



## D4.6 Decision Support tools II

Document Identification			
<b>Status</b>	Final	<b>Due Date</b>	31/08/2022
<b>Version</b>	0.8	<b>Submission Date</b>	01/09/2022

<b>Related WP</b>	WP4	<b>Document Reference</b>	D4.6
<b>Related Deliverable(s)</b>	D2.2 Pledger Requirements Analysis, D2.3 Pledger Overall Architecture, D4.3 Decision Support Tools I, D3.4 Performance Measurements and classification tools II, D3.5 Edge/Cloud orchestration tools II, D3.6 QoS and SLA assessment and negotiation tools II, D5.6 Pledger integrated demonstrator II	<b>Dissemination Level (*)</b>	PU
<b>Lead Participant</b>	ENG	<b>Lead Author</b>	Francesco Iadanza (ENG)
<b>Contributors</b>	ENG, I2CAT	<b>Reviewers</b>	Lara Lopez (ATOS) Panagiotis Matzakos (INTRA)

### Keywords:

DSS, smart orchestration, resource management, end-to-end latency, SLA violation, energy efficiency

<b>Document name:</b>	D4.6 Decision Support tools II		<b>Page:</b>	1 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b> 0.8
			<b>Status:</b>	Final

## Document Information

List of Contributors	
Name	Partner
Estela Carmona	I2CAT
Gabriele Giammatteo	ENG
Francesco Iadanza	ENG
Keven Kearney	ENG

Document History			
Version	Date	Change editors	Changes
0.1	21/06/2022	Francesco Iadanza (ENG)	ToC with progress since D4.3
0.2	15/07/2022	Estela Carmona (I2CAT), Gabriele Giammatteo (ENG), Francesco Iadanza (ENG), Keven Kearney (ENG)	Draft with all sections covered
0.3	29/07/2022	Panagiotis Matzakos (INTRA)	First internal review
0.4	02/08/2022	Francesco Iadanza (ENG)	Update after internal review
0.5	08/08/2022	Lara López (ATOS)	Second internal review
0.6	25/08/2022	Francesco Iadanza (ENG)	Version ready for quality check
0.7	31/08/2022	Carmen San Román (ATOS)	Quality assurance review
0.8	01/09/2022	Lara López (ATOS)	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Francesco Iadanza (ENG)	25/08/2022
Quality manager	Carmen San Román (ATOS)	31/08/2022
Project Coordinator	Lara López (ATOS)	01/09/2022

# Table of Contents

---

Document Information .....	2
Table of Contents .....	3
List of Tables.....	5
List of Figures .....	6
List of Acronyms.....	7
Executive Summary .....	8
1 Introduction .....	9
1.1 Purpose of the document.....	9
1.2 Relation to the other project work.....	9
1.3 Structure of the document.....	9
1.4 Progress for the second project period.....	9
1.5 Glossary adopted in this document .....	10
2 Functional description .....	11
2.1 Objective of the DSS.....	11
2.2 Description of the functional components .....	14
2.3 DSS interactions with Pledger core components .....	18
2.4 Outline of the DSS innovations.....	20
3 Technical description.....	21
3.1 Baseline technologies and dependencies .....	21
3.2 Components Architecture.....	21
3.3 Class diagram.....	24
3.4 DSS insight .....	27
3.5 DSS data model, REST API and integration example.....	29
4 Installation and usage guides.....	31
4.1 Requirements .....	31
4.2 Installation.....	31
4.3 Usage.....	32
4.4 Test and validation.....	32
4.5 License .....	33
4.6 Source code repository .....	33
5 Demonstration .....	34
5.1 Configuration service.....	34
5.1.1 First project period .....	34
5.1.2 Second project period.....	34
5.2 DSS .....	34
5.2.1 First project period .....	34

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	3 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

5.2.2	Second project period.....	35
5.3	Validation and Verification.....	37
6	Conclusions .....	38
7	References .....	39
8	Annex .....	41
8.1	Configuration UI screenshots.....	41
8.1.1	Administrator.....	41
8.1.2	Infrastructure Provider.....	43
8.1.3	Service Provider. ....	44

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	4 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

## List of Tables

---

*Table 1 : Pledger Configuration subsystem functional components implemented by the DSS ..... 13*  
*Table 2 : Pledger Orchestrator subsystem functional components implemented by the DSS ..... 13*

<b>Document name:</b>	D4.6 Decision Support tools II				<b>Page:</b>	5 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8	<b>Status:</b> Final

## List of Figures

---

Figure 1: DSS high-level schema	11
Figure 2: Pledger functional subsystems	12
Figure 3: Pledger functional layer components: DSS and ConfService	12
Figure 4: DSS decision tree for “resources” optimization	15
Figure 5: DSS optimizations and possible scenario	16
Figure 6: DSS not managing edge resources exclusively	17
Figure 7: ECODA + "resources" algorithm	17
Figure 8: EA-ECODA algorithm	18
Figure 9: DSS used as standalone component, for tests validation	19
Figure 10: DSS interactions with the other core components	20
Figure 11: DSS component diagram	22
Figure 12: Sequence diagram about configuration data exchange: SLA Manager	23
Figure 13: DSS sequence diagram to receive SLA violations	24
Figure 14: DSS sequence diagram to send commands to the Orchestrator	24
Figure 15: Configuration service and DSS main class diagram	25
Figure 16: Monitoring configuration example	27
Figure 17: Service optimization configuration example	28
Figure 18: App management configuration example	28
Figure 19: DSS Swagger-UI accessible from the UI – App REST resource example	29
Figure 20: DSS integration with NodeRed example	30
Figure 21: DSS detailed install instructions	31
Figure 22: Snapshot of the DSS documentation folders on Gitlab	32
Figure 23: Setup of a Kubernetes cluster to test the DSS	33
Figure 24: Documentation about how to run/validate tests on a Kubernetes cluster	33
Figure 25: Configuration and DSS service demos for the first project period on YouTube	35
Figure 26: DSS service demos for the second project period on YouTube	36
Figure 27: Pledger Integration Demo #1	36
Figure 28: Administrator manages Pledger users	41
Figure 29: Users with roles	41
Figure 30: Users login audit	42
Figure 31: Definition of Pledger SP and IP users	42
Figure 32: Infrastructure with monitoring plugin and resources capacity	43
Figure 33: Nodes with feature auto-discovery features working to identify HW availability	43
Figure 34: Catalog applications	44
Figure 35: DApp composed by multiple services with K8S descriptor	44
Figure 36: Service with descriptor and initial configuration about resources (Kubernetes)	45
Figure 37: SLA definition	45
Figure 38: Guarantees definition	46
Figure 39: ServiceConstraints, setup by the SP, to express deployment preferences	46

## List of Acronyms

Abbreviation / acronym	Description
API	Application Programming Interface
CI/CD	Continuous Integration and Deployment
DApp	Decentralised App
DB	Data Base
DLT	Distributed Ledger Technology
DSS	Decision Support System
Dx.y	Deliverable number y belonging to WP x
EA-ECODA	Energy Aware Edge-to-Cloud Offloading Decision Algorithm
ECODA	Edge-to-Cloud Offloading Decision Algorithm
IaaS	Infrastructure as a Service
IP	Infrastructure Provider
JDBC	Java Data Base Connectivity
JSON	JavaScript Object Notation
MAPE	Monitoring, Analysis, Planning and Execution
MVP	Minimum Viable Product
Mx	Project Month x
OS	Operating System
QoS	Quality of Service
RDBMS	Relational Database Management System
REST	REpresentational State Transfer
SaaS	Software as a Service
SLA	Service Level Agreement
SP	Service Provider
SQL	Simple Query Language
SUC	System Use Case
TTODA	Three-Tier Offloading Decision Algorithm
Tx.y	Task number y belonging to WP x
UI	User Interface
WP	Work Package

## Executive Summary

---

This document accompanies the delivery of the final release of the Decision Support System (DSS) developed in Task 4.3 “Decision Support mechanisms for Edge/Cloud computation moving” (M6-M33) which includes Pledger’s Configuration and Recommender functional subsystems described in D2.3.

DSS is a component designed and developed in Pledger to allow service providers allocate resources for Decentralised Applications while reducing Service Level Agreement violations, end-to-end latency for cloud, edge, and far-edge infrastructures, taking into consideration the energy consumption on the edge.

Some of the introduced optimisations have been accompanied by novel IEEE scientific publications, referred later in this document, as well as proper documentation on Gitlab and demos on YouTube, used to promote the results in an IEEE ICC demo conference paper in 2022.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	8 of 46		
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8	<b>Status:</b>	Final

# 1 Introduction

---

## 1.1 Purpose of the document

---

The purpose of this deliverable (D4.6) is to provide functional and technical description of the main functionalities, functional components, and services of the Pledger Decision Support tools for task T4.3, named as Decision Support System (DSS) within this document for convenience.

This deliverable accompanies the tools final release and provides installation, documentation, and usage guidelines, as well as demos of the main functionalities.

## 1.2 Relation to the other project work

---

The DSS and the present deliverable are guided by the work performed in WP2 about the requirement analysis (D2.2[1]) and the architecture (D2.3[2]), and use the first release of this document, the D4.3[3], as a baseline to present the complete and updated list of features available in the final DSS release.

The DSS is coupled with the core Pledger components developed in WP3: in particular, it relies on the Benchmarking and AppProfiler (D3.4[4]) for the selection of the best infrastructure to host a Decentralised Application (DApp), on the Orchestrator (D3.5[5]) to deploy DApps on an infrastructure, on the Service Level Agreement (SLA) Manager and the Monitoring Engine (D3.6[6]) to receive SLA violations and monitoring data.

Finally, DSS integration details with the other Pledger core components is documented in D5.6[7].

## 1.3 Structure of the document

---

This document is structured in 6 sections :

- ▶ Section 2 presents the DSS functional description aligned with the overall architecture of Pledger and the new features available.
- ▶ Section 3 presents the technical description of the DSS with reference to the initial features described in D4.3 and focus on those introduced in the final release.
- ▶ Section 4 presents the updated installation and usage guide of the DSS.
- ▶ Section 5 presents the updated DSS demos.
- ▶ Section 6 presents the conclusions.

## 1.4 Progress for the second project period

---

The first release of the Pledger Decision Tools has been presented in in D4.3 in M20 covering the **first project period** (until M20), where such tools focused more on the configuration options as well as the management of the data exchanged with other Pledger core components. It also includes a first version of the optimization algorithm to introduce the updates needed to manage private, scarce, and diverse edge resources using SLA violations as feedback.

For the **second project period** (from M21 until M33), this release has been used as a baseline for the design and implementation of new optimization algorithms, some accompanied by novel IEEE scientific publications, along with finer-grained Service Level Agreements (SLA) violations management.

As it will be further detailed in this document, the major advancements in this period have been:

- ▶ improved SLA violations management with the options to process, ignore or suspend them within the DSS optimizations.
- ▶ refactoring of the initial optimization presented in D4.3 to allow finer grained control.
- ▶ an optimization algorithm (named “**ECODA**”) to reduce end-to-end latency and resource optimization in cloud-edge infrastructure, based on a novel IEEE publication [29].
- ▶ an optimization algorithm (named “**TTODA**”) to reduce end-to-end latency and resource optimization in cloud-edge-faredge infrastructure, based on a novel IEEE publication [30].

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	9 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

- ▶ an optimization algorithm (named “EA-ECODA”) to reduce end-to-end latency, energy consumption on the edge and resource optimization in cloud-edge infrastructure.
- ▶ Extended configuration to support the above listed updates.

## 1.5 Glossary adopted in this document

---

- ▶ Pledger core system. The system that aggregates all the core results and developed components of the project. Corresponding components result in the Minimum Viable Product (MVP) of Pledger, the system that will be freely available to users and other projects and initiatives. Most of the core components represent the exploitable part of the project and contain all the main features that will deem Pledger successful as a project.
- ▶ Pledger core subsystems. The subsystems that belong to the Pledger core system.
- ▶ (Functional) Component. A module / component of the system offering a specific functionality.
- ▶ Asset. A tool already available by one of the partners of the consortium for the materialisation of functional components or subsystems.
- ▶ (Functional) Subsystem. A group of interrelated functional components.
- ▶ Decision Support System (DSS). The Pledger Decision Support tools developed in task T4.3 and described in this document.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	10 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

## 2 Functional description

The Decision Support tools have been specifically designed for Pledger and allow service providers to optimize resource usage, latency and energy consumption on the edge based on the different needs of the Decentralised Applications (DApps), with the goal to minimize the SLA violations and fulfill the agreed Quality of Service (QoS).

For convenience, in this document Decision Support Tools are named “Decision Support System” (**DSS**); also, the key features described in D4.3 for the first project period are reported in this document to provide a comprehensive description of the DSS, highlighting the progress achieved since the first release.

### 2.1 Objective of the DSS

The DSS implements a complete Monitoring, Analysis, Planning and Execution (MAPE) loop for cloud-edge-faredge infrastructures. The goal is to find the best node to deploy DApps, while providing different optimizations and supported scenarios to the service provider, backed up by continuous SLA violation monitoring to fulfil the agreed QoS. Figure 1 shows the high-level schema of the DSS.

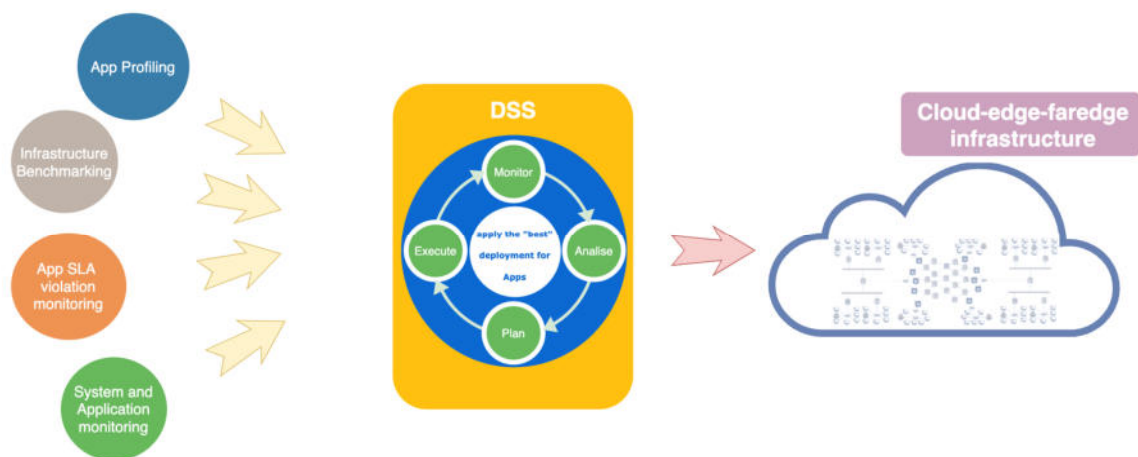


Figure 1: DSS high-level schema

Within Pledger architecture presented in D2.3, the DSS is positioned as part of the Management layer, spanning over the **Configuration** and the **Orchestration** subsystems, as shown in Figure 2.

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	11 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

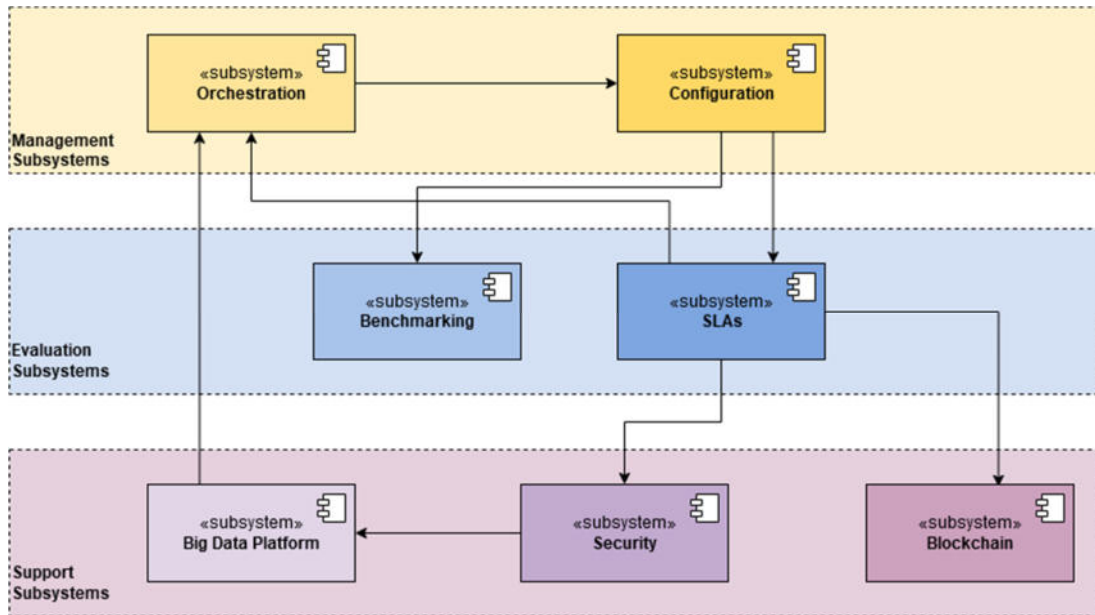


Figure 2: Pledger functional subsystems

Similarly, Figure 3 extracted from the architecture deliverable (D2.3), identifies the functional components integrated with the DSS : highlighted in **blue** those for the decision making, for simplicity later referred as **DSS** ; highlighted in **red** those related to the configuration to support the DSS activities, later referred as **ConfService**.

