, which is also the main entry point to this Cloud environment.

On the other side, the tool responsible for managing and checking that QoS requirements are met during runtime is the SLA Manager. QoS constraints, defined when launching a Rotterdam task, are used at deployment time to generate SLAs, which are then evaluated by this tool. If the SLA detects a violation, it sends this information to Rotterdam so it can execute the required actions, like scaling in or out the application. To do this evaluation, the SLA Manager relies on other monitoring tools, like Prometheus or Prometheus Pushgateway[11], which are the tools that are continuously gathering metrics from Cloud and Edge infrastructures and the applications running on them.

Cloud Infrastructure

The cloud infrastructure layer is composed by the following components:

- The container-orchestration systems.
- And the Hypervisor or devices needed to run these orchestration platforms.

This layer is required by Rotterdam and the other management and monitoring tools to run and manage containerized applications, and it is based on the following technologies:

- Docker (container technology)
- OpenShift / Kubernetes (container management)
- VMware (Hypervisor)

The main Cloud infrastructure environment used in the project is located in Modena Data Center, and it is composed by a cluster of four Virtual Machines, and an Openshift OKD container orchestrator.

Edge devices

Finally, the Edge devices layer is composed by one or more devices connected to Rotterdam, and it is based on the following technologies:

- Linux distribution: ubuntu 16.04, 18.04
- Container-orchestration system: MicroK8s
- Serverless framework: Kubeless (and Knative)

[11] The Prometheus Pushgateway allows applicatiopns and batch jobs to expose their metrics to Prometheus: https://github.com/prometheus/pushgateway

3.3.1 Rotterdam

Rotterdam is the CaaS application responsible for deploying, configuring and managing the tasks running in the Cloud and Edge infrastructures layers. This application is composed by the following subcomponents:

- CaaS API Gateway
- Deployment Engine
- Adaptation Engine
- Infrastructures Manager

All these subcomponents were introduced in D4.2, except the Infrastructures Manager, which has been added in this final release. Figure 3 shows the internal architecture of Rotterdam:

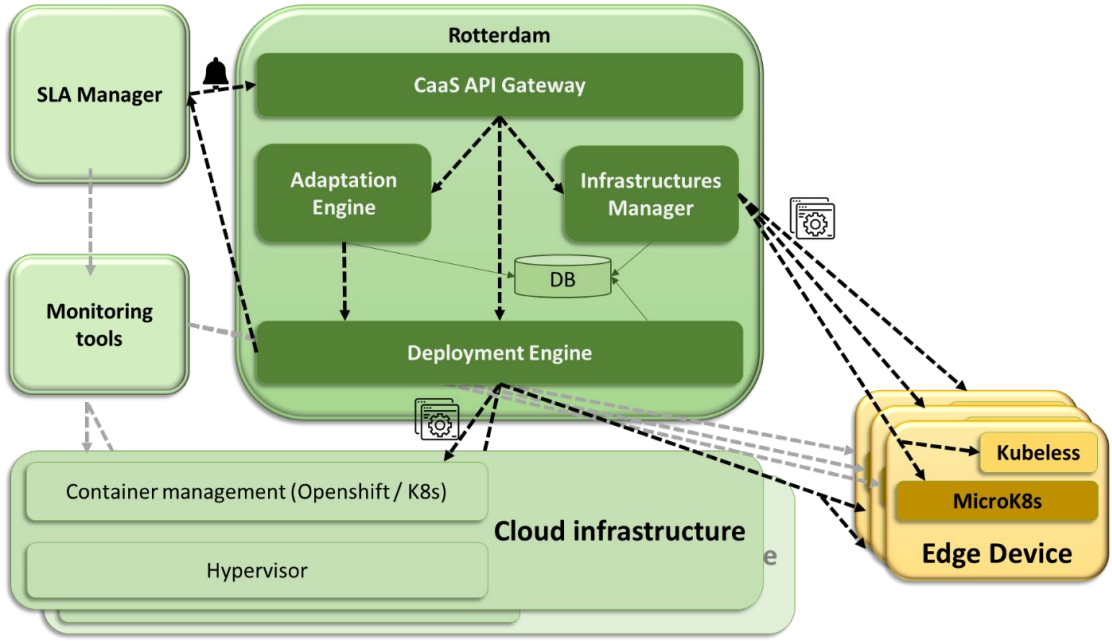


Figure 3 – Rotterdam and internal components

CaaS API Gateway

The CaaS API Gateway exposes a REST API with all the methods needed to deploy and manage applications, serverless functions and infrastructures in Cloud and Edge layers. Previous release already provided the methods needed to deploy and manage Rotterdam tasks in the Cloud infrastructure. Now, these methods have been improved and extended to include more features and capabilities. Apart from that, new methods have been added to allow final users the management of Edge infrastructures with MicroK8s where to deploy applications and serverless functions.

Rotterdam tasks deployment and management requests are redirected to the Deployment Engine, which oversees these operations. SLA violations are redirected to the Adaptation Engine, which is the responsible for handling these operations by deciding the actions that must be taken. And finally, infrastructures management requests, like the deployment of MicroK8s in a remote Edge device, are redirected to the Infrastructures Manager module.

Deployment Engine

The Deployment Engine is responsible for deploying and managing applications and functions. It connects to the containers' orchestrators located in the Cloud and Edge layers, and send them the applications deployment and management instructions. The previous release of this component only supported the management of containerized applications on one cluster at the same time. This has been improved, and now Rotterdam is able to distribute and manage applications and serverless functions in multiple clusters and remote devices at a time.

Adaptation Engine

This subcomponent processes the SLA violations and decides which actions must be taken according to the QoS defined for the applications that generate these violations. For example, if one workflow is generating violations because it hasn't enough workers to execute all the internal tasks, then the Adaptation Engine will send the Deployment Engine a request to scale out the number of workers of this workflow.

Infrastructures Manager

This new subcomponent processes the requests related to the creation and management of infrastructures (orchestrators and Edge devices). On one hand, it is responsible for creating and managing new connections to existing orchestrators, like Openshift or Kubernetes clusters, and the connections to Edge devices where to deploy later a MicroK8s (and Kubeless) instance. On the other hand, this subcomponent is responsible for installing at runtime MicroK8s and Kubeless instances in these Edge devices.

3.3.2 SLA Manager & Monitoring tools

The SLA Manager is the module responsible for creating and evaluating the SLAs associated to the Rotterdam tasks. This module relies on the information provided by a set of monitoring tools which are continuously monitoring the Cloud and Edge infrastructures and applications. These applications are composed by the following subcomponents depicted in Figure 4:

- The **REST API** exposes all the methods needed to create and manage SLAs, templates, metrics and connections to monitoring tools.
- The **Generator** creates SLAs based on the QoS / SLA requirements defined in the JSON files accepted by the REST API methods.
- The **Monitor** gets the metrics from the monitoring tools.
- The **Evaluator** is the module responsible for checking that these metrics comply with the SLAs guarantees defined by users.
- If there is a violation, the **Notifier** is the subcomponent responsible for sending this information to other tools, i.e. Rotterdam.
- The **Monitoring tools** are responsible for gathering infrastructure and applications metrics.

The subcomponents updated or implemented during this project's period are described hereunder.

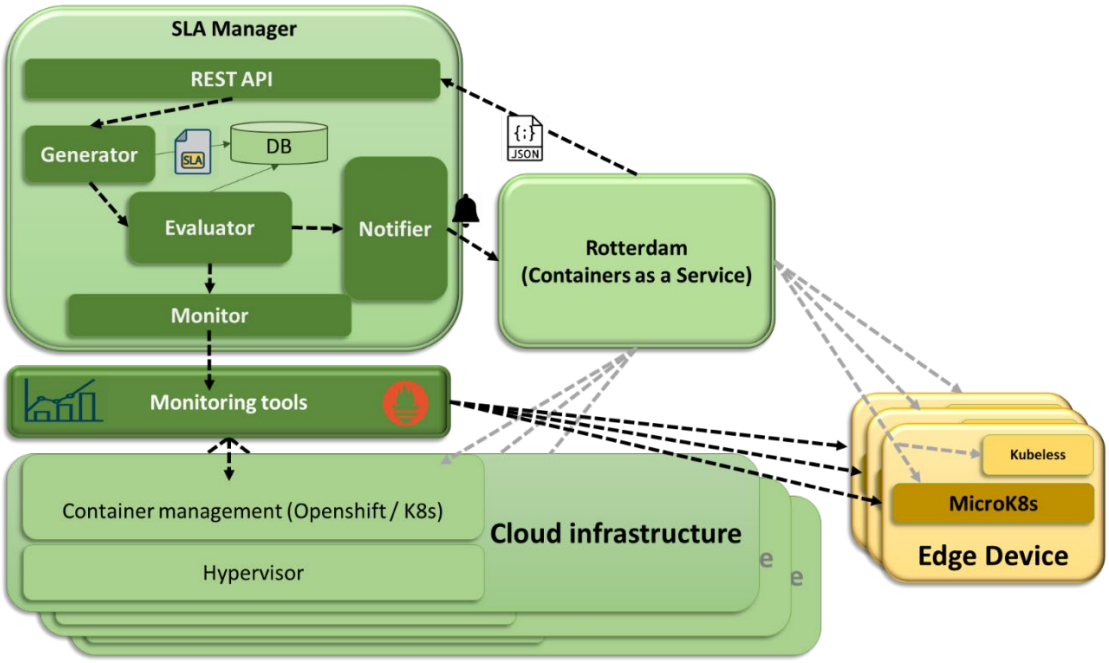


Figure 4 – SLA Manager and monitoring tools components

Monitor

The monitor module offers a set of interfaces, which can be implemented to adapt the SLA Manager to a specific monitoring tool. In the case of CLASS, this component has been adapted to get a list of metrics from Prometheus. These metrics can be defined using the SLA Manager REST API.

Notifier

The Notifier is another module that also offers a set of interfaces that have to be implemented to connect the SLA Manager with external tools. In this case, it has been adapted to connect this tool with Rotterdam. This way, if the Evaluator detects a violation, the SLA Manager can send this information to Rotterdam.

Monitoring tools

Finally, the monitoring tools that are being used in this release are the following:

- Prometheus, used to get metrics from infrastructures and applications.
- Prometheus Pushgateway, used by COMPSs Workflows to push their metrics to Prometheus.
- Grafana, used to visualize the metrics monitored by the platform.

3.4 Deployment Diagrams

The figures in this section show the deployment diagrams of the Cloud standalone environment for Data Analytics Service Management and Scalability Infrastructure deployed in Modena Data Center. First, this environment can be deployed in a single cluster, as shown in Figure 5. This kind of deployment includes the following sub-components: Rotterdam, the SLA Manager and the Monitoring tools.

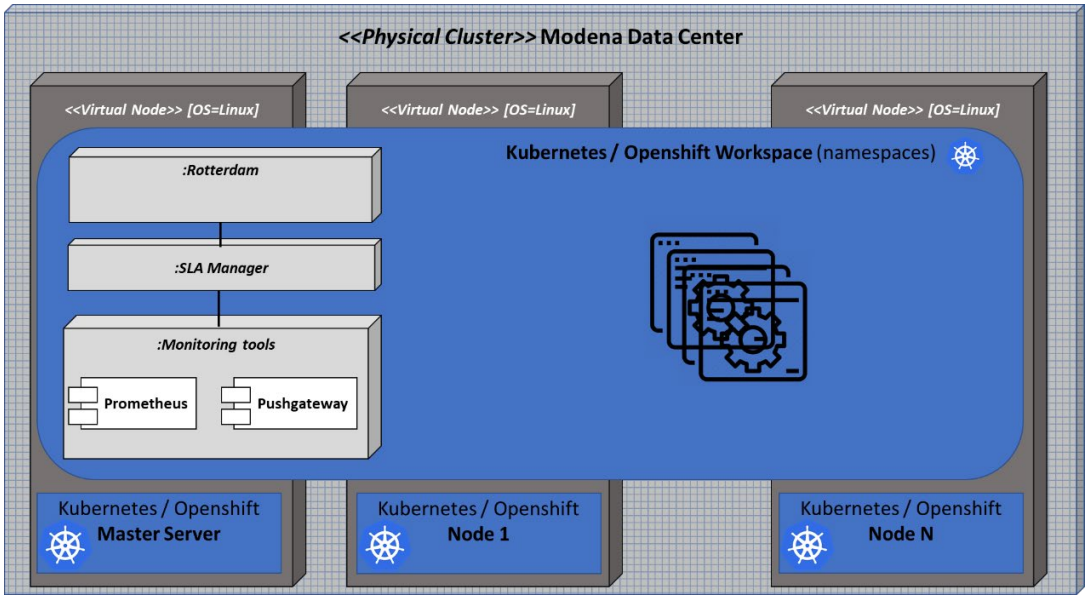


Figure 5 – Data Analytics Service Management and Scalability Cloud Infrastructure

The cloud infrastructure deployment relies on a cluster composed by two or more servers. In the case of the Modena Data Center testbed used during all the project phases to run the platform and its applications, there are four servers used to install Openshift, one master and three nodes (as shown in Figure 7). The namespaces (called "projects" in Openshift) used to logically group the applications running in Openshift comprehend the four servers. One of these namespaces was used to deploy Rotterdam, the SLA Manager and the monitoring tools. Applications deployed and managed by Rotterdam run on different namespaces.

During runtime, Rotterdam users can create new connections to remote containers orchestrators and devices (Edge devices with MicroK8s). Figure 6 depicts this scenario, where Rotterdam manages multiple orchestrators:

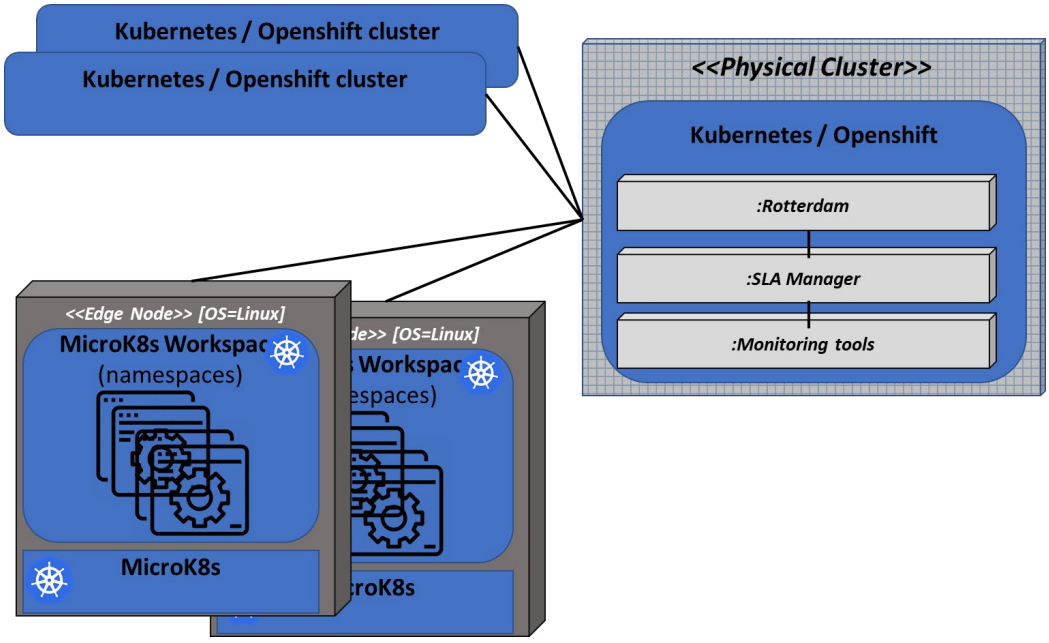


Figure 6 – Data Analytics Service Management and Scalability Multi Cloud and Edge Infrastructure

One single Rotterdam instance deployed in a Cloud environment (e.g. Openshift cluster from Modena Data Center) can manage multiple orchestrators located in Edge and Cloud.

3.4.1 CLASS Cloud standalone environment in Modena Data Center

Figure 7 shows the deployment used in Modena Data Center, including the characteristics of the VMs used for the deployment:

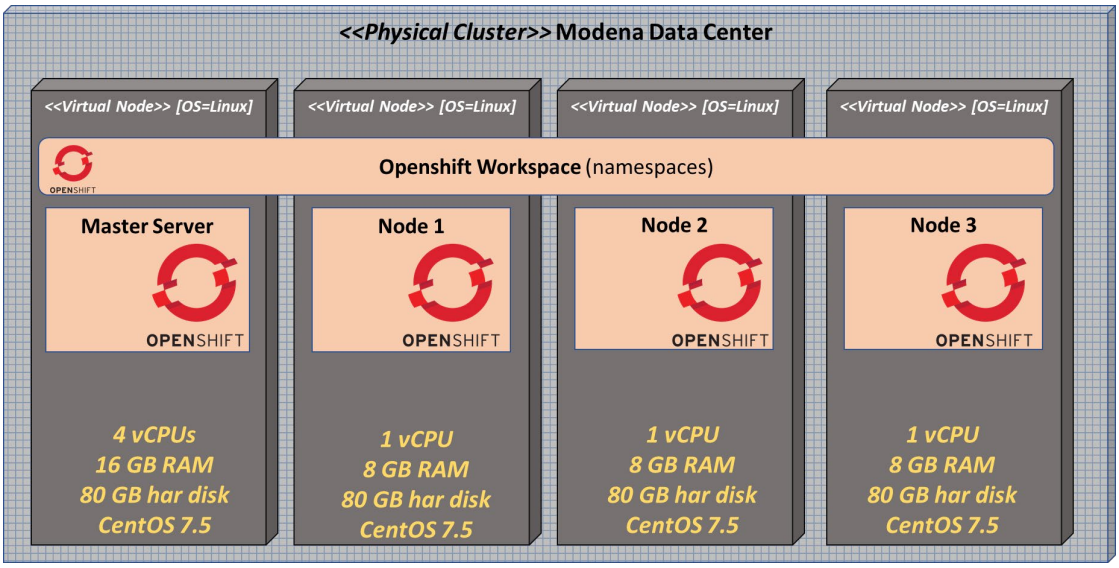


Figure 7 – Modena Data Center deployment

In this case, Openshift was used instead of Kubernetes. Rotterdam and all the tools were installed in the *default* project (namespace), and the applications managed by these tools were launched in the *class* namespace.

