PLEDGER

Automated deployment and orchestration on the edge

Pledger Handbook

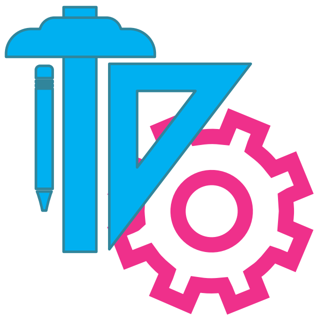


Table of Contents

List of Tables

List of Figures

1 PLEDGER in a nutshell

With the number of connected devices continuously growing, the need for processing data near to them to avoid latency issues and provide real-time responses is also increasing. Translating this growth into numbers, the global edge computing market will reach USD 8.96 billion by 2023, growing at a CAGR of 32.6%[1]. Only at European level this growth will represents USD 1.94 billion by the same time, growing at a CAGR of 29.3%[2]. This is mainly due to the increased demand for low-latency and real-time solutions to avoid network traffic bottlenecks. However, cloud computing will not disappear, but its capabilities will be extended closer to the edge in the so-called fog computing paradigm (also known as fogging or edge computing).

In this context, PLEDGER delivers a new architectural paradigm and a toolset that paves the way for next generation edge computing infrastructures, tackling the modern challenges faced today and coupling the benefits of low latencies on the edge, with the robustness and resilience of cloud infrastructures. This toolset allows third parties to act as independent validators of QoS features in IoT applications, enabling new decentralised applications and business models, thus filling a large gap in the emerging edge/IoT computing market landscape.

For doing so, PLEDGER brings a set of innovations explained as follows:

Smart contracts and blockchain

PLEDGER deploys and leverages a suite of blockchain related technologies along with smart contracts in order to implement the requested features of the system. Solid innovation concepts rely on the ability to couple Distributed Ledger technologies together with concepts and implementation from the edge and cloud computing paradigms. In particular, the following:

- Private transactions in permissioned/enterprise blockchain networks. Unlike public blockchains, these platforms constitute a business-suitable blockchain component that also provides private transactions.
- Decentralised Applications executing on its blockchain exploiting all the legitimate edge user data exchange and delivering high quality systems services information adding to the PLEDGER's overall value. This kind of applications are performing secure and immutable blockchain transactions and can provide ways to automate metric measurements and other important data retrieval through well-developed smart contracts and submitted transactions without intermediaries.
- Privacy enhancing technologies supported by an advanced anonymous personalization service framework. In cloud and edge environments a recommender system functionality along with blockchain network technologies add to PLEDGER's completeness in the sense of the combination of different and useful capabilities.

Smart contracts on blockchains that embody SLA terms

SLA terms are of vital importance for any service offered through cloud environments. The specific terms describe the level of quality of the specific service offered by the cloud providers. The way to extend such SLA terms to edge computing environments necessitates new, innovative ways on how the terms are being expressed. Moreover, by leveraging blockchain transactions time efficiency and the ability to have pieces of code being executed when specific conditions are met provides an innovative framework for a new generation of SLA monitoring and enforcement frameworks.

[1] R. a. Markets, «Global Edge Computing Market (2018-2023)». Available: https://www.researchandmarkets.com/reports/4667633/global-edge-computing-market-2018-2023#rela0-4667632.

[2] R. a. Markets, «Europe Edge Computing Market (2018-2023)». Available: https://www.researchandmarkets.com/research/4t4m3w/the_edge?w=4.

Edge/Cloud SLA monitoring

There are several tools, or approached, already in the market for SLA monitoring. Some of them don't take into account application faults for the calculation or don't consider different providers' definition.

PLEDGER proposes an abstracted approach to monitor and manage SLAs regardless of the provider. Thus, the main innovation potential of PLEDGER relies on auditing contracts in a provider compliant manner, enable compensation claiming and introducing novelties to the market.

Edge/Cloud offloading and trade-off

Based on the current state of the art technologies, PLEDGER proposes an innovative decision support mechanism for offloading activities taking into account several criteria, such as energy, cost, bandwidth and computing requirements. It also considers additional parameters, such as SLA terms and trust, smart contract terms, features of the deployed applications and privacy requirements. All these considerations are used to take decisions about what should be moved from the edge up to the cloud.

Edge/Cloud benchmarking

Benchmarking is a common technique for assessing application performance that is affected by several factors external to the user. Based on already existing tools, PLEDGER extends them for edge infrastructures taking into account the variety of edge nodes with different capabilities and software available.

Privacy, trust and security on edge computing

PLEDGER approach includes a set of tools to offer multi-layer security protection at the edge, including blockchain and machine learning algorithms to detect anomalies and prioritise risks. These tools also support traceability, access control and GDPR- related functionalities.

Network slicing in 5G (and other) networks

Current trends turn the network into a platform to host services. More specifically, to run multiple logical networks as virtually independent business operations into one physical infrastructure (network slicing). These network slices must be able to run isolate, without affecting other slices. For this, PLEDGER incorporate mechanisms to measure the degree of isolation by different providers, and how to enforce isolation for each of them, over edge and fog infrastructures.

More information about how these innovations are part of the PLEDGER provided tools and how users can benefit of them based on some applicability examples are further described in the next sections.

1.1 Main Objectives and added value

PLEDGER has 7 main objectives to add value for both, service and infrastructure edge/cloud providers, as explained below:

Obj.1 Enhance edge/fog computing provider resource management practices that lead to improved stability of offered services and QoS/QoE for end users.

PLEDGER provides a toolkit for end-to-end implementation of resource management practices that can aid edge/fog computing providers in making informed decisions about application deployment and service offer, via co-scheduling complementary applications from a resource utilisation point of view for optimising the performance characteristics of the application at runtime. The tools cover aspects such as resource usage monitoring, modelling of application types runtime characteristics, collaborating with the typical cloud management toolkits for offering recommendation services regarding service containers grouping on physical nodes on the cloud.

Obj.2 Architect and implement the coupling of edge/fog computing with blockchains and distributed ledger technologies.

PLEDGER target leveraging state-of-the-art architectural approaches and implementations to a higher abstraction layer and common APIs, thus hiding the provider-specific details and technology specific differentiations. This way, it may be used across the various toolkits included in the project without modifications and aid in a faster uptake and easier deployment of the involved mechanisms. This relates to measurement frameworks, deployment aspects (of applications and the toolkits) and monitoring characteristics. The innovative architectural elements that PLEDGER introduces are structured around the following axes:

- Achieving higher levels of trust in open trustless edge/fog infrastructures through the adoption of blockchain related technologies and methodologies based on “proof-of-*” techniques (e.g., proof-of-work, proof-of-sake, etc.) that enable edge resources orchestrate themselves in a trusted way.
- Allowing the implementation of smart contracts in edge infrastructure elements, combining the abilities of high automation in executing code portions on the edge while combining at the same time smart contracts with Service Level Agreements contractual negotiations.
- Allowing the deployment of decentralised applications (DApps) in large edge infrastructure deployments as a new paradigm of applications for suitable use cases that can harness the benefits of the distributed nature of edge computing.

Obj.3 Enhance cloud/edge services behaviour measurability and predictability in order to increase operational trust, thus enabling their use in more critical applications and increase the synergies between cloud/edge infrastructures.

PLEDGER offers tools to monitor the overall performance stability and efficiency of the resources (as organized by the outcomes of Objective 1) through a suitable measurement and benchmarking framework that may be used by the adopters. This aspect ensures and validates that experienced performance guarantees are indeed met and thus enable providers to issue performance-based SLAs, an aspect that is missing from the modern edge computing landscape, but also evaluate provider performance. Hence, it will enable direct comparability, abstracted at the application level. The measurement tools provided in PLEDGER will cover both the computing and the network domains, leveraging (in the case of the network) novel SDN frameworks to gather network telemetry.

Obj.4 Define and apply sets of metrics that are meaningful to the end users for QoE and QoS levels.

PLEDGER defines and includes an extended set of metrics to characterise provider QoS/QoE levels and offered SLAs in a way that is both comparable between the various definitions and easily grasped by the end users of edge computing infrastructures. These metrics are abstracted from the backend technical details that are needed for their definition and will be used to further abstract decision making and service rating at the edge/cloud user and application level. Furthermore, they are configurable based on user preferences and adapt provider rankings automatically, visualising them towards the end user. In edge environments, the network cannot be overprovisioned like in a data centre, thus PLEDGER edge providers leverage techniques such as SDN to orchestrate the connectivity between distributed edge functions required to provide the required level of QoS/QoE.

Obj.5 Define and implement new ways of increasing security and privacy on edge, filtering out (during the cloud moving process) with intelligent way the data that have to be kept on the edge.

PLEDGER engineers new levels of security and trust in large scale autonomous and trustless multipurpose edge/cloud platforms and integrates Privacy Enhancing Technologies (PETs) at the design level of PLEDGER’s architecture. It defines and implements a lightweight blockchain ledger and the trust-ensuring mechanisms that enables machine-to-machine and human-to-machine transactions with transparency, and all necessary security aspects (authentication, authorization, accounting, etc.). PLEDGER provides applicable solutions on how PET is applied on the edge resources, thus suggesting innovative ways on how end-to-end security can be achieved across the whole path from IoT sensor to edge and cloud computing and over public multipurpose infrastructures guaranteeing privacy and confidentiality of the information flow.

Obj.6 Pilot and demonstrate the applicability and scalability of the tools on large scale edge/cloud infrastructures and for applications that typically need specialised oversized infrastructures.

PLEDGER tools are validated in edge/fog/cloud infrastructures of considerable size in order to demonstrate both the scalability of the involved tools and the ability to utilize heterogeneous and general usage edge/cloud computing setups for application cases that up to now demand specialised and typically oversized infrastructures. The benefit of the usage of tools is further explained in this document based on the pilots' experience.

Obj.7 Enable the extension of the edge/cloud combined business model and provide an extensive replicability and best practices framework.

PLEDGER aims at extending business models for the edge/fog/cloud combination as well as easing the applicability and adaptation of such infrastructures to user requirements. In this context, it enables new consulting roles and the tools to implement them, new entities to exploit and perform analytics on top of concentrated data in terms of SLA, smart contracts and performance monitoring and new application categories to exploit the benefits of the combined edge/cloud model. PLEDGER also provides best practices and a wider replicability framework (both in technical and organizational terms) that is expected to significantly contribute towards the future wider adoption of its findings in multiple business environments.

1.2 PLEDGER Architecture

PLEDGER consists of two main types of subsystems which can be identified as follows:

- **Core subsystems:** The Pledger core system is considered 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.
- **Use Cases subsystems:** The Pledger Use Case subsystems are considered the subsystems that are developed for the pilots of the project. Use Case subsystems are considered the subsystems that are used only for the sake of completing successfully the objectives of the pilots per se (and not of the project as a whole).

As far as the core subsystems are concerned, the main ones that have been identified, are the following:

1. **Configuration subsystem:** Subsystem responsible for providing and storing the infrastructure and application configuration details.
2. **Orchestration subsystem:** Subsystem responsible for managing the orchestration of containerized applications (actions related to the deployment of applications, infrastructure scale up/down, etc.).
3. **Benchmarking subsystem:** Subsystem responsible for providing performance data of configured infrastructures to better characterize them and to optimize the orchestration with suggestions on application performance.
4. **SLAs subsystem:** Subsystem responsible for creating, managing and evaluating the SLAs associated to the applications running on the Pledger Cloud and Edge environment.
5. **Blockchain subsystem:** Subsystem responsible for providing the Pledger DLT and the corresponding middleware (set of tools offering features such as smart contracts and cryptocurrencies).
6. **Big Data & Communication Platform subsystem:** Subsystem responsible for exposing a high-performance distributed streaming platform for interconnecting, storing, and transforming.
7. **Security subsystem:** Subsystem responsible for enhancing the security of the overall system as a whole.

The above subsystems are split in three functional groups:

1 **Management subsystems:** Subsystems that are focused on the management of IaaS and SaaS.
2 **Evaluation subsystems:** Subsystems that are focused on the evaluation of IaaS.
3 **Support subsystems:** Supporting subsystems related to communications and security.

The following figure gives the subsystems view of the Pledger Core System. The view is not hierarchical and the interrelations between the subsystems that are provided are the main ones.

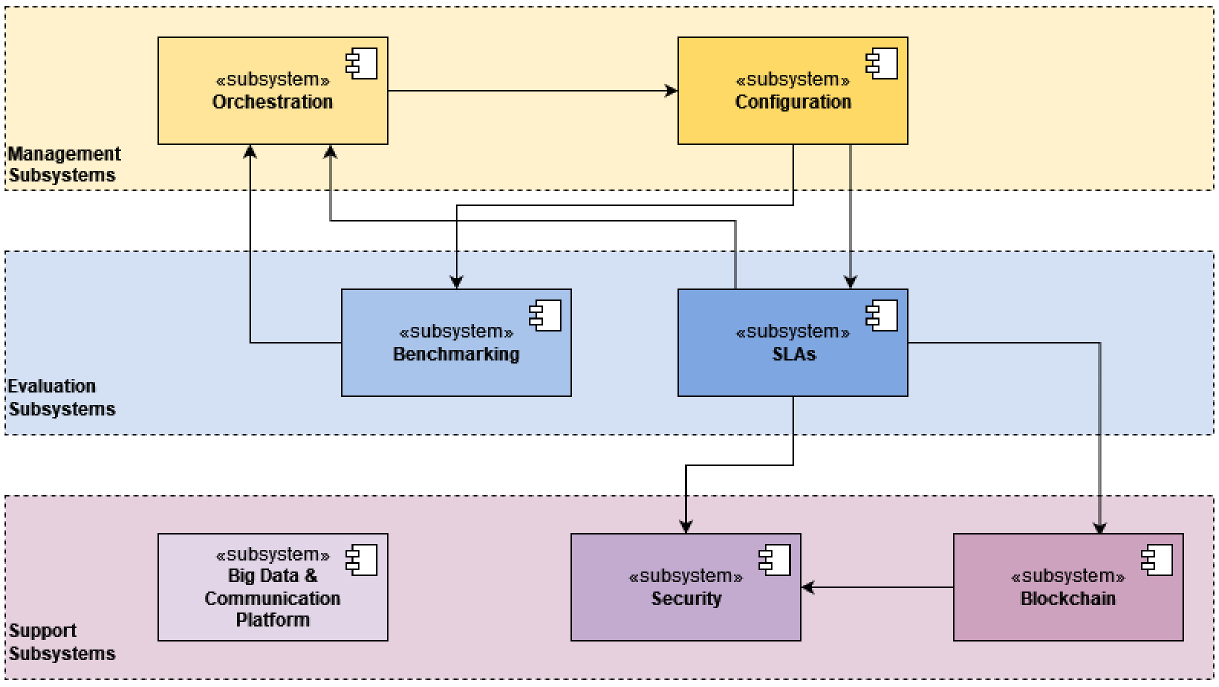


Figure 1: subsystems view of the Pledger Core System

The following figure gives the overall view of the Pledger Core System in the form of a Component diagram. The diagram depicts all the different core components of the system as well as the associations between them (mostly input/output relations), with some of the associations being omitted to make the diagram easier to read. It also depicts the different subsystems as well as the different subsystem groups.