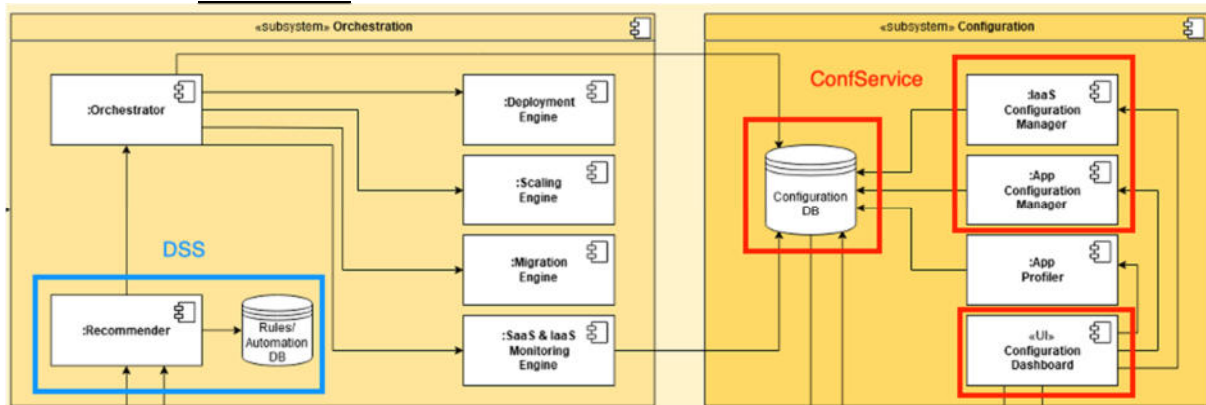


Figure 3: Pledger functional layer components: DSS and ConfService

For the **Configuration** subsystem (with no major updates since the **first project period**) the following functional components : IaaS and App Configuration Manager, merged into the Configuration Service backend, the Configuration Dashboard, which is the Configuration Service User Interface (UI), and the Configuration Data Base (DB) used to persist data.

The list of the Configuration subsystem functional components implemented by the DSS, and their roles are reported in Table 1, already presented in D4.3 and reported here for the sake of readability.

Table 1 : Pledger Configuration subsystem functional components implemented by the DSS

ID	Component	Functionality
FC.1.1	<b>IaaS Configuration Manager</b>	The IaaS Configuration Manager is responsible for the management of infrastructure configuration, spanning from the credentials to the main topology and properties of the infrastructure (such as the servers URL, the master/worker properties for Kubernetes, GPU type, CPU model, etc.) that could impact scheduling decisions, as well as resource limits configured from the Infrastructure as a Service (IaaS) providers.
FC.1.2	<b>App Configuration Manager</b>	The App Configuration Manager is responsible for the management of the app configuration, which includes generic information used to match (Software as a Service) SaaS providers' preferences expressed in their profiles, along with those specific related to QoS and SLAs. QoS keys are listed for each application and SLA values/ thresholds are stored to allow the SLA Manager check their violations and prioritize them.
FC.1.4	<b>Configuration DB</b>	The Configuration DB is responsible for storing the aforementioned configuration information and sharing it through specific Application Programming Interface (API) to the other Pledger subsystems, such as the Recommender in the Orchestration subsystem, the Benchmarking and the SLA creators' components.
FC.1.5	<b>Configuration Dashboard</b>	The Configuration Dashboard is the UI provided to IaaS and SaaS users to allow the proper configuration of the aforementioned data and also includes reports to allow the SaaS users to have a detailed view of the infrastructures and applications status and the recommendations for the orchestration.

In the **second project period**, the advancement has been in FC.1.4 and FC.1.5 to support the configuration needed by the novel DSS optimisations later described in the next subsection.

Similarly, for the **Orchestration** subsystem (with some updates since the **first project period**, later described) the DSS implements the following functional components : the Recommender and Rules/Automation DB. The latter connects to the Configuration DB to access configuration data and produce actions for the Orchestrator (eg., scaling or offloading). The list of the Orchestrator subsystem functional components implemented by the DSS, as well as their roles are reported in Table 2, which is already presented in D4.3.

Table 2 : Pledger Orchestrator subsystem functional components implemented by the DSS

ID	Component	Functionality
FC.2.6	<b>Recommender</b>	The Recommender component is responsible to provide <b>automations</b> to the SaaS providers related to the app orchestration. In particular, the automations focus on the most efficient allocation within the available infrastructures, taking into account : <ul style="list-style-type: none"> <li>▶ the infrastructure and app properties from the configuration subsystem,</li> <li>▶ the infrastructure and app status from the IaaS Monitoring component and the SLA Manager</li> <li>▶ the user preferences stored again in the configuration subsystem</li> </ul>
FC.2.7	<b>Rules/Automation DB</b>	The Rules/Automation DB component stores the Recommender <b>configuration</b> and includes rules to decline the decision-making process (with regard to the user preferences) and provide automations.

In the **second project period**, the advancement has been in FC.2.6 and FC.2.7 in the design and implementation of additional and more sophisticated algorithms to provide automatic scaling and offloading based on different infrastructure and application metrics as described in the next subsection.

## 2.2 Description of the functional components

---

In Pledger, the DSS relies on the orchestrator and the monitoring components to offer an end-to-end smart orchestration on hybrid edge and cloud infrastructures, where benchmarking and application profiling allow to deal with hardware diversity.

The main scenario explored is with limited resources on the edge, where different applications might compete to keep computation on the edge, with the goal to reduce end-to-end latency and energy consumption on the edge while fulfilling the agreed QoS.

In the **first project period**, release of the DSS presented in D4.3 focused on the aggregation of the different data used to take decisions and the integration with the other Pledger components. It also focused on the configuration of the service providers preferences to prioritise deployment options, for example to prefer edge infrastructures over cloud, or specific nodes based on their location or hardware. This feature allows a great degree of configurability through a dedicated User Interface (UI).

The first DSS release also included a first optimisation algorithm inspired by the Kubernetes QoS model based on **reserved resources** and **autoscaling**, but assuming **limited resources** on the edge. It relies on SLA violations as feedback to drive the decisions about either to scale or offload depending on resource availability and service provider preferences.

In fact, Kubernetes was initially designed for the cloud, where adequate capacity planning ensures resources availability and application can scale transparently, whereas on the edge this is not always possible. So, while on the cloud autoscaling perfectly matches the needs to provide some QoS relying on unlimited resources, in Pledger their main scenario explored has been with scarce edge resources, where proper allocation is crucial and can heavily affect agreed QoS.

This first optimization, later referred as “**resources**” optimization, was provided to set the ground for the others now available in the final DSS release. In “**resources**” optimization, according to the Service Provider (SP) preferences, which in Pledger might prioritise edge over cloud, the DSS also verifies if a higher priority deployment option is available and, in this case, it triggers the placing of services on different nodes. Mainly, this means horizontally/vertically autoscale services within the edge, then offload to cloud if there is no more space, and the opposite if resources become available on the edge. Similarly, the DSS checks if the DApp is requesting more resources than necessary, meaning the actual usage is far below the resource reserved and no SLA warning/violations have been received for a specific amount of time. In this case, the DSS gradually decreases resources with a **percentage** which can be configured. At the same time, the DSS gradually increases resources allocated to each DApp if they produce any SLA warning/violation. Eventually this will lead to saturating the edge resources and result in offloading to the cloud. The aforementioned checks about increasing/decreasing resources are done in parallel by the DSS in order to find the optimal resource allocation : Figure 4 shows the decision tree used by the DSS for “resources” optimization.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	14 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

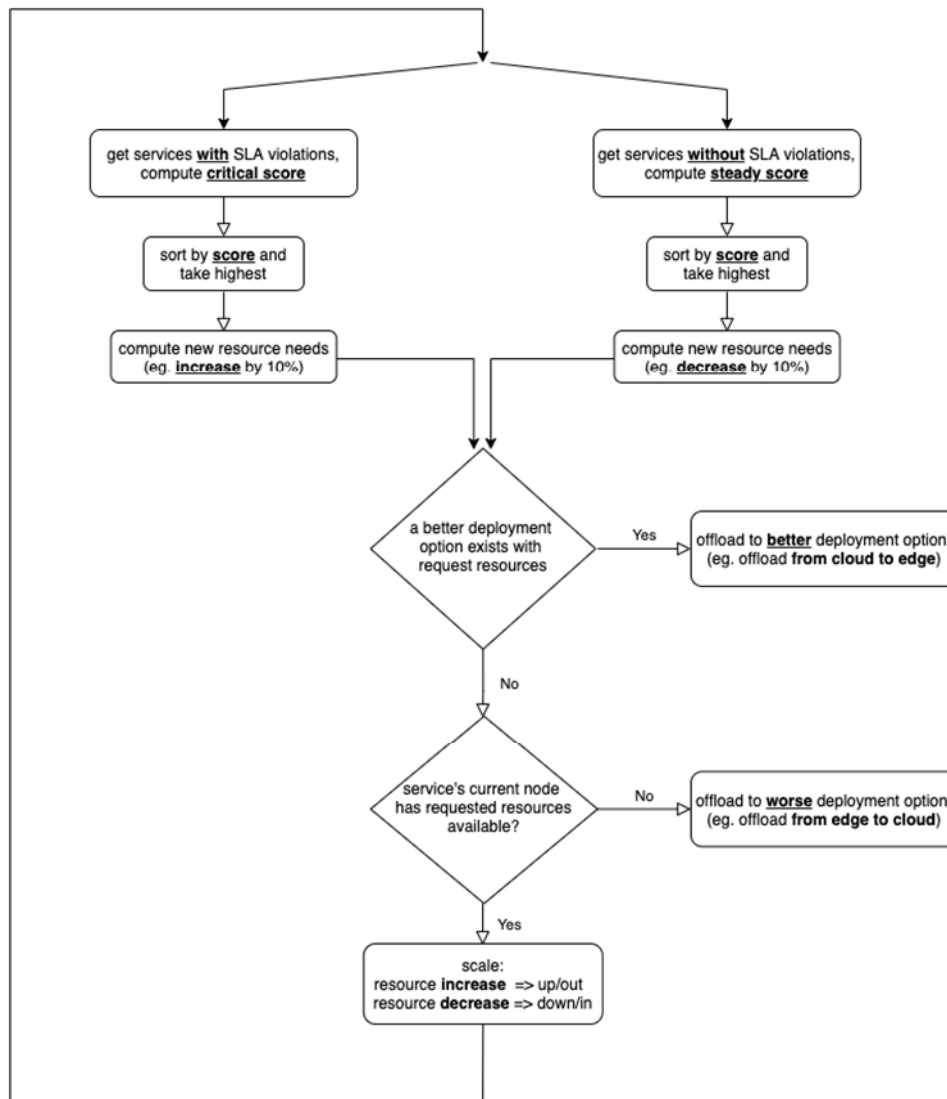


Figure 4: DSS decision tree for “resources” optimization

To summarize, “resources” optimization was introduced in the first DSS release to focus on resource saturation on the edge and the usage of SLA violations as feedback to achieve agreed QoS. This also allowed to test the end-to-end flow and set the ground for the next activities.

Also, this initial optimization allowed to manage all possible autoscaling (both **horizontal** and **vertical**) as well as all possible offloading combination (eg. **edge-to-edge**, **edge-to-cloud**) based on the priorities set by the SP, although all UCs and demonstrators have been configured to privileged edge over cloud and use edge resources as much as possible.

In the **second project period**, the latency measurements retrieval using GoldPinger[14] has been consolidated to monitor end-to-end latency, DApps boot time has also been monitored through Metrics-server[13], so that they are now used by the DSS optimization algorithms. Also, SLA configuration has been improved to allow multiple ways to handle the violations, so it is now possible to include them within the optimization, or completely ignore them (for example, because they need to be used by other components) or let the optimization process be suspended (for example, because they notify a major inconvenience on the network which is excluded by an SLA).

With these additional features in place, the DSS has been improved to include several additional optimization algorithms, some described in novel IEEE publications and facing **different scenarios** shown in Figure 5, taken from the online documentation available together with the source code on the

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	15 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

DSS public repository on Gitlab [27] and the corresponding demo videos published on YouTube[28] reflecting the status of the development at the moment of writing this document (August 2022).

best scenario for each DSS optimisation	how many tiers is the infrastructure made of (2-3)?	are edge resources managed by the DSS exclusively?	use SLA violations as feedback to achieve agreed QoS?	is edge HW very variable?	is latency critical?	is edge energy consumption critical?	custom optimisation needed?
resources	edge-cloud	yes	yes	yes	no	no	no
offloading	edge-cloud	no	yes	yes	no	no	no
scaling	edge-cloud	no	yes	yes	no	no	no
latency	edge-cloud	yes	no	no	yes	no	no
resources_latency	edge-cloud	yes	yes	yes	yes	no	no
latency_faredge	faredge-edge-cloud	yes	no	no	yes	no	no
resources_latency_faredge	faredge-edge-cloud	yes	yes	yes	yes	no	no
resources_latency_energy	edge-cloud	yes	yes	yes	yes	yes	no

Figure 5: DSS optimizations and possible scenario

In particular, Figure 5 shows the different scenarios which might be faced by service providers, along with the suggested DSS optimization.

The supported **scenarios** are the following:

- ▶ the number of tiers the infrastructure is made of: it considers whether nodes are only cloud/edge or cloud/edge/faredge.
- ▶ whether edge resources are managed by the DSS in an exclusive way: this allows the DSS to either manage all edge resources or to support scenarios where high priority DApps **must** stay on edge and possibly **scale** under control of separate orchestrator, leaving to the DSS the management of the remaining resources.
- ▶ whether SLA violations are used as feedback to the DSS.
- ▶ whether latency is critical.
- ▶ whether HW on edge is homogeneous. If not, benchmarking is used together with AppProfiler to choose the best node possible for a given DApp.
- ▶ whether to include **energy** consumption consideration.

In the end, the DSS also includes a **custom optimization** through webhook calls to the DSS API to deal with specific needs outside those listed.

The DSS combines the above-mentioned scenarios and **provide the following optimisations** that can be configured through the UI:

- ▶ “resources”
- ▶ “scaling”
- ▶ “offloading”
- ▶ “latency”
- ▶ “resources\_latency”
- ▶ “latency\_faredge”
- ▶ “resources\_latency\_faredge”
- ▶ “resources\_latency\_energy”
- ▶ “webhook”

Along with “**resources**” optimization, whose algorithm has been described in D4.3 and summarized before, “**scaling**” and “**offloading**” represents two simplifications required by Pledger Use Cases (UC) where resources are not managed by the DSS exclusively, so “**resources**” optimization algorithm is

applied on resource quotas which change over time, then actions taken are limited to just scaling or offloading because of specific UC constraints. For example, in UC1 there is no option to offload to the cloud for lack of compatible infrastructure based on Windows operating system, whereas in UC3 the action requested has been to avoid scaling, offloading on the cloud when resources are not available, to grant edge resources to higher priority processes not managed by the DSS.

To summarize, the simplified scenario supported for the UCs requires higher priority services to stay on the edge, while the DSS manages the rest of the resources, keeping low priority services on the edge as much as possible and using SLA violations as feedback to trigger offloading/scaling.

Figure 6 shows the “scaling” and “offloading” optimizations with resources not managed by the DSS in an exclusive way.

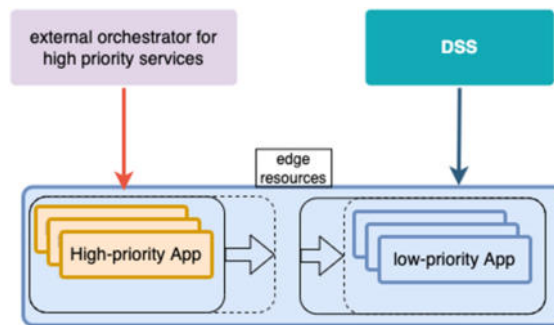


Figure 6: DSS not managing edge resources exclusively

The next optimization, “**latency**”, implements the edge-to-cloud offloading decision algorithm (ECODA), presented in a novel IEEE paper presented at **IEEE Globecom 2021 [29]**, is based on “resources” optimization and adds the reduction of end-to-end latency on a two-tiers infrastructure (edge and cloud) and resource usage using Lagrangian optimization. More details are provided in the above-mentioned paper.

The “**resources\_latency**” optimization is a combination of “resources” and “latency” ones, which is ideal when edge devices have very different hardware capabilities and there is specific QoS to fulfill based on SLA warnings and violations used as feedback. The goal in this case is to guarantee such QoS, saturate the edge resources as much as possible and provide, at the same time, the minimal end-to-end latency.

Figure 7 shows the ECODA workflow in combination with the “resources” optimization. The continuous monitoring process done by ECODA assigns scores to the different services using Lagrangian optimization based on latency, priority, and reserved resources as parameters. Such scores are then used to prioritize scaling/offloading based on the available resources.

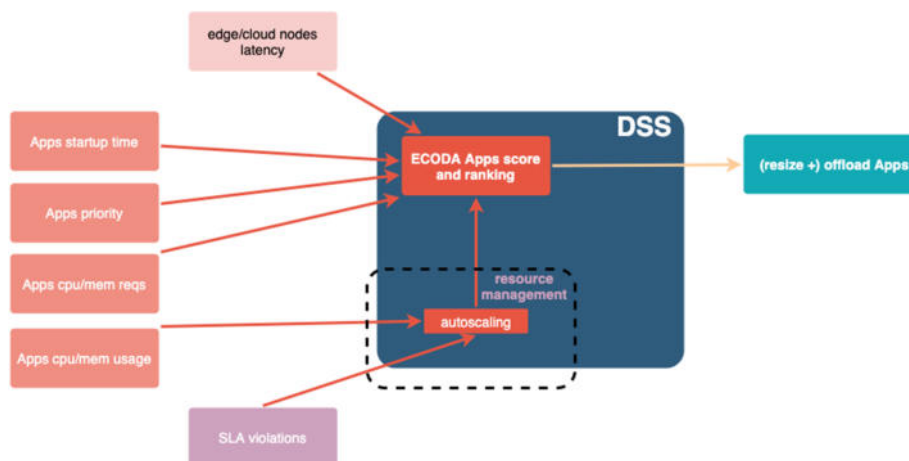


Figure 7: ECODA + "resources" algorithm

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	17 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

The “**latency\_faredge**” optimization implements the three-tier offloading decision algorithm (TTODA) algorithm, presented in a novel IEEE paper at **IEEE WCNC 2022 [30]**, and focuses on the optimization of the end-to-end latency and resource usage on a three-tier infrastructure (far-edge, edge and cloud). TTODA assigns scores to the different services using Lagrangian optimization based on latency, priority, and reserved resources. Differently to ECODA, TTODA simultaneously optimizes the usage of resources at the faredge and edge tiers, and the DApps running on each tier are prioritized accordingly. More details are provided in the above-mentioned paper.

The “**resources\_latency\_faredge**” optimization is a combination of “resources” and “latency\_faredge” ones and works similarly to “resources\_latency” using three-tier infrastructures.