3.5 Interfaces Provided

This section describes the REST interfaces provided by the Cloud and Edge environment components, in particular the Rotterdam and SLA Manager applications. Interfaces provided by the other tools or platforms such as, for example, Kubernetes or Prometheus, can be found in their respective web sites and documentations.

3.5.1 Rotterdam

Rotterdam exposes a REST interface to external users and applications to manage the deployment and lifecycle of containerized applications. This REST API presents many changes with respect to the one presented in the previous release. New functions for the management of infrastructures and serverless functions have been added.

Rotterdam Tasks

These are the functions responsible for managing Rotterdam tasks running in the platform. One of the main changes that present these methods is that properties like the dock identifier have been moved from the request parameters to the body parameters.

Method	URI	Description
GET	/tasks/**{id}**	Returns all the information of a Rotterdam Task
DELETE	/tasks/**{id}**	Deletes a Rotterdam Task from the system
GET	/tasks/**{id}**/all	Gets a Rotterdam Task, including deployment info
GET	/tasks	Returns all the current Rotterdam tasks (from all infrastructures / clusters)
POST	/tasks	Deploys a new Rotterdam Task in the cloud / edge platform. The user can specify the cluster identifier in the body parameters

QoS / SLA operations

The methods responsible for creating and managing the QoS templates, used to create the SLAs associated to Rotterdam tasks, are the following:

Method	URI	Description
POST	/sla/tasks/**{id}**/guarantee/**{guarantee}**	Process SLA Manager violations. Method used by the SLA Manager to receive the violations and notifications.
GET	/qos/definitions/**{name}**	Returns the information of a QoS template definition
GET	/qos/definitions	Returns the list of all QoS templates
POST	/qos/definitions	Creates a new QoS template

The first method is responsible for getting the violations generated by external tools, i.e. the SLA Manager.

Infrastructures

The following methods are used to create and manage the information needed to connect to other container orchestrators located in remote clusters or Edge devices. They also include the methods responsible for deploying MicroK8s in these Edge devices.

Method	URI	Description
GET	/imec	Returns the list of all infrastructures / clusters connected to Rotterdam
POST	/imec	Creates a new connection to an infrastructure
GET	/imec/**{id}**	Returns the information of an infrastructure
PUT	/imec/**{id}**	Updates the information of an infrastructure

DELETE	/imec/**{id}**	Deletes an infrastructure connection
GET	/imec/**{id}**/cluster	Returns the orchestrator information running on an infrastructure
POST	/imec/**{id}**/cluster	Deploys a new orchestrator (MicroK8s) in an infrastructure
DELETE	/imec/**{id}**/cluster	Deletes the orchestrator from an infrastructure

Serverless functions

The following methods are used to create and manage serverless functions in Kubeless (MicroK8s) instances.

Method	URI	Description
GET	/functions/**{id}**	Returns a function
DELETE	/functions/**{id}**	Deletes a function
POST	/functions/**{id}**	Calls a function
GET	/functions	Returns a list of all functions managed by Rotterdam
POST	/functions	Creates a new function

Other operations

Method	URI	Description
GET	/	Get the status of the REST API server
GET	/config	Get the current Rotterdam configuration
GET	/version	Get the current Rotterdam version
GET	/status	Get the current Rotterdam status

3.5.2 SLA Manager

The SLA Manager also exposes a REST API to external users and applications to create and manage SLAs, the templates used to create SLAs, and the metrics that will be monitored or evaluated.

Agreements

Creation and management of SLAs:

Method	URI	Description
GET	/agreements	Returns all agreements
GET	/agreements/**{id}**	Gets the basic information of an agreement

POST	/agreements	Creates an SLA
PUT	/agreements/**{id}**/start	Starts the SLA's evaluation
PUT	/agreements/**{id}**/stop	Stops the SLA's evaluation
PUT	/agreements/**{id}**/terminate	Terminates the SLA's evaluation
PUT	/agreements/**{id}**	Updates an SLA
DELETE	/agreements/**{id}**	Deletes an SLA
GET	/agreements/**{id}**/details	Gets the details of an agreement
POST	/create-agreement	Creates an agreement from a template

Templates

Creation and management of SLA templates:

Method	URI	Description
GET	/templates	Get the list of templates
GET	/templates/**{id}**	Get a template
POST	/templates	Creates a new template

Metrics

Management of the metrics evaluated by the SLA:

Method	URI	Description
GET	/metrics	Get the list of all metrics that are being monitored
POST	/metrics/**{id}**	Adds a new metric (to be monitored / gathered from monitoring tools)
DELETE	/metrics/**{id}**	Deletes a metric

Sources

The following methods are responsible for managing the Prometheus sources used by the SLA Manager:

Method	URI	Description
GET	/sources/prometheus	Get the list of all Prometheus instances connected to the SLA Manager
POST	/sources/prometheus	Adds a connection to a Prometheus instance
DELETE	/sources/prometheus/**{id}**	Deletes a connection to a Prometheus instance

4 Installation and usage guides

Apart from the containers' orchestrator and the monitoring tools, the Cloud Data Analytics Service Management and Scalability platform is composed of two main builds. The first one is **Rotterdam**, and it includes the CaaS API Gateway, the Deployment Engine, the Infrastructures Manager and the Adaptation Engine. The other main build is the **SLA Manager**, which is one of the baseline tools included in this project.

This section describes how to install the complete Cloud environment, in particular the tools developed by ATOS: Rotterdam and the SLA Manager.

4.1 Packages distribution and requirements

Both the **SLA Manager** and **Rotterdam** are provided as docker images and their code is available in GitHub, in the following link: https://github.com/class-euproject. These applications do not require to be installed in a container orchestrator, or in the same host / location of the orchestrator. They just need to be able to connect to the orchestrator managed by them. Thus, the basic requirements for these two applications are the following:

- **Docker** (if using docker images)
  - https://docs.docker.com/engine/install/
- **Golang** (if using Github code to compile and generate the executable)
  - https://golang.org/doc/install

The requirements for installing the other tools that are part of the cloud platform, Openshift (or Kubernetes) and the monitoring tools can be found in the following links:

- **Openshift** (version 3.10)
  - https://docs.openshift.com/container-platform/3.10/install/prerequisites.html
- **Kubernetes**
  - https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
- **Prometheus**, Prometheus **Pushgateway** and **Grafana**
  - As these tools are provided as docker images, they can be installed in the container orchestrator (Openshift or Kubernetes). Thus, in this kind of environment, they only require the containers orchestrator.

4.2 Installation

4.2.1 Container orchestrator

To install an Openshift cluster, users can use the following guide: https://docs.openshift.com/container-platform/3.10/install/index.html

This guide is specific for Openshift version 3.10, the one used in the Modena Data Center cluster during the project. At the time of writing this document there is a newer version of Openshift: 4.4

As Openshift requires certain security features, the installation will require more steps and will be more constrained to certain Linux releases, like Centos or RHEL.

By contrast, the installation of a Kubernetes cluster is less complex. Users can follow the guides provided by Kubernetes, which include the installation of Kubernetes in multiple or single hosts (MicroK8s): https://kubernetes.io/docs/setup/

4.2.2 Monitoring tools

To install **Prometheus** in Openshift or Kubernetes, users can follow the guide available in the following link:

https://prometheus.io/docs/prometheus/latest/installation/

In the case of the cluster deployed in Modena Data Center, ATOS used the playbooks (ansible) provided by Openshift:

https://docs.openshift.com/container-platform/3.10/install_config/cluster_metrics.html#openshift-prometheus-deploy

Prometheus **Pushgateway** and **Grafana** can be installed using their docker images:

- Grafana: https://grafana.com/docs/grafana/latest/installation/docker/

```
docker pull grafana/grafana
docker run -d -p 3000:3000 grafana/grafana
```

- Prometheus Pushgateway: https://github.com/prometheus/pushgateway

```
docker pull prom/pushgateway
docker run -d -p 9091:9091 prom/pushgateway
```

After installing the monitoring tools, they have to be configured. For instance, in the case of Prometheus, users must update the "*prometheus.yml*" file (configuration) to enable the connection between Prometheus and tools like the Pushgateway. The same way Grafana has to be connected to Prometheus using its setup options.

4.2.3 Rotterdam

Rotterdam is provided as a docker image that can be found in the following link: https://hub.docker.com/r/atosclass/rotterdam-caas

Rotterdam code is also available in the following GitHub link: https://github.com/class-euproject/Rotterdam

Last version of Rotterdam (docker image) is: ***atosclass/rotterdam:latest***

There are several ways to install the application:

1. Using Openshift-OKD GUI (version 3.10) to deploy and launch the application:
    - In OKD Web interface, as shown in Figure 8, go to a project (namespace), e.g. _default_ project, and select "***Add to project*** > ***Deploy Image***".

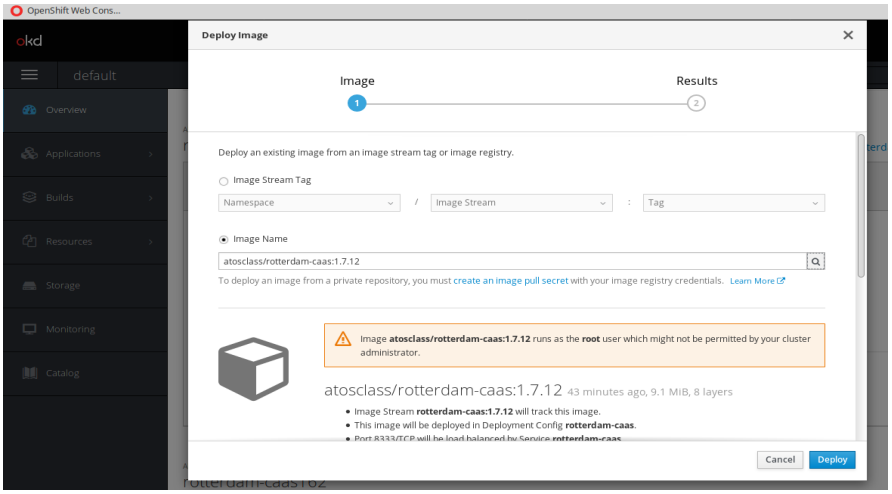


Figure 8 – OKD Web interface – Rotterdam deployment

- Select "***Image Name***" option: ***atosclass/rotterdam:latest***
- Add the following environment variables:
  - **OpenshiftOauthToken**: the value of this token has to be created manually after installing Openshift and configuring users and permissions. This variable is supported only when using Openshift. In the case of Kubernetes this variable is not needed.
  - **SLALiteEndPoint**: this is the URL of the SLA Manager
  - **PrometheusPushgatewayEndPoint** (optional): if there are applications that need to use this tool to push metrics to Prometheus, then the value has to be set
  - **MaxAllowed** (optional): Maximum number of violations allowed before sending a notification to the Adaptation Engine. This is a default value used by the platform to decide when to generate a violation in order to take the required actions.
  - **MaxReplicas** (optional): Default maximum number of replicas allowed per application. In the case one application needs to be scaled our, this value is used to limit the number of replicas.
- Deploy the image
- Create a Route to access the REST API (CaaS API Gateway), e.g. "rotterdam-cass.192.168.1.2.xip.io".

Figure 9 – OKD Web interface – Rotterdam

2. Using Docker to start the application:

```
docker pull atosclass/rotterdam-caas:latest

docker run [OPTIONS] atosclass/rotterdam-caas: latest [COMMAND] [ARG...]
```

   Apart from the environment variables used when deploying Rotterdam application in Openshift, in the case of Docker, users must define the following environment variables:
   - **KubernetesEndPoint**: URL of Kubernetes REST API. This variable is supported only when using Kubernetes.
   - **OpenshiftEndPoint**: URL of Openshift REST API. This variable is supported only when using Openshift.
   - **ServerIP**: IP address of the orchestrator master node.
3. Using Golang command line
   After downloading the repository (GitHub), go to "atos/rotterdam" folder, and execute the following command:

```
go run main.go
```

   The environment variables described in steps *1* and *2* have to be set in the system before launching the application.

4.2.4 SLA Manager

The SLA Manager is also provided as a docker image, and it can be found in the following link: https://hub.docker.com/r/atosclass/slalite

Last version of the SLA Manager (docker image) is: ***atosclass/slalite:latest***

There are two ways to install the SLA application:

1. Using Openshift-OKD GUI (version 3.10) to deploy and launch the application:
   - In OKD Web interface, go to a project (namespace), e.g. _default_ project, and select "***Add to project*** > ***Deploy Image***".
   - Select "***Image Name***" option: ***atosclass/slalite:latest***
   - Add the following environment variables:
     - **UrlRotterdam**: URL of Rotterdam REST API, e.g. "rotterdam-cass.192.168.1.2.xip.io". The SLA Manager needs to know where to send the violations.
     - **UrlPrometheus**: this is the URL of the main Prometheus instance, the one deployed in the main cloud container orchestrator (i.e. Openshift). Users can add more connections to other Prometheus instances during runtime.
     - **MetricsPrometheus** (optional): the list of initial metrics that will be requested to Prometheus. Users can add more metrics during runtime.
   - Deploy the image
   - Create a Route to access the SLA Manager REST API, e.g. "sla-manager.192.168.1.2.xip.io".

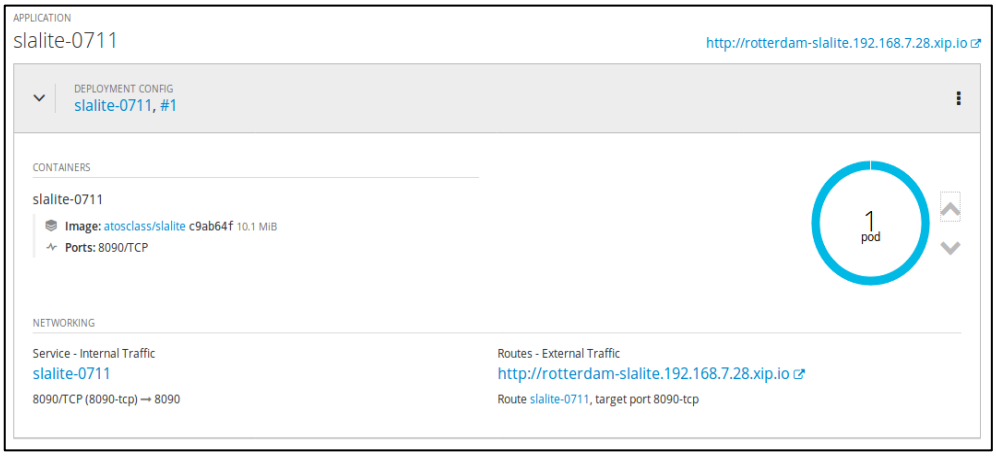


Figure 10 – OKD Web interface – SLA Manager

2. Using Docker to start the application

```
docker pull atosclass/slalite:latest

docker run [OPTIONS] atosclass/ slalite: latest [COMMAND] [ARG...]
```

At the end of the installation in OKD, these two applications should appear in the selected project or namespace (Figure 11):

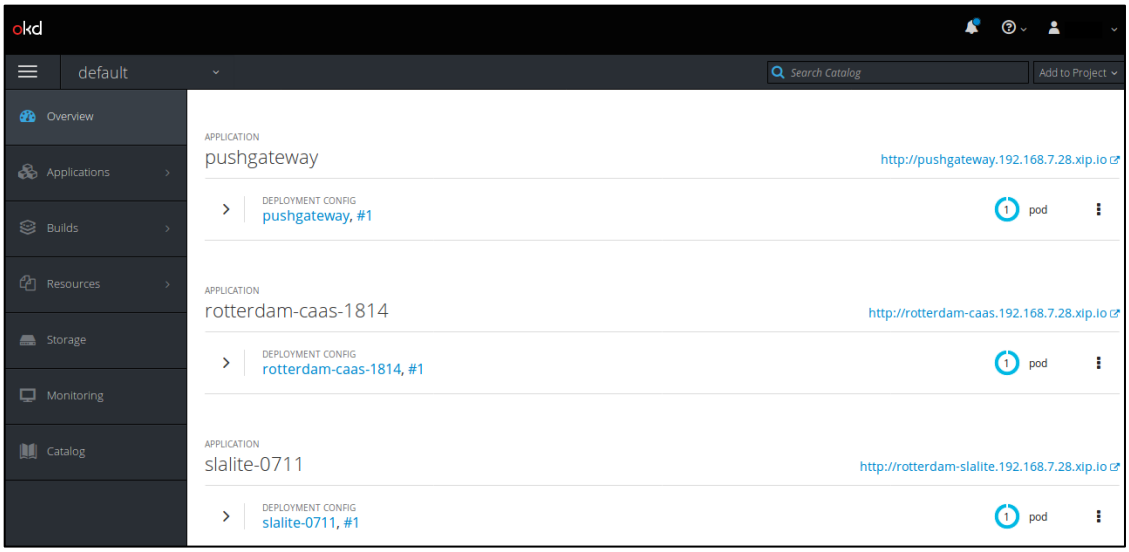


Figure 11 – OKD Web interface - Rotterdam and SLA Manager running in "default" namespace