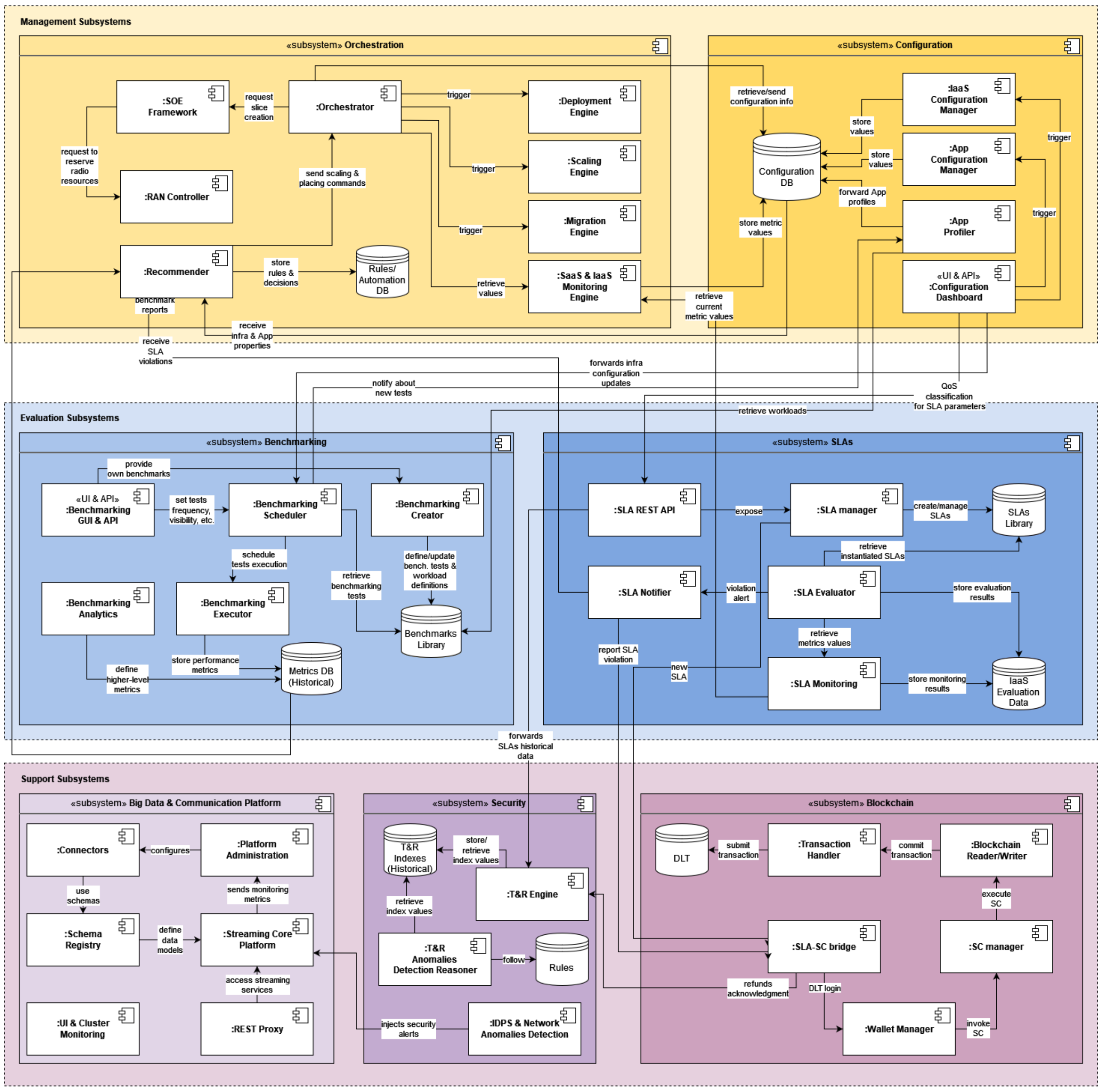


Figure 2: Pledger Core System in the form of a Component diagram

The different subsystems and components are presented now in more detail:

1.2.1 Configuration subsystem

Configuration subsystem: The main responsibility of the Configuration subsystem is to store the infrastructure and application configuration that is managed either manually by the IaaS/ SaaS providers or by tools that support automatic discovery features (e.g., the App Profiler).

The main information stored is users' configuration, infrastructure configuration, app configuration, multitenancy and authorizations for users to access specific infrastructures within resource limitations (e.g., resource quotas).

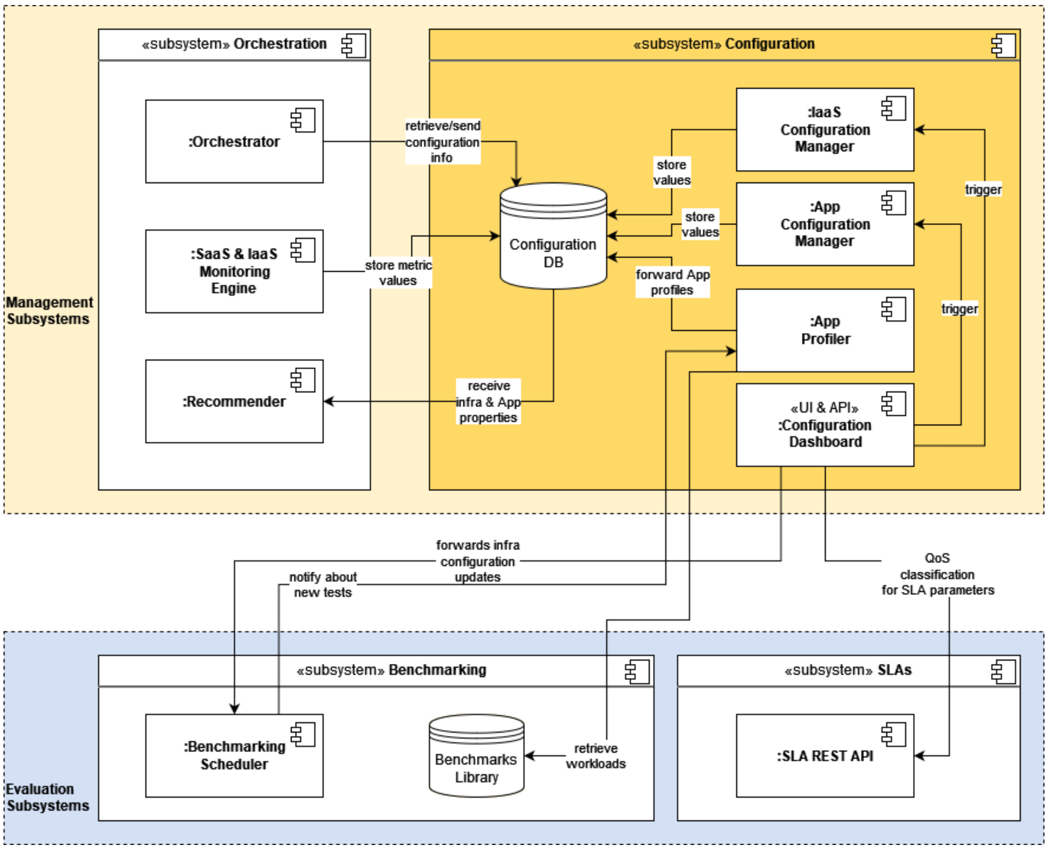


Figure 3: Configuration subsystem Component Diagram

1.2.2Orchestration subsystem

Orchestration subsystem: The Orchestration subsystem is the link between the Decision Support System (DSS) and the underlying infrastructure, providing an abstraction layer on top of infrastructure management tools of choice to the DSS. This way, it allows to decouple the DSS from the technical implementation details and facilitates interoperability with diverse technical solutions and makes it possible to transparently update and change low level infrastructure management tools. The DSS ("Recommender" in the diagrams) will implement the intelligence of decision-making and the Orchestrator will be the executor arm of DSS decisions (implementing on demand the necessary actions).

This subsystem will manage orchestration of containerized applications (i.e. relying on Kubernetes implementation) to handle its management. The actions foreseen for this subsystem are related to the deployment of applications, infrastructure scale up/down and migration as well as operations in relation to the operational life-cycle of applications (start, stop, update and get current status of the application).

In addition, this subsystem also considers the management of complete clusters at different infrastructure levels (cloud, edge, on premise) as well as the deployment of cluster orchestration software (for single-node cluster) in the edge. The subsystem also features the necessary tools for slice creation and management. The slicing concept applied in Pledger refers to the reservation of computation and radio resources from a pool of available (physical and virtual) resources of an infrastructure to form an isolated, logical cluster of nodes on top of which services can be deployed, while enabling end-to-end connectivity between the radio access and the services deployed in the computation nodes.

The Orchestration subsystem will be in charge of selecting the best nodes of a cluster (best effort) to run an application based on the SaaS provider's preferences or the profile of the application (e.g. app requires GPU to run) or the recommendations added by DSS at runtime. The Orchestrator will also have receive input from a Monitoring Engine to collect different metrics of the infrastructure in a time series database. With these functionalities we can settle the base for a QoS control system.

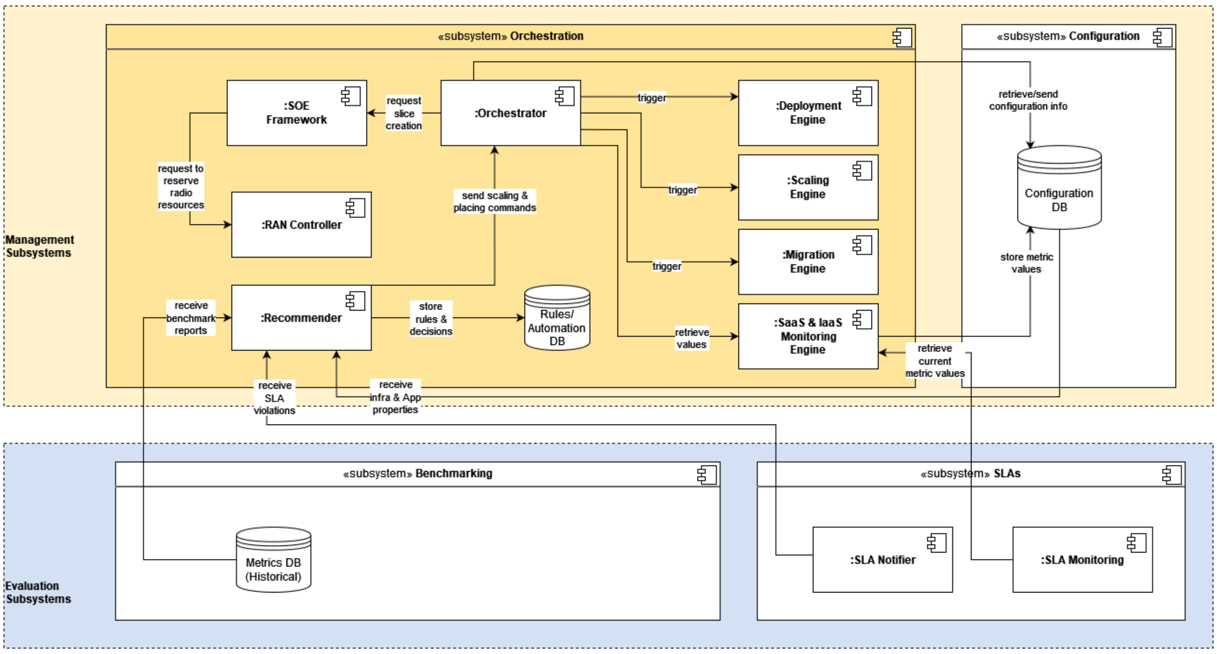


Figure 4: Orchestration subsystem Component Diagram

1.2.3 Benchmarking subsystem

Benchmarking subsystem: The main responsibility of the Benchmarking subsystem is to provide performance data of configured infrastructures to better characterise them and to optimise the orchestration with suggestions on application performance.