The “**resources\_latency\_energy**” optimization implements the energy-aware edge-to-cloud offloading decision algorithm (EA-ECODA) which delivers low-latency services on battery and solar-powered devices on the edge, using ECODA as a baseline, adding the dynamic change of the max resource threshold to limit energy consumption due to computation while considering the amount of energy available on batteries as well as the battery charging rate. This work will be submitted by the end of September 2022 to IEEE Transactions on Vehicular Technology, a scholarly journal dedicated to vehicular technology, covering topics such as Wireless Networks and Mobile Services, and Connected and Autonomous Vehicles Systems, with impact factor of 6.239 [32], ranked 7th among 288 IEEE journals [33].

Figure 8 shows the EA-ECODA workflow, where far-edge resource **thresholds** are continuously changed according to the energy consumption. Then, the same workflow as in ECODA + “resources” optimisation is applied, and the far-edge resource thresholds are fed to ECODA during each iteration.

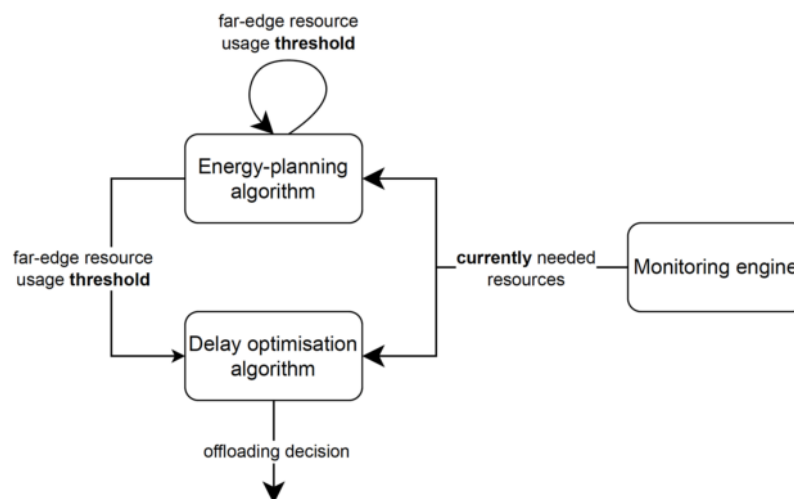


Figure 8: EA-ECODA algorithm

Finally, the “**webhook**” optimization is left to developers to allow custom automation that might be domain specific. To enable this process, the DSS exposes a dedicated REpresentational State Transfer (REST) API which simplifies the integration, as described in Section 3.5.

## 2.3 DSS interactions with Pledger core components

In the **second project period**, the integration with the Orchestrator has been consolidated to support different platforms such as Kubernetes and Docker, and to pass over the configuration needed to manage the Slice Orchestration Engine (SOE) framework; also, AppProfiler integration have been finalized with the parsing of the messages needed to evaluate the best node to offload DApps.

To demonstrate the replicability of the results, the DSS has been engineered with focus on modularity, so that it is possible either to run both as a **standalone** component or **in conjunction with Pledger core components**.

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	18 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

As a **standalone** component, the DSS implements a complete MAPE loop for cloud-edge Kubernetes multi-clusters via direct API calls to :

- ▶ Kubernetes[24] (and any implementation for cloud/edge) to orchestrate DApps.
- ▶ Metrics-server[13] for infrastructure and application monitoring.
- ▶ Goldpinger[14] for latency monitoring.
- ▶ Prometheus[15] and AlertManager[16] for the SLA monitoring, using an adapter or bash scripts to send violations.
- ▶ Kafka[12] for communication/integration.

Figure 9 shows the interactions with external open source components for tests validation. For convenience, DSS features related with configuration are separated in a different block named “**ConfService**”. This separation will be further described in section 3 about the DSS technical implementation.

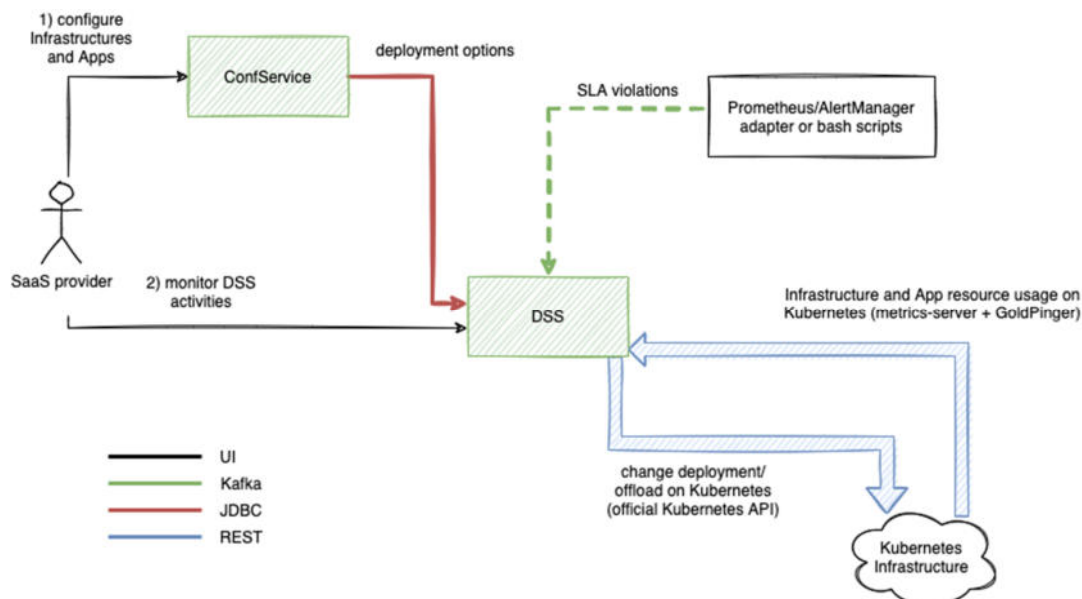


Figure 9: DSS used as standalone component, for tests validation

When **working in conjunction with Pledger core components**, it gives its best, and :

- ▶ orchestrates (virtually) any possible infrastructure, thanks to the Orchestrator which supports Kubernetes, Docker and also custom interfaces.
- ▶ automatically detects the most performant nodes for DApps, thanks to the functionalities provided by the Benchmarking Suite and the AppProfiler.
- ▶ benefits from SLA managed together with Distributed Ledger Technology (DLT), thanks to SLA Manager which integrates with other Pledger components to support DLT.
- ▶ benefits from improved reliability on control and data plane, thanks to the StreamHandler, which overcomes the features provided by vanilla Kafka instance.

The main interactions described above are shown in Figure 10.



## 3 Technical description

---

This section describes the DSS from a technical perspective, with highlights on the baseline technologies and dependencies, the architecture and interfaces provided to external components, the main class diagram, and the REST API available to the developers for possible integration.

In terms of progress, baseline technologies and dependencies, as well as components architecture and interfaces did not change since the first DSS release presented in the D4.3, as most of the initial work was indeed focused on setting the ground for the next development.

So, this section mainly reports what was described already in D4.3 and is presented for convenience. For the **second project period**, the only updates are about the class diagram, to support the new optimizations and their configuration.

### 3.1 Baseline technologies and dependencies

---

The DSS is composed by a **UI frontend** to allow Service Provider (SP) and Infrastructure Provider (IP) to change the configuration, a **Configuration backend** service to store the configuration needed by the DSS, the **DSS backend** service that implements the optimization algorithms and a Relational Database Management System (**RDBMS**) to persist data.

The main technologies used for the DSS are NodeJS[19] v11, Angular[10] v.11 for the frontend, OpenJDK[8] v.11 and SpringBoot[9] v2.4 for the backend, MySQL[11] v5.5 for the RDBMS. It also includes Kafka[12] connectors to communicate with the StreamHandler, used for asynchronous communication with the other Pledger components as shown in the next subsection.

All the DSS services are deployed on Kubernetes using Pledger's Continuous Integration and Deployment (CI/CD) pipeline to leverage on continuous integration and manage the whole development and deployment lifecycle. It also includes a custom component that allows the re-creation of table structure and initial configuration data according to the desired version. This is meant to easily restore the configuration status to a consistent version shared at the source repository.

The DSS also references a local instance of NodeRed[17] with some custom blocks interfacing with the DSS API and the StreamHandler to facilitate the creation of custom automation by the SP.

### 3.2 Components Architecture

---

Figure 11 describes the main components of the DSS, with UI, Configuration service, DSS service, and RDBMS, with detail about how they are interfacing with the core Pledger components to provide the common configuration through a REST API using a publish/subscribe communication with the StreamHandler.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	21 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

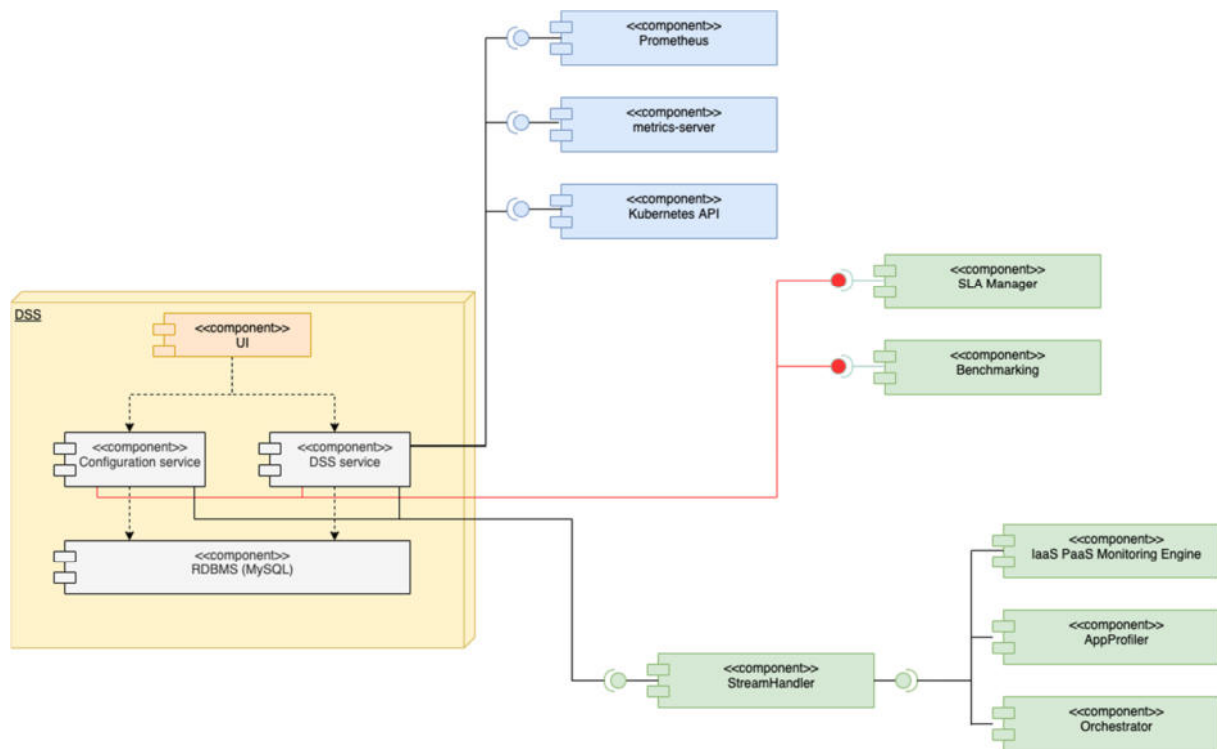


Figure 11: DSS component diagram

Configuration and DSS services interface with external services with four different approaches:

- ▶ **send notification** to the StreamHandler to alert interested components which can query Configuration service REST API;
- ▶ **send simple control data** to the StreamHandler;
- ▶ **receive simple control data** from the StreamHandler;
- ▶ **execute REST calls**.

In particular:

- ▶ **Configuration service sends notifications** about new configurations available so that SLA Manager and the Benchmarking can query configuration data through Configuration service API.
- ▶ **DSS service sends control data** to the Orchestrator and receives **control data** from IaaS Monitoring Engine and AppProfiler to gather infrastructure monitoring and app profile data.
- ▶ It also **executes REST calls** to Prometheus, Metrics-server and Kubernetes API to get monitoring data and orchestrate clusters in case of “managed” DApps.

The sequence diagram in Figure 12 shows the main flow for the resource configuration. The example reported is about the SLA Manager, although it is the same workflow used for Benchmarking.

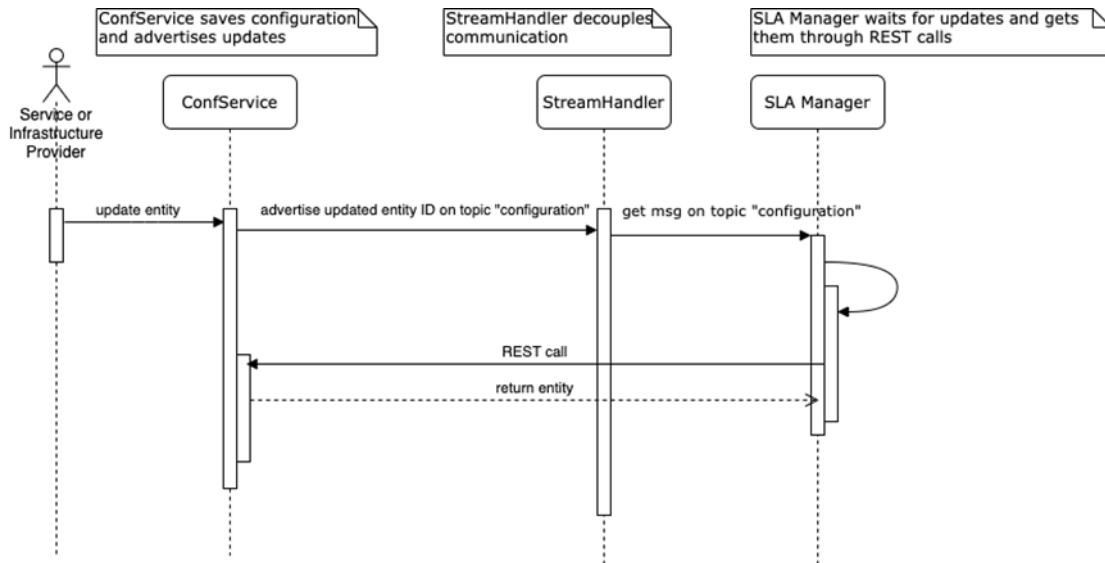


Figure 12: Sequence diagram about configuration data exchange: SLA Manager

To summarize, the Configurator main interfaces are:

- ▶ (A) with the StreamHandler to send notifications when a configuration entity changes.
- ▶ (B) with the SLA Manager and Benchmarking, to provide the configuration using REST API

The interface A) is a publish/subscribe protocol that uses the topic “configuration” to advertise updates on any REST entity with a JavaScript Object Notation (JSON) message sent to Kafka in the format:

```
{'id': ID, 'entity': 'ENTITY_NAME', 'operation': 'OPERATION_NAME'}
```

The format is straightforward:

- ▶ ID is the unique identifier in the Configuration service, used to possibly filter only the interested entities.
- ▶ ENTITY\_NAME is the REST resource name.
- ▶ OPERATION\_NAME is the operation executed, UPDATE for creation/update or DELETE for deletion.

The interface B) is the Configuration service REST API available via Swagger[18] to facilitate integration and described in section 3.5.

Similarly, the DSS **interfaces** with the StreamHandler to:

- ▶ **receive** SLA violations, sent by the SLA Manager
- ▶ **receive** Benchmarking reports, sent by the Benchmarking
- ▶ **receive** the best node for a specific DApp, sent by the AppProfiler
- ▶ **send** start/stop, scaling, offloading actions to the Orchestrator, to actuate them

The sequence diagram in Figure 13 shows the main flow for the SLA Manager, which is identical to Benchmarking and AppProfiler.

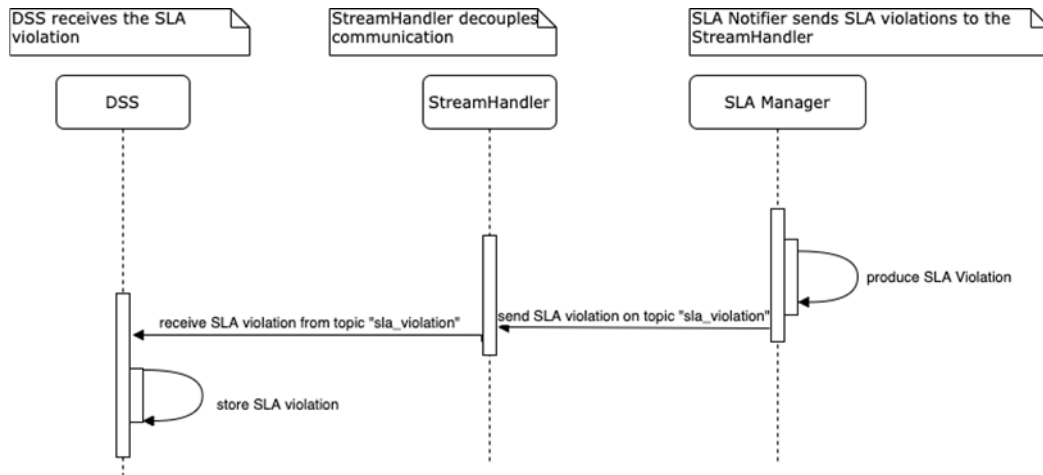


Figure 13: DSS sequence diagram to receive SLA violations

Similarly, the DSS uses the StreamHandler to publish commands for the Orchestrator to actuate the scaling/offloading actions, as shown in Figure 14.