4.3 Usage

After installing and configuring all the platform components, final users should have access to the Rotterdam CaaS Gateway (e.g. http://rotterdam-cass.192.168.1.2.xip.io). This REST API is also offered as a Swagger[12] interface, which offers a Web interface with access to all methods exposed by Rotterdam and a description of all the parameters needed to call these methods correctly. Figure 12 and Figure 13 show some of the CaaS Gateway method exposed in the Swagger interface:

[12] https://swagger.io/

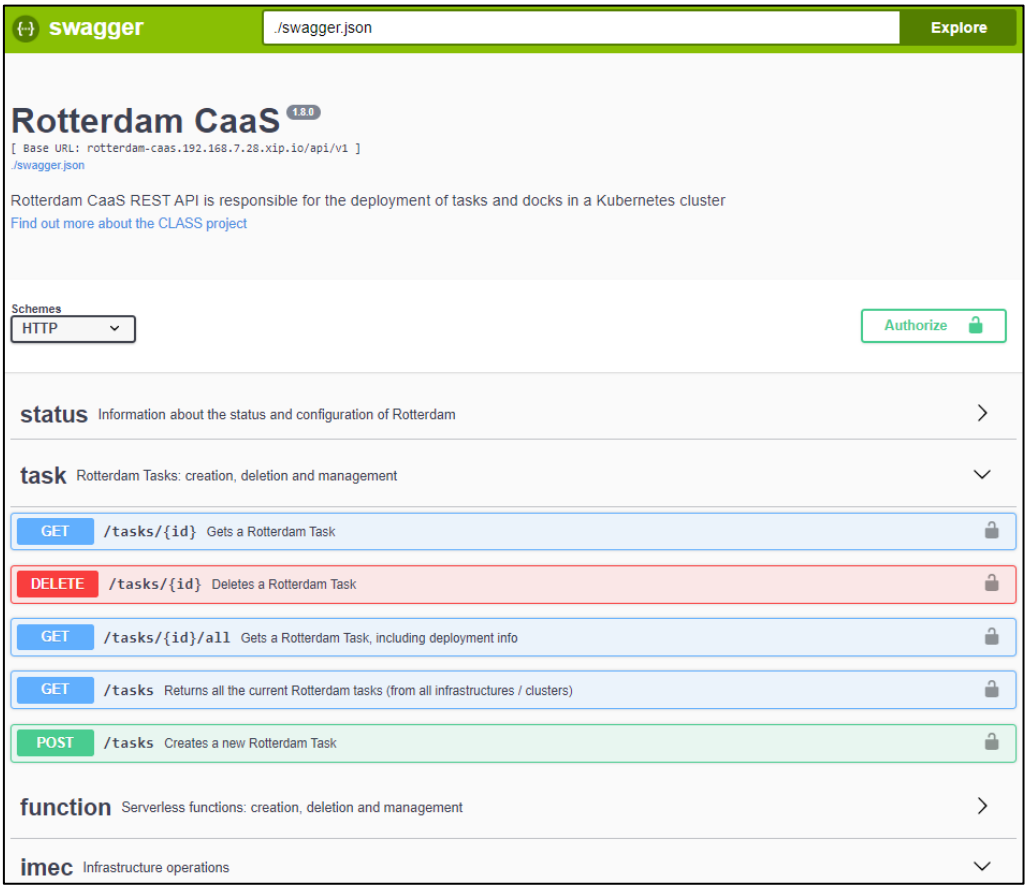


Figure 12 – Rotterdam Swagger REST API – Tasks methods

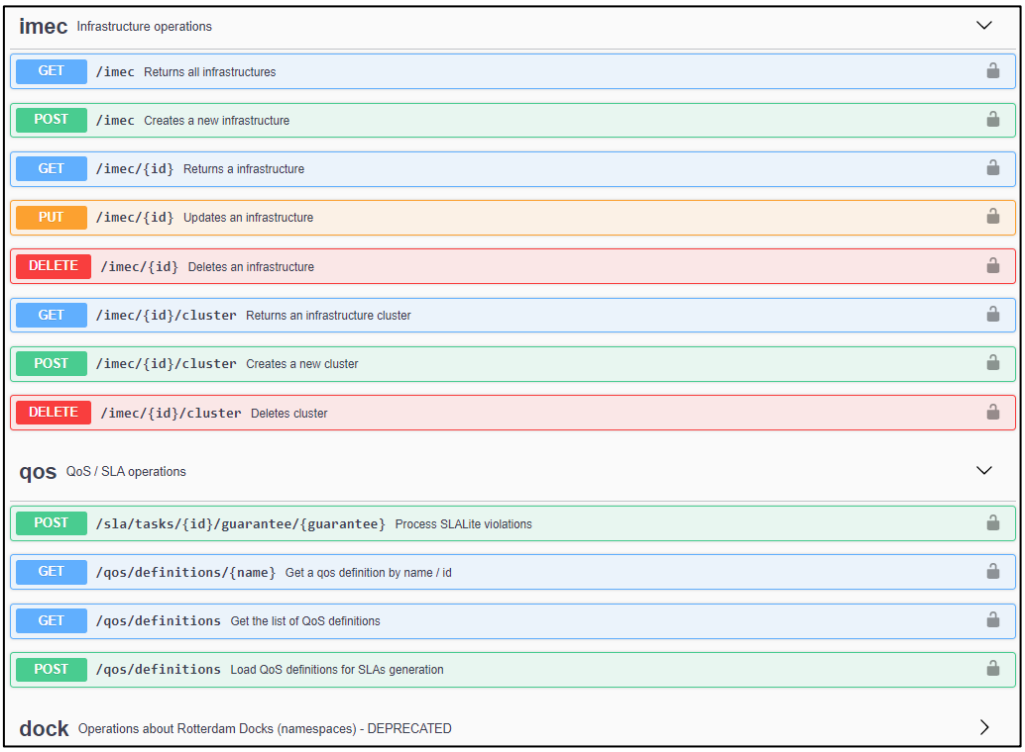


Figure 13 – Rotterdam REST API – Infrastructure and QoS methods

The SLA Manager also offers a REST API to users and applications. This REST API can be accessed via Web browser (e.g. GET methods: http://sla-manager.192.168.1.2.xip.io/):


```
{
   "agreements": {
      "Method": "GET",
      "Path": "/agreements",
      "Help": "Agreements"
   },
   "providers": {
      "Method": "GET",
      "Path": "/providers",
      "Help": "Providers"
   },
   "templates": {
      "Method": "GET",
      "Path": "/templates",
      "Help": "Templates"
   },
   "metrics": {
      "Method": "GET",
      "Path": "/metrics",
      "Help": "Metrics"
   },
   "sources": {
      "Method": "GET",
      "Path": "/sources",
      "Help": "Sources"
   }
}
```


Section 3.5, “Interfaces Provided”, describes all the available methods. The following subsections describe the JSON content of the tasks, QoS templates and all the other elements (with examples) used in the REST API methods to create and manage Rotterdam tasks and all the other elements. These JSON files are used in the body content of some of the “POST” and “PUT” calls exposed by the REST API.

4.3.1 Rotterdam tasks

Rotterdam tasks managed by the platform have to be defined in a JSON format. These are the formats accepted by this tool (definition of a **nginx** server[13] application):

- Long format (deprecated):

```
{
        "name": "nginx-app",
        "dock": "default",
        "cluster": "microk8s_1",
        "qos": {
                "name": "KubeletTooManyPods",
                "description": "scale down task if cluster pods > 50"
        },
        "replicas": 2,
        "containers": [ {
                "name": "nginx",
                "image": "nginx",
                "ports": [{
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                       }] ,
                "environment": [{
                        "name": "TEST_VALUE",
                        "value": "1.2.3"}]]}
```

This "long" definition requires the following properties:

  - **Name**: name of the application / COMPSs workflow
  - **Dock**: name of the namespace where to deploy the application (e.g. "class")
  - **Cluster**: identifier of the cluster / infrastructure / device
  - **Qos**: identifier of the QoS template used to generate the SLA
  - **Replicas**: number of instances. In the case of COMPSs workflows, this is the number of workers used by the master.
  - **Containers**: this property contains all the information about the containerized application
    - **Image**: URL of the containerized application (e.g. docker hub URL of the application)
    - **Ports**: ports used by the application
    - **Environment**: Environment variables

  Previous example defines a nginx application that uses port 80 to expose its services. It will use a QoS template, called "KubeletTooManyPods", to generate the SLA.
- Simple format: These JSON files present a simplified version of the previous JSON format, and they aim to simplify as much as possible the number of

[13] https://www.nginx.com/

properties used by COMPSs workflows and similar applications. The following examples define a “redis” application.

```
{
        "name": "redis-app",
        "cluster": "microk8s_1",
        "replicas": 4,
        "image": "redis",
        "qos": [{"qosid": "KubeletTooManyPods"}],
        "ports": [6379]
}
```

```
{
        "name": "redis-app",
        "cluster": "microk8s_1",
        "replicas": 4,
        "image": "redis",
        "qos": [ {"qosid": "deadlines001",
                "metric": "missed_deadlines",
            "comparator": “<",
            "value": 2,
            "action": "scale_out",
            "maxreplicas": 25,
            "minreplicas": 2,
            "scalefactor": 1.5,
             "maxallowed": 2}],
        "ports": [6379]
}
```

The main difference between these two files is that one uses a QoS template identifier, and the other one defines directly the QoS. These new formats presented in this final release require the following properties:

- **Name**: name of the application
- **Cluster**: identifier of the orchestrator
- **Image**: URL of the containerized application
- **QoS**: identifier / definition of the QoS template
- **Replicas**: number of instances / workers
- **Ports**: ports used by the application

4.3.2 QoS templates

QoS templates (JSON) used by the platform to generate the SLAs and the actions that will be taken if there is a SLA violation have the following properties:

```
{
        "type": "app-compss",
        "guaranteeName": "DeadlinesMissed_2",
        "maxAllowed": 0,
        "action": "scale_out",
        "scaleFactor": 1.5,
        "guarantees": [
                {
                        "name": "deadlines_missed_1",
                        "constraint": "deadlines_missed < 5"},
                {
                        "name": "deadlines_missed_2",
                        "constraint": "deadlines_missed < 2"}]}
```

- **Type**: this field defines if the QoS template is applied to applications or infrastructure.
- **guaranteeName**: name of the guarantee
- **maxAllowed**: number of violations allowed before raising a violation
- **action**: what action needs to be taken if there is a violation in the SLA, e.g. scaling in or out the instances of the application.
- **scaleFactor**: How to scale out or in the application
- **guarantees**: the guarantees define the conditions that applications must fulfill.

4.3.3 Infrastructures

Infrastructures managed by Rotterdam have also to be defined in a JSON format. This subsection presents two different files. The first one is the JSON used to create a connection to a container orchestrator or to an “empty” (no orchestrator installed) device:

```
{
        "name": "Name",
        "description": "Infrastructure description",
        "type": "microk8s",
        "so": "ubuntu18",
        "defaultDock": "default",
        "KubernetesEndPoint": "http://10.0.5.12:8001",
        "PrometheusPushgatewayEndPoint": "http://pushgateway.192.168.7.28.xip.io",
        "OpenshiftEndPoint": "http://10.0.5.12:8001",
        "OpenshiftOauthToken": "eyJhbGciOiJSUzI.. JhbGciO",
        "hostIP": "192.168.1.12",
        "hostPort": 22,
        "user": "user_name",
        "password": "user_password"
```

The following properties have to be defined:

- **Name**, **Description**: name and description of the infrastructure / cluster / device
- **Type**: type of the infrastructure. Following values are accepted: “Kubernetes”, “Openshift” and “MicroK8s”
- **SO**: Operating System. This field is needed when deploying a MicroK8s instance in a remote or Edge device.
- **defaultDock**: default namespace
- **hostIP**: IP address
- **hostPort**: Port used to connect to this infrastructure
- **user**: username
- **password**: user password
- **OpenshiftOauthToken** (optional): token used to connect to Openshift and manage applications
- **OpenshiftEndPoint** (optional): REST API endpoint of Openshift
- **KubernetesEndPoint** (optional): REST API endpoint of Kubernetes
- **PrometheusPushgatewayEndPoint** (optional): REST API endpoint of Prometheus Pushgateway

“HostIP”, “HostPort”, “User” and “Password” fields are used for deploying at runtime a MicroK8s instance in an Edge device.

The following JSON is used for installing MicroK8s in an “empty” device at runtime:

```
{
        "type": "microK8s",
        "apiPort": 8001
}
```

- **Type**: type of the installation: MicroK8s, MicroK8s and Kubeless, MicroK8s and Knative etc.
- **apiPort**: REST API port where the MicroK8s instance will be listening.

4.3.4 Serverless functions

Finally, serverless functions managed by Rotterdam are defined also in a JSON format. This is a first prototype for basic serverless functions:

```
{
        "name": "helloworld",
        "cluster": "microk8s_cluster",
        "runtime": "python2.7",
        "function": "def foo(event, context):\n    return \"hello world\"\n"
}
```

- **Name**: name of the function
- **Cluster**: identifier of the orchestrator (i.e. MicroK8s device)
- **Runtime**: runtime needed to execute the function
- **function**: function code

4.3.5 Usage example: deployment and scalability

This section presents an example of how to use the platform to deploy a (containerized) nginx server application in the default Cloud environment (i.e. Openshift – Modena Data Center). This example includes the use and collaboration between Openshift, Rotterdam, the SLA Manager and Prometheus.

1. The nginx server application will be deployed in the "class" project (namespace) of Openshift. In this example the "class" project is empty, as is shown in the following figure:

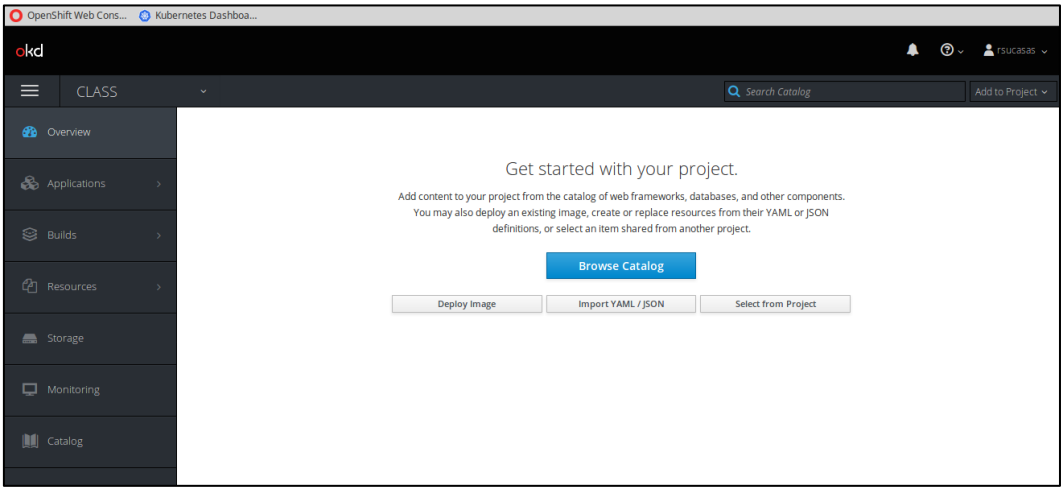


Figure 14 – OKD GUI – empty "class" project / namespace

2. To deploy this application, we will use Rottedam (CaaS API Gateway) application deployed in the "default" project (namespace) together with the SLA Manager, Prometheus, Grafana and the Prometheus Pushgateway.

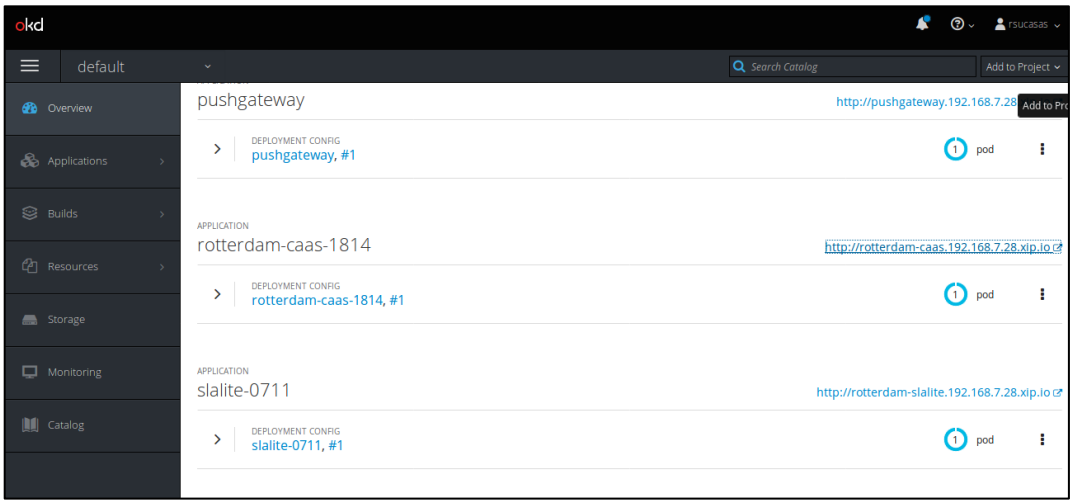


Figure 15 – OKD GUI – "default" project / namespace with Rotterdam and the SLA Manager

3. First, we have to create a QoS template. The following template will be used to scale in the application if there are too many pods running in the platform. This example uses the metric "*kubelet_running_pod_count*" gathered by Prometheus to simulate that if the infrastructure is being stressed, then applications like the

nginx-server should use fewer resources by reducing their number of instances or replicas.