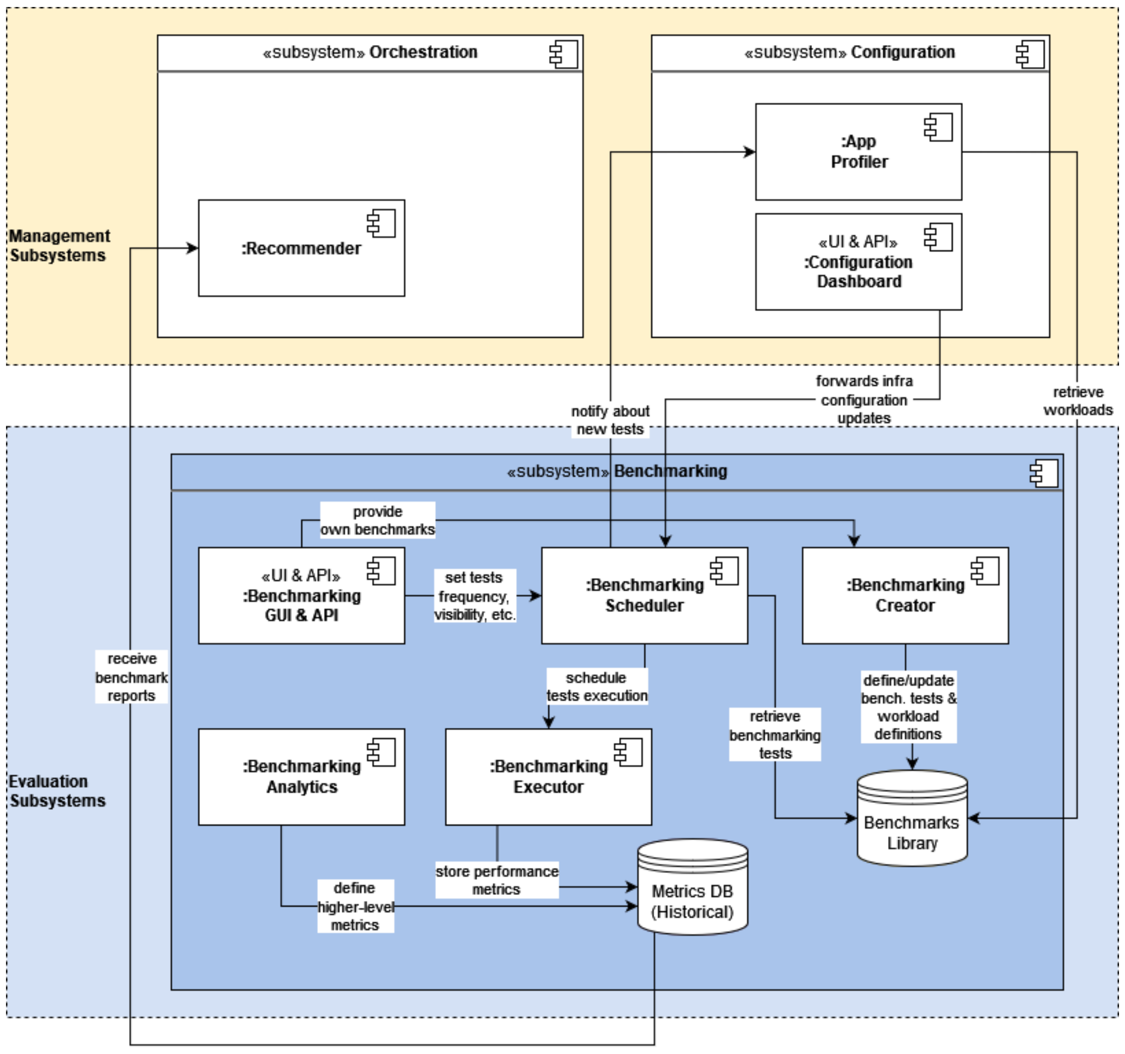

Management Subsystems
«subsystem» Orchestration
:Recommender
«subsystem» Configuration
:App Profiler
«UI & API» :Configuration Dashboard
forwards infra configuration updates
notify about new tests
retrieve workloads
Evaluation Subsystems
«subsystem» Benchmarking
provide own benchmarks
«UI & API» :Benchmarking GUI & API
set tests frequency, visibility, etc.
:Benchmarking Scheduler
:Benchmarking Creator
receive benchmark reports
schedule tests execution
retrieve benchmarking tests
define/update bench. tests & workload definitions
:Benchmarking Analytics
:Benchmarking Executor
Benchmarks Library
store performance metrics
define higher-level metrics
Metrics DB (Historical)

Figure 5: Benchmarking subsystem Component Diagram

1.2.4 SLA subsystem

SLAs subsystem: The SLAs subsystem is the subsystem responsible for creating, managing and evaluating the SLAs associated to the applications running on the Pledger Cloud and Edge environment. This subsystem relies on the information gathered by external monitoring tools, which are used to continuously evaluate the SLAs, and to notify other components about violations or other relevant information related to the QoS of these applications.

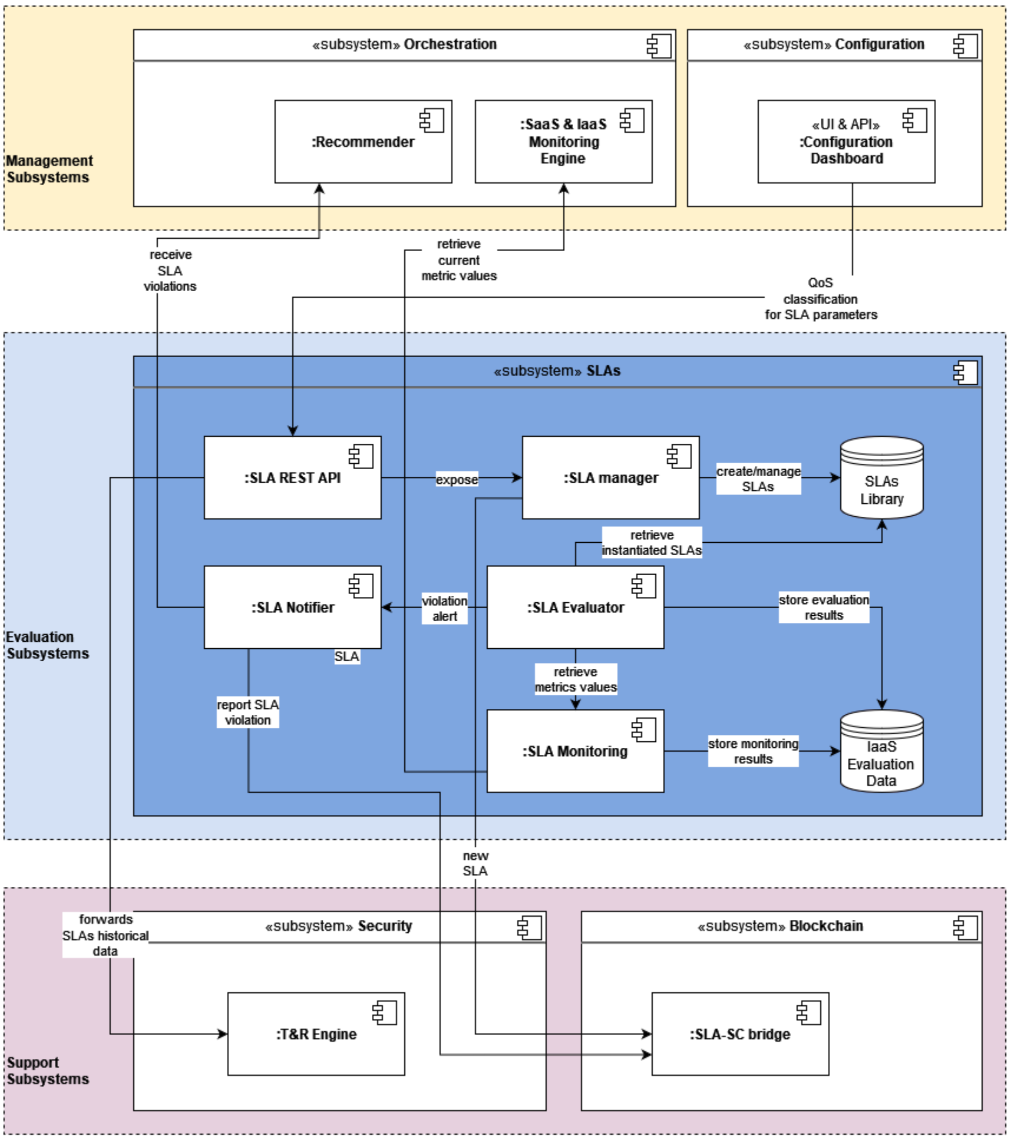


Figure 6: SLA subsystem Component Diagram

1.2.5 Blockchain subsystem

Blockchain subsystem: Pledger will provide its own Blockchain/ Distributed Ledger, taking under consideration features such as openness, access, speed, security, use of consensus/ security mechanisms, etc. Depending on the implementation scenario chosen by the users (Blockchain as a Service – BaaS, vertical solutions, etc.) Pledger will also provide a set of tools offering features such as smart contracts and cryptocurrencies, thus enabling a wide range of applications under different business models.

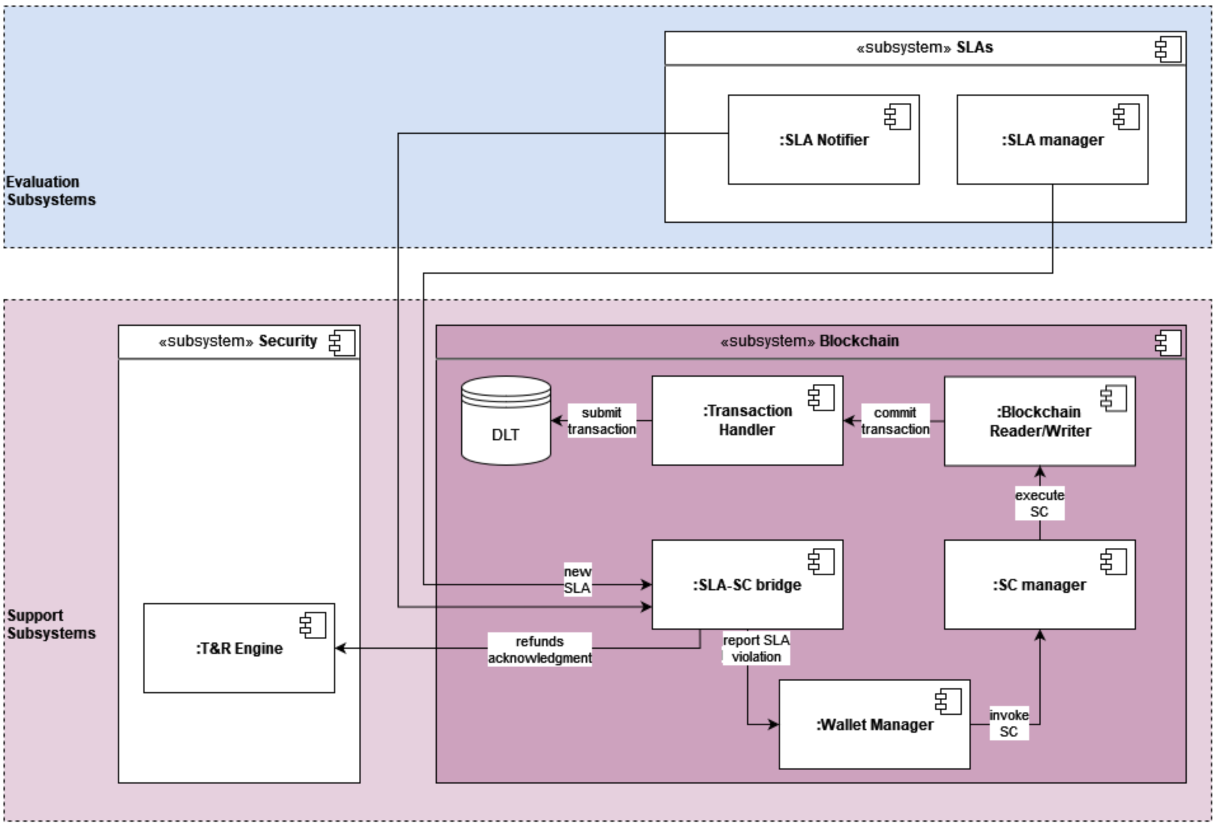


Figure 7: Blockchain subsystem Component Diagram

1.2.6 Big Data Platform and Communication subsystem

Big Data Platform & Communication subsystem: The Big Data & Communication Platform subsystem is the subsystem that exposes a high-performance distributed streaming platform for interconnecting, storing, transforming. It can efficiently ingest and handle massive amounts of data into processing pipelines, for both real-time and batch processing. The publish-subscribe mechanisms that this platform offers, are used in order to facilitate the integration of the different subsystems of the Pledger Core Platform. Producer applications can produce streams of data that are passed through the Big Data & Communication Platform and written into topics. Then, one or more consumers' applications may read data from topics. This way, the software components of the Pledge Core system may communicate with connections to the topics of the Big Data & Communication sub-system with a fault-tolerant way.

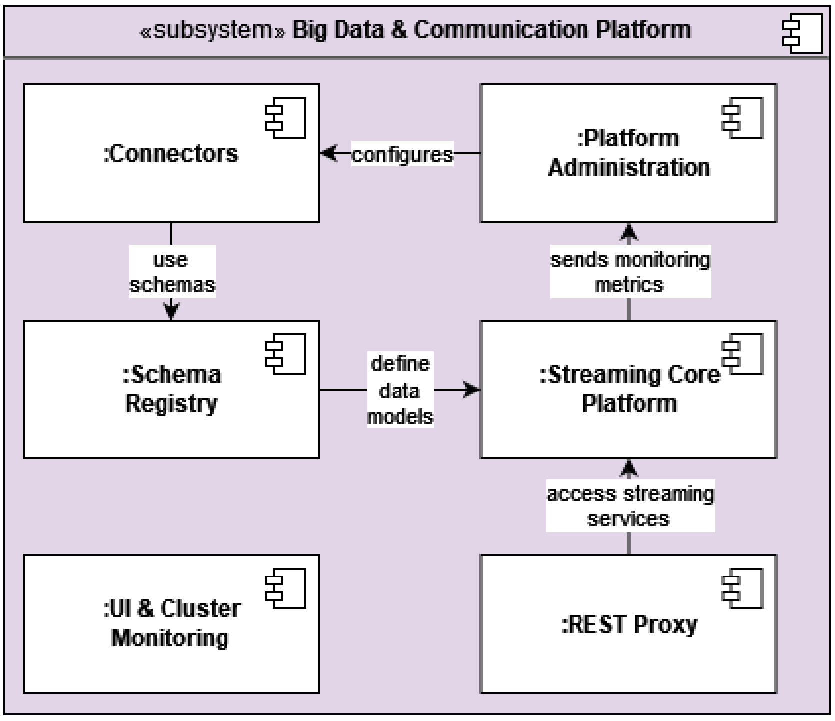


Figure 8: Big Data Platform subsystem Component Diagram

1.2.7 Security subsystem

Security subsystem: By using blockchains one can also mitigate several of the trust issues. However, a multi-layer approach is required to address security challenges at the edge. The Security subsystem includes components that enhance the security of the overall system as a whole. Of course, several other security features and mechanisms are included in other components of other subsystems, but this subsystem is specifically focused on the security aspects of Pledger.