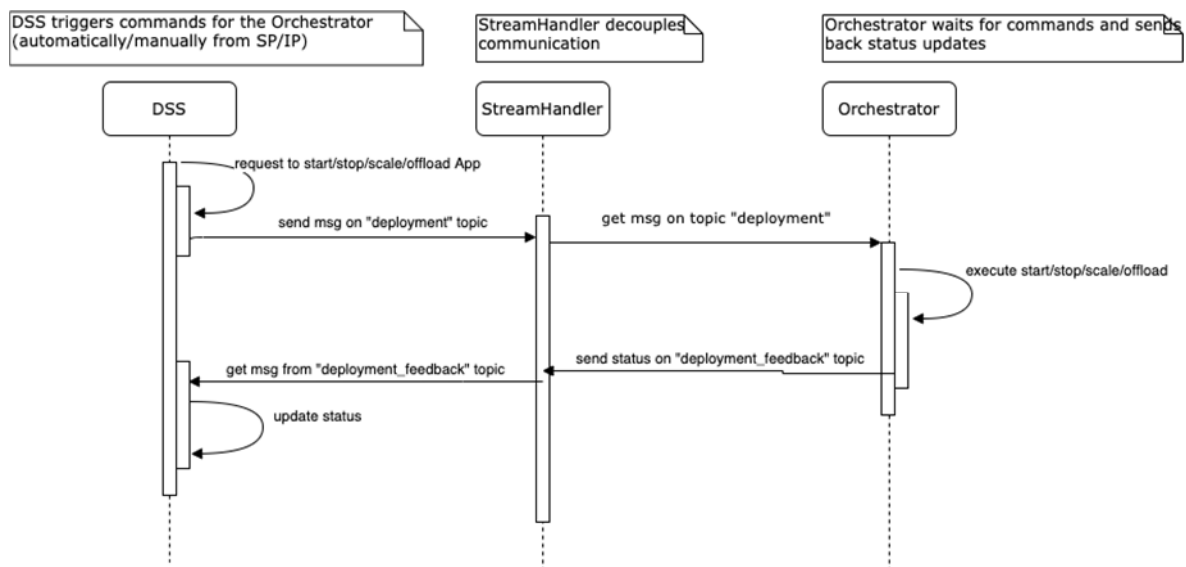


Figure 14: DSS sequence diagram to send commands to the Orchestrator

### 3.3 Class diagram

As the Configuration service role is to provide the configuration data needed by the DSS to work, **Configuration** and **DSS** services share the same model and data is shared internally via Java Data Base Connectivity (JDBC) protocol. The main entities and relations are shown in the class diagram in Figure 15.

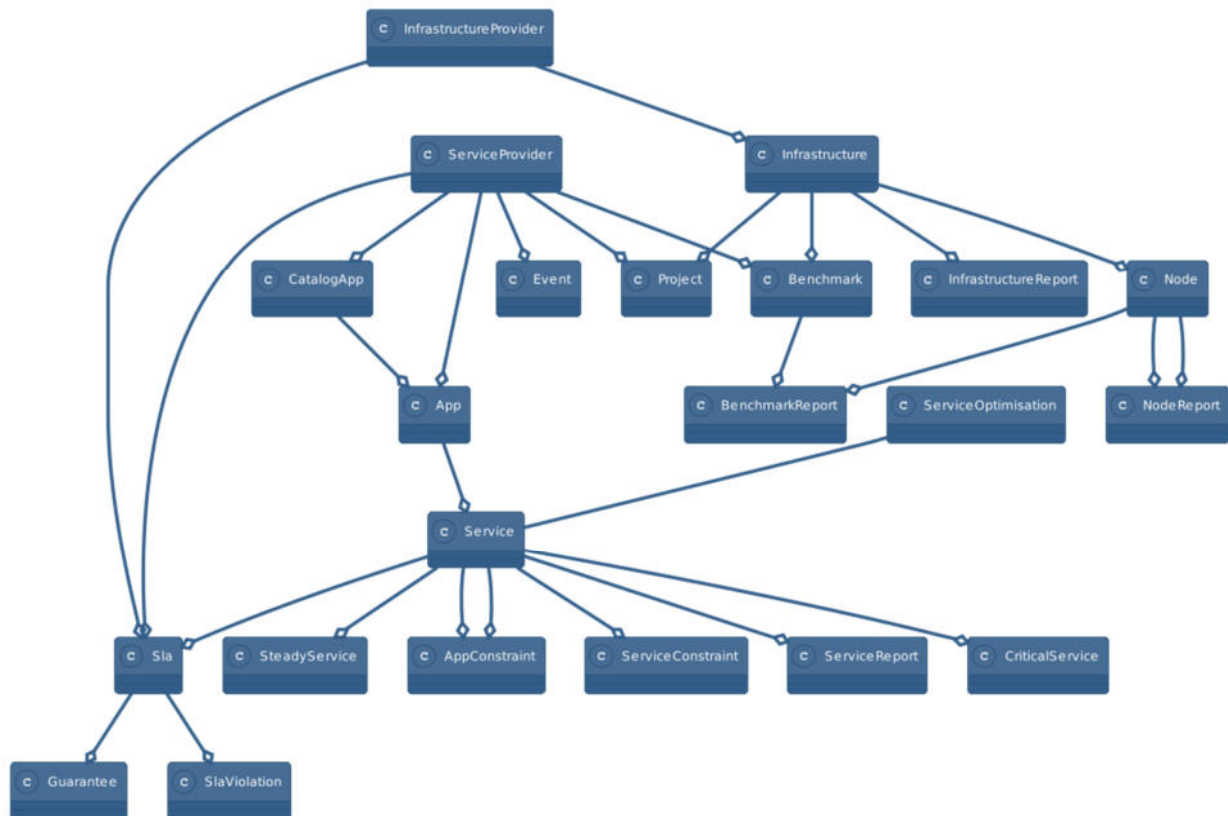


Figure 15: Configuration service and DSS main class diagram

The entity list and their role are described below :

- ▶ **App**: represents a DApp, an application that is tied to an SLA and used by a Service Provider (SP).
- ▶ **AppConstraint**: represents the DApp constraints about infrastructure and node properties
- ▶ **Benchmark**: represents a benchmark used to estimate the performance of an infrastructure and its Nodes.
- ▶ **BenchmarkReport**: represents the historical data produced by the benchmarks.
- ▶ **CatalogApp**: represents the applications available in Pledger.
- ▶ **Event**: represents an event that is logged and is available for further investigation.
- ▶ **Guarantee**: represents the list of constraints agreed within a specific SLA.
- ▶ **Infrastructure**: represents the infrastructure used, possibly owned by separate providers, with type describing whether it is Kubernetes, Docker, etc.
- ▶ **InfrastructureProvider**: represents the responsible entity for the infrastructure; such an entity is introduced to identify one of the two parties involved in the definition of an SLA.
- ▶ **InfrastructureReport** represents the resource usage about infrastructures.
- ▶ **Node**: represents one element of an infrastructure which might have specific resources available so that it is possible to differentiate the orchestration (scaling and placing) of applications according to the DApps needs.
- ▶ **NodeReport** represents the resource usage about nodes.
- ▶ **Project**: represents a high-level contract between SP and Infrastructure Provider (IP) which allows a SP to use an infrastructure with a resource quota.
- ▶ **Service**: represents an executable component of an App.
- ▶ **ServiceConstraint**: represents a constraint on a specific Service, such as the number of resources requested.
- ▶ **ServiceOptimisation**: represents the optimisation configured for a specific Service. This is a new entity added in the second project period to configure the optimizations available.
- ▶ **ServiceProvider**: represents the provider of the services in Pledger.

- ▶ **ServiceReport**: contains report about the Service resource usage.
- ▶ **SLAViolation**: represents the violation of an agreement (Guarantee) within an SLA.
- ▶ **SLA**: represents a contract between an InfrastructureProvider and a ServiceProvider about the execution of a specific App. An advancement in the second period has been made about how to manage SLA violations, with the options to use them as feedback, suspend for some time or ignore them completely. This allows finer degree of control to the service providers.
- ▶ **CriticalService**: represents a Service that is eligible to resource increment or offloading to a lower priority Node if not available on the Node currently hosting the service.
- ▶ **SteadyService**: represents a Service that is eligible for resource reduction or offloading to a higher priority Node, if available, with respect to the Node currently hosting a service.

Three main roles are supported: “Administrator”, “ServiceProvider”, “InfrastructureProvider”.

**Administrator** is responsible for:

- ▶ the management of the IP and SP users.
- ▶ the creation of Projects to bind SP with Infrastructures and define the quota of resources assigned

This role allows to manage Pledger users and assign them resources.

**ServiceProvider** is responsible for:

- ▶ getting access to his/her own configuration data.
- ▶ configuring his/her profile with preferences used by the DSS.
- ▶ configuring CatalogApps and DApps, either public or private, and their priorities.
- ▶ configuring Benchmarks, either public or private.
- ▶ configuring SLA, their Guarantees and Penalties.
- ▶ reading the SLA violations.
- ▶ getting information about services that have their resources reduced (SteadyService) or increased (CriticalService) by the DSS.
- ▶ getting information about Service actual reserved usage and metrics about latency (ServiceReport).
- ▶ setting constraints and priorities about the preferred deployments (ServiceConstraints) and monitor which options are available due to the actual resources (DeploymentOptions).

**InfrastructureProvider** is responsible for:

- ▶ defining the Infrastructure and Node topology.
- ▶ configuring custom properties on Nodes about hardware availability and so on.
- ▶ reading reports about Infrastructure and Node resource availability.

Limitations are added to allow each SP to access only their own private data, as well as public data.

Private data is considered any entity that is tied to a specific instance of a SP according to the relationships shown in the entity diagram in Figure 15. This means, for example, that a specific SP can access only the App and Service that are attached to that instance. The same holds for indirect relations as well, such as Infrastructure and so on. In the latter case, a SP can access only the Infrastructure that is attached to Projects assigned to that SP instance.

Public data is considered any entity that is not tied to any specific instance of a SP. This for example might happen with Benchmark or CatalogApp whenever the relation with SP is not populated.

Such a scenario is included in section 5 about the configuration demo where SP and IP have access to the Configuration UI, where DApps (and linked entities like Services and so on) belonging to one SP are not visible to others.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	26 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

### 3.4 DSS insight

This subsection describes at the DSS engine and the different optimizations that have been implemented during the second project period.

The DSS acts as a MAPE loop, where the different stages are handled by the following main Java packages :

- ▶ **“monitoring”**, responsible for the extraction of the relevant information for the DSS, that is configuration data, monitoring data (resource usage, resource requests, latency) and so on, covering the MAPE Monitoring stage.
- ▶ **“optimization”**, responsible for the MAPE Analysis and Planning stages, defining the next actions to take, as scaling/offloading activities.
- ▶ **”scheduler”**, responsible for the MAPE Execution stage, commanding the scaling/offloading activities.

For the **“monitoring”** package, it is worth to report that support has been added for different infrastructure and application monitoring services, such as Prometheus, AlertManager, Metrics-server and, of course, to Pledger core component MonitoringEngine through the StreamHandler using Kafka protocol. Configuration is done through the ConfService, for each Infrastructure, in the “Monitoring Plugin” field, as shown in Figure 16.

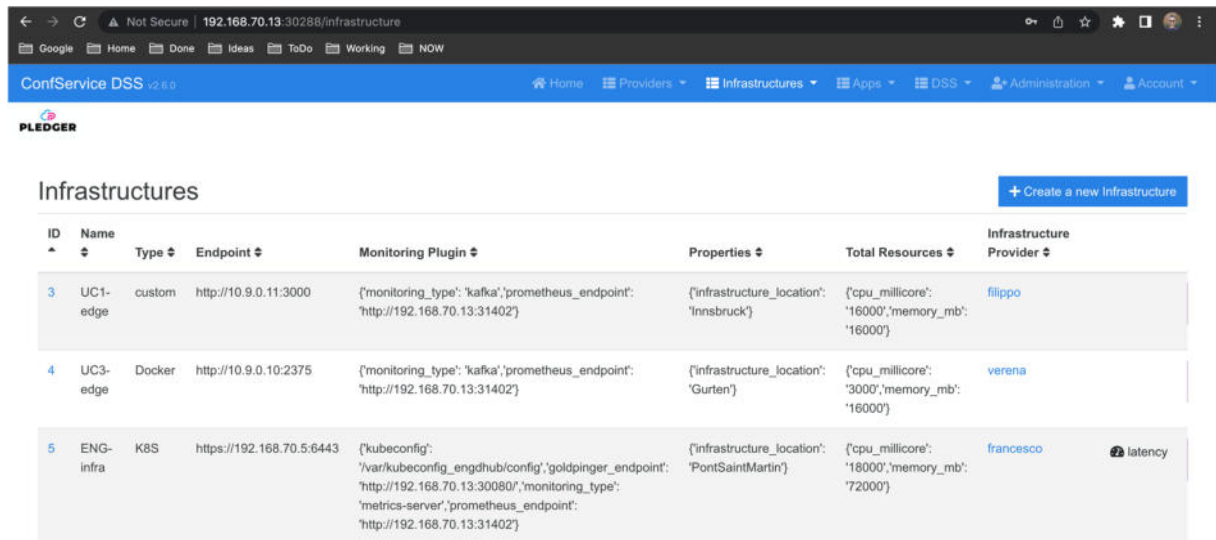


Figure 16: Monitoring configuration example

For the **“optimization”** package, each optimization algorithm is implemented in a separate class, with configuration managed through the ConfService as shown in Figure 17.

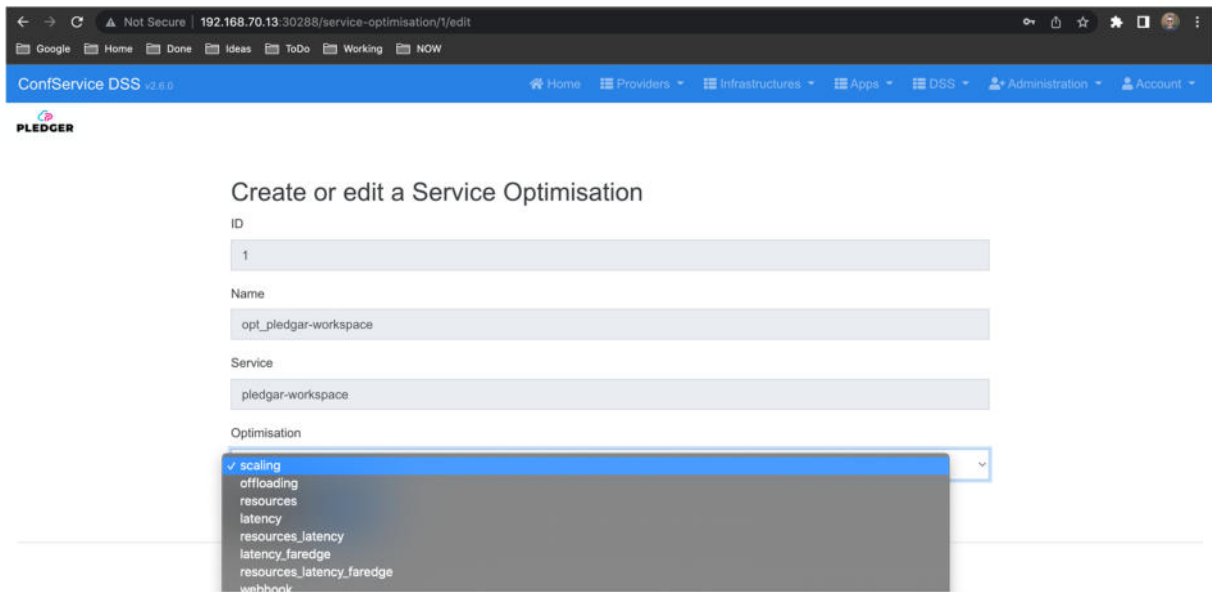


Figure 17: Service optimization configuration example

For the “**scheduler**” package, pluggable support has been added to manage scaling/offloading execution, either via direct API call to Kubernetes API or through messages sent to the Orchestrator via the StreamHandler. Also in this case, the choice on whether to have direct API calls or rely on Pledger orchestrator is configured on the ConfService as shown in Figure 18, where the service provider can choose respectively an App to be “MANAGED” or “DELEGATED”.

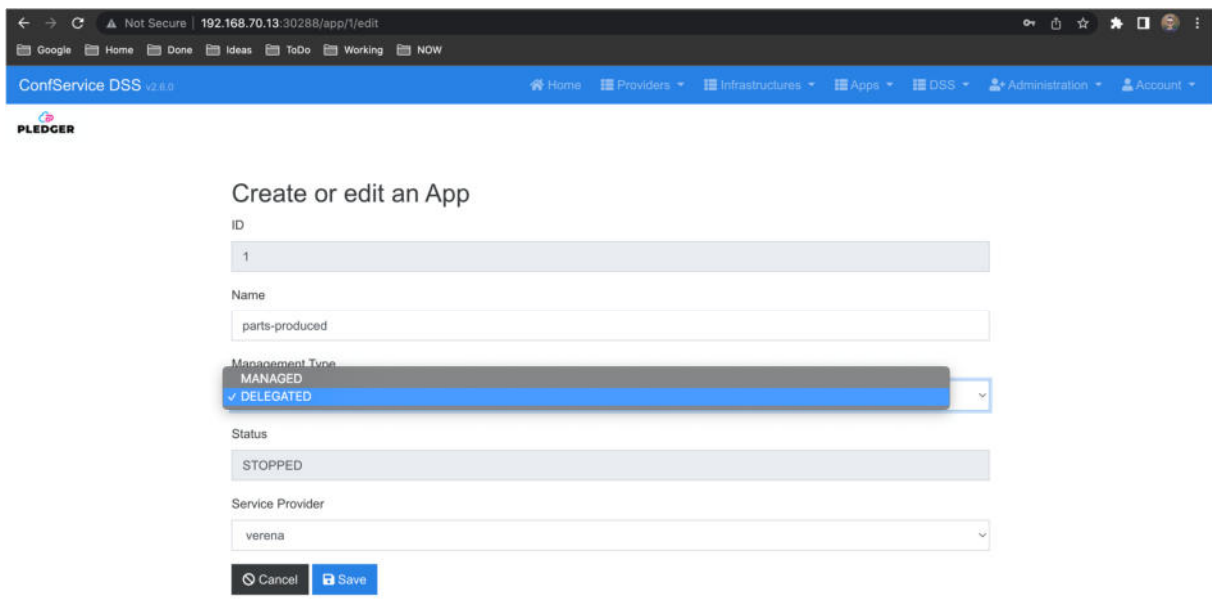


Figure 18: App management configuration example

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	28 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

### 3.5 DSS data model, REST API and integration example

The DSS entities listed in subsection 3.3 are published as **REST resources** with methods that allow to: get a complete/filtered list, get a specific instance, save, update or delete one instance using **HTTP GET, POST, PUT, DELETE** methods.