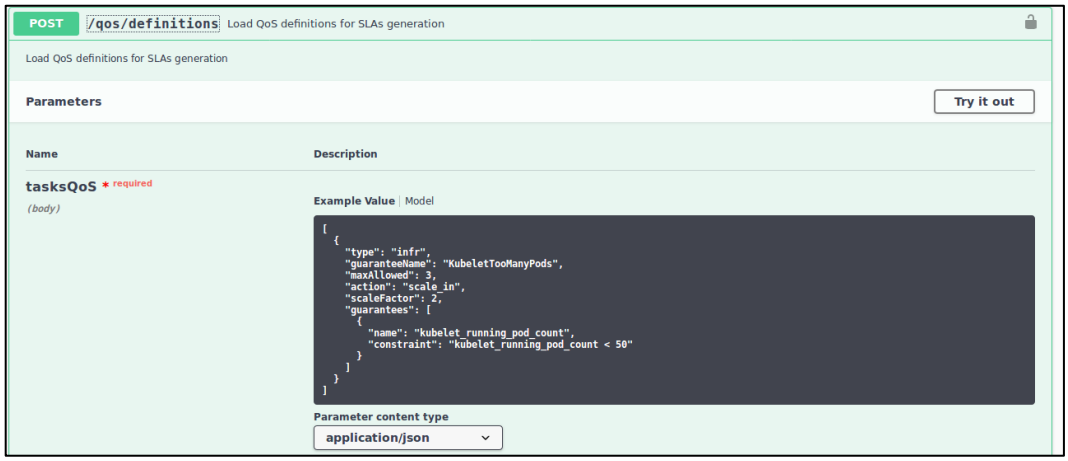


Figure 16 – Rotterdam (swagger) REST API – QoS template definition

After defining the template, we use the POST method "/qos/definitions", available in the swagger REST API, to create this QoS template.

4. Then, after successfully creating the QoS template, we can proceed with the task (nginx) definition where we can associate it to this QoS template. In the task definition we can also specify the number of instances ("*replicas*" property), for example we can set this value to 55 to force the violations.

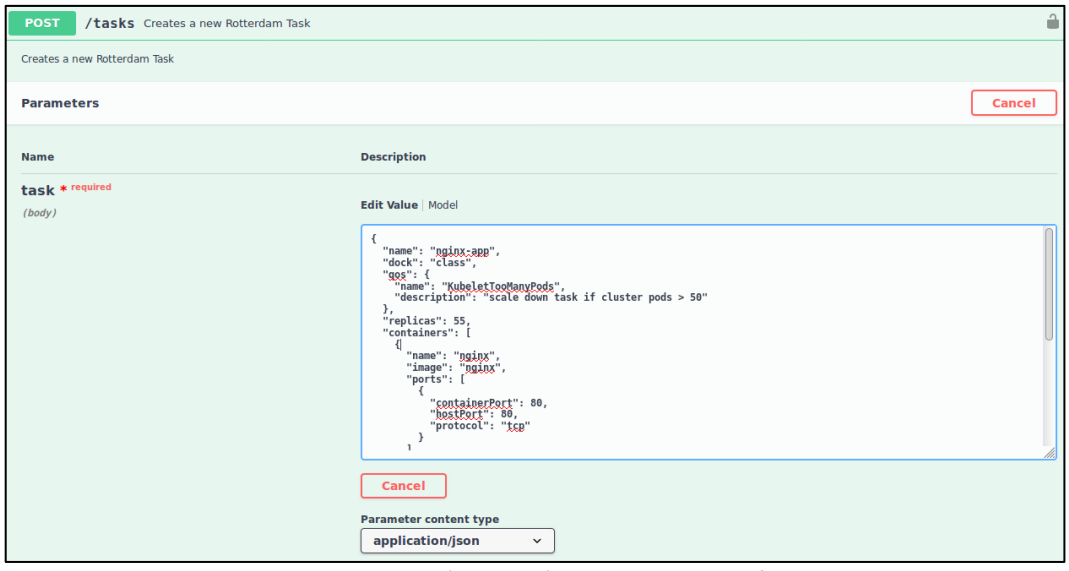


Figure 17 – Rotterdam (swagger) REST API - Task definition

5. After launching the POST "*/tasks*" request we get a response (also in JSON format) like the one presented in Figure 18. This response includes the identifier of the Rotterdam task, needed to get later the information and status of this task, and it also includes, among other properties, the URL where users can access the nginx server application.

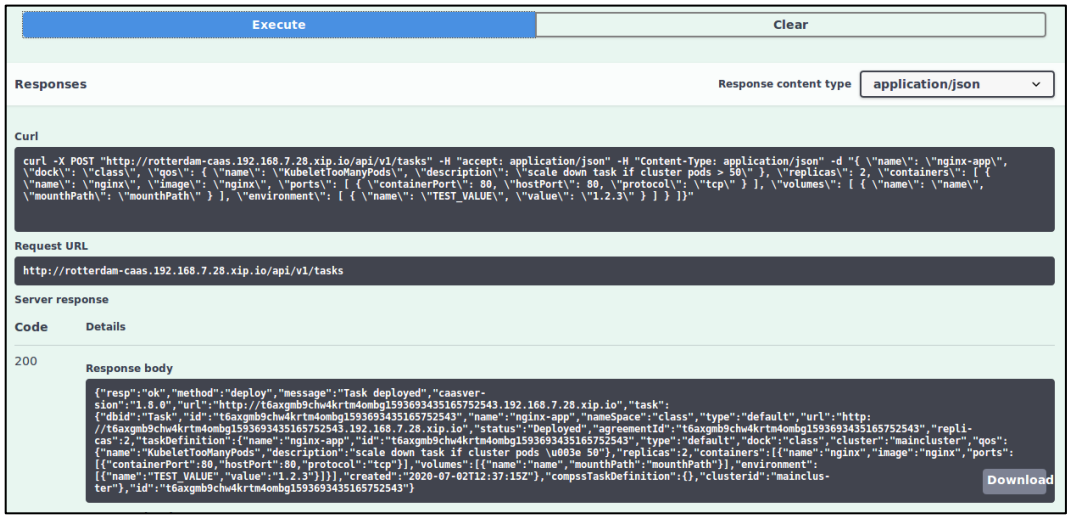


Figure 18 – Rotterdam (swagger) REST API – Task deployment result / response

6. Back in the Openshift-OKD Web interface, we can see that the "class" namespace / project now shows the deployment (Figure 19 and Figure 20) of the nginx server application and all the internal elements, e.g., service, pods, routes.

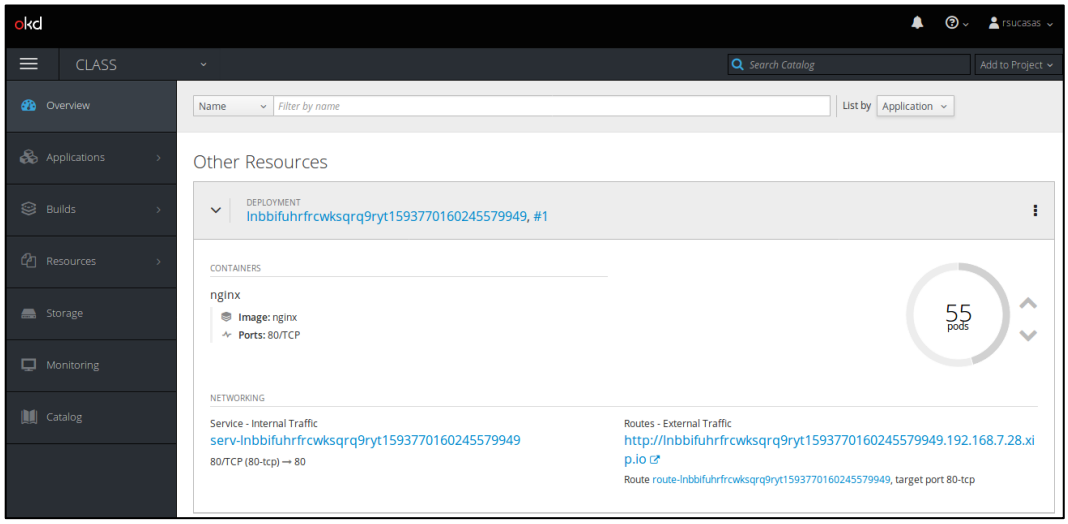


Figure 19 – OKD GUI – nginx server deployment in "class"namespace

If Openshift needs to download the application image from docker hub, then this operation can take a few minutes. In the case the docker image is already downloaded in Openshift, this deployment operation takes less than a minute.

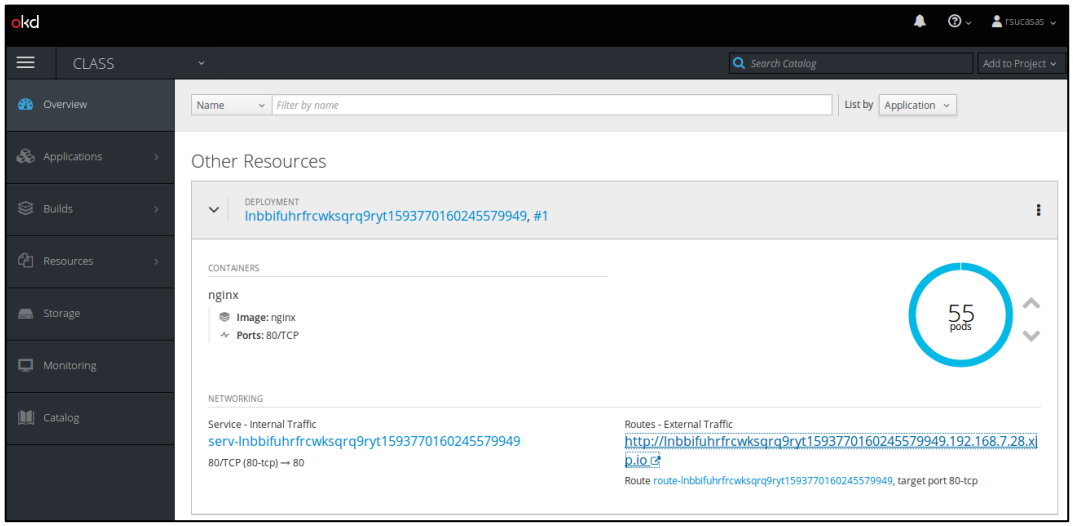


Figure 20 – OKD GUI - nginx server deployed and ready

Once the application is deployed and ready, users can access it through the web browser (Figure 21). Other type of applications, such as COMPSs workflows or applications like redis, will be accessed through their published ports. This information can be obtained by calling the GET "*/task/{id}*" method.

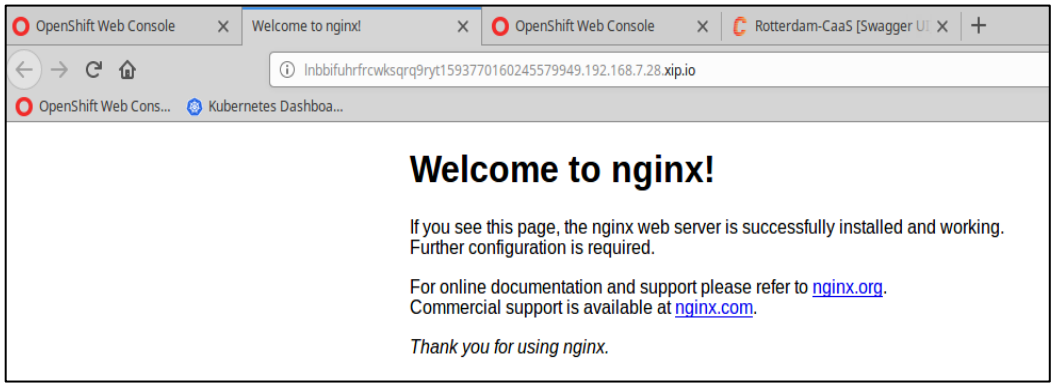


Figure 21 – nginx server application

7. The SLA generated by the system can be accessed by calling the SLA Manager REST API, e.g. using the web browser. Next figure shows the SLA generated in this example. It contains the status (e.g. "started"), the expiration and creation time, the identifier of the SLA, and the guarantees defined by the user in step 3.
   Internally, the SLA Manager is continuously evaluating the SLAs. In this case, the SLA Manager asks Prometheus every 15s or 30s for the metrics defined in the guarantees. If it detects that these guarantees are not met, then it generates a notification or violation and sends it back to Rotterdam's Adaptation Engine.
   In this example, the SLA Manager will detect a violation, and it will send it to the Adaptation Engine. This Adaptation Engine will take the actions defined in the QoS template used for this application. In this case, it will halve the number of instances of the nginx server application (see Figure 23Figure 23).

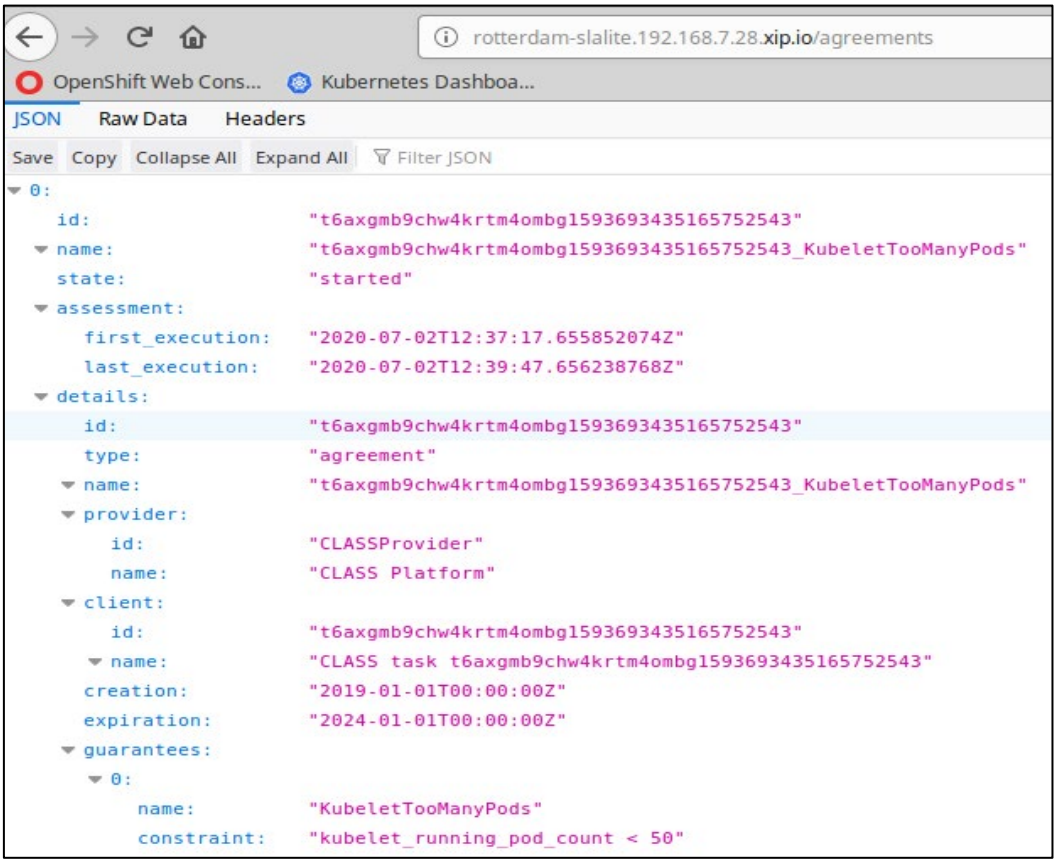


Figure 22 – SLA Manager REST API – SLA

8. The Adaptation Engine takes the required actions after detecting a violation with the nginx server application:

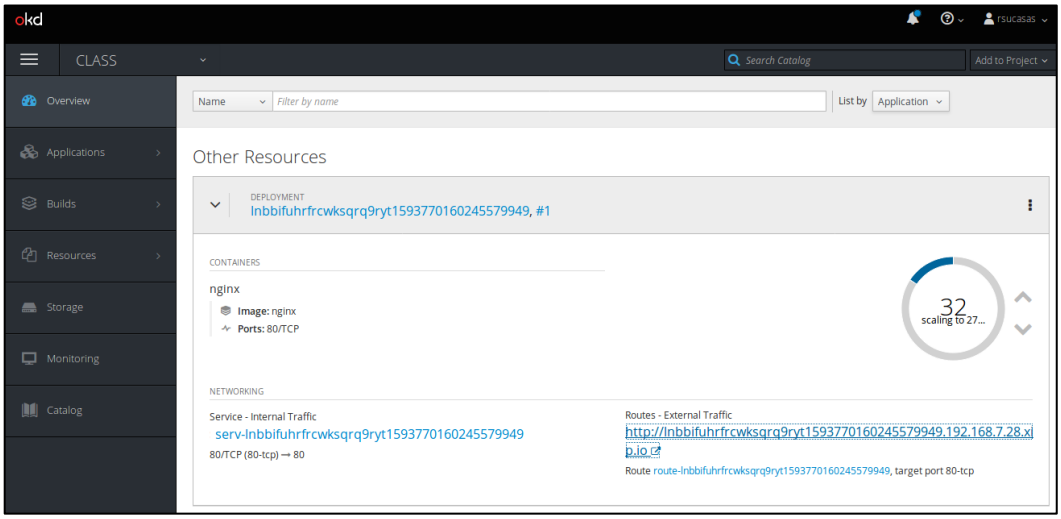


Figure 23 – OKD GUI - nginx server's instances are halved after SLA violation

4.3.6 Usage example: MicroK8s in Edge device

This section presents an example of how to use the platform to first deploy a MicroK8s instance in an Edge device, and then how to deploy in this Edge device a (containerized) nginx server application.