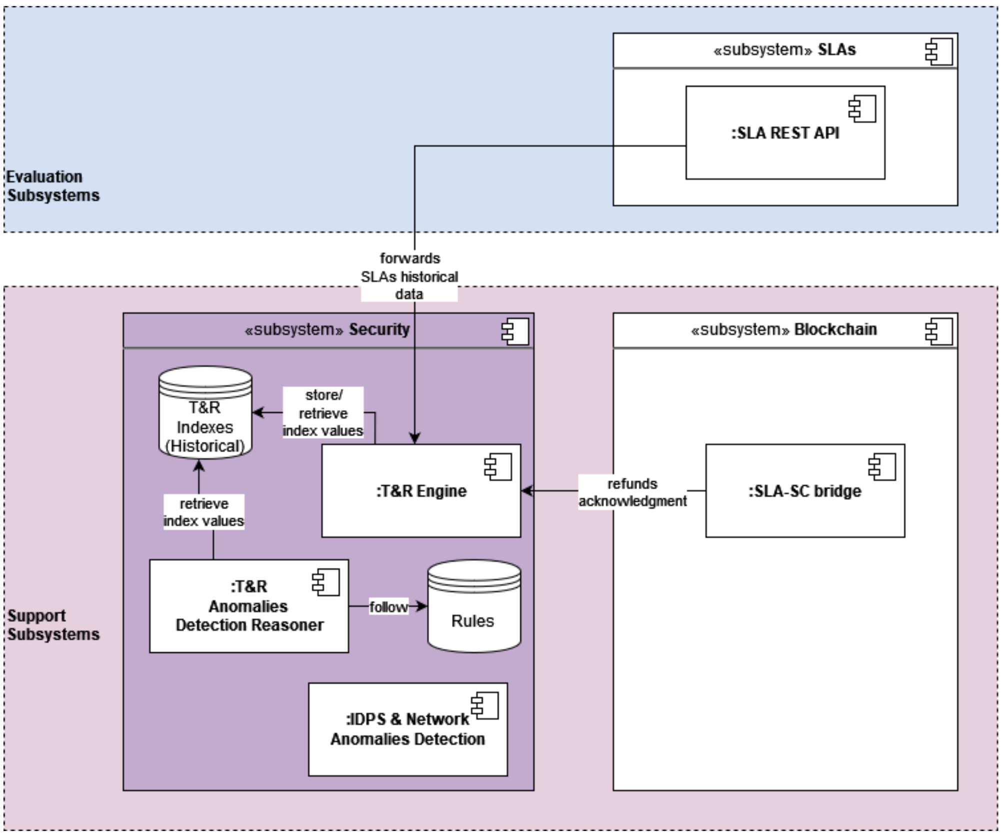


Figure 9: Security subsystem Component Diagram

2 PLEDGER Step by step

This step by step guide enables through specific steps the deployment and operation of all PLEDGER components. It contains all the important requirements and instructions in order to deploy an application through PLEDGER system, configure the intelligence modules (DSS, Benchmarking Suite and App Profiler) and finally the configuration of continues monitoring of SLA and automatic penalties operation.

2.1 Accessing the system

The **objective** of this section is to show the main system configuration is performed. The main component involved is ConfService.

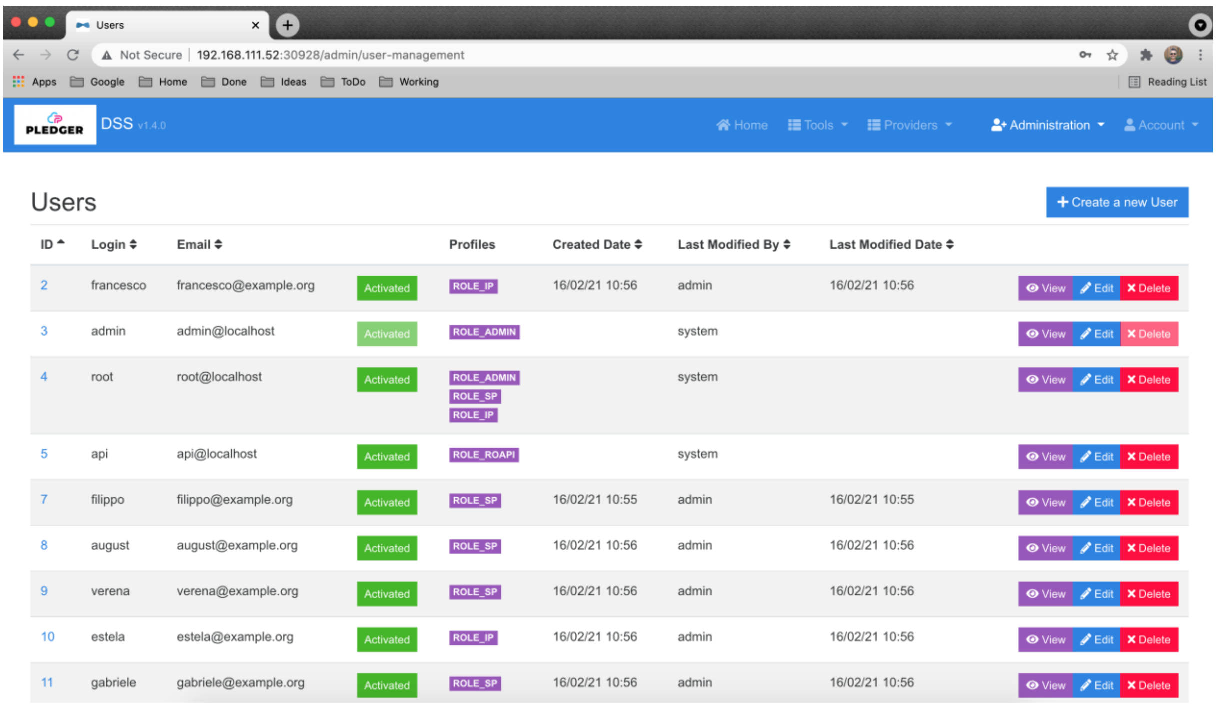
The main elements configured are:

1) users (service and infrastructure providers),
2) infrastructures and nodes,
3) Dapps.

As it is shown below, each step is executed by a **different role**.

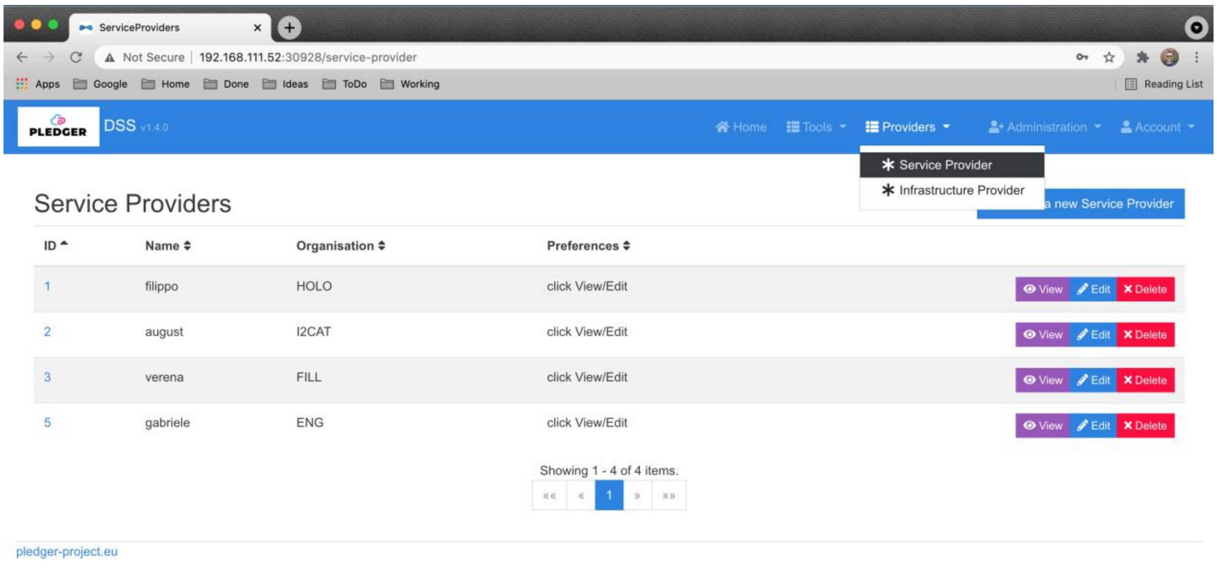
2.1.1 An admin configures service providers and infrastructure provider's users.

The Administrator is responsible for the creation of Pledger users and for the monitoring of authentication activities. Figure 10 shows the user dashboard where the Administrator can assign roles to users and enable/disable them. Figure 11 the list of Service provider users.



ID	Login	Email		Profiles	Created Date	Last Modified By	Last Modified Date	
2	francesco	francesco@example.org	Activated	ROLE_IP	16/02/21 10:56	admin	16/02/21 10:56	View Edit Delete
3	admin	admin@localhost	Activated	ROLE_ADMIN		system		View Edit Delete
4	root	root@localhost	Activated	ROLE_ADMIN ROLE_SP ROLE_IP		system		View Edit Delete
5	api	api@localhost	Activated	ROLE_ROAPI		system		View Edit Delete
7	filippo	filippo@example.org	Activated	ROLE_SP	16/02/21 10:55	admin	16/02/21 10:55	View Edit Delete
8	august	august@example.org	Activated	ROLE_SP	16/02/21 10:56	admin	16/02/21 10:56	View Edit Delete
9	verena	verena@example.org	Activated	ROLE_SP	16/02/21 10:56	admin	16/02/21 10:56	View Edit Delete
10	estela	estela@example.org	Activated	ROLE_IP	16/02/21 10:56	admin	16/02/21 10:56	View Edit Delete
11	gabriele	gabriele@example.org	Activated	ROLE_SP	16/02/21 10:56	admin	16/02/21 10:56	View Edit Delete

Figure 10: ConfService: users with roles



Service Providers

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

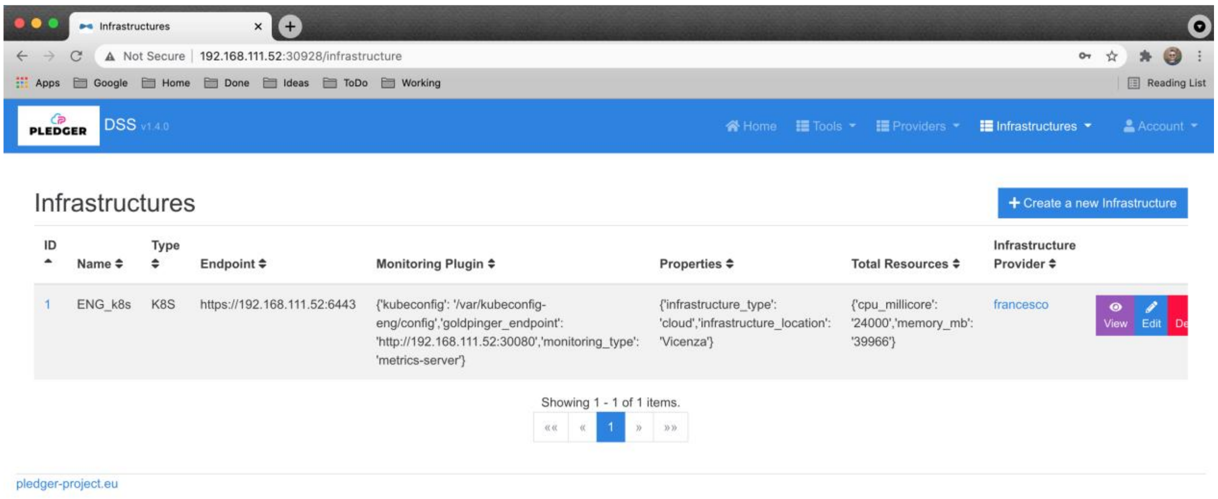
Showing 1 - 4 of 4 items.

pledger-project.eu

Figure 11: ConfService: definition of Pledger SP and IP users

2.1.2 An infrastructure provider access to the system to configure infrastructures.

The Infrastructure provides are responsible for the definition of the infrastructure and nodes in Pledger, along with the required configuration to allow its monitoring. Figure 12 shows the monitoring plugin configuration and Figure 13 shows the nodes with the hardware features automatically discovered by Pledger.



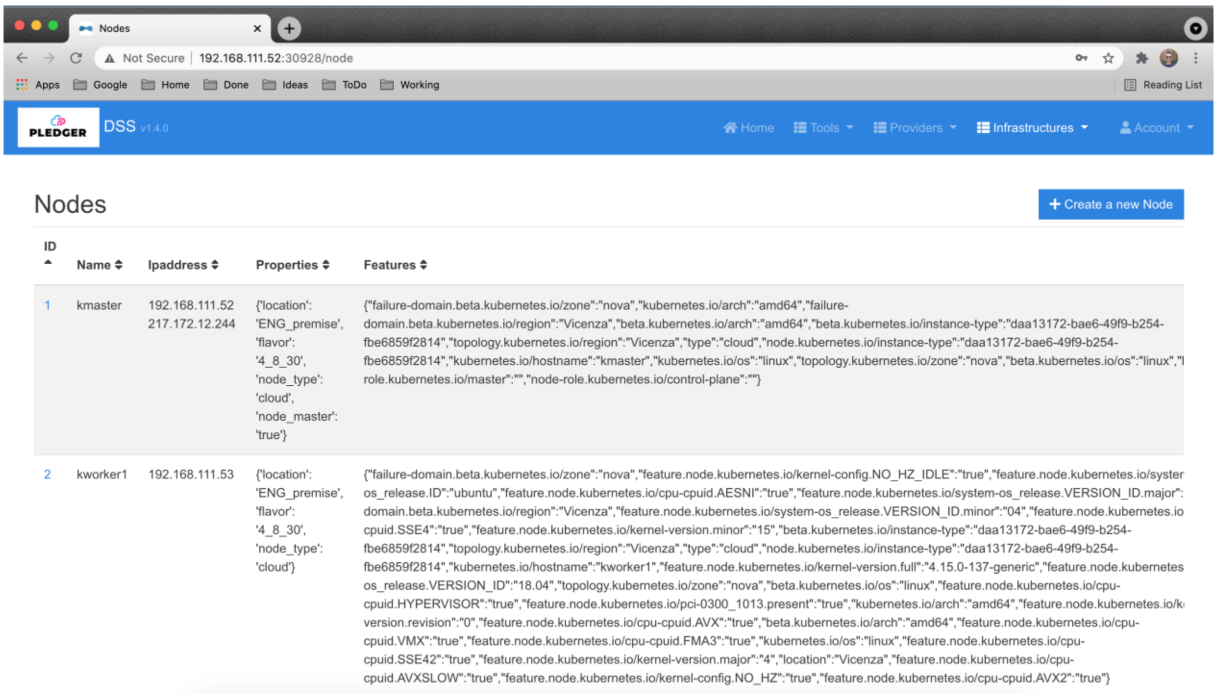
Infrastructures

ID	Name	Type	Endpoint	Monitoring Plugin	Properties	Total Resources	Infrastructure Provider	
1	ENG_k8s	K8S	https://192.168.111.52:6443	{'kubeconfig': '/var/kubeconfig-eng/config','goldpinger_endpoint': 'http://192.168.111.52:30080','monitoring_type': 'metrics-server'}	{'infrastructure_type': 'cloud','infrastructure_location': 'Vicenza'}	{'cpu_millicore': '24000','memory_mb': '39966'}	francesco	View Edit De