The only exceptions are the InfrastructureReport, NodeReport, Event, SteadyService and CriticalService which are read-only and available only through **HTTP GET method**.

The complete list of entities is available through Swagger-UI to facilitate the integration with third-party components. Swagger also exposes the DSS data model. Figure 19 shows an example of “App” entity **REST methods** and its **data model**.

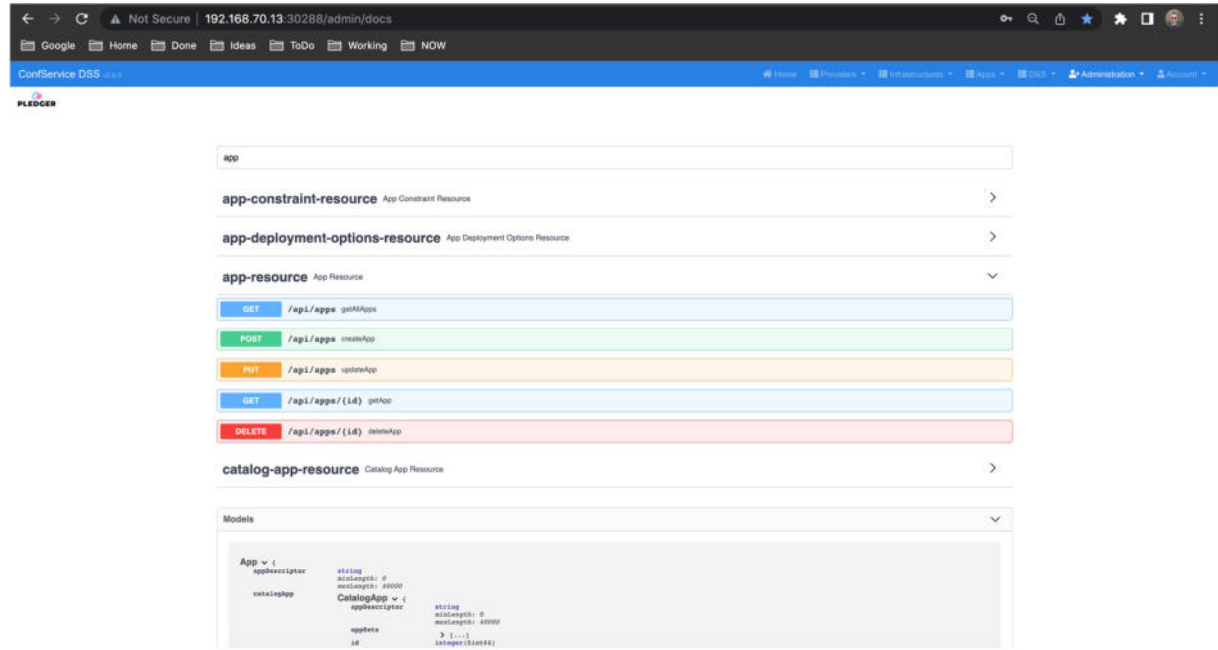


Figure 19: DSS Swagger-UI accessible from the UI – App REST resource example

An example of integration with the DSS is provided with a sample NodeRed project where 1) a **webhook** is defined to receive custom automation calls from the DSS, 2) authenticates on the DSS API, 3) reads SLA violations, then 4) connects to the StreamHandler to publish/subscribe data on a topic, with all the building blocks a developer can use to make a custom automation with business logic. The sample project is shown in Figure 20.

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	29 of 46	
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	
	<b>Version:</b>	0.8	<b>Status:</b>	Final

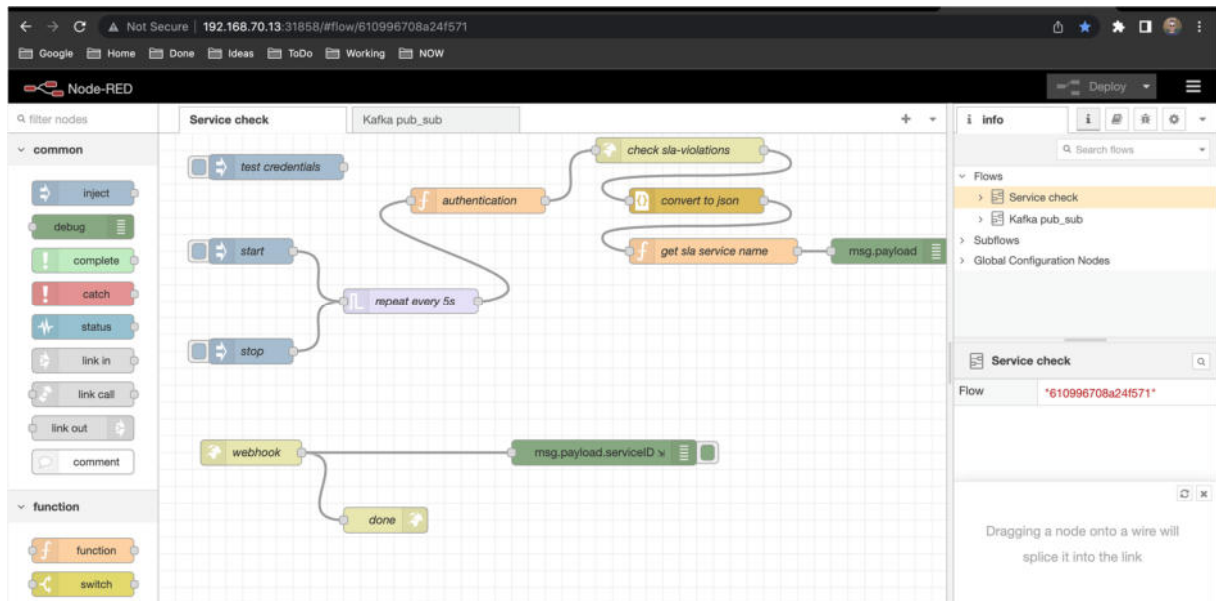


Figure 20: DSS integration with NodeRed example

Document name:	D4.6 Decision Support tools II	Page:	30 of 46
Reference:	D4.6	Dissemination:	PU
	Version:	0.8	Status:
			Final

## 4 Installation and usage guides

The requirements and installation did not change since the first DSS release, as most of the work focused on development, so most of this section corresponds to what was described in D4.3 and is reported again for convenience.

A new subsection has been added to report the test and validation activities, produced for a IEEE-ICC 2022 demo [31].

### 4.1 Requirements

The DSS required the following packages to be correctly installed and executed:

- ▶ OpenJDK[8] v11.0
- ▶ NodeJS[19] v12.0
- ▶ Npm[20] v6.0

In addition, Maven[21] v3 is needed for the correct building of the DSS application. As shown in the next sub-section, a Jenkins[23] build file is also provided to automate the **build** process and YAML files to automate the **deployment** on Kubernetes, so that any committed change in the DSS code is automatically deployed and made available using the CI/CD pipeline.

### 4.2 Installation

The DSS and its components are packaged as a full-stack Java application using the CI/CD services with the image made available on the Pledger Docker registry[22].

The source code and the configuration file used for the build with Jenkins is also stored in GitLab[27] and the build is triggered automatically whenever a change in the code is committed.

Kubernetes[24] descriptors are also shared on GitLab to allow versioning and automatic deployment and they can be used to replicate the installation on any other instance of Kubernetes outside Pledger infrastructure. Figure 21 shows the detailed installation instructions on Gitlab.

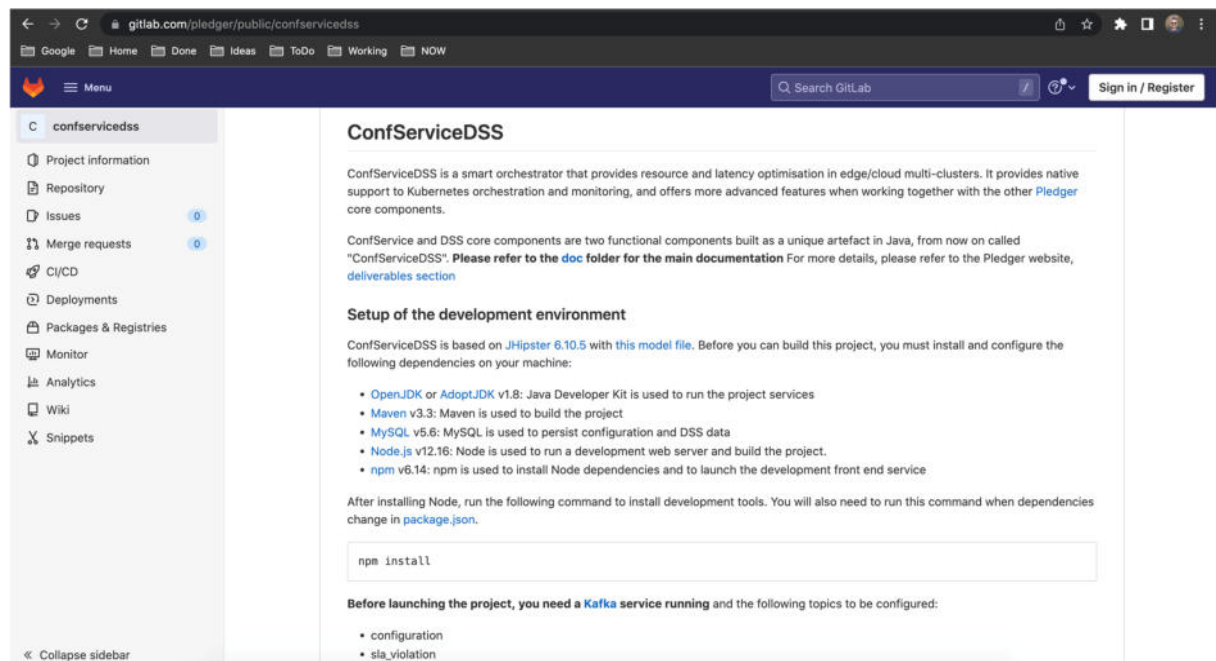


Figure 21: DSS detailed install instructions

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	31 of 46	
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	
	<b>Version:</b>	0.8	<b>Status:</b>	Final

### 4.3 Usage

Once the installation is in place, some initial Simple Query Language (SQL) data is also provided to show a basic configuration and allow some tests about assignments of resources to Service Providers and so on.

For the actual usage of the DSS, an infrastructure, the Orchestrator, the Benchmarking and some DApps need to be configured according to the scenario described in section 5.

### 4.4 Test and validation

Documentation as well as validation tests have been produced to promote the DSS to the IEEE ICC 2022 demo. The goal was to facilitate the DSS usage and the replicability of the DSS optimizations.

For this reason, the documentation on Gitlab has been structured, starting from “/doc”, with subfolders to describe (among other things):

- ▶ the DSS architecture,
- ▶ how to simulate a Kubernetes cluster with multiple cloud, edge and far-edge nodes, on a physical machine, with KinD[26],
- ▶ documentation about each optimization,
- ▶ documentation about how to replicate the tests and validate the results

A snapshot of the DSS “/doc” folder on Gitlab is shown in Figure 22.

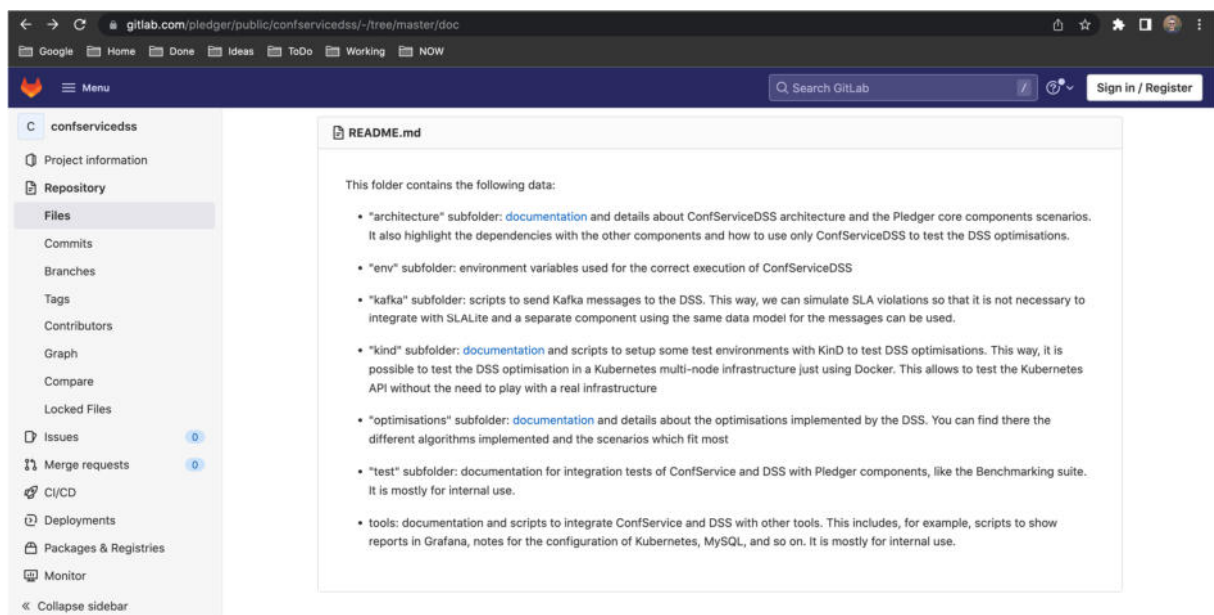


Figure 22: Snapshot of the DSS documentation folders on Gitlab

Similarly, Figure 23 shows the documentation about the setup and configuration of KinD to create a test Kubernetes cluster with multiple cloud, edge, and far-edge nodes, whereas Figure 24 shows the documentation about the test validation for cloud-edge-faredge Kubernetes cluster.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	32 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

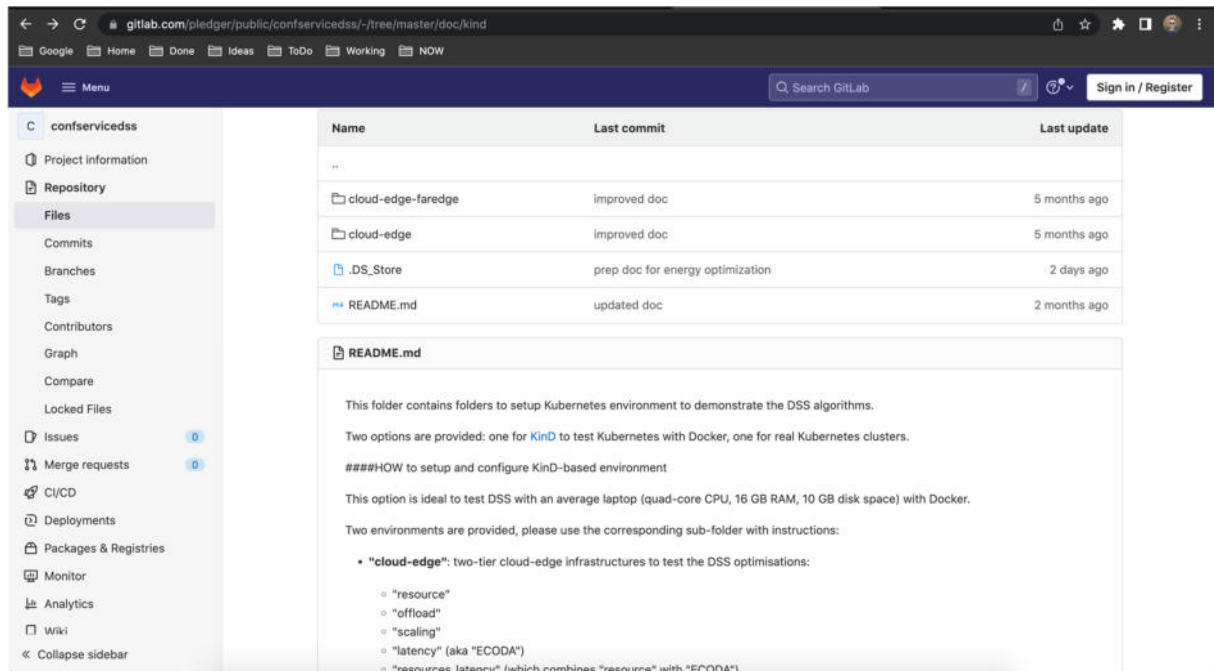


Figure 23: Setup of a Kubernetes cluster to test the DSS

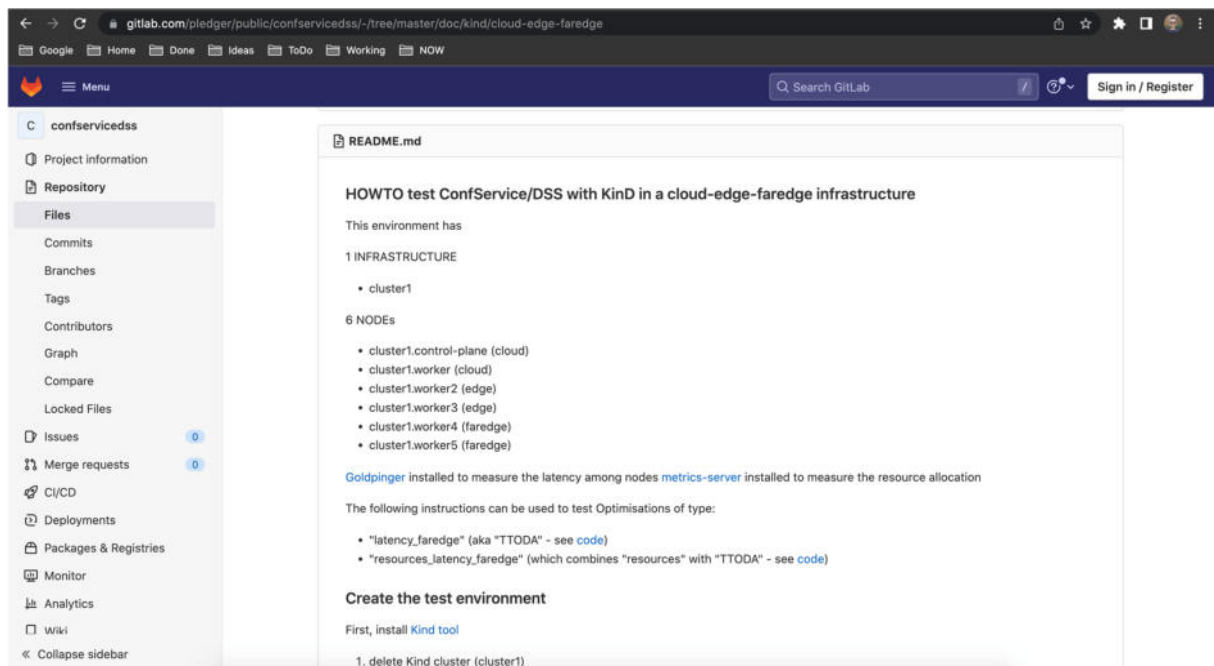


Figure 24: Documentation about how to run/validate tests on a Kubernetes cluster

## 4.5 License

All the DSS code is provided with Apache license v2.0 [25].