1. First, we need to create the connection to the Edge device. Here we define the Operating System, the IP address, and the username and password to access this host.

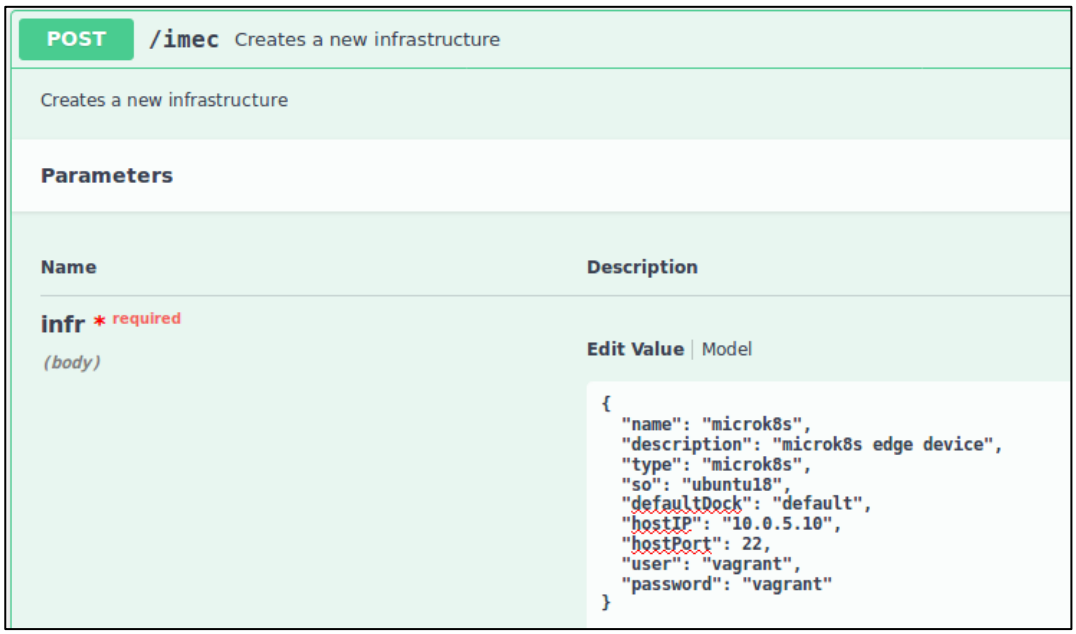


Figure 24 – Rotterdam (swagger) REST API – Infrastructure creation

2. After creating the location using the POST "*/imec*" method, we get an identifier in the JSON response (Figure 25). This identifier will be used to deploy there MicroK8s.

Figure 25 – Rotterdam (swagger) REST API – Infrastructure creation response

3. From the Swagger REST API we can get the list of infrastructures managed by the platform (using GET "*/imec*" method). After creating this new connection, we get two elements: the default cluster (i.e. the main Openshift cluster deployed in Modena Data Center), and the new Edge device.

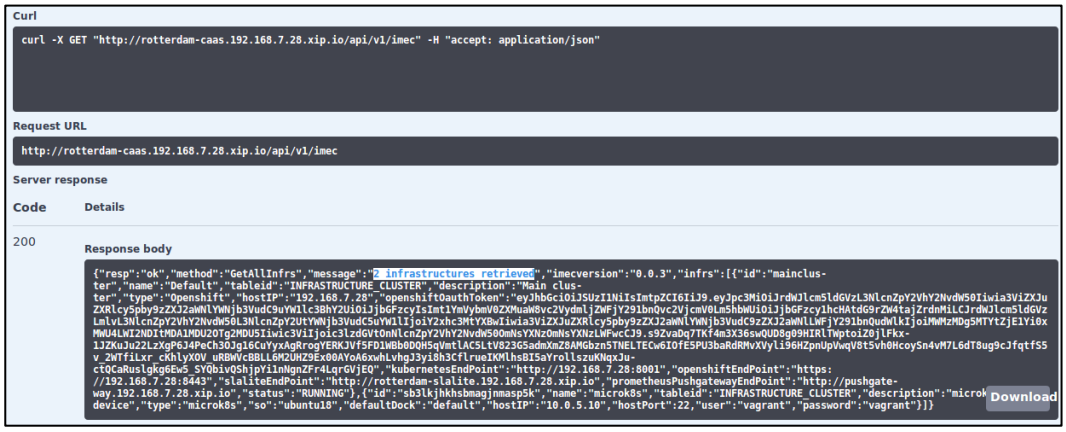


Figure 26 – Rotterdam (swagger) REST API – list of managed infrastructures

4. In order to install MicroK8s in this cluster, we can use the POST "*/imec/{id}/cluster*" method, as is shown in the next picture:

Figure 27 – Rotterdam (swagger) REST API – MicroK8s deployment

5. Going back to the Openshift – OKD Web interface, we can access Rotterdam's logs. In these logs (see next figure), we can see the information related to the MicroK8s installation in the selected device. This installation can take a few minutes.

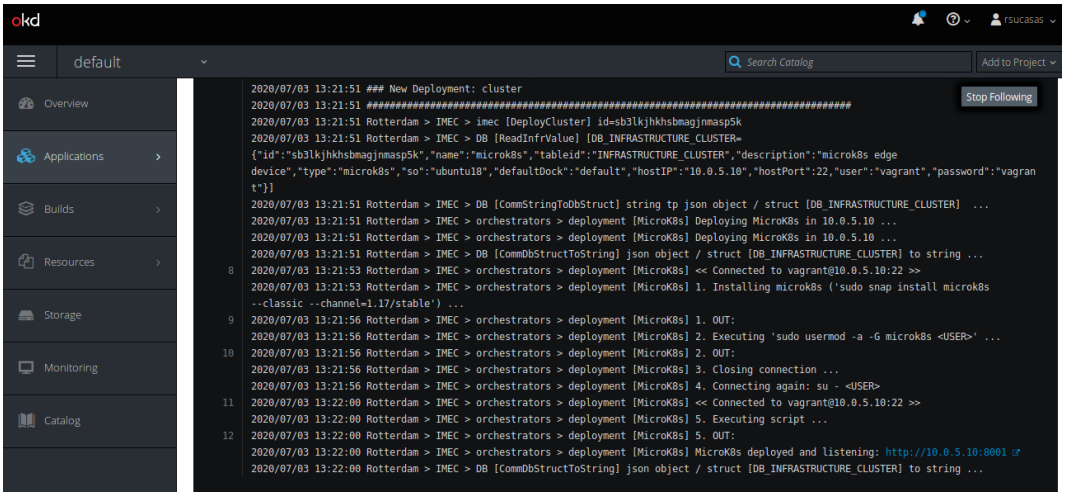


Figure 28 – OKD GUI – Rotterdam logs (MicroK8s deployment)

6. After installing MicroK8s we can start the deployment and management of tasks in this new orchestrator. In this example, we will deploy again an nginx server application using the same POST "*/tasks*" method we used in the previous example in Section 4.3.5.

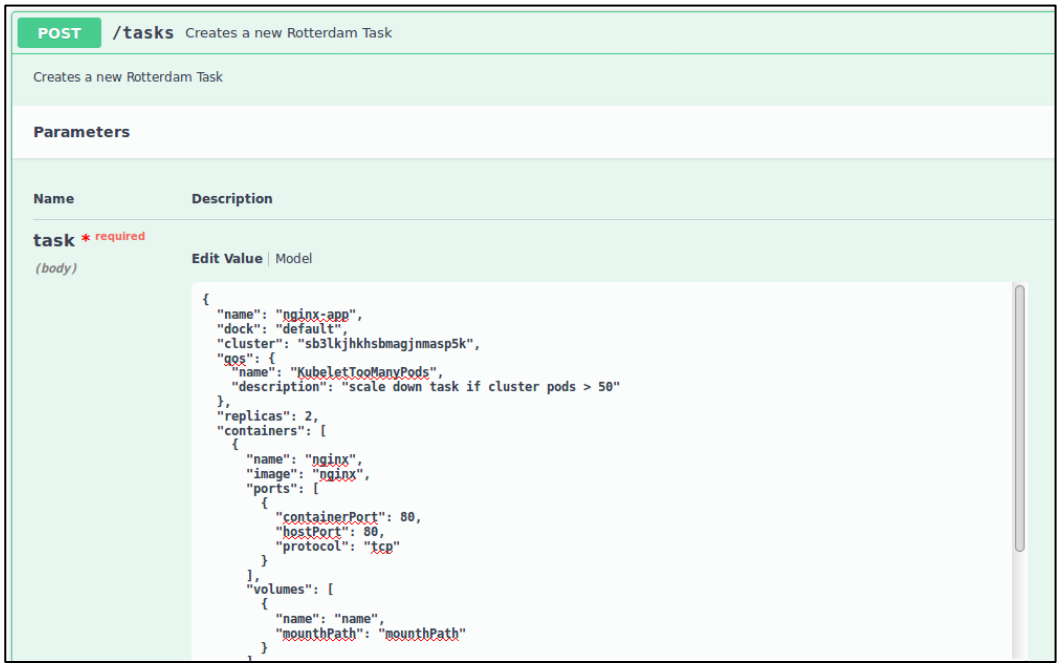


Figure 29 – Rotterdam (swagger) REST API – task deployment

This time we have to specify the identifier of the orchestrator (cluster field) in the task definition.

7. After deploying the Rotterdam task, we get the URL where we can access the nginx server, as shown in Figure 30.

Figure 30 – Rotterdam (swagger) REST API –response of task deployment

8. Once the application is deployed and ready, users can access it through the web browser:

Figure 31 – nginx server application running in the Edge device

9. Finally, if we take a look at the Edge device console, we can see MicroK8s installed and listening at port 8001 (the one we defined in step *4*), and also get the elements that correspond to the nginx server application (pods, services, deployment and replicaset).

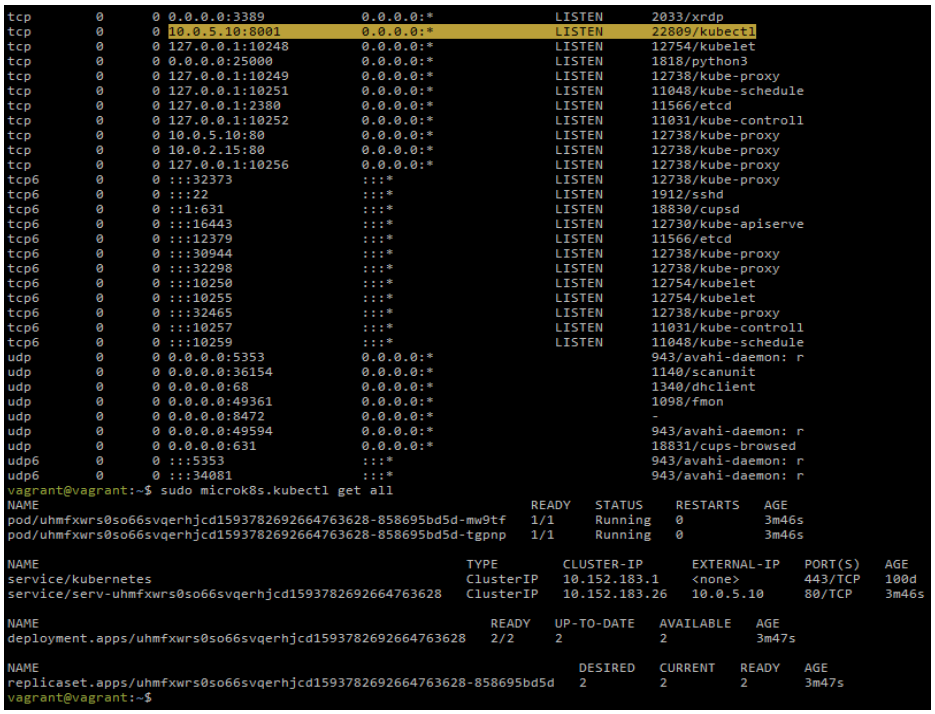


Figure 32 – Edge device console

5 Demonstration

Demonstration of the Cloud (and Edge) Data Analytics Service Management components final release considers the following video demos:

- The first one shows the **integration of the Cloud Data Analytics Service Management components and COMPSs**. We present two videos of this integration made in M26, where the deployment of multiple workflows in the Cloud using Rotterdam is shown:
  - *integrationcompssrotterdamconverted.mp4*
  - *integrationcompssrotterdam_x2_-_part_1.mp4*
    integrationcompssrotterdam_x2_-_part_2.mp4
- The other demo shows the **deployment and management of applications in multiples Cloud and Edge orchestrators**. There is also a video of this second demo:
  - *rotterdammultipleorchestratorsconverted.mp4*

These demonstrators are available at the CLASS intranet:

https://class-project.eu/user/login

A dedicated user has been created for demonstration purposes, with limited access to deliverables and related videos. The credentials to access this service are the following:

Username: ***EC_user***

Password: ***@Hz.52qXXF#K23***

After log in, click on “Intranet”, the demonstration videos and files of this deliverable are located in “PU_D4-7Demo” directory.

5.1 Scenario description

For these demos we have used the Cloud platform deployed in the Modena Data Center (described in section *CLASS Cloud standalone environment in Modena Data Center*), and an Edge Device with Ubuntu 18 and 4 GB RAM. Figure 33 depicts the elements used in these demos:

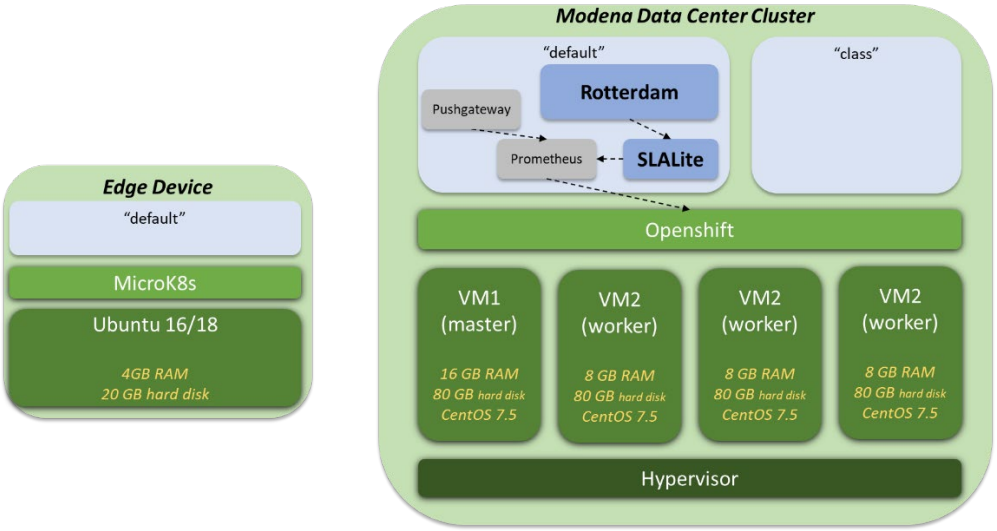


Figure 33 – Openshift (Modena Data Center) and Edge device used in the demos

Rotterdam, the SLA Lite and the monitoring tools are deployed in "default" project (namespace) of the Openshift cluster (Cloud). These monitoring tools are gathering metrics from the cluster and also the applications. COMPSs master application will generate custom metrics using the Prometheus Pushgateway. Tasks deployed by Rotterdam will run in "class" project.

A remote (Edge) device will be connected to Rotterdam, and will be used to deploy there containerized applications. Tasks deployed by Rotterdam in this device will run in "default" project.

5.2 Integration with COMPSs

This demo shows the integration of COMPSs and the Cloud Data Analytics Service Management platform. First, a COMPSs master application was used to launch a set of workers in the Cloud to run a workflow (see Figure 34). Then, two COMPSs master applications instead of only one, were used to launch two workflows in the Cloud using Rotterdam Caas API Gateway. This demo includes the coordination of Rotterdam, Openshift, the SLA Manager, COMPSs, the COMPSs workers, Prometheus, and the Prometheus Pushgateway, and it shows the following features:

- Integration of Rotterdam, SLA Manager, Prometheus, Prometheus Puschgateway and COMPSs.
- Execution of multiple COMPSs workflows in the cloud platform.
- How COMPSs workflows are automatically scaled based on the monitoring of real time QoS objectives (SLAs) ➔ missed deadlines generated by workflow's tasks.
- How platform metrics can be viewed on Prometheus and Prometheus Pushgateway.