Showing 1 - 1 of 1 items.

pledger-project.eu

Figure 12: ConfService: infrastructure with monitoring plugin and resources capacity



Nodes
Not Secure | 192.168.111.52:30928/node
Apps Google Home Done Ideas ToDo Working
Reading List
PLEDGER
DSS v1.4.0
Home Tools Providers Infrastructures Account
Create a new Node

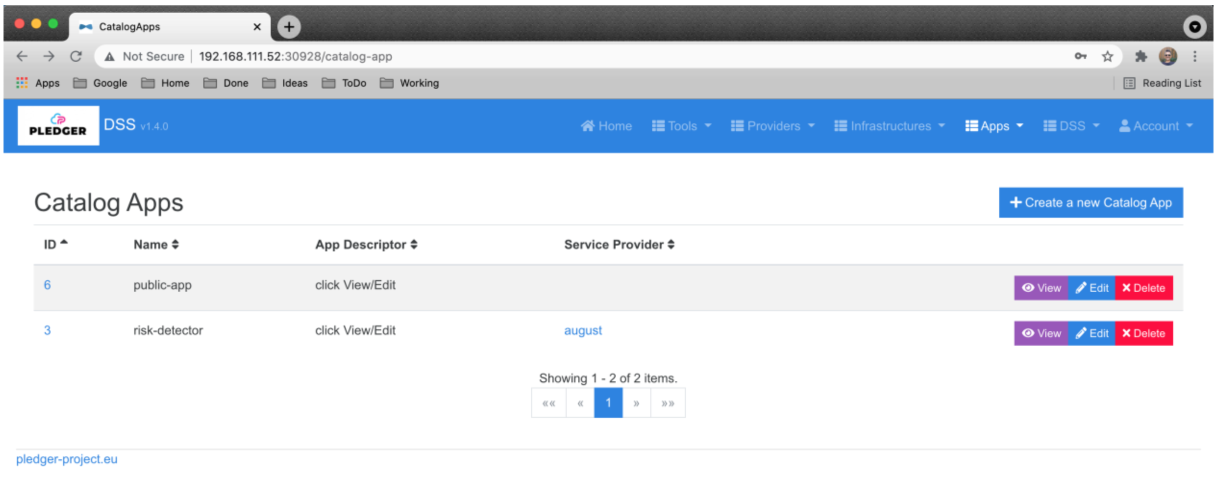
Nodes

ID	Name	Ipaddress	Properties	Features
1	kmaster	192.168.111.52 217.172.12.244	{'location': 'ENG_premise', 'flavor': '4_8_30', 'node_type': 'cloud', 'node_master': 'true'}	{"failure-domain.beta.kubernetes.io/zone":"nova","kubernetes.io/arch":"amd64","failure-domain.beta.kubernetes.io/region":"Vicenza","beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/instance-type":"daa13172-bae6-49f9-b254-fbe6859f2814","topology.kubernetes.io/region":"Vicenza","type":"cloud","node.kubernetes.io/instance-type":"daa13172-bae6-49f9-b254-fbe6859f2814","kubernetes.io/hostname":"kmaster","kubernetes.io/os":"linux","topology.kubernetes.io/zone":"nova","beta.kubernetes.io/os":"linux","I role.kubernetes.io/master":"","node-role.kubernetes.io/control-plane":""}
2	kworker1	192.168.111.53	{'location': 'ENG_premise', 'flavor': '4_8_30', 'node_type': 'cloud'}	{"failure-domain.beta.kubernetes.io/zone":"nova","feature.node.kubernetes.io/kernel-config.NO_HZ_IDLE":"true","feature.node.kubernetes.io/syster os_release.ID":"ubuntu","feature.node.kubernetes.io/cpu-cpuid.AESNI":"true","feature.node.kubernetes.io/system-os_release.VERSION_ID.major": domain.beta.kubernetes.io/region":"Vicenza","feature.node.kubernetes.io/system-os_release.VERSION_ID.minor":"04","feature.node.kubernetes.io cpuid.SSE4":"true","feature.node.kubernetes.io/kernel-version.minor":"15","beta.kubernetes.io/instance-type":"daa13172-bae6-49f9-b254-fbe6859f2814","topology.kubernetes.io/region":"Vicenza","type":"cloud","node.kubernetes.io/instance-type":"daa13172-bae6-49f9-b254-fbe6859f2814","kubernetes.io/hostname":"kworker1","feature.node.kubernetes.io/kernel-version.full":"4.15.0-137-generic","feature.node.kubernetes os_release.VERSION_ID":"18.04","topology.kubernetes.io/zone":"nova","beta.kubernetes.io/os":"linux","feature.node.kubernetes.io/cpu-cpuid.HYPERVISOR":"true","feature.node.kubernetes.io/pci-0300_1013.present":"true","kubernetes.io/arch":"amd64","feature.node.kubernetes.io/k version.revision":"0","feature.node.kubernetes.io/cpu-cpuid.AVX":"true","beta.kubernetes.io/arch":"amd64","feature.node.kubernetes.io/cpu-cpuid.VMX":"true","feature.node.kubernetes.io/cpu-cpuid.FMA3":"true","kubernetes.io/os":"linux","feature.node.kubernetes.io/cpu-cpuid.SSE42":"true","feature.node.kubernetes.io/kernel-version.major":"4","location":"Vicenza","feature.node.kubernetes.io/cpu-cpuid.AVXSLOW":"true","feature.node.kubernetes.io/kernel-config.NO_HZ":"true","feature.node.kubernetes.io/cpu-cpuid.AVX2":"true"}

Figure 13: ConfService: nodes with feature auto-discovery features working to identify HW availability

2.1.3 A service provider access to the system to configure DApps and deployment preferences

Service providers are responsible for the configuration of the DApps, their preferences about the deployment options to filter and prioritize the DSS options. Figure 14 and Figure 15, show the configuration of catalog apps and services.



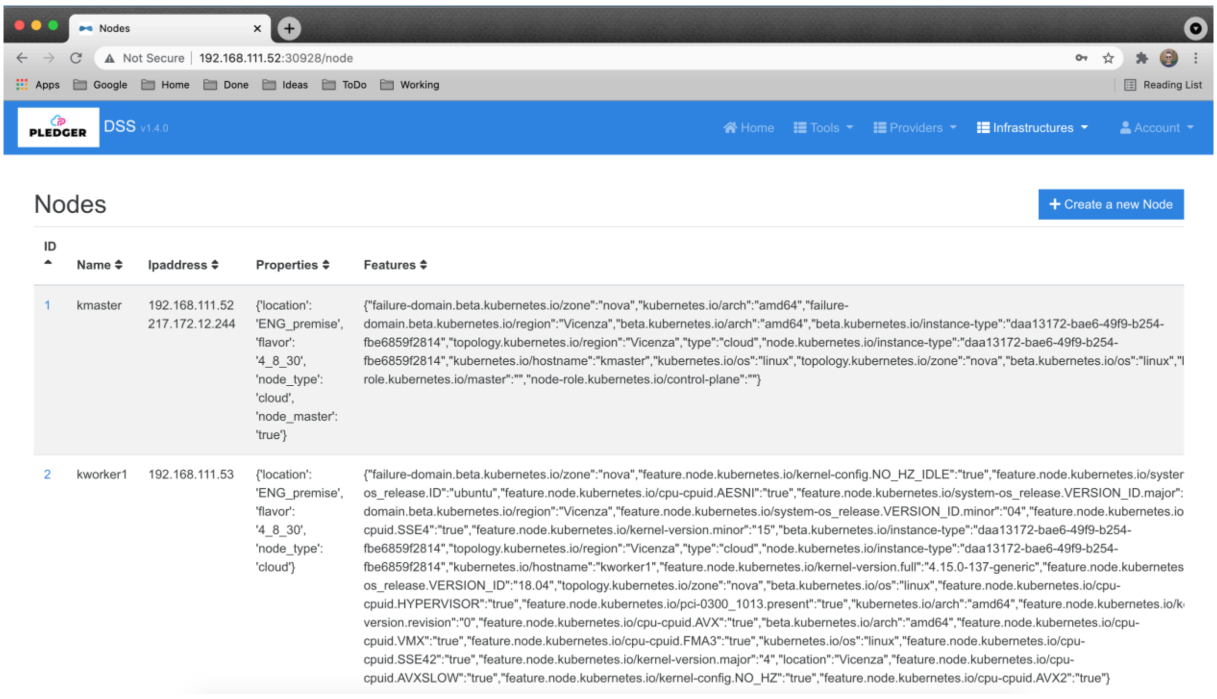
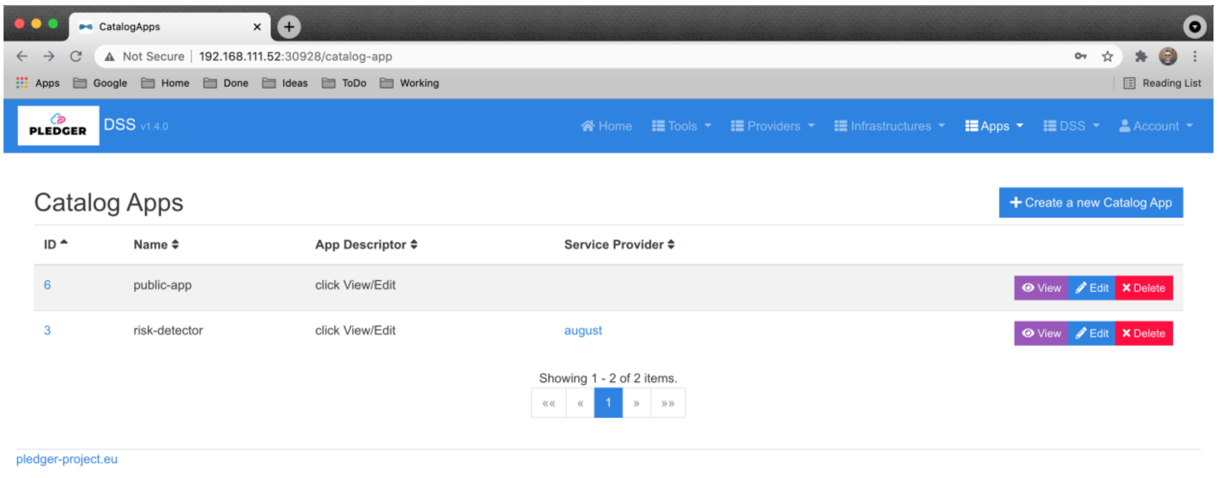
CatalogApps
Not Secure | 192.168.111.52:30928/catalog-app
Apps Google Home Done Ideas ToDo Working
Reading List
PLEDGER
DSS v1.4.0
Home Tools Providers Infrastructures Apps DSS Account
Create a new Catalog App
View Edit Delete
«« « 1 » »»
pledger-project.eu

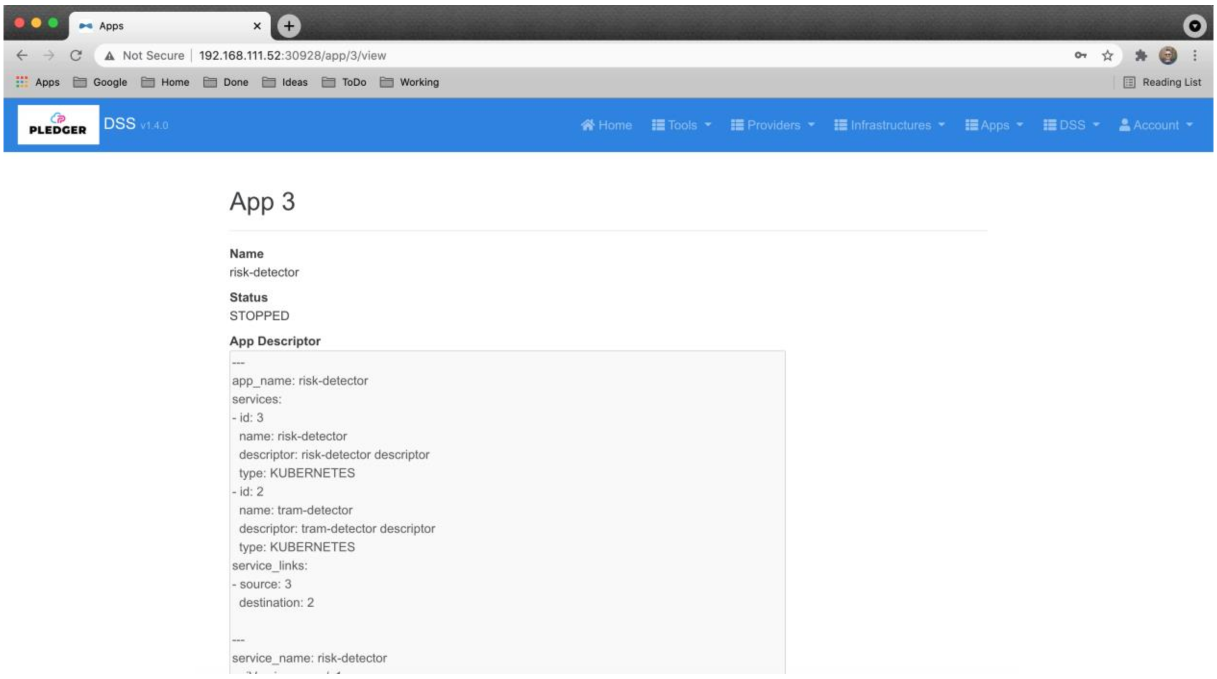
Catalog Apps

ID	Name	App Descriptor	Service Provider
6	public-app	click View/Edit	
3	risk-detector	click View/Edit	august

Showing 1 - 2 of 2 items.