## 4.6 Source code repository

The DSS source code is publicly available on Gitlab repository, with documentation and **changelog** to track new source code **tags** changes, at the URL <https://gitlab.com/pledger/public/confservicedss>.

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	33 of 46	
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	
	<b>Version:</b>	0.8	<b>Status:</b>	Final

## 5 Demonstration

---

This section contains the references to the demo published on the project YouTube channel[28], grouped by functional components (**Configuration service** and **DSS**) and by project period, with a reference to the System Use Cases (SUC) defined in the MVP.

### 5.1 Configuration service

---

#### 5.1.1 First project period

In the **first project period**, most of the work concerned the Configuration service and UI along with initial DSS features. For the readers' convenience, the Configuration screenshots (already presented in D4.3) are reported in Annex 8.1.

#### 5.1.2 Second project period

In the **second project period**, the Configuration UI did not change much. The only update provided has been to support DApps orchestration done either by the DSS or the Orchestrator, and to allow the selection and configuration of the different DSS optimizations. More details are provided in section 3.4.

### 5.2 DSS

---

Several demos have been prepared during the project, all supported by videos published on YouTube [28] and grouped into two different playlists to distinguish between **first** and **second** project period.

#### 5.2.1 First project period

For the **first project period**, videos have been updated and now include a slide to describe demo scenario. The demos available for this period are:

- ▶ “**Pledger ConfService #1**“, showing the Administrator configuring additional users.
- ▶ “**Pledger ConfService #2**“, showing the InfrastructureProvider configuration.
- ▶ “**Pledger ConfService#3**“, showing the ServiceProvider configuring an App, Service, with SLA and Guarantees.
- ▶ “**Pledger ConfService#4**“, showing the ServiceProvider configuring the deployment options to prioritize edge over cloud.
- ▶ “**Pledger DSS#1**“, showing the DSS doing a scaling down
- ▶ “**Pledger DSS#2**“, showing the DSS scaling up, offloading to the cloud, then back to the edge

Figure 25 shows the introduction slide for the “**Pledger DSS#2**” demo and the list of demo videos on the right for the first period at M20.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	34 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

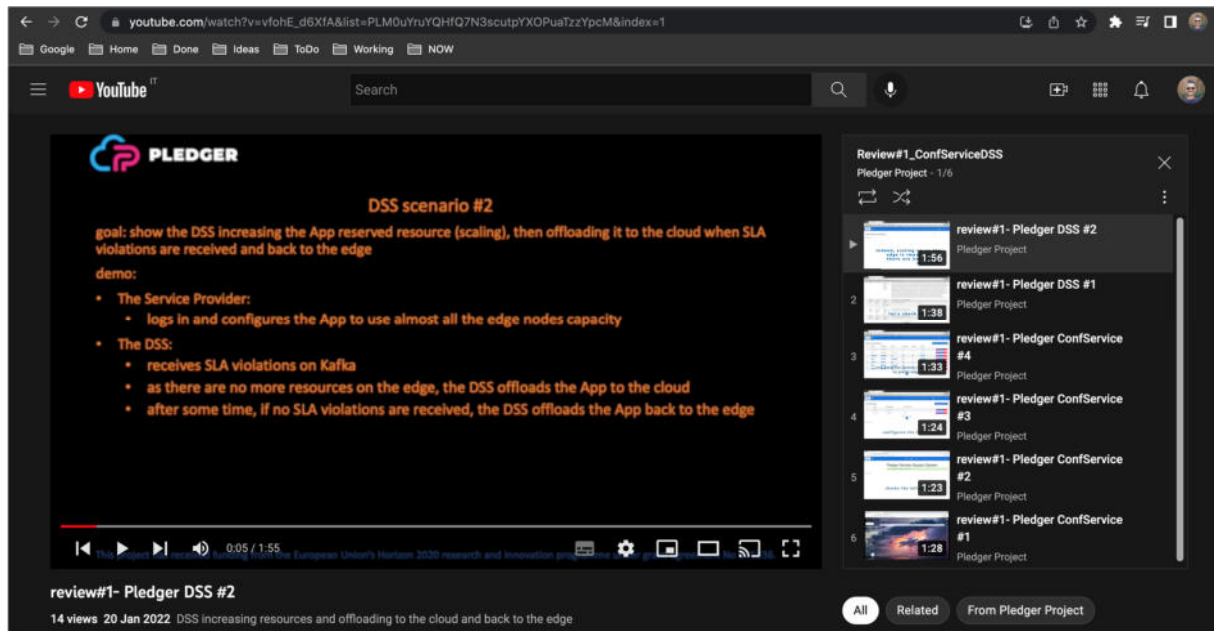


Figure 25: Configuration and DSS service demos for the first project period on YouTube

### 5.2.2 Second project period

For the **second project period**, videos have been prepared to demonstrate the different optimizations. Some also describe the 1) **installation process** in detail, 2) how to **create a test environment** simulating a Kubernetes cluster with multiple cloud, edge and far-edge nodes using KinD[26] and 3) the detailed steps to **test the optimizations** and **validate the results**. The demos available for this period are:

- ▶ “DSS #3”, about DSS installation from source.
- ▶ “DSS #4”, about KinD cloud-edge Kubernetes environment setup and DSS manual operations.
- ▶ “DSS #5”, about DSS scaling up/down based on SLA violations.
- ▶ “DSS #6”, about DSS scaling out based on SLA violations.
- ▶ “DSS #7”, about DSS offloading to the cloud based on SLA violations.
- ▶ “DSS #8”, about DSS optimising latency on cloud-edge Kubernetes using “ECODA” algorithm.
- ▶ “DSS #9”, about DSS optimising latency on cloud-edge Kubernetes using “ECODA” algorithm and SLA violations.
- ▶ “DSS #10”, about KinD cloud-edge-faredge Kubernetes environment setup.
- ▶ “DSS #11”, about DSS optimising latency on cloud-edge-faredge Kubernetes, using “TTODA” algorithm.
- ▶ “DSS #12”, about DSS optimising latency on cloud-edge-faredge Kubernetes, using “TTODA” algorithm and SLA violations.
- ▶ “DSS #13”, about DSS optimising energy consumption on the edge, latency and resource usage on cloud-edge Kubernetes using “EA-ECODA” algorithm.

Figure 26 shows the introduction slide for the “Pledger DSS#10” demo and the list of demo videos on the right for the second project period.

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	35 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

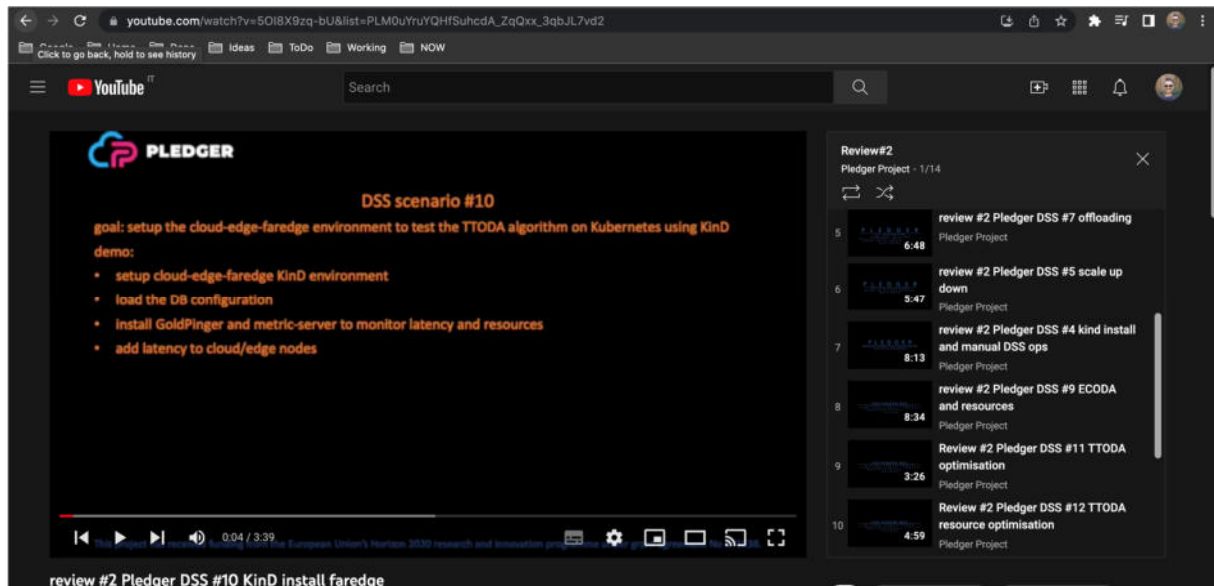


Figure 26: DSS service demos for the second project period on YouTube

Also, two end-to-end integration demo videos have been published to demonstrate the integration with DSS, SLA Manager, Orchestrator, Benchmarking and AppProfiler. These are:

- ▶ “Pledger Integration demo #1”, focused on the DSS work in conjunction with SLA Manager and Orchestrator to launch a DApp on an infrastructure.
- ▶ “Pledger Integration demo #2”, focused on Benchmarking and AppProfiler work, where AppProfiler sends a notification to the DSS with the most performant node for a given DApp.

Figure 27 shows the “Pledger Integration Demo #1” with the demo description.

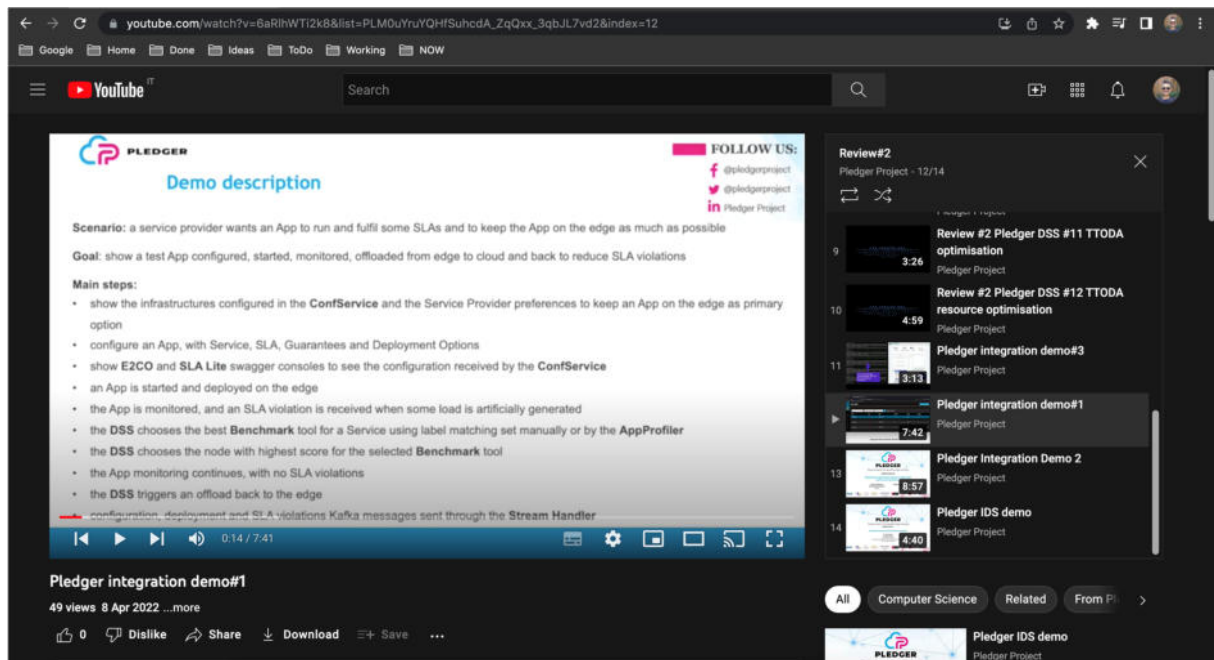


Figure 27: Pledger Integration Demo #1

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	36 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b> Final

### 5.3 Validation and Verification

The MVP requirements addressed by the DSS tools are reported, showing the SUC that are enabled by the final release, using the list taken from D4.3 and updated with the finalized integration with the other Pledger components. All are now in “**completed**” status.

- ▶ SUC.01 “provide App profile details”. Covered by the DSS with both the manual insertion of DApp properties and those provided by the AppProfiler described in the integration activities of T5.2.
- ▶ SUC.02 “deploy App components”. Covered by the DSS which supports Kubernetes with direct API calls for “managed” DApps and virtually any possible infrastructure type when using the Orchestrator for “delegated” DApps, as described in T5.2.
- ▶ SUC.03 “view available IaaS resources”, covered by the Configuration service and its UI.
- ▶ SUC.04 “get recommendations for IaaS resources”, covered by the Configuration service which allows the service providers to provide deployment options and by the DSS that has different optimizations available and supporting documentation to choose the one that better fits a given scenario.
- ▶ SUC.05 “select fitting IaaS resources” covered by the DSS which receives the best node for a given DApp and uses such information during the optimization process.
- ▶ SUC.08 “get notified about unwanted events”, covered by the DSS receiving SLA violations by the SLA Manager.
- ▶ SUC.09 “handle unwanted events. Covered by the DSS which chooses better deployment options in case of SLA Violations and warnings.
- ▶ SUC.10 “view suggested recovery actions”. Covered by the DSS taking autonomous decisions. The selection is done by using the Service Provider preferences which give different priorities to the actions.
- ▶ SUC.11 “choose/ perform recovery action(s). Same as SUC.10.
- ▶ SUC.14 “choose to migrate (parts of) components of App”. Covered by the DSS working on the App composed of multiple Services and deployment options with preferences.
- ▶ SUC.15 “choose to scale up/ down (components of) App”. Covered by the DSS as it supports resizing of App resources.
- ▶ SUC.16 “configure infrastructure”. Covered by the Configuration service with support to feature auto discovery.
- ▶ SUC.17 “make IaaS available”. Covered by the Configuration service which provides IaaS to the DSS.
- ▶ SUC.19 “IaaS monitoring”. Covered by the DSS that extracts the PaaS and IaaS metrics from Prometheus, Metrics-server, Goldpinger and the Monitoring Engine. It also includes latency metrics for cloud offloading and time metrics to measure the DApp time to boot.
- ▶ SUC.20 “read infrastructure configurations”. Covered by the Configuration service, which shares the updates with the other core components.
- ▶ SUC.21 “get IaaS evaluation”. Covered by the DSS service, which supports scaling/offloading depending on the IaaS resource availability.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	37 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

## 6 Conclusions

---

This deliverable reports the work done in T4.3 “Decision Support mechanisms for Edge/Cloud computation moving (M6-M33) and highlights the progress since D4.3, released at M20. It presents the main functional and technical components of the DSS final release, as well as a description of how to install and use it including references to video demonstrations.

During the second project period, the major advancements have been about new implemented DSS optimizations, some based on novel technical publications, along with a finer grained control over SLA violation management as part of the integration done within T5.2.

This work will be used for the last phase about the Use Cases evaluation using the Pledger platform environment, so the development will continue to support the integration and end-to-end demos that will be prepared by the end of the project.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	38 of 46		
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8	<b>Status:</b>	Final

## 7 References

- [1] PLEDGER, “D2.2: Pledger Requirements Analysis”, Voutyras, Orfefs (2022), <http://pledger-project.eu/D2.2.pdf> [retrieved date 01/08/2022]
- [2] PLEDGER, “D2.3: Pledger Overall Architecture”, Voutyras, Orfefs (2022), <http://pledger-project.eu/D2.3.pdf> [retrieved date 01/08/2022]
- [3] PLEDGER, “D4.3: Decision Support Tools I”, Ianzada, Francesco (2021)[to be published on <http://pledger-project.eu>]
- [4] PLEDGER, “D3.4: Performance Measurements and classification tools II”, Giammatteo, Gabriele (2022) [to be published on <http://pledger-project.eu>]
- [5] PLEDGER, “D3.5: Edge/Cloud orchestration tools II”, Psychas, Alexandros (2022) [to be published on <http://pledger-project.eu>]
- [6] PLEDGER, “D3.6: QoS and SLA assessment and negotiation tools II”, Sucasas, Roi (2022) [to be published on <http://pledger-project.eu>]
- [7] PLEDGER, “D5.6: Pledger integrated demonstrator II”, Matzakos, Panos; Sarris, Ioannis; Segou, Olga (2022) [to be published on <http://pledger-project.eu>]
- [8] OpenJDK homepage, <https://openjdk.java.net/> [retrieved date 01/08/2022]
- [9] Spring homepage, <https://spring.io/projects/spring-boot/> [retrieved date 01/08/2022]
- [10] Angular homepage, <https://angular.io> [retrieved date 01/08/2022]
- [11] MySQL homepage, <https://www.mysql.com/> [retrieved date 01/08/2022]
- [12] Kafka homepage, <https://kafka.apache.org/> [retrieved date 01/08/2022]
- [13] Metrics-server, <https://github.com/kubernetes-sigs/metrics-server> [retrieved date 01/08/2022]
- [14] Goldpinger, <https://github.com/bloomberg/goldpinger> [retrieved date 01/08/2022]
- [15] Prometheus, <https://prometheus.io/> [retrieved date 01/08/2022]
- [16] Alert Manager, <https://github.com/prometheus/alertmanager> [retrieved date 01/08/2022]
- [17] NodeRed homepage, <https://nodered.org/> [retrieved date 01/08/2022]
- [18] Swagger homepage, <https://swagger.io/> [retrieved date 01/08/2022]
- [19] NodeJS homepage, <https://nodejs.org/> [retrieved date 01/08/2022]
- [20] Npm homepage, <https://www.npmjs.com/> [retrieved date 01/08/2022]
- [21] Maven homepage, <https://maven.apache.org/> [retrieved date 01/08/2022]
- [22] Docker homepage, <https://docs.docker.com/registry/> [retrieved date 01/08/2022]
- [23] Jenkins homepage, <https://www.jenkins.io/> [retrieved date 01/08/2022]
- [24] Kubernetes homepage, <https://kubernetes.io/> [retrieved date 01/08/2022]
- [25] Apache license v2.0, <https://www.apache.org/licenses/LICENSE-2.0> [retrieved date 01/08/2022]
- [26] KinD, <https://kind.sigs.k8s.io/> [retrieved date 01/08/2022]
- [27] DSS Gitlab repo, <https://gitlab.com/pledger/public/confservicedss> [retrieved date 01/08/2022]
- [28] Pledger project YouTube channel, <https://www.youtube.com/channel/UCXV6V9rJ0ZvWhXeoWvDsArQ> [retrieved date 01/08/2022]
- [29] E. Carmona Cejudo and M. S. Siddiqui, “An Optimization Framework for Edge-to-Cloud Offloading of Kubernetes Pods in V2X Scenarios” 2021 IEEE Globecom Workshops (GC Wkshps).
- [30] E. Carmona-Cejudo, F. Ianzada and M. S. Siddiqui, "Optimal Offloading of Kubernetes Pods in Three-Tier Networks," 2022 IEEE Wireless Communications and Networking Conference (WCNC), 2022, pp. 280-285, doi: 10.1109/WCNC51071.2022.9771724.