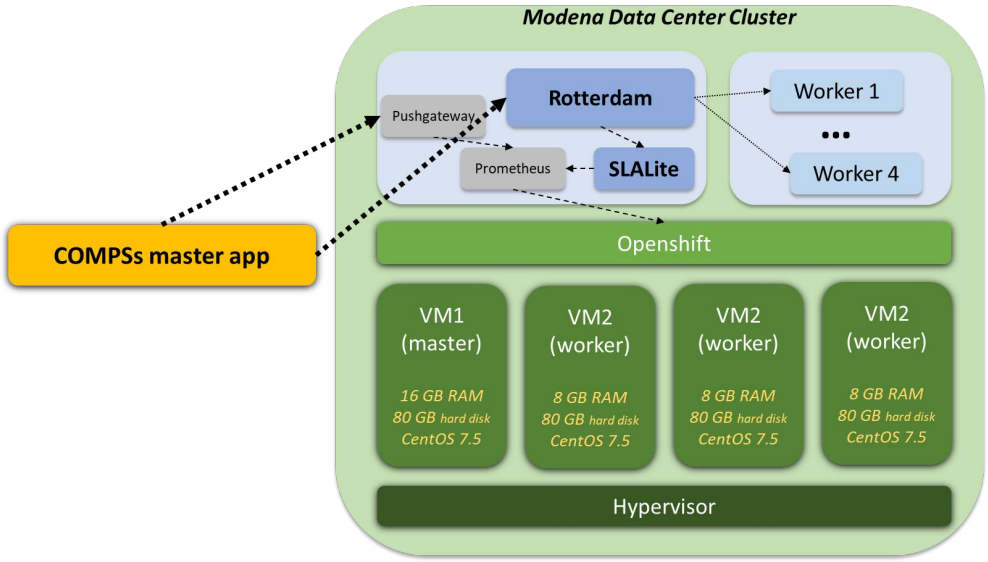


Figure 34 – Integration with COMPSs master application

The demonstration steps presented in the following subsection make use of five different graphical interfaces and one console window:

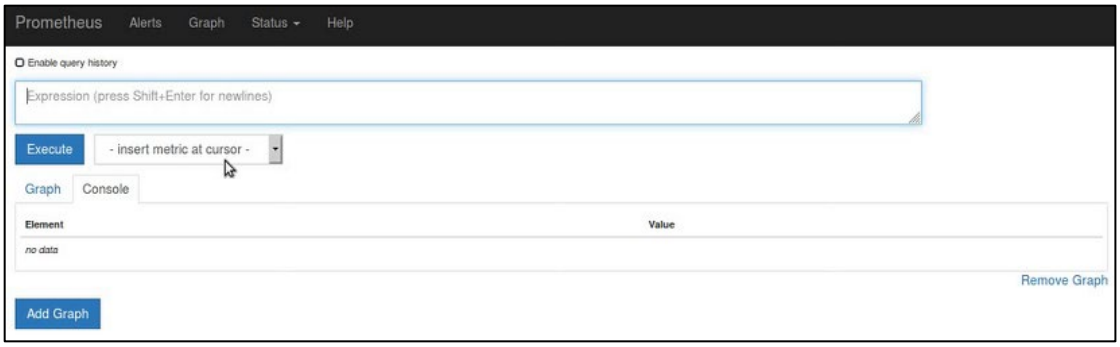
- OKD-Openshift Web UI of the Cloud platform
- Rotterdam Swagger Web Interface
- SLA Manager interface
- Prometheus and Pushgateway interfaces
- Console window of COMPSs master application

5.2.1 Demo

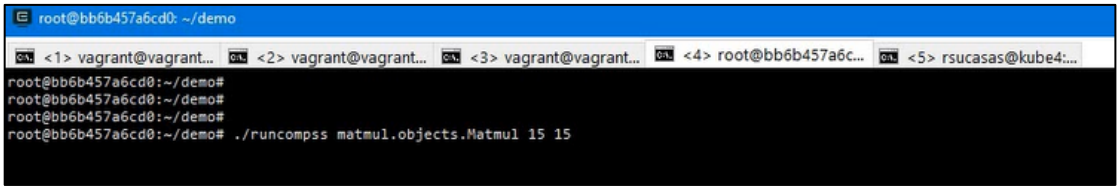
The recorded demo can be accessed in the following link:

integrationcompssrotterdamconverted.mp4

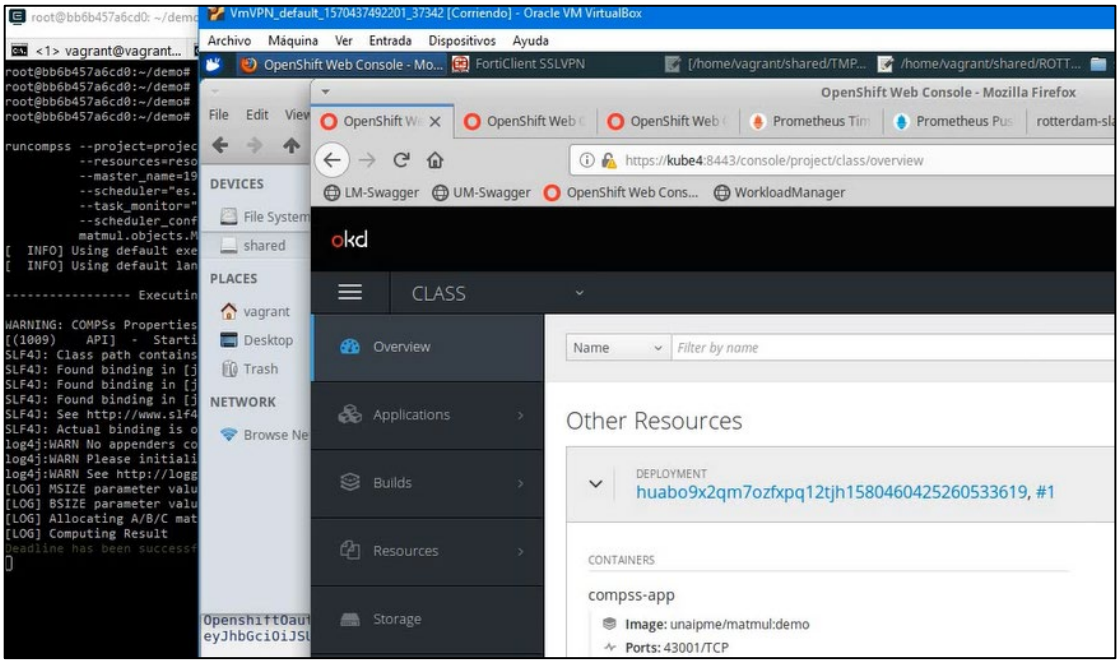
Step 1: Checking Prometheus metrics (min 00:25) - *Prometheus Web Interface*



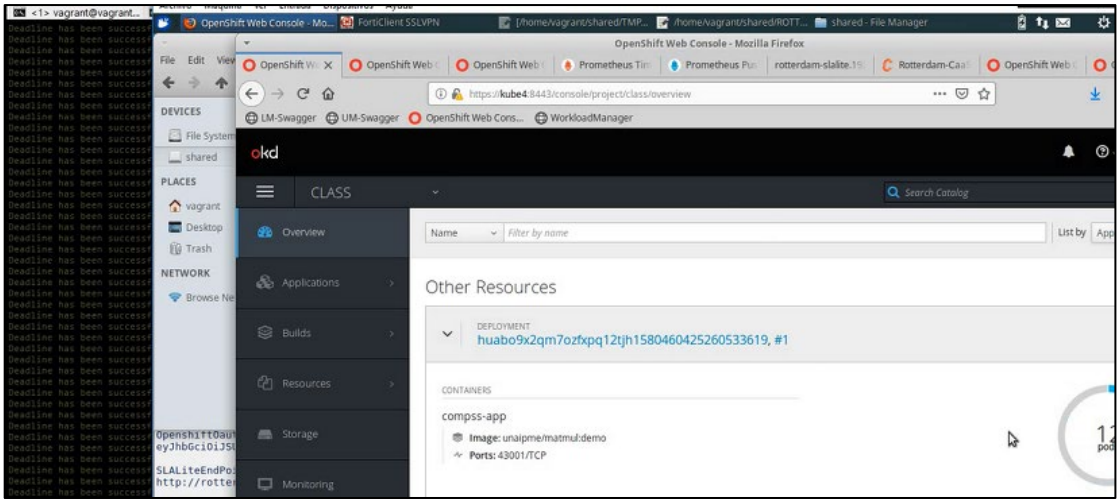
Step 2: Launching COMPSs master application (min 00:35) – *COMPSs master application*



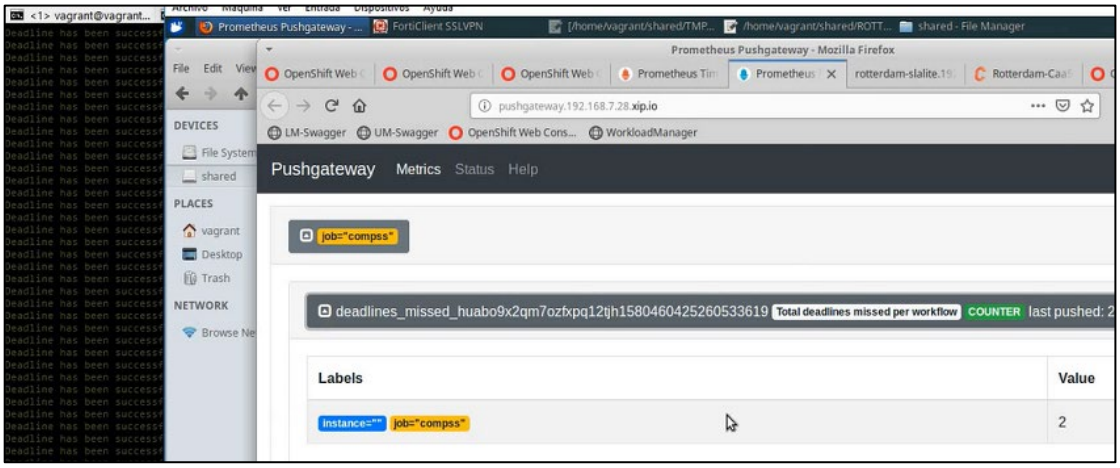
Step 3: COMPSs workers deployed on Openshift platform (min 00:51) - *OKD Web Interface & COMPSs master application*



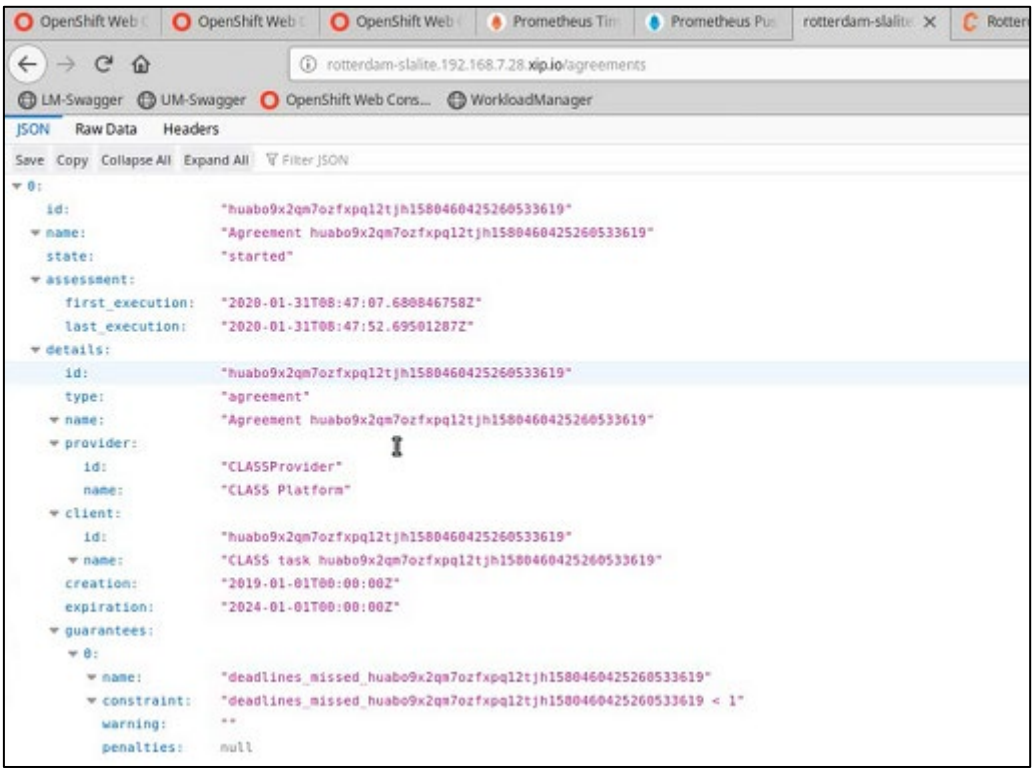
Step 4: Rotterdam scales out the workers after violations (min 00:58) - *OKD Web Interface & COMPSs master application*



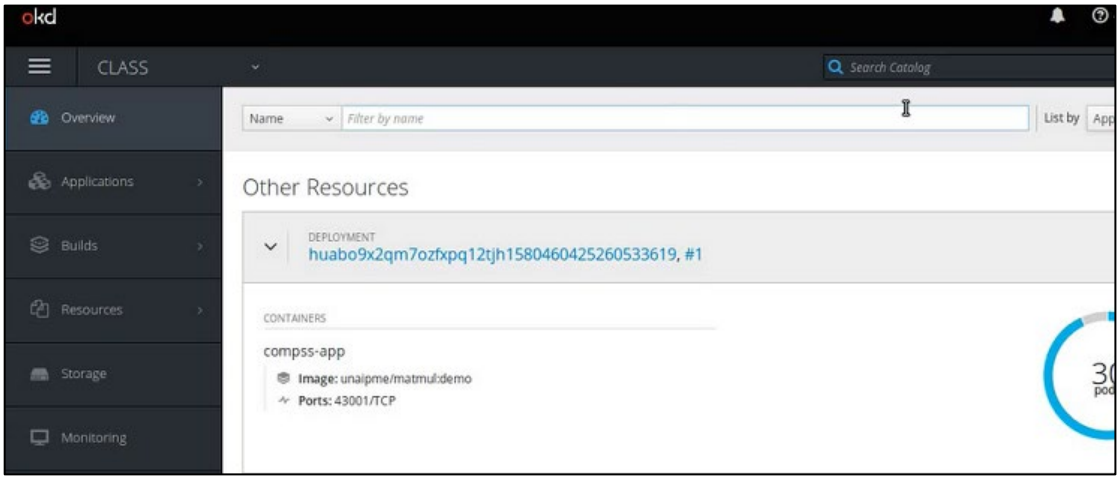
Step 5: COMPSs master updated metrics in Prometheus Pushgateway → 2 deadlines missed (min 01:10) - *Pushgateway Web Interface & COMPSs master application*



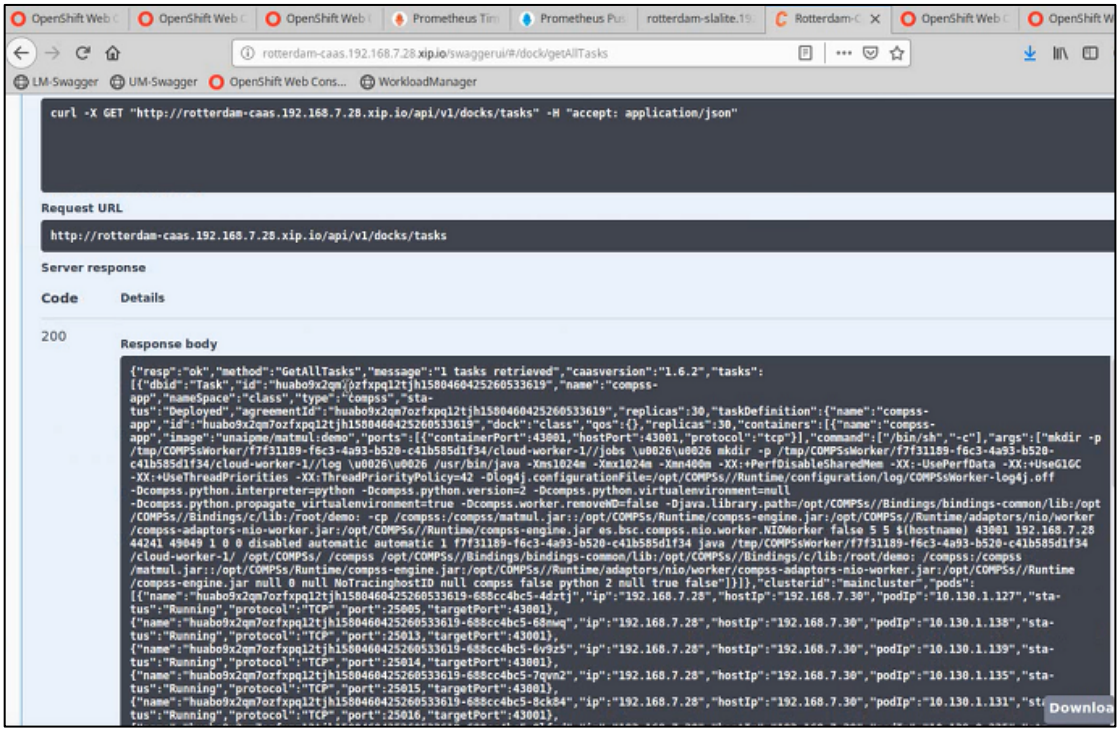
Step 6: Checking SLA (min 01:30) – *SLA Manager Web Interface*



Step 7: Rotterdam continues scaling out the workers after more violations are generated (min 01:42) - *OKD Web Interface*



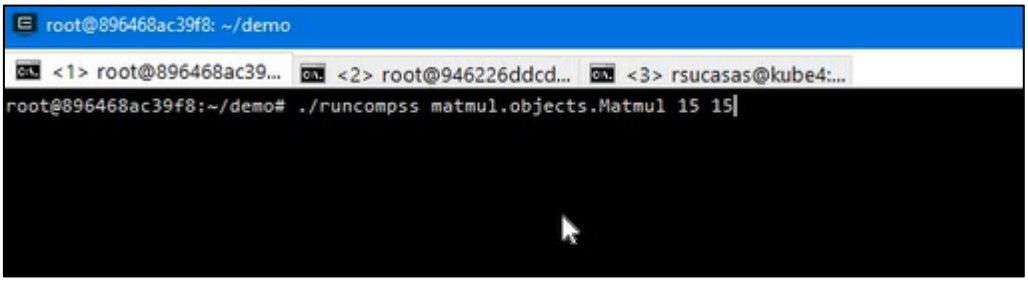
Step 8: Checking COMPSs workers ports exposed by Rotterdam. These ports are used by COMPSs master to execute the workflow tasks (min 02:28) - *Rotterdam Swagger Web Interface*