Figure 14: ConfService: catalog apps



App 3

Name
risk-detector

Status
STOPPED

App Descriptor

```
---
app_name: risk-detector
services:
- id: 3
  name: risk-detector
  descriptor: risk-detector descriptor
  type: KUBERNETES
- id: 2
  name: tram-detector
  descriptor: tram-detector descriptor
  type: KUBERNETES
service_links:
- source: 3
  destination: 2

---
service_name: risk-detector
```

Figure 15: ConfService: DApp composed by multiple services with K8S descriptor

Table 1 reports the demos on YouTube about the configuration done by administrators, infrastructure and service provides.

Title	Link
ConfService_#1: Admin configuring users	https://www.youtube.com/watch?v=SQ2uZOrXtDs
ConfService_#2: IP configuring infrastructures	https://www.youtube.com/watch?v=wR5Job68AXU
ConfService_#3: SP configuring Apps	https://www.youtube.com/watch?v=V609jhGZw2Q

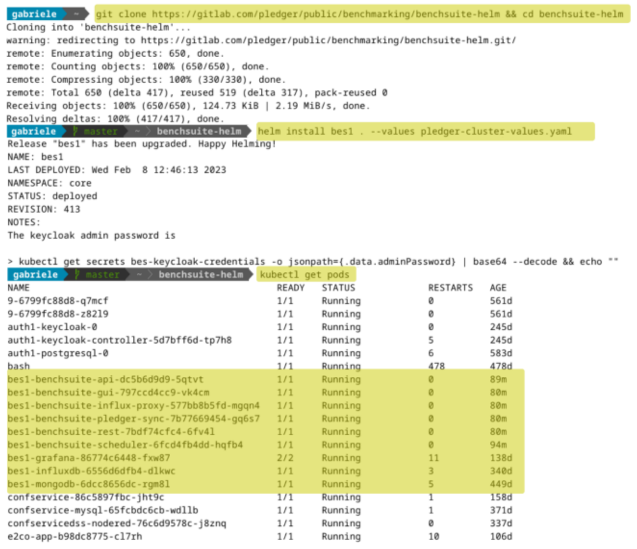
Table 1: ConfService demos on YouTube

2.2 Infrastructure benchmarking

In order to collect and analyse the performance of the infrastructures managed by a Pledger system, the Benchmarking Suite service is needed. It is a service that runs in the background, reads the configuration of the infrastructures from the ConfService, automatically execute the tests and analyse the results and finally, publishes performance reports in the DSS. In the basic scenario, the interaction with the user is minimal (section **Error! Reference source not found.**2.2.1) and all operations (sections 2.2.2, 2.2.3) are automated. However, it is possible to customize the behaviour of the Benchmarking Suite (section 2.2.4) to have more precise and reliable results.

The Benchmarking Suite is an open-source software produced in Pledger and available on the Pledger source code repository at: https://gitlab.com/pledger/public/benchmarking. The deployment of the software has been extremely simplified thanks to the implementation of an Helm Chart that automates the deployment and the configuration of the software in Kubernetes clusters. For most of the cases, it is enough to download the chart package (or download its source code) and execute the "helm install" command (Figure 16). It will automatically download and start all the required containers. A mandatory configuration that must be provided through the "Helm values" mechanism is the list of the endpoints for the ConfService, the DSS and the StreamHandler to allow the Benchmarking Suite to communicate with the other Pledger components.

The Helm Chart source code and more detailed installation and configuration instructions are available here: https://gitlab.com/pledger/public/benchmarking/benchsuite-helm.



```
gabriele  ~  git clone https://gitlab.com/pledger/public/benchmarking/benchsuite-helm && cd benchsuite-helm
Cloning into 'benchsuite-helm'...
warning: redirecting to https://gitlab.com/pledger/public/benchmarking/benchsuite-helm.git/
remote: Enumerating objects: 650, done.
remote: Counting objects: 100% (650/650), done.
remote: Compressing objects: 100% (330/330), done.
remote: Total 650 (delta 417), reused 519 (delta 317), pack-reused 0
Receiving objects: 100% (650/650), 124.73 KiB | 2.19 MiB/s, done.
Resolving deltas: 100% (417/417), done.
gabriele  master  ~  benchsuite-helm  helm install bes1 . --values pledger-cluster-values.yaml
Release "bes1" has been upgraded. Happy Helming!
NAME: bes1
LAST DEPLOYED: Wed Feb  8 12:46:13 2023
NAMESPACE: core
STATUS: deployed
REVISION: 413
NOTES:
The keycloak admin password is

> kubectl get secrets bes-keycloak-credentials -o jsonpath={.data.adminPassword} | base64 --decode && echo ""
gabriele  master  ~  benchsuite-helm  kubectl get pods
NAME                                            READY   STATUS      RESTARTS   AGE
9-6799fc88d8-q7mcf                              1/1     Running     0          561d
9-6799fc88d8-z8219                              1/1     Running     0          561d
auth1-keycloak-0                                1/1     Running     0          245d
auth1-keycloak-controller-5d7bff6d-tp7h8        1/1     Running     5          245d
auth1-postgresql-0                              1/1     Running     6          583d
bash                                            1/1     Running     478        478d
bes1-benchsuite-api-dc5b6d9d9-5qtvt             1/1     Running     0          89m
bes1-benchsuite-gui-797ccd4cc9-vk4cm            1/1     Running     0          80m
bes1-benchsuite-influx-proxy-577bb8b5fd-mgqn4   1/1     Running     0          80m
bes1-benchsuite-pledger-sync-7b77669454-gq6s7   1/1     Running     0          80m
bes1-benchsuite-rest-7bdf74cfc4-6fv4l           1/1     Running     0          80m
bes1-benchsuite-scheduler-6fcd4fb4dd-hqfb4      1/1     Running     0          94m
bes1-grafana-86774c6448-fxw87                   2/2     Running     11         138d
bes1-influxdb-6556d6dfb4-dlkwc                  1/1     Running     3          340d
bes1-mongodb-6dcc8656dc-rgm8l                   1/1     Running     5          449d
confservice-86c5897fbc-jht9c                    1/1     Running     1          158d
confservice-mysql-65fcbdc6cb-wdllb              1/1     Running     1          371d
confservicedss-nodered-76c6d9578c-j8znq         1/1     Running     0          337d
e2co-app-b98dc8775-cl7rh                        1/1     Running     10         106d
```

Figure 16: Benchmarking Suite installation steps

The full admin and user documentation of the Benchmarking Suite is available online at: https://benchmarking-suite.readthedocs.io/en/latest/.

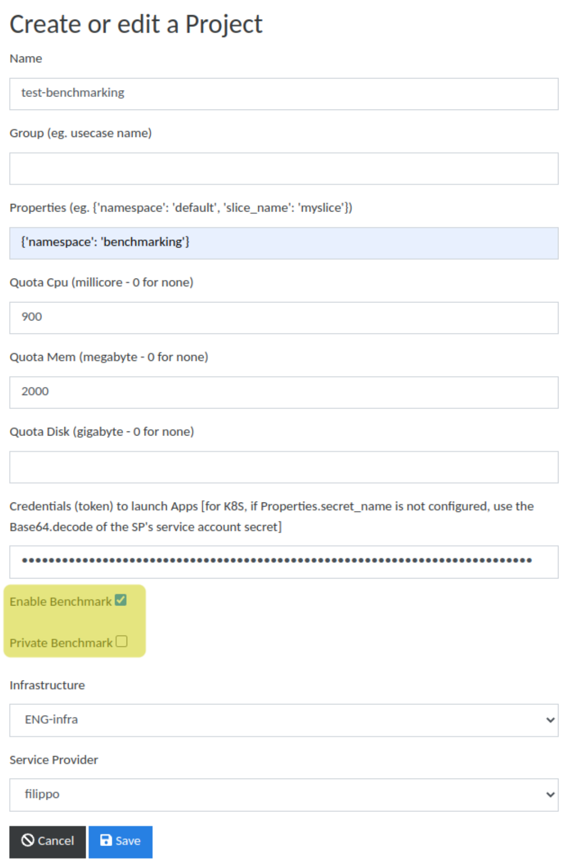
In addition, two relevant videos have been published on the usage of the Benchmarking Suite:

- https://www.youtube.com/watch?v=1R6QYYjn6OM
- https://www.youtube.com/watch?v=oJGZCa8SUUI

2.2.1 Enable benchmarking of a specific infrastructure.

To let the Benchmarking Suite start to collect performance data for an infrastructure, it is necessary to enable the benchmarking of the infrastructure in the ConfService. This operation can be executed only after an infrastructure has been added to the system (see section 2.1.2). This functionality is available both for infrastructure providers and service providers. Infrastructure providers can activate benchmarking on their infrastructure with the objectives of collecting data to assess and tune the infrastructure or to offer performance data publicly to the service providers that might want to deploy applications on their infrastructures. Service providers, on the other hand, can activate benchmarking on the infrastructures they want to utilize to optimize the deployments of their applications .

To enable benchmarking, the user should connect to the ConfService and create a new "Project" entity by selecting "Apps → Projects → Create a new Project" from the main navigation menu. In the form (Figure 17), the user needs to specify the target infrastructure and the credentials to access the resources and he/she needs to the "Enable Benchmarking" option. In addition, it is possible to specify limits for the usage of resources and whether the calculated performance data can be public or must remain private.



Create or edit a Project

Name: test-benchmarking

Group (eg. usecase name):

Properties (eg. {'namespace': 'default', 'slice_name': 'myslice'}): {'namespace': 'benchmarking'}

Quota Cpu (millicore - 0 for none): 900

Quota Mem (megabyte - 0 for none): 2000

Quota Disk (gigabyte - 0 for none):

Credentials (token) to launch Apps [for K8S, if Properties.secret_name is not configured, use the Base64.decode of the SP's service account secret]: ••

Enable Benchmark ☑

Private Benchmark ☐

Infrastructure: ENG-infra

Service Provider: filippo

Figure 17: ConfService Project Creation and benchmarking enabling

Instead of creating a new "Project" specific for the benchmarking of the infrastructure, it is also possible to enable benchmarking on the Project that the user previously created to deploy an application. The first approach, however, has the advantage that it is possible to specify benchmarking-specific credentials and configuration (e.g., restricting the resources usable or the privileges on the infrastructure).

2.2.2 Benchmarking Suite is notified and starts the execution of benchmarking tests.

The Benchmarking Suite is notified when the user configuration changes in the ConfService. In the case of the activation of benchmarking for a given infrastructure (see previous section), the notification trigger an automatic re-configuration of the Benchmarking Suite that schedule the execution of a pre-defined set of benchmarking tests in the infrastructure. The tests will start automatically using the credentials and infrastructure parameters specified in the ConfService. The Benchmarking Suite will collect the performance data measured on the infrastructure during the execution of tests and will analyse this data (aggregating and comparing the data with previous data – if available – for the same infrastructure). The diagram in Figure 18 shows the sequence of steps executed that triggered by the enabling of the benchmarking by the user.

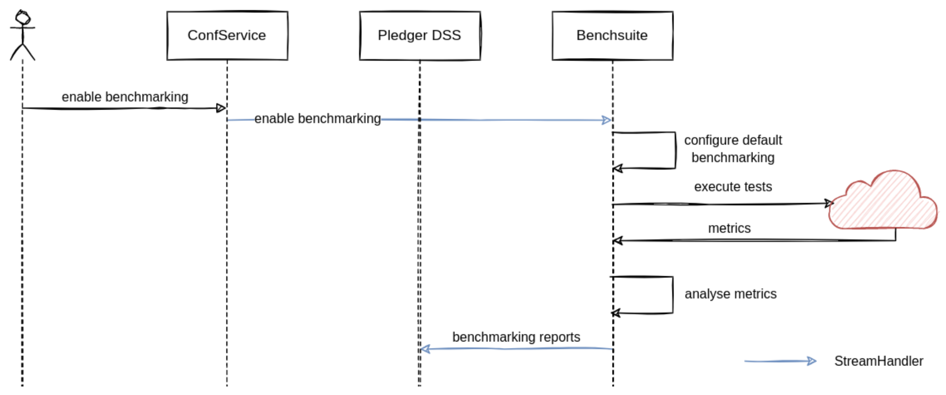
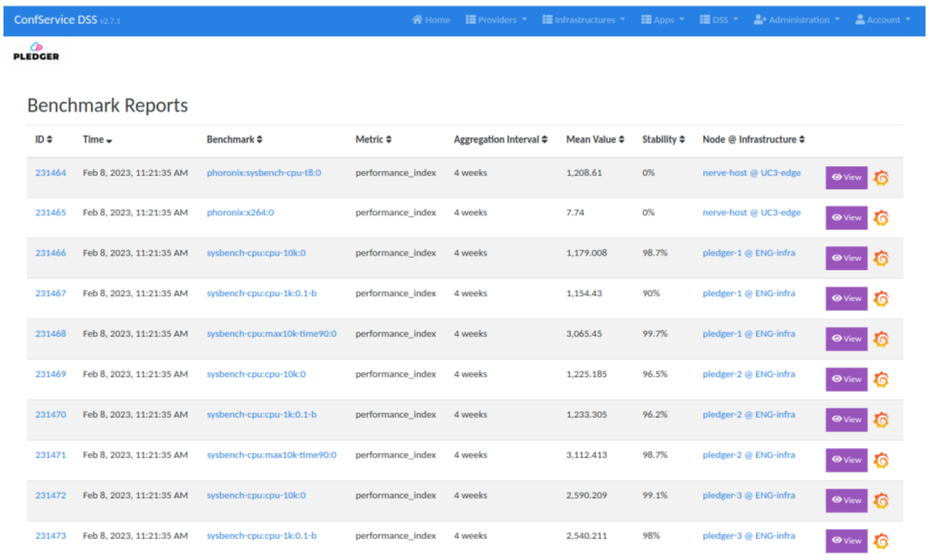


Figure 18: Benchmarking execution sequence diagram

It is worth to note that, once enabled, benchmarking of an infrastructure will stay on and tests will be repeated on regular basis (once a week, by default). This guarantees to have more reliable and stable performance data.