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	39 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8
				<b>Status:</b>	Final

- [31] Carmona Cejudo, Estela & Iadanza, Francesco. "Demo: A Decision Support System for Task Offloading Optimization in Cloud-to-Far-Edge Kubernetes Networks". 2022 IEEE International Conference on Communications, Seoul, South Korea
- [32] IEEE Transactions on Vehicular Technology,  
<https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=25> [retrieved date 01/08/2022]
- [33] IEEE Transactions on Vehicular Technology, impact factor  
<https://vtsociety.org/publication/ieee-transactions-vehicular-technology> [retrieved date 01/08/2022]

<b>Document name:</b>	D4.6 Decision Support tools II			<b>Page:</b>	40 of 46		
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU	<b>Version:</b>	0.8	<b>Status:</b>	Final

## 8 Annex

### 8.1 Configuration UI screenshots

This subsection collects some screenshots from the Configuration UI, grouped by **role**, already described in D4.3 and reported here for readers' convenience to show the main configuration features provided to each role.

#### 8.1.1 Administrator.

Administrators are responsible for the creation of Pledger users and for the monitoring of authentication activities. Figure 28 shows the Administrator drop-down menu while Figure 29 shows the user dashboard where the Administrator can assign roles to users and enable/disable them. Figure 30 shows audit logs and Figure 31 the list of Service provider users.

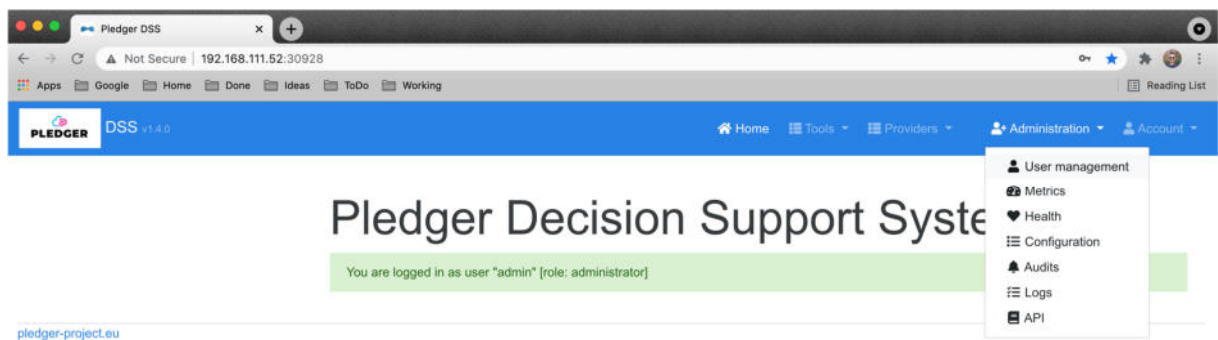


Figure 28: Administrator manages Pledger users

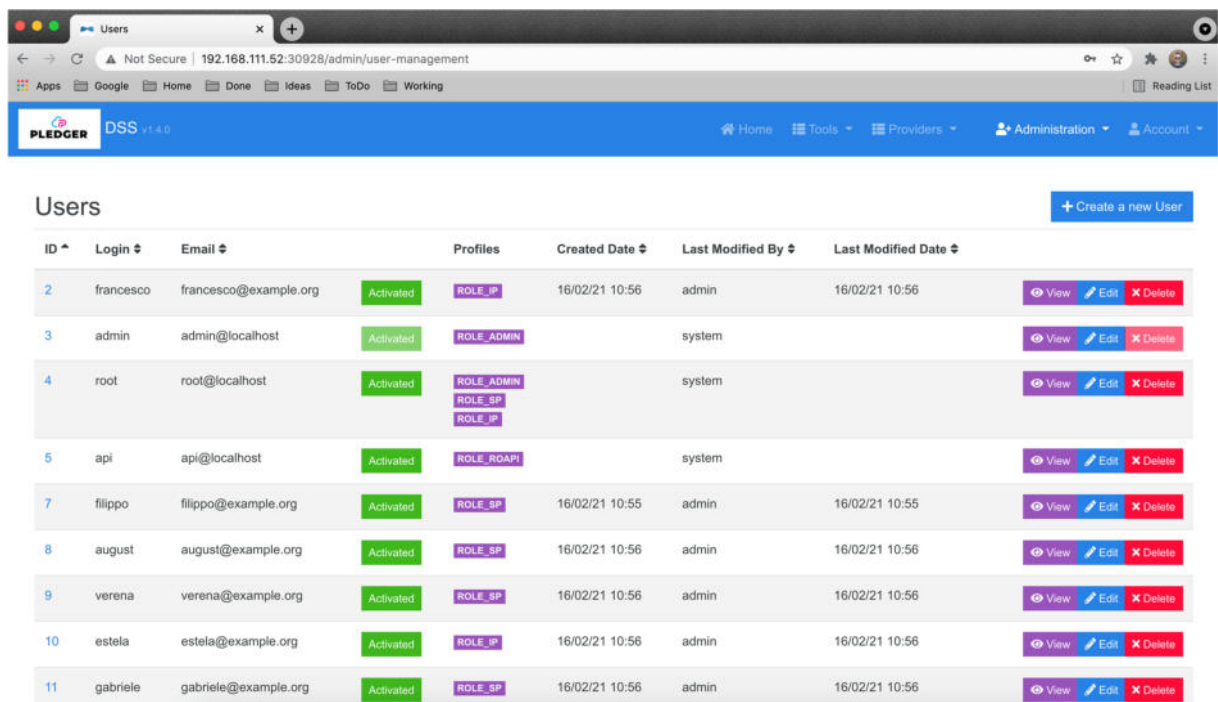
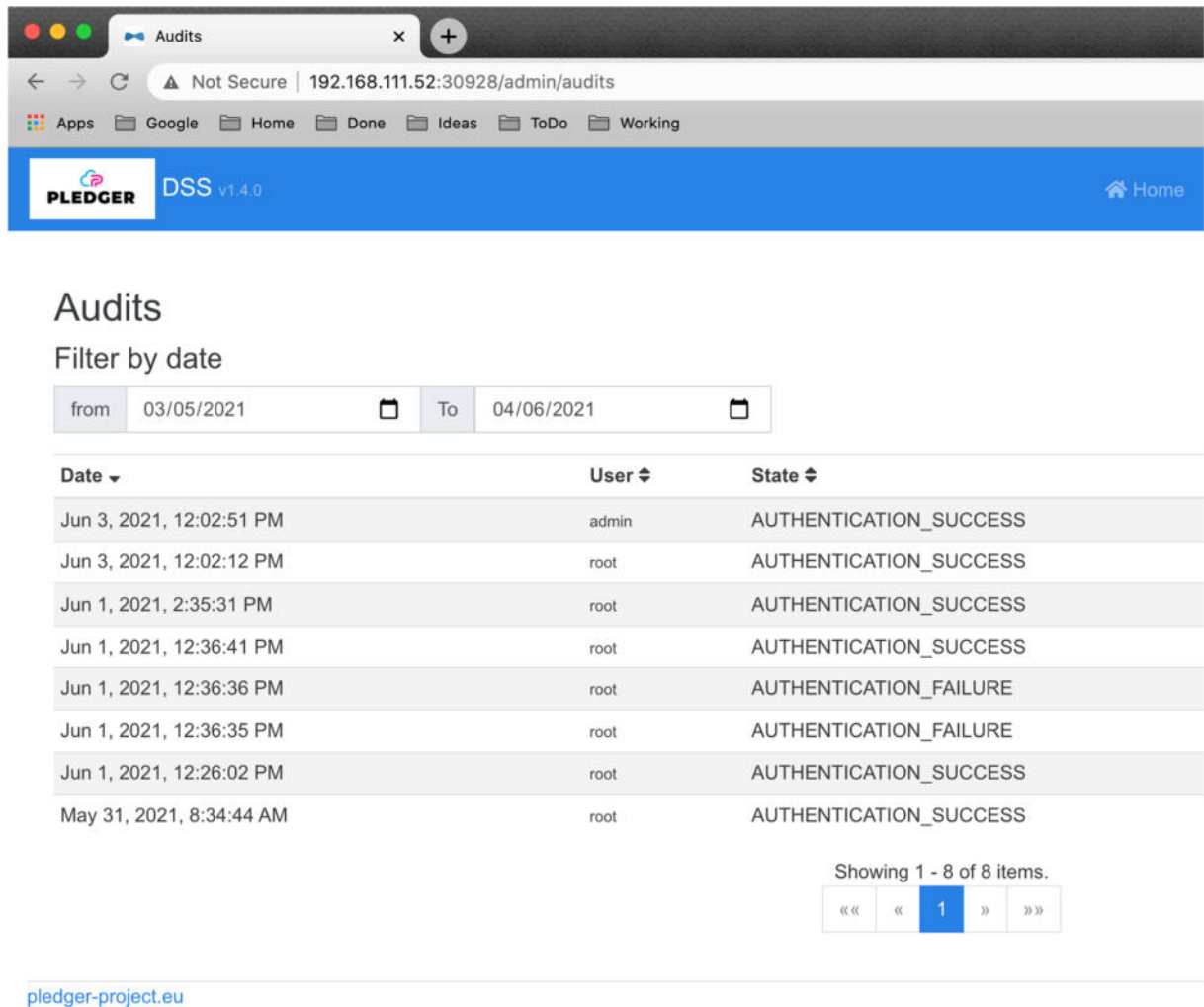


Figure 29: Users with roles

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	41 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
<b>Version:</b>	0.8	<b>Status:</b>	Final



**Audits**

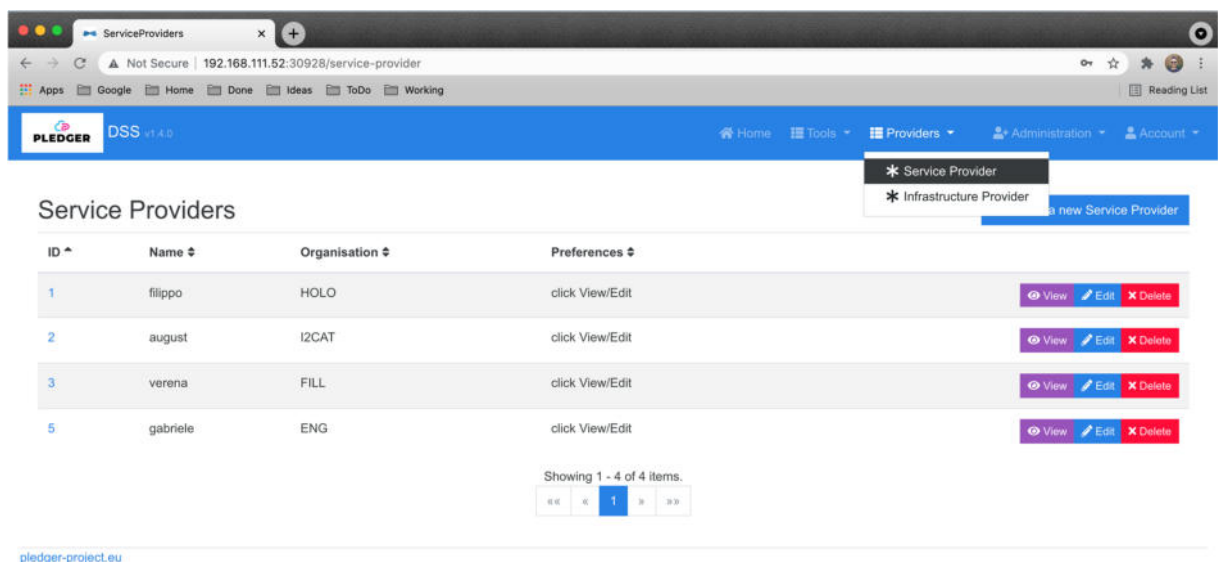
Filter by date

from 03/05/2021 To 04/06/2021

Date	User	State
Jun 3, 2021, 12:02:51 PM	admin	AUTHENTICATION_SUCCESS
Jun 3, 2021, 12:02:12 PM	root	AUTHENTICATION_SUCCESS
Jun 1, 2021, 2:35:31 PM	root	AUTHENTICATION_SUCCESS
Jun 1, 2021, 12:36:41 PM	root	AUTHENTICATION_SUCCESS
Jun 1, 2021, 12:36:36 PM	root	AUTHENTICATION_FAILURE
Jun 1, 2021, 12:36:35 PM	root	AUTHENTICATION_FAILURE
Jun 1, 2021, 12:26:02 PM	root	AUTHENTICATION_SUCCESS
May 31, 2021, 8:34:44 AM	root	AUTHENTICATION_SUCCESS

Showing 1 - 8 of 8 items.

Figure 30: Users login audit



**Service Providers**

ID	Name	Organisation	Preferences
1	filippo	HOLO	click View/Edit
2	august	I2CAT	click View/Edit
3	verena	FILL	click View/Edit
5	gabriele	ENG	click View/Edit

Showing 1 - 4 of 4 items.

Figure 31: Definition of Pledger SP and IP users

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	42 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

### 8.1.2 Infrastructure Provider.

Infrastructure providers are responsible for the definition of the infrastructure and nodes in Pledger, along with the required configuration to allow its monitoring. Figure 32 shows the monitoring plugin configuration and Figure 33 shows the nodes with the hardware features automatically discovered by Pledger.

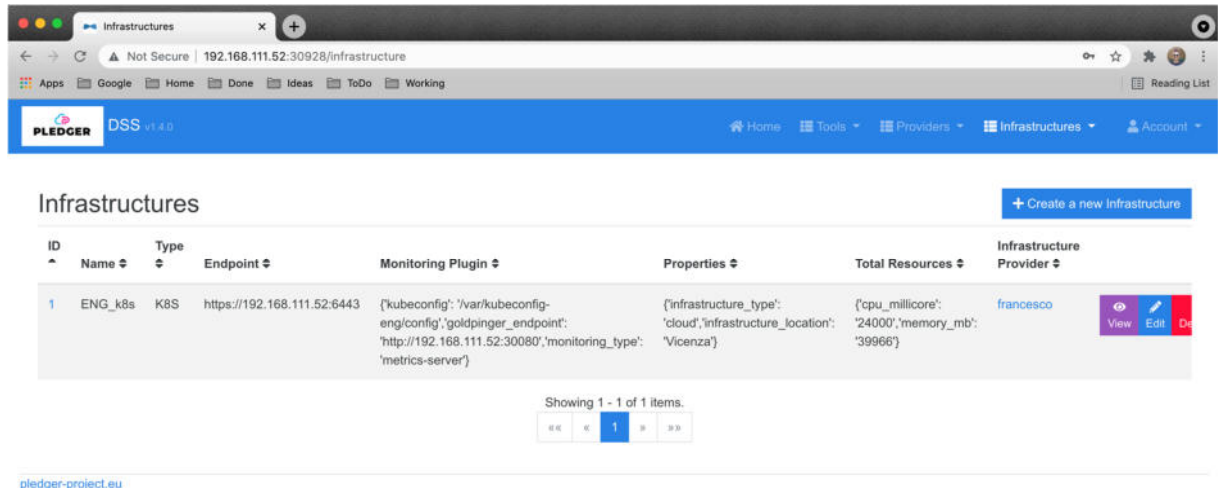


Figure 32: Infrastructure with monitoring plugin and resources capacity

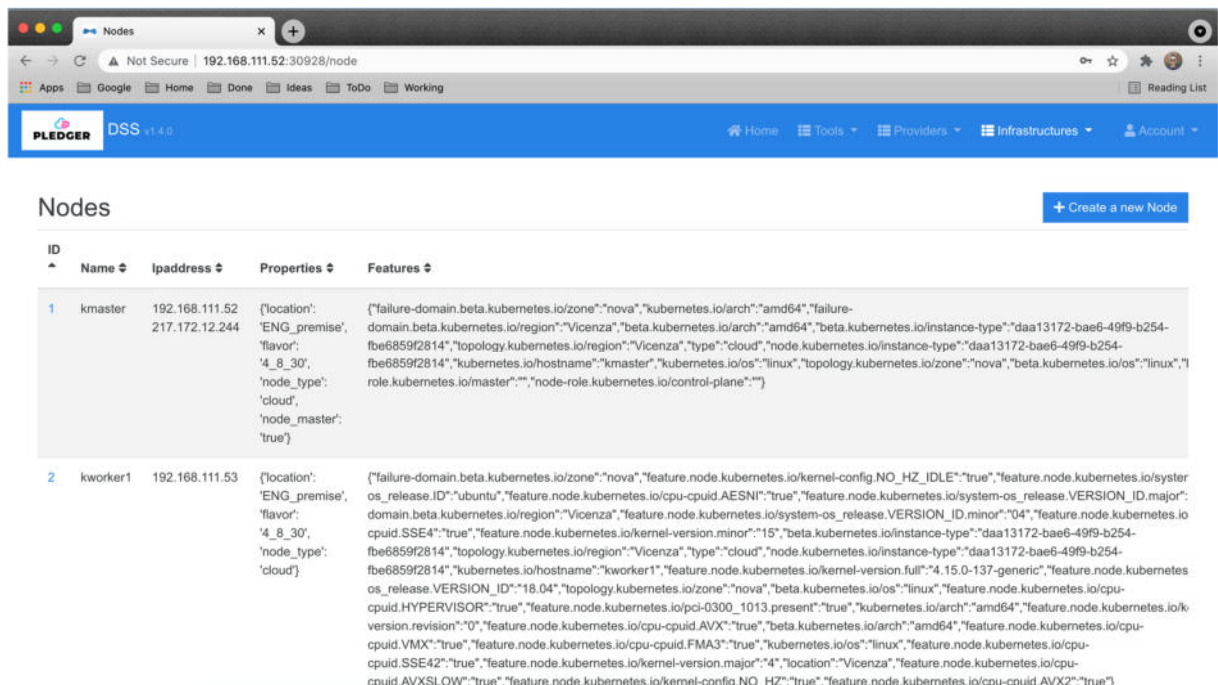


Figure 33: Nodes with feature auto-discovery features working to identify HW availability

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	43 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

### 8.1.3 Service Provider.

Service providers are responsible for the configuration of the DApps, their preferences about the deployment options to filter and prioritize the DSS options. Figure 34, Figure 35, Figure 36 show the configuration of catalog applications, services. Figure 37, Figure 38 and Figure 39 show the configuration of SLA and Guarantees.

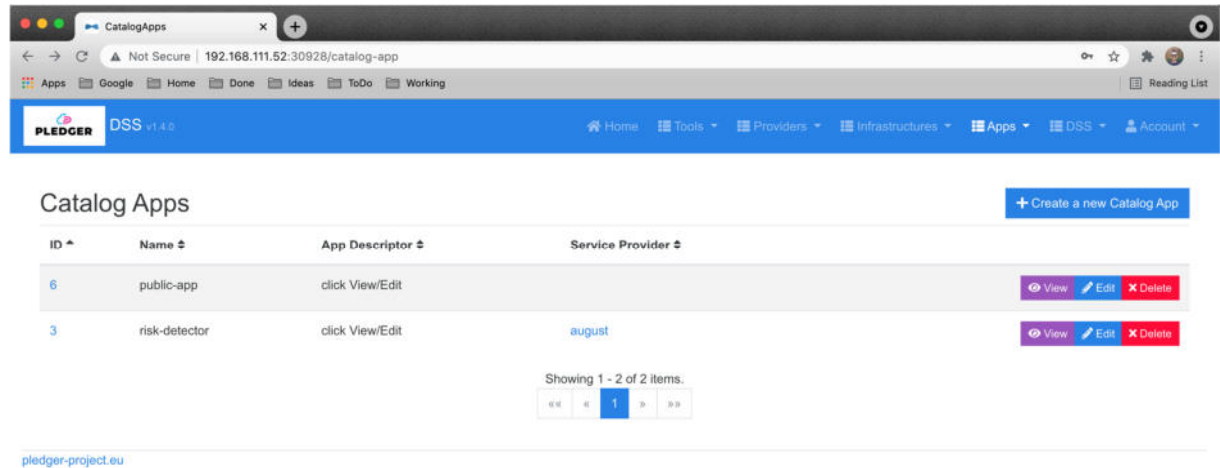


Figure 34: Catalog applications

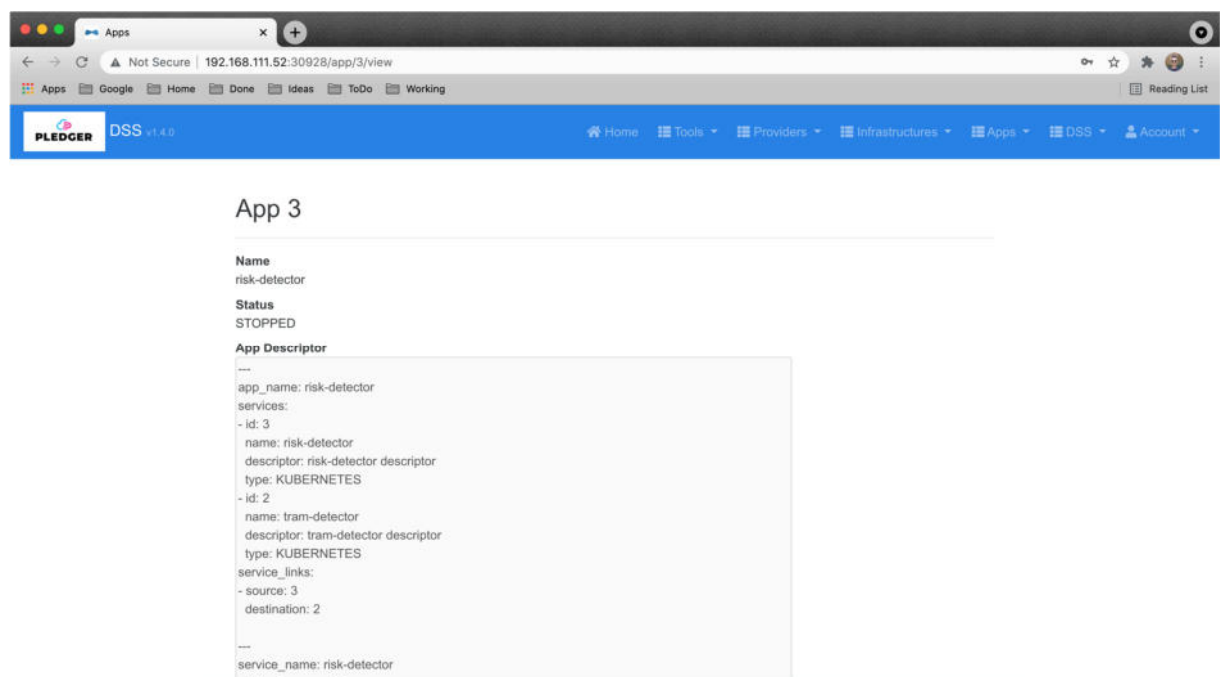


Figure 35: DApp composed by multiple services with K8S descriptor

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	44 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final

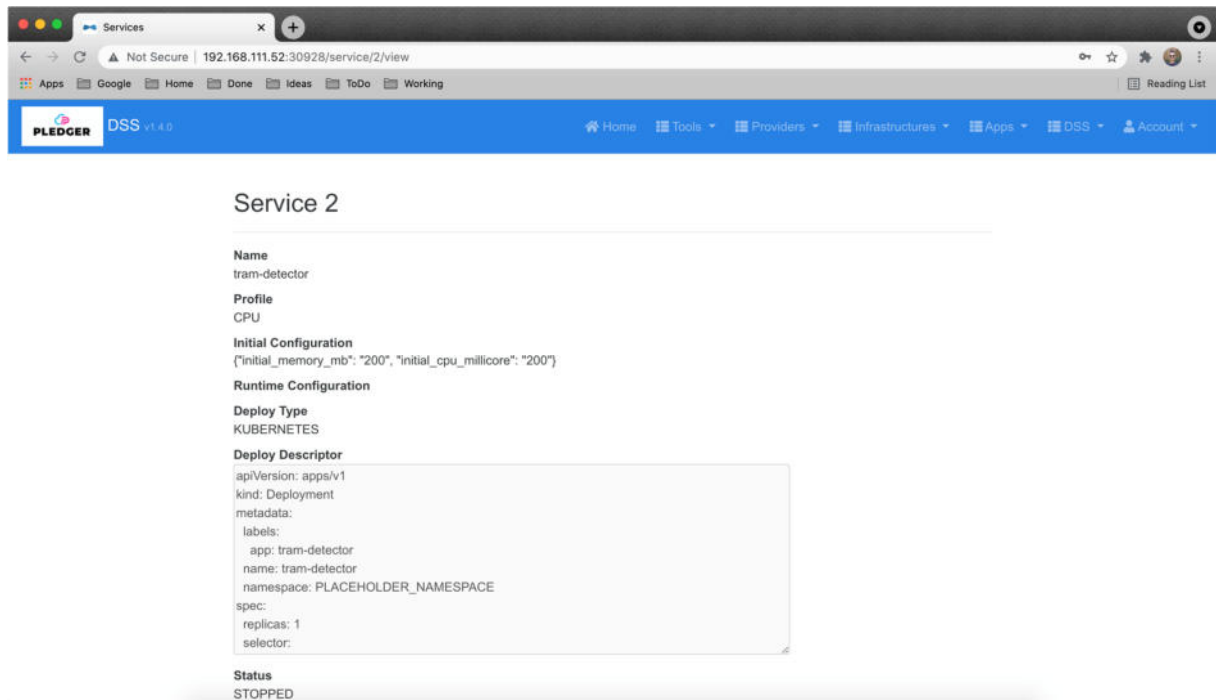


Figure 36: Service with descriptor and initial configuration about resources (Kubernetes)

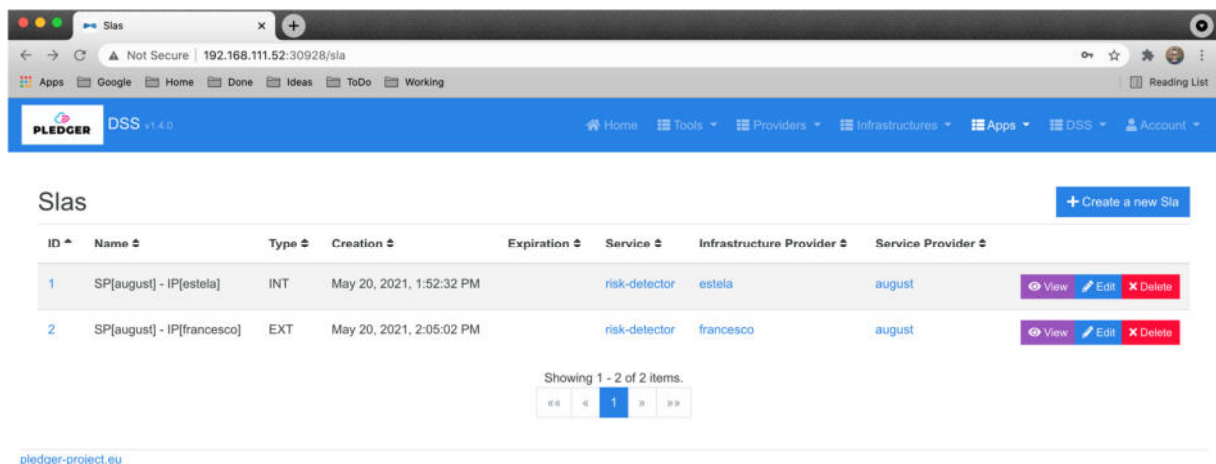
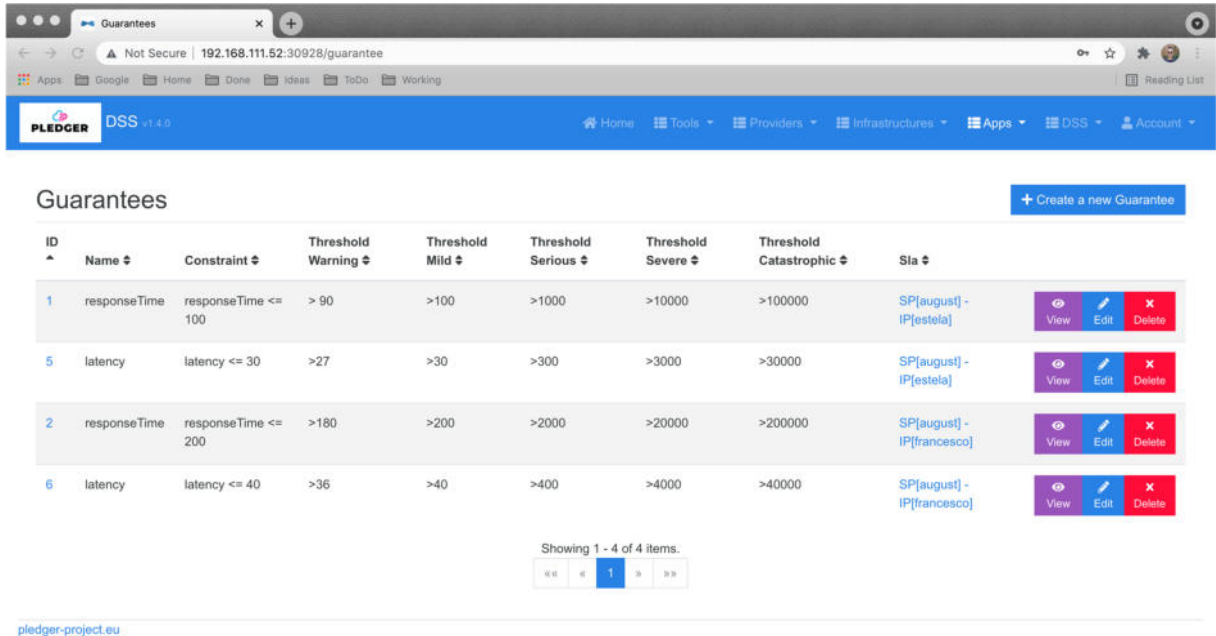


Figure 37: SLA definition

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	45 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final



Guarantees

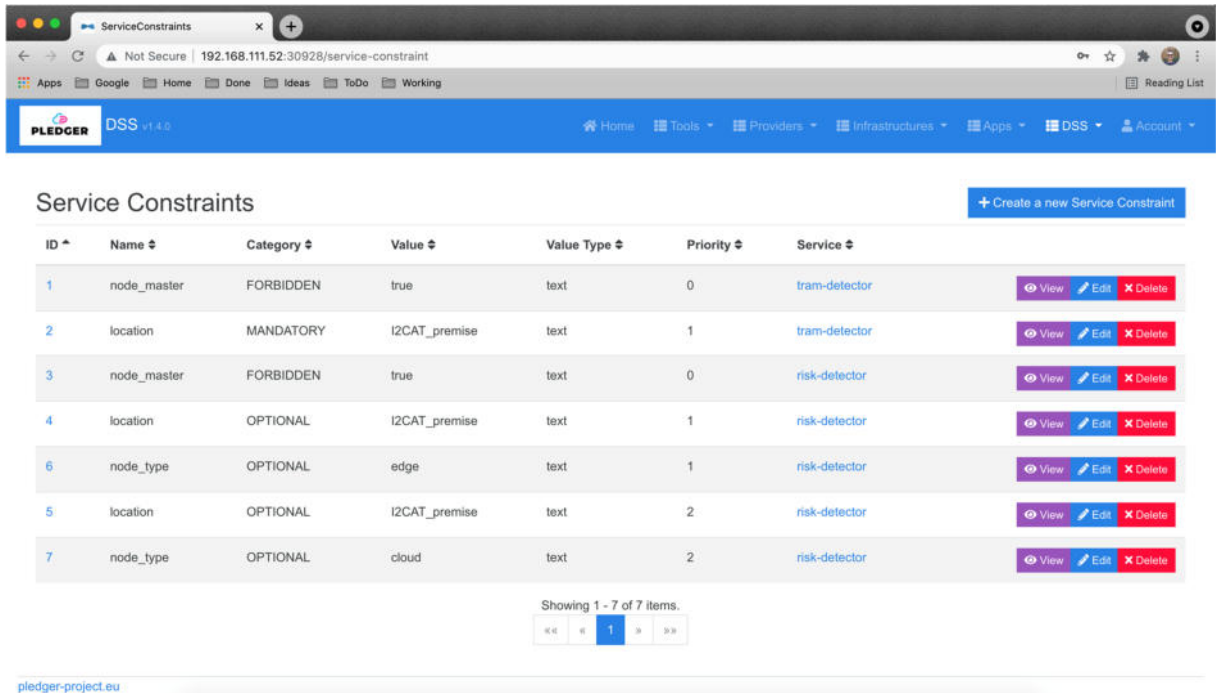
+ Create a new Guarantee

ID	Name	Constraint	Threshold Warning	Threshold Mild	Threshold Serious	Threshold Severe	Threshold Catastrophic	Sla	
1	responseTime	responseTime <= 100	> 90	>100	>1000	>10000	>100000	SP[august] - IP[estela]	View Edit Delete
5	latency	latency <= 30	>27	>30	>300	>3000	>30000	SP[august] - IP[estela]	View Edit Delete
2	responseTime	responseTime <= 200	>180	>200	>2000	>20000	>200000	SP[august] - IP[francesco]	View Edit Delete
6	latency	latency <= 40	>36	>40	>400	>4000	>40000	SP[august] - IP[francesco]	View Edit Delete

Showing 1 - 4 of 4 items.

pledger-project.eu

Figure 38: Guarantees definition



ServiceConstraints

+ Create a new Service Constraint

ID	Name	Category	Value	Value Type	Priority	Service	
1	node_master	FORBIDDEN	true	text	0	tram-detector	View Edit Delete
2	location	MANDATORY	I2CAT_premise	text	1	tram-detector	View Edit Delete
3	node_master	FORBIDDEN	true	text	0	risk-detector	View Edit Delete
4	location	OPTIONAL	I2CAT_premise	text	1	risk-detector	View Edit Delete
6	node_type	OPTIONAL	edge	text	1	risk-detector	View Edit Delete
5	location	OPTIONAL	I2CAT_premise	text	2	risk-detector	View Edit Delete
7	node_type	OPTIONAL	cloud	text	2	risk-detector	View Edit Delete

Showing 1 - 7 of 7 items.

pledger-project.eu

Figure 39: ServiceConstraints, setup by the SP, to express deployment preferences

<b>Document name:</b>	D4.6 Decision Support tools II	<b>Page:</b>	46 of 46
<b>Reference:</b>	D4.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	0.8	<b>Status:</b>
			Final