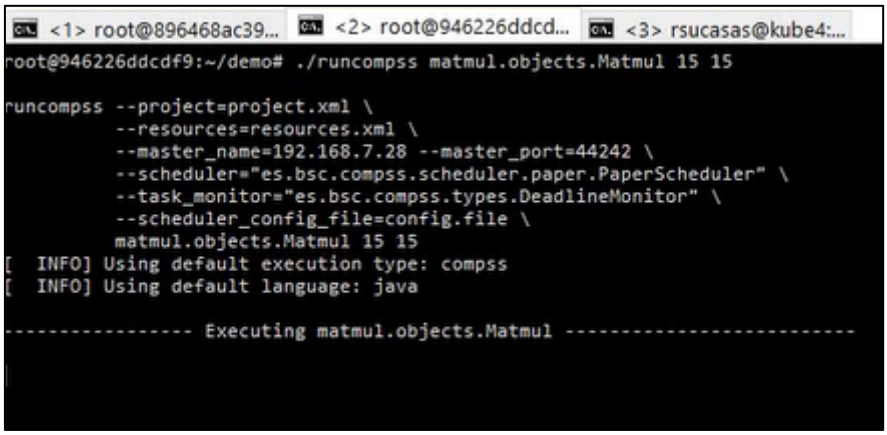
integrationcompssrotterdam_x2_-_part_1.mp4

integrationcompssrotterdam_x2_-_part_2.mp4

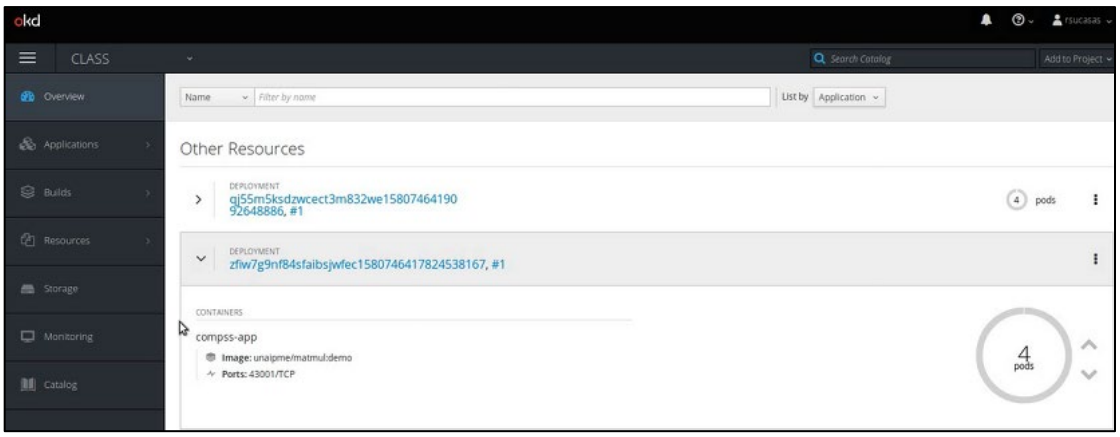
Step 1: Launching COMPSs master **A** application (min 00:33) – *COMPSs master application A*



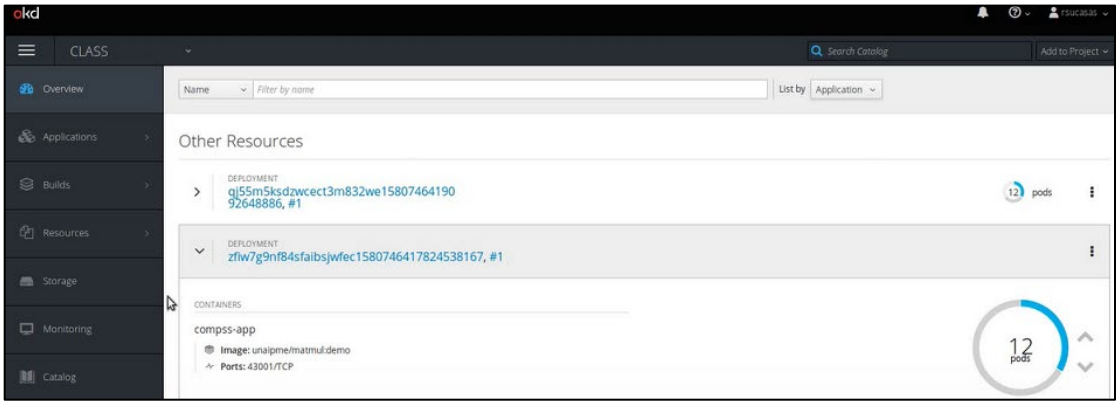
Step 2: Launching COMPSs master **B** application (min 00:35) – *COMPSs master application B*



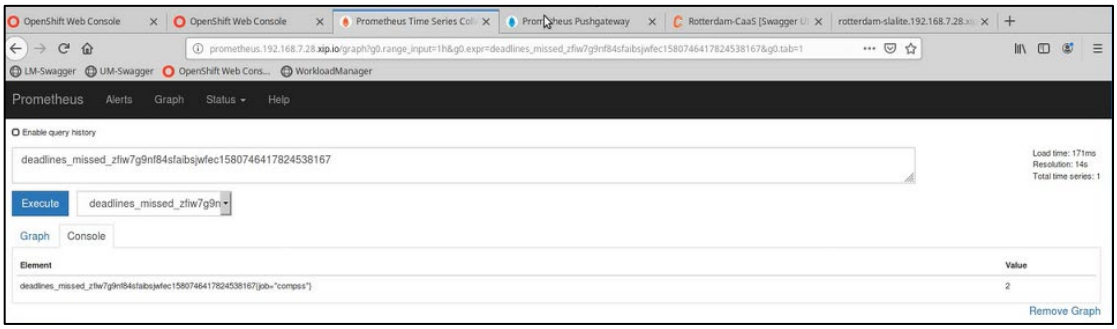
Step 3: Rotterdam deploys in "class" namespace the workers of the two workflows (min 00:39) - *OKD Web Interface*



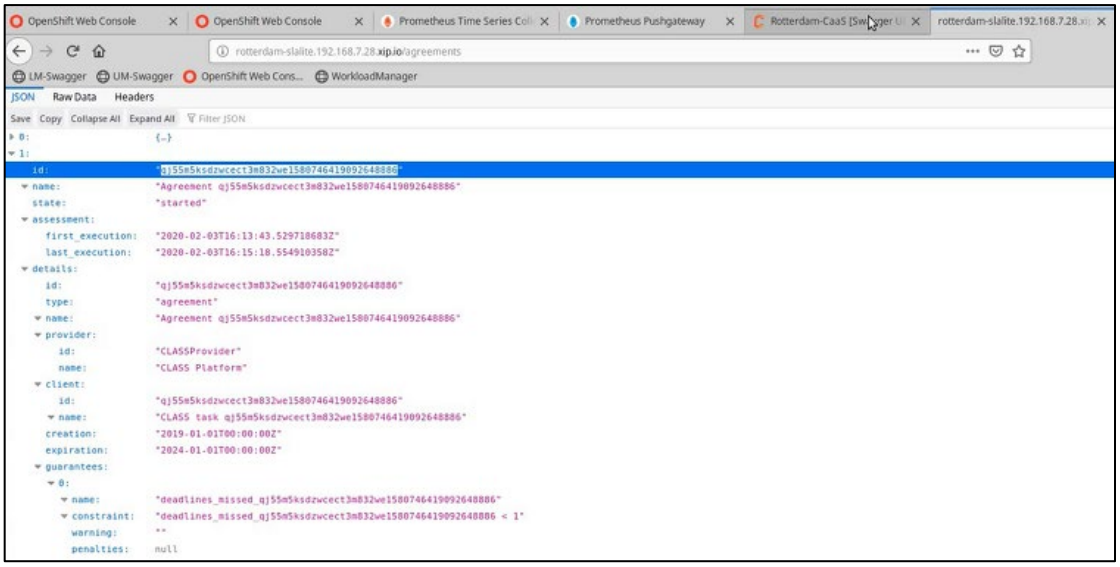
Step 4: Rotterdam scales up the number of workers of the two workflows after getting violations from the SLA Manager (min 01:04) - *OKD Web Interface*



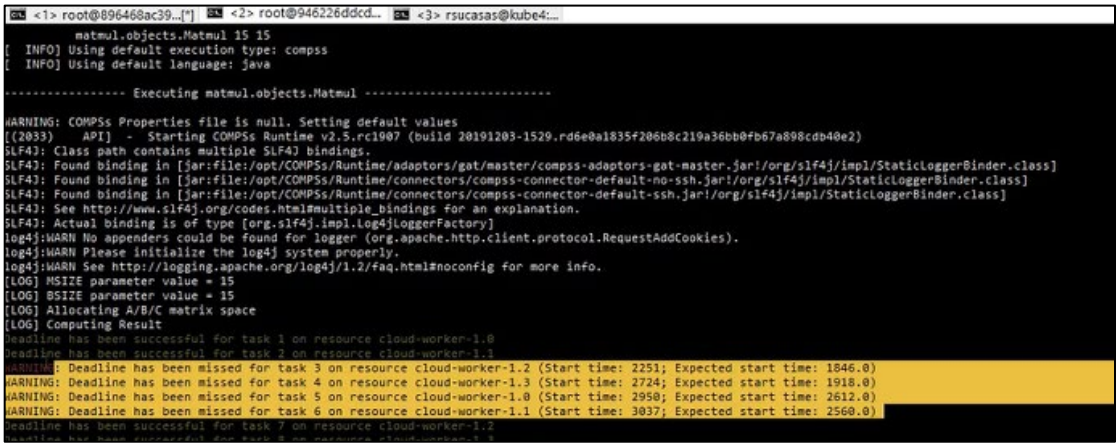
Step 5: Prometheus shows the metrics that generated the violations (min 02:01) - *Prometheus Web Interface*



Step 6: Two SLAs were created. One for each workflow (min 02:29) – *SLA Manager Web Interface*



Step 7: Violations are shown in master **B** application console (min 03:095) – *COMPSs master application B*



5.3 Management of multiple clusters in Edge and Cloud

This demo shows the management of multiple orchestrators and applications from the main Rotterdam instance deployed in the Cloud Data Analytics Service Management platform from Modena Data Center. This demo includes the coordination of Rotterdam, Openshift, MicroK8s, the SLA Manager and Prometheus.

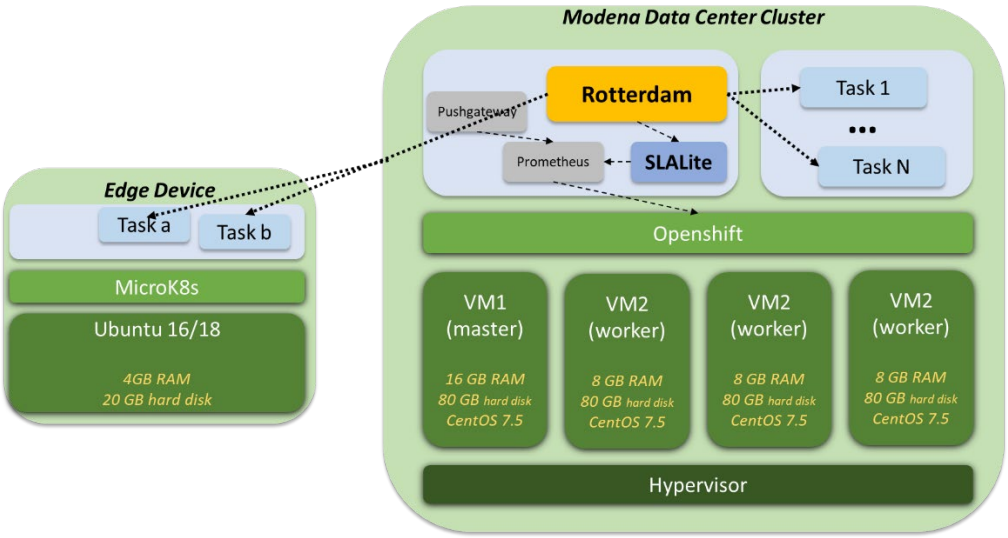


Figure 35 – Management of multiple clusters and applications

The following features are shown in this demo:

- Management of multiple infrastructures: main cluster's orchestrator and one Edge device
- Deployment of a MicroK8s orchestrator in an Edge device
- Execution of multiple applications in the Cloud platform and in the Edge device

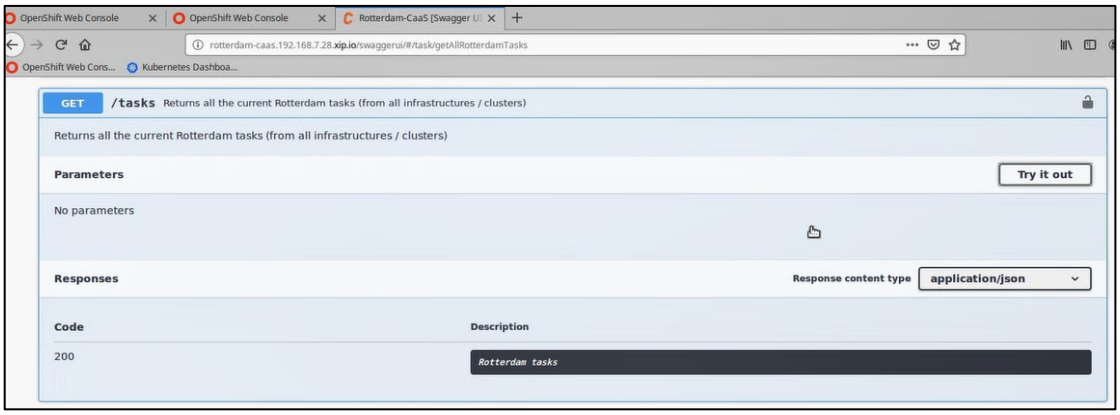
The demonstration steps presented in the following subsection make use of two different graphical interfaces and one console window:

- OKD-Openshift Web UI of the Cloud platform
- Rotterdam Swagger Web Interface
- Console window of the Edge device

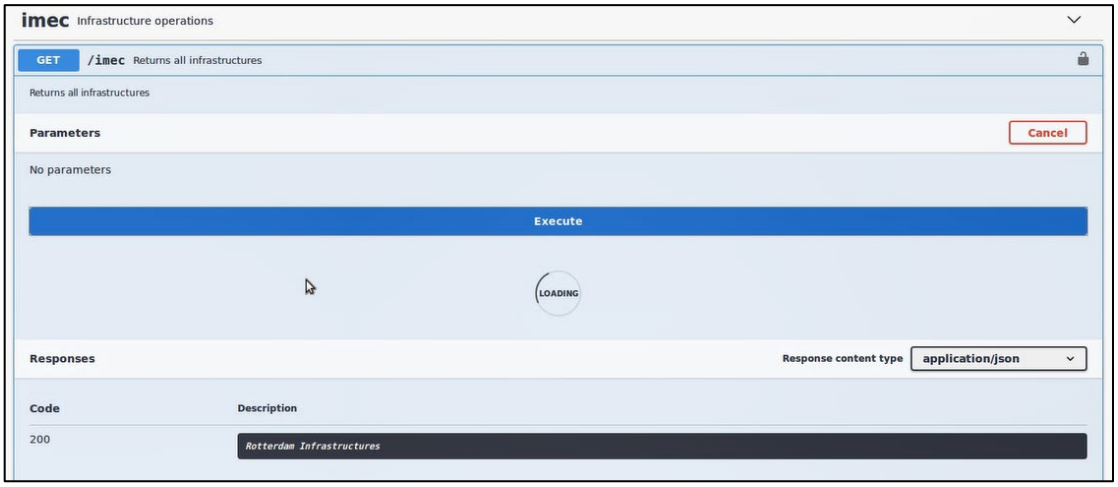
5.3.1 Demo

rotterdammultipleorchestratorsconverted.mp4

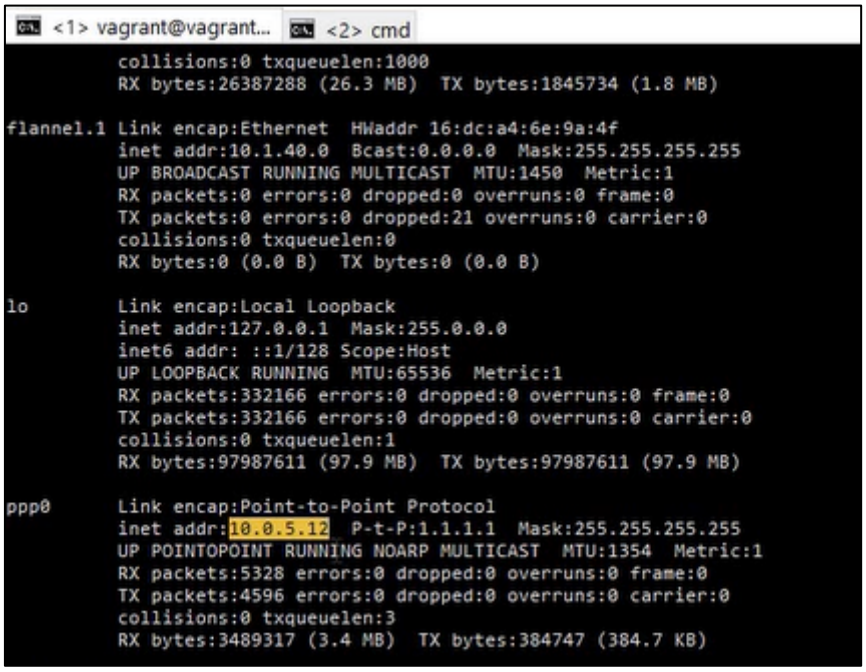
Step 1: Check existing Rotterdam tasks (min 00:34) - *Rotterdam Swagger Web Interface*