2.2.3 Report is shared with the Decision Support System (DSS).

As last step, in Figure 18, the Benchmarking Suite will send a report on the performance of the infrastructure to the DSS in a machine-readable way. This is repeated every time a new test is executed. The reports (shown in Figure 19) includes also a stability indicator that express how much this value changed from the past. The data is later used by the DSS (together with the data from the AppProfiler) to suggest the best infrastructure and node to deploy a given application.



Benchmark Reports

ID	Time	Benchmark	Metric	Aggregation Interval	Mean Value	Stability	Node @ Infrastructure	
231464	Feb 8, 2023, 11:21:35 AM	phoronix:sysbench-cpu-t8:0	performance_index	4 weeks	1,208.61	0%	nerve-host @ UC3-edge	View
231465	Feb 8, 2023, 11:21:35 AM	phoronix:x264:0	performance_index	4 weeks	7.74	0%	nerve-host @ UC3-edge	View
231466	Feb 8, 2023, 11:21:35 AM	sysbench-cpu:cpu-10k:0	performance_index	4 weeks	1,179.008	98.7%	pledger-1 @ ENG-infra	View
231467	Feb 8, 2023, 11:21:35 AM	sysbench-cpu:cpu-1k:0.1-b	performance_index	4 weeks	1,154.43	90%	pledger-1 @ ENG-infra	View
231468	Feb 8, 2023, 11:21:35 AM	sysbench-cpu:max10k-time90:0	performance_index	4 weeks	3,065.45	99.7%	pledger-1 @ ENG-infra	View
231469	Feb 8, 2023, 11:21:35 AM	sysbench-cpu:cpu-10k:0	performance_index	4 weeks	1,225.185	96.5%	pledger-2 @ ENG-infra	View
231470	Feb 8, 2023, 11:21:35 AM	sysbench-cpu:cpu-1k:0.1-b	performance_index	4 weeks	1,233.305	96.2%	pledger-2 @ ENG-infra	View
231471	Feb 8, 2023, 11:21:35 AM	sysbench-cpu:max10k-time90:0	performance_index	4 weeks	3,112.413	98.7%	pledger-2 @ ENG-infra	View
231472	Feb 8, 2023, 11:21:35 AM	sysbench-cpu:cpu-10k:0	performance_index	4 weeks	2,590.209	99.1%	pledger-3 @ ENG-infra	View
231473	Feb 8, 2023, 11:21:35 AM	sysbench-cpu:cpu-1k:0.1-b	performance_index	4 weeks	2,540.211	98%	pledger-3 @ ENG-infra	View

Figure 19: Benchmarking Reports stored in the DSS

2.2.4 Manual tuning of benchmarking tests and analysis of results

The Benchmarking Suite deployment also includes a web portal GUI that can be used in advanced use cases to:

change the default list of benchmarking tests to execute and change their configuration

define new benchmarking tests

customize the scheduling (e.g., the execution frequency) of the tests

see raw and calculated performance metrics for each execution.

For a full description and guide of all functionalities, there is an online guide at: The full admin and user documentation of the Benchmarking Suite is available online at: https://benchmarking-suite.readthedocs.io/en/latest/.

2.3 Application profiling

The objective of this section is to show how the PLEDGER Application Profiler is used in order to provide helpful information to the DSS about applications that are going to be deployed. To achieve this, it communicates the prediction result to the ConfService. The app profiler is a standalone application but in the PLEDGER ecosystem it communicates with the other PLEDGER components through Kafka.

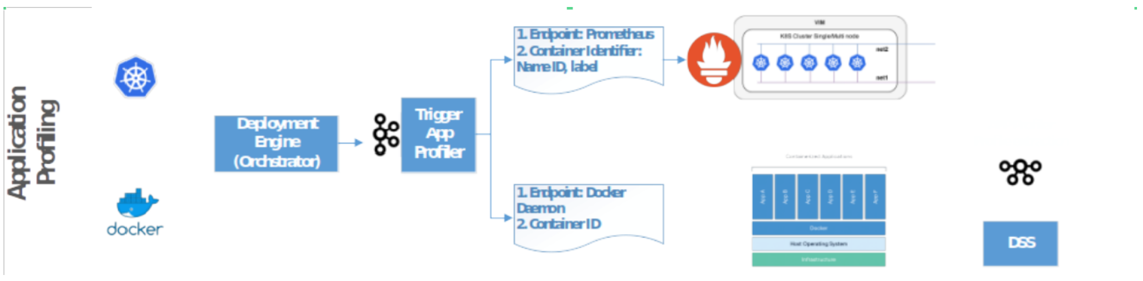
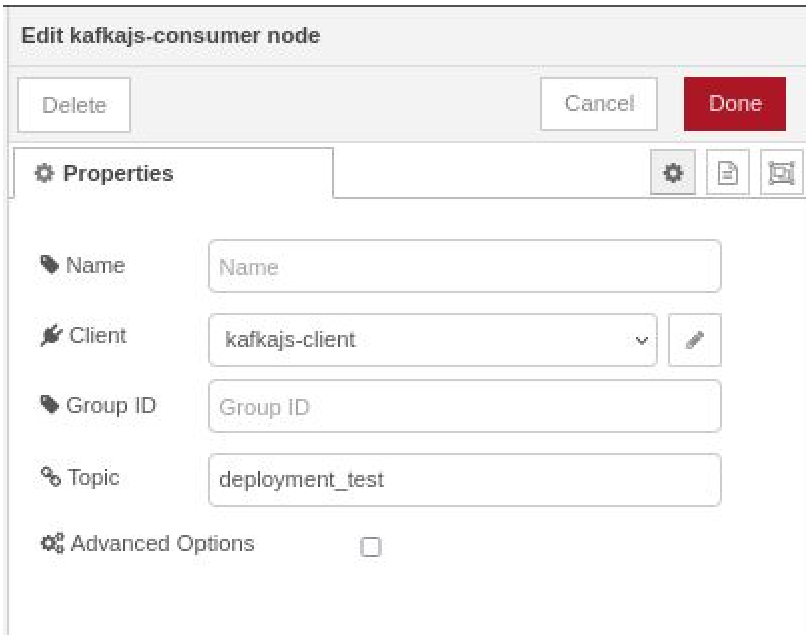


Figure 20: Application profiling diagram

2.3.1The App Profiler is automatically started when Application is deployed.

The App Profiler is constantly listening to the Kafka topic that sends all the deployments information. This topic must be set in “Kafka flow”, “kafkajs-consumer node” as seen in Figure 21. Thus, every time a new application is deployed, the Profiler will be triggered and the profiling will start automatically. More specifically, the ConfService sends all the needed information so that the profiler can get access to the low-level metrics of the newly deployed application. The configuration that is needed pertains to the Kafka consumer and producer nodes and the corresponding paths to the trustore and keystore files as depicted in Figure 22 and is described in the following url:

https://gitlab.com/pledger/public/app-profiler#kafka

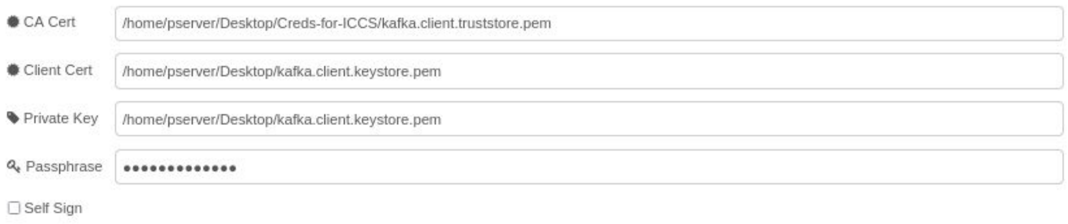

Edit kafkajs-consumer node

Delete | Cancel | Done

Properties

Name	Name
Client	kafkajs-client
Group ID	Group ID
Topic	deployment_test
Advanced Options	☐

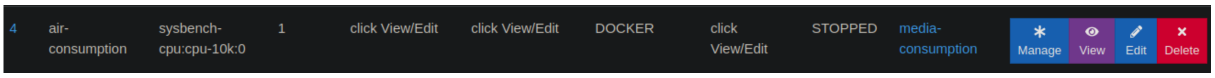
Figure 21: kafkajs-consumer node properties



● CA Cert	/home/pserver/Desktop/Creds-for-ICCS/kafka.client.truststore.pem
● Client Cert	/home/pserver/Desktop/kafka.client.keystore.pem
Private Key	/home/pserver/Desktop/kafka.client.keystore.pem
Passphrase	•••••••••••••
☐ Self Sign	

Figure 22: kafkajs-consumer trustosre and keystore paths configuration

After calculating a profile prediction, the App Profiler informs the ConfService about the result, by tagging the running service to which the prediction corresponds as seen in Figure 23



4	air-consumption	sysbench-cpu:cpu-10k:0	1	click View/Edit	click View/Edit	DOCKER	click View/Edit	STOPPED	media-consumption	Manage View Edit Delete

Figure 23: Application service tagged with sysbench prediction

2.3.2 Resource usage data is collected and analysed.

After the profiling process has been triggered, the collection of the required data starts. This operation can be done either through the monitoring engine by querying the Prometheus database, or by collecting the metrics in real time from the running container with the usage of the Docker API. The default process is the first one and is performed automatically as already described in 2.3.1. However, the App Profiler can also run independently of the PLEDGER system and by providing the exact location of a running Docker container it can start collecting and analyzing its performance metrics. The different options can be utilized through the App Profiler REST API as fully detailed described in its documentation on GitLab in the following url:

https://gitlab.com/pledger/public/app-profiler

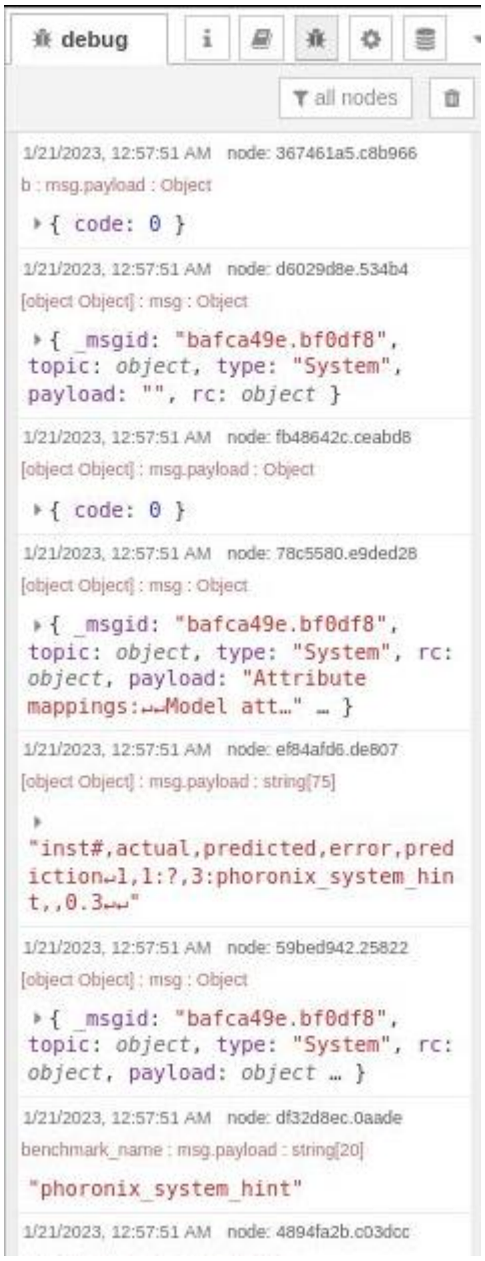
A simple user interface is also available as seen in :

https://gitlab.com/pledger/public/app-profiler#profiler-ui

2.3.1 App Profiler composes a profile vector and classifies the application.

When the analysing of the collected data has finished, a profile vector has been created which is fed to the profiler prediction model. This prediction will be transmitted automatically to the ConfService but can also be inspected in the debug logs of the Node-RED platform. This can be seen in the last debug message in Figure 24. In case the metrics have already been collected in the past and have been stored in a csv file, a prediction can again be calculated by setting the appropriate path to the input file as it is described in the following url:

https://gitlab.com/pledger/public/app-profiler#random-forest-weka-classifier



```
1/21/2023, 12:57:51 AM   node: 367461a5.c8b966
b : msg.payload : Object
▸{ code: 0 }

1/21/2023, 12:57:51 AM   node: d6029d8e.534b4
[object Object] : msg : Object
▸{ _msgid: "bafca49e.bf0df8",
topic: object, type: "System",
payload: "", rc: object }

1/21/2023, 12:57:51 AM   node: fb48642c.ceabd8
[object Object] : msg.payload : Object
▸{ code: 0 }

1/21/2023, 12:57:51 AM   node: 78c5580.e9ded28
[object Object] : msg : Object
▸{ _msgid: "bafca49e.bf0df8",
topic: object, type: "System", rc:
object, payload: "Attribute
mappings:↵↵Model att…" … }

1/21/2023, 12:57:51 AM   node: ef84afd6.de807
[object Object] : msg.payload : string[75]
▸
"inst#,actual,predicted,error,pred
iction↵1,1:?,3:phoronix_system_hin
t,,0.3↵↵"

1/21/2023, 12:57:51 AM   node: 59bed942.25822
[object Object] : msg : Object
▸{ _msgid: "bafca49e.bf0df8",
topic: object, type: "System", rc:
object, payload: object … }

1/21/2023, 12:57:51 AM   node: df32d8ec.0aade
benchmark_name : msg.payload : string[20]
"phoronix_system_hint"

1/21/2023, 12:57:51 AM   node: 4894fa2b.c03dcc
```

Figure 24: Node-RED debug console

2.3.2 Classification results are shared with the DSS.

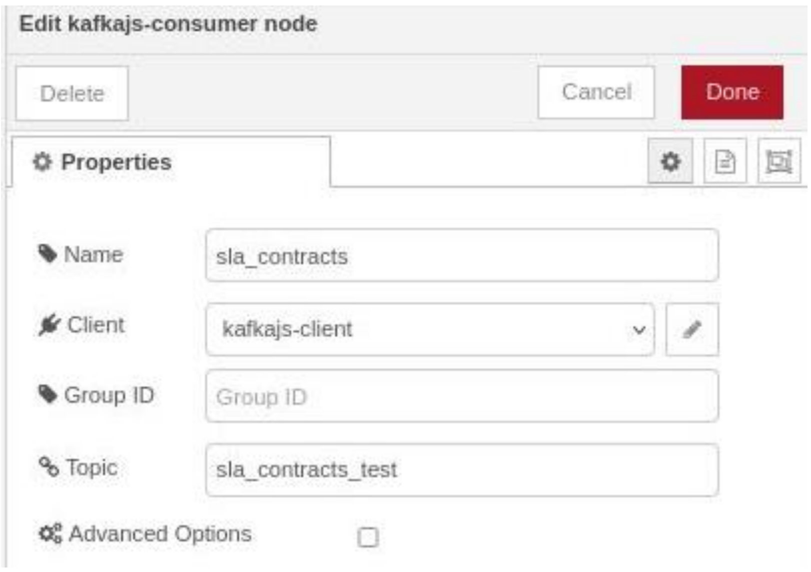
When the classification has finished, the result must be shared with the DSS. To achieve this, the prediction is automatically transmitted to the ConfService . This results in tagging the corresponding service with the actual prediction as depicted in Figure 23. Again, the appropriate Kafka topic must be set in the producer node. From there on, the DSS can make use of the information of the prediction whenever it is needed in order to support its deployment decisions.

2.4 Provider selection

Infrastructure providers for the deployment of new applications are selected based on their past behaviour. Therefore, the T&R engine ranks the available providers taking into account the individual Trust that has been developed through interactions with a specific client, but also the community Reputation which takes into account the experiences of all clients with the specific provider. The results of this analysis are presented to the client so that he can choose the best available provider.

2.4.1 T&R engine constantly analyses the SLA violations information for each of the infrastructure providers.

The T&R engine listens to the Kafka topic that communicates all the new SLA contracts as well as the information about every SLA violation for active SLA contracts that occurs. All this knowledge is stored in a database and the Trust and Reputation indexes are constantly updated based on it. The engine is implemented in node-RED flows, thus the appropriate Kafka topic must be set in the corresponding kafkajs-consumer node as depicted in Figure 25



Name	sla_contracts
Client	kafkajs-client
Group ID	Group ID
Topic	sla_contracts_test
Advanced Options	☐

Figure 25: kafkajs-consumer node properties

2.4.2 It ranks the available provides and presents them through a user interface to the client so that he can select the most suitable one for deploying a specific application.

By accessing the T&R engine UI in the following url:

{Node-RED address}/ui

the user can select to be presented either with the top 10 providers ranked by reputation, or to ask for the Trust or Reputation indexes for specific clients and providers. This is shown in Figure 26, Figure 27 and Figure 28

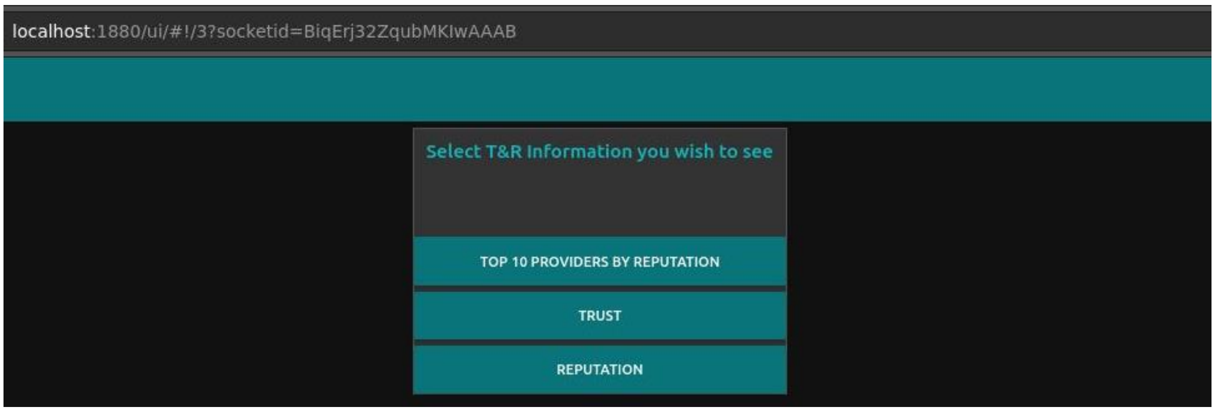

localhost:1880/ui/#!/3?socketid=BiqErj32ZqubMKIwAAAB
Select T&R Information you wish to see
TOP 10 PROVIDERS BY REPUTATION
TRUST
REPUTATION

Figure 26: Basic T&R menu

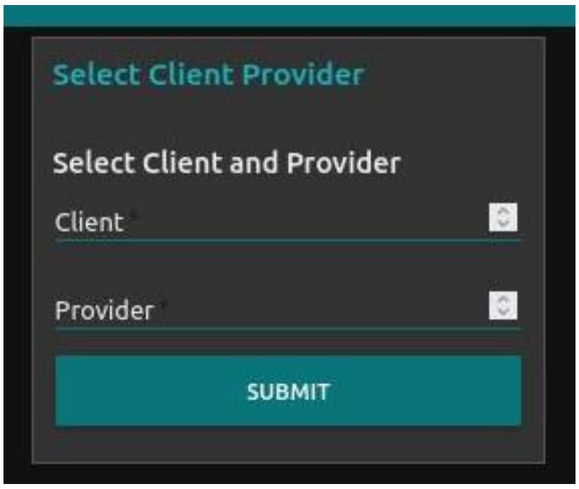


Figure 27: Trust querying form

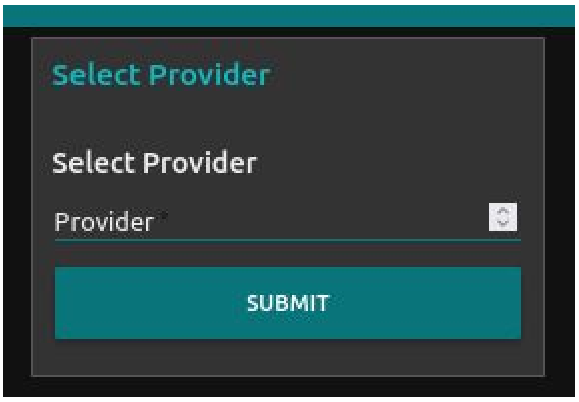


Figure 28: Reputation querying form

The form of the calculated results can be seen in Figure 29 and Figure 30

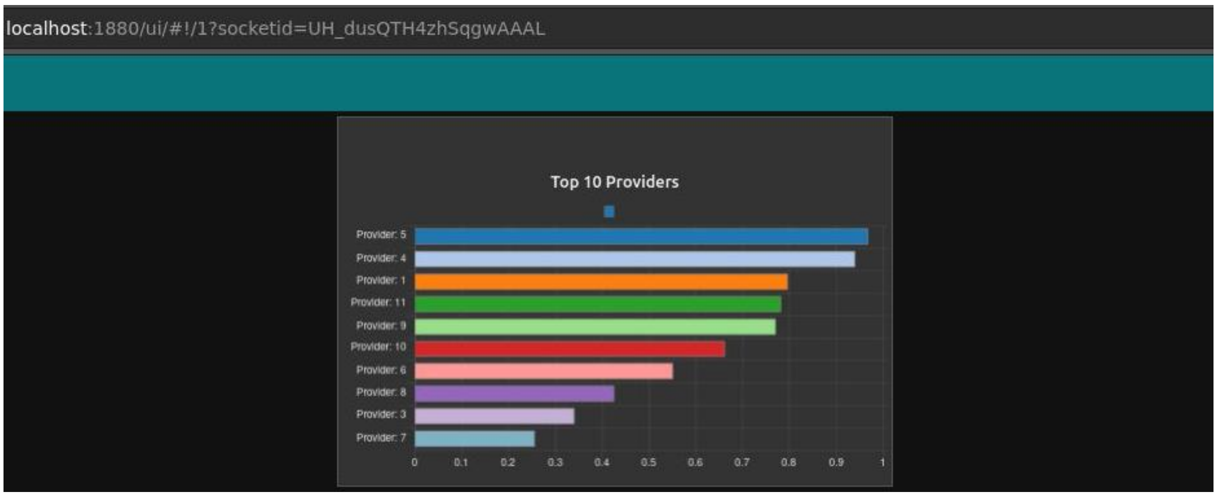


Figure 29: Top 10 providers by reputation

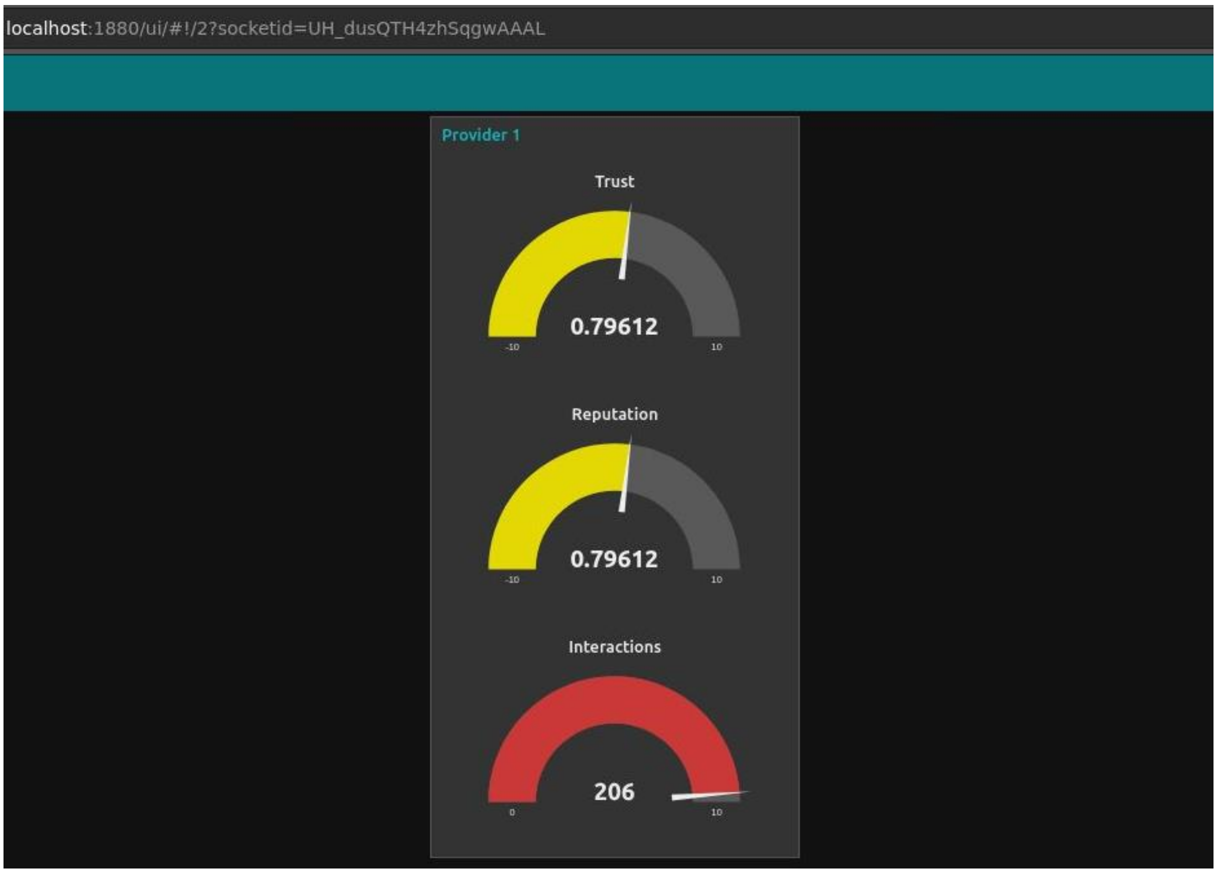


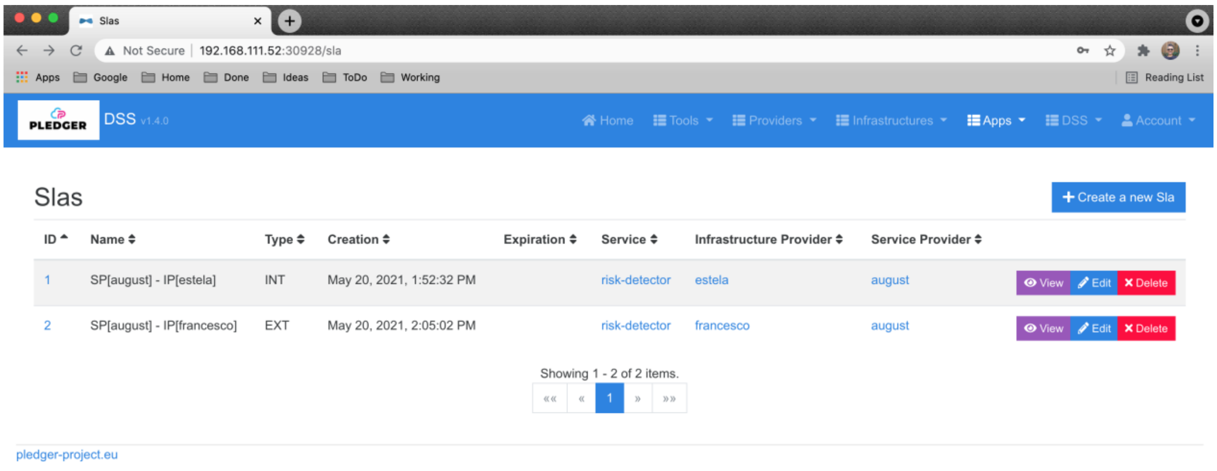
Figure 30: Client-provider trust and reputation indexes

2.5 Configuring QoS

The **objective** of this section is to show how SLA are configured in the system. The components involved are the ConfService and the SLA Lite.

2.5.1 A service provider states the guarantees for an application.

Figure 31, Figure 32, Figure 33 and Figure 34 show the configuration of the SLA and Guarantees happening on the ConfService UI which, in turn, sends notifications to the SLA Lite to sync configuration data. First, the SLA has to be created. During the creation the SLA is associated to a service. Then the guarantees terms must be created. These guarantee terms have to be associated to an existing SLA.


Slas
Not Secure | 192.168.111.52:30928/sla
Apps Google Home Done Ideas ToDo Working Reading List
PLEDGER DSS v1.4.0
Home Tools Providers