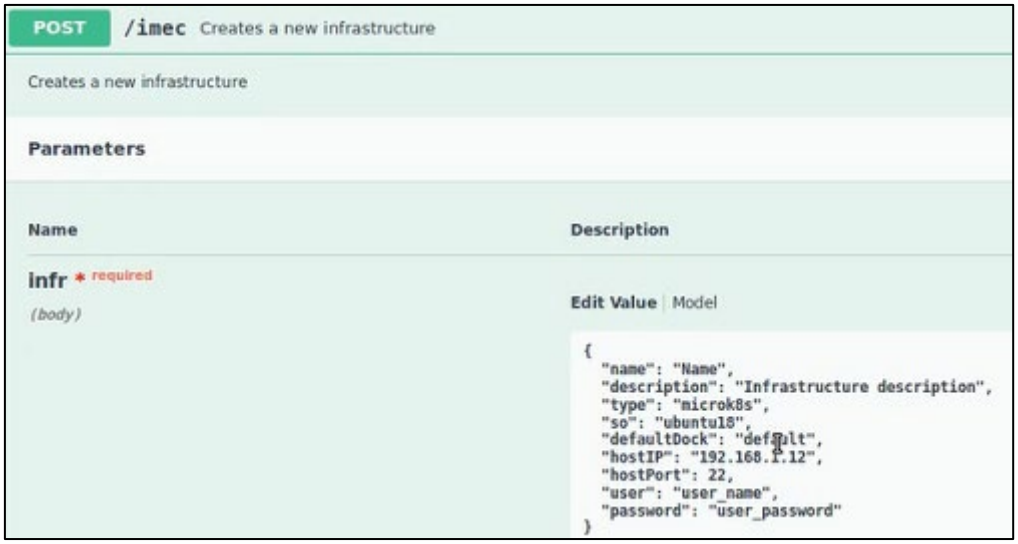
Step 2: Check existing infrastructures / orchestrators managed by Rotterdam (min 00:59) - *Rotterdam Swagger Web Interface*



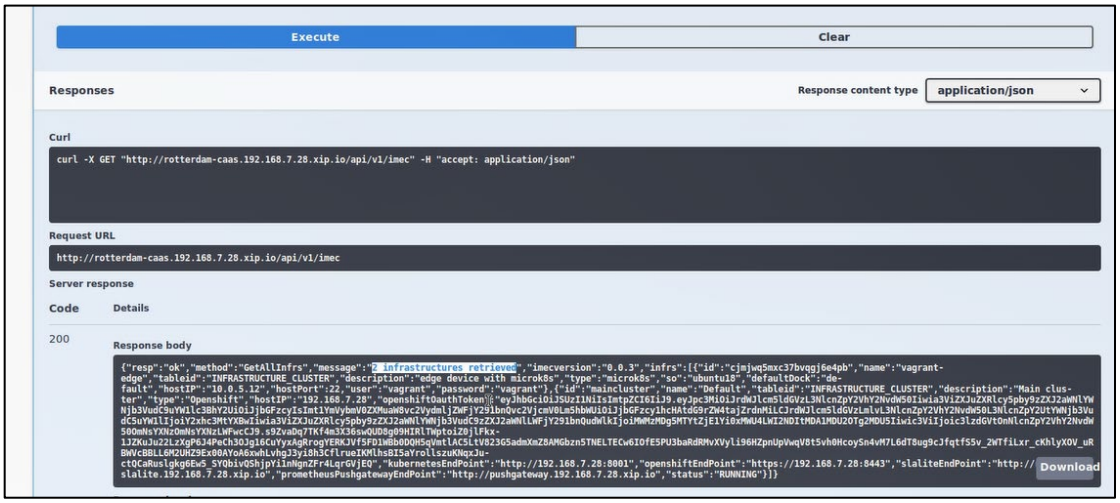
Step 3: Getting IP address from Edge device (min 01:34) - *Edge device console*



Step 4: Creating a new infrastructure / connection to the Edge device (min 01:49) - *Rotterdam Swagger Web Interface*

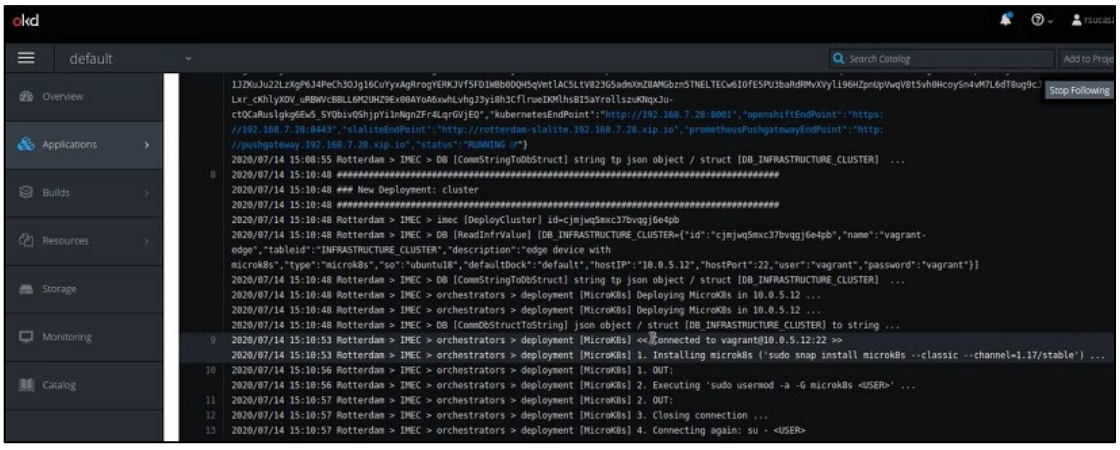


Step 5: Check existing infrastructures / orchestrators managed by Rotterdam (min 03:23) - *Rotterdam Swagger Web Interface*

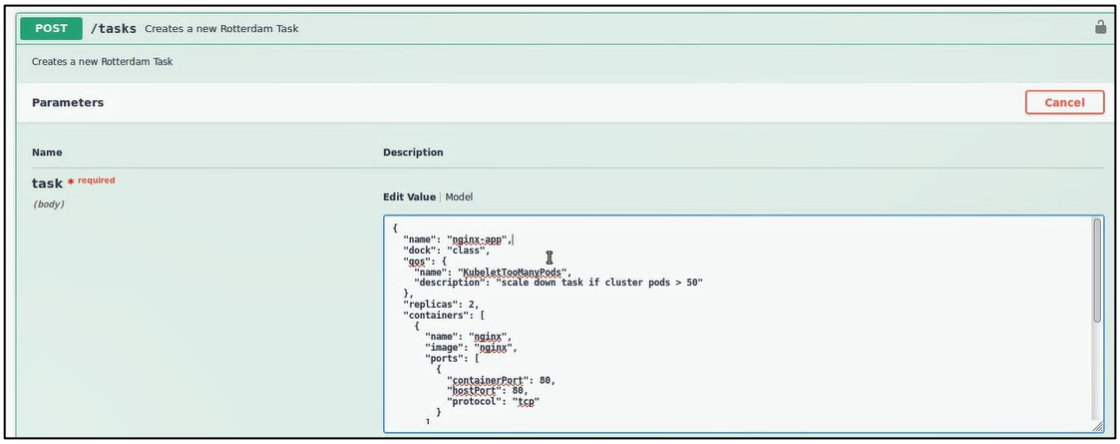


Step 6: Deploying a MicroK8s instance in the Edge device (min 03:33) - *Rotterdam Swagger Web Interface*

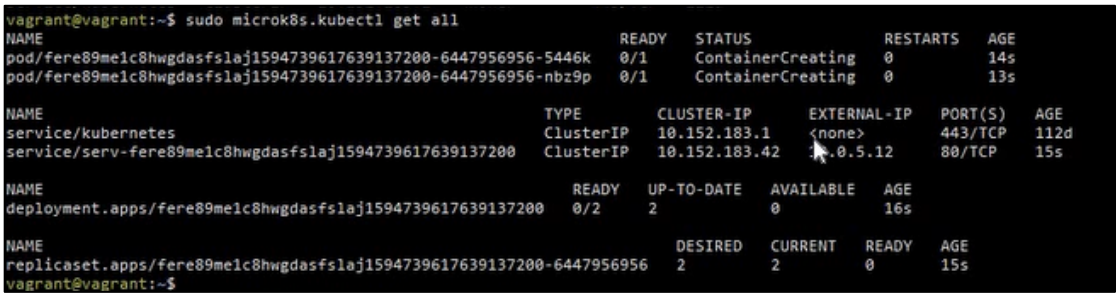
Step 7: Checking Rotterdam logs – MicroK8s installation (min 05:24) – *OKD Web Interface*



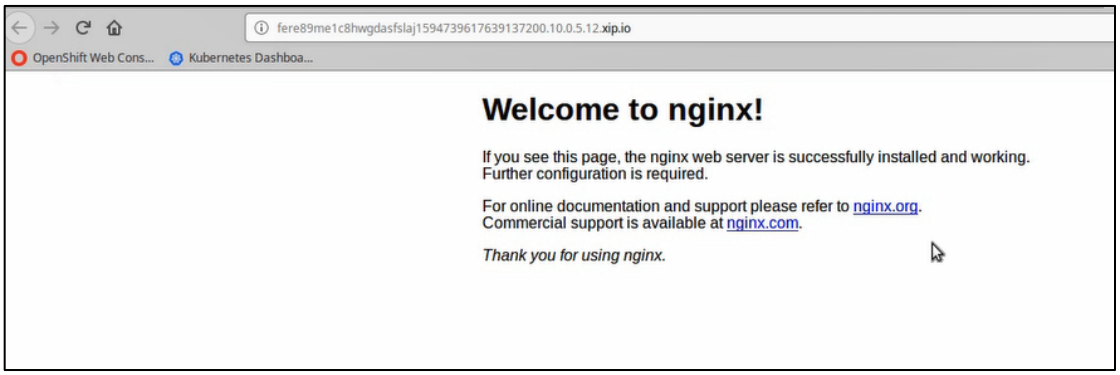
Step 8: Deploying an nginx server application on the Edge device (min 07:15) - *Rotterdam Swagger Web Interface*



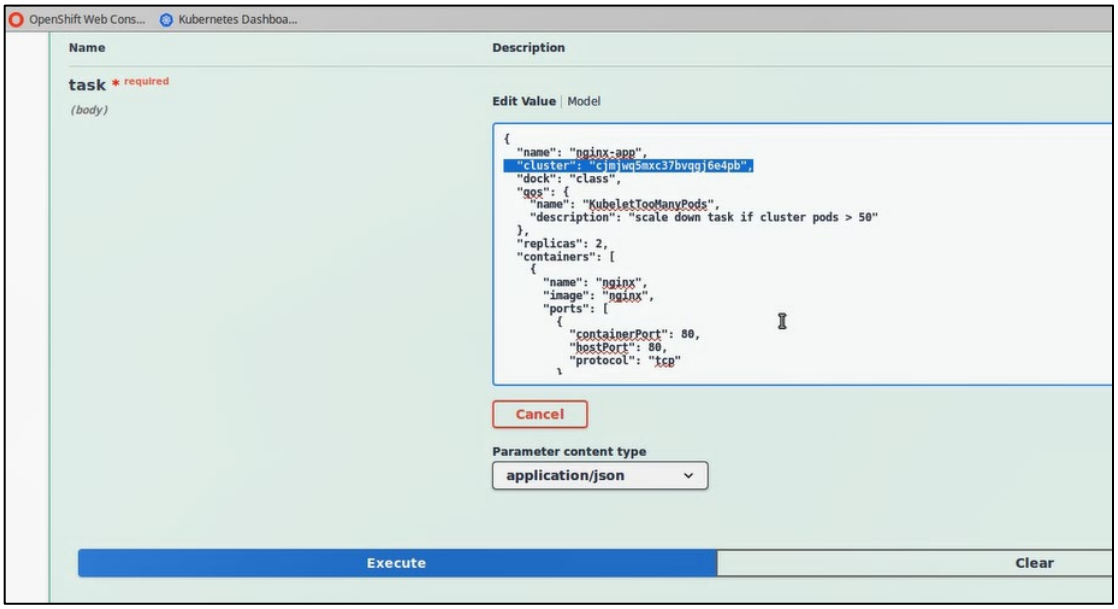
Step 9: Checking tasks deployed on the Edge device (min 08:23) - *Edge device console*



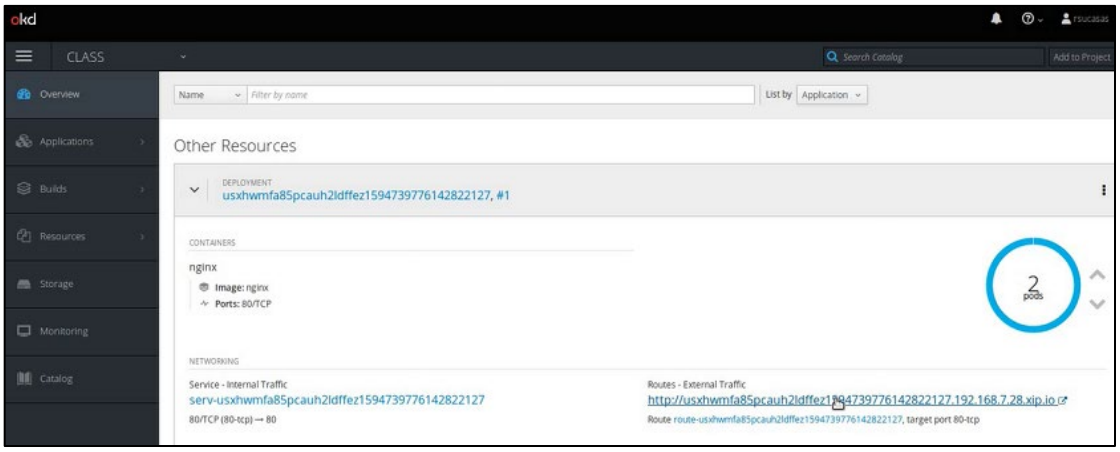
Step 10: Nginx server application from Edge device (min 09:20)



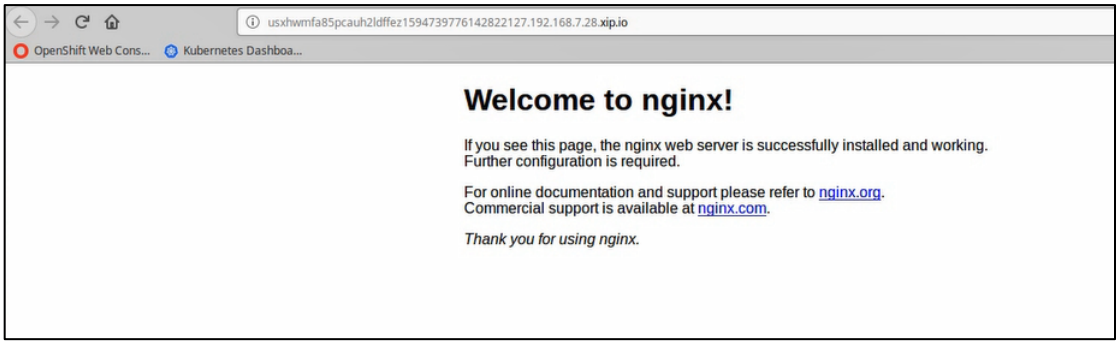
Step 11: Deploying an nginx server application on the main cluster - Openshift (min 10:02) - *Rotterdam Swagger Web Interface*



Step 12: Checking tasks deployed on the Openshift device – "class" namespace / project (min 10:25) - *OKD Web Interface*



Step 13: Nginx server application from Openshift - "class" namespace (min 10:37)



6 Conclusion

This deliverable reported on the work done in WP4 from M16 to M29, and the contributions done by ATOS to WP3 during the same period. The target at milestone MS3 of tasks 4.2 and 4.3 has been successfully achieved and documented in this deliverable: A cloud and edge environment for data analytics service management and scalability. The same applies to task 3.2, “Develop, experiment and evaluate edge platform agent for analytics”.

This deliverable also presented the code, guides and the instructions to install and manage the complete Cloud and Edge environment platform, including use examples and two demonstrations (videos), one of them made to the project team in Madrid’s face to face meeting (February 2020).

The progress done in the last two milestones will pave the way for the next phase where the goal will be evaluating the Use Cases using the Cloud and Edge platform environment presented in this deliverable. For this purpose, we will continue the development and improvement of the platform.

References

[1] C. -. DoW, “Edge&CLoud Computation: A Highly Distributed Software Architecture forBig Data AnalyticS (proposal document)”.

[2] CLASS, “D4.1 Cloud Requirement Specification and Definition,” June 2018.

[3] CLASS, “D4.2. First release of the Cloud Data Analytics Service Management components,” March 2019.

[4] CLASS, “D4.4. First release of the Cloud Data Analytics Service Scalability components,” March 2019.

[5] Atos-SLALite, “https://www.mf2c-project.eu/wp-content/uploads/2017/12/D4.5-mF2C-Platform-Manager-block-and-microagents-integration-IT-1.pdf (Section 3.3)”.

[6] Atos-mF2C, “mF2C (Towards an Open, Secure, Decentralized and Coordinated Fog-to-Cloud Management Ecosystem),” [Online]. Available: https://www.mf2c-project.eu/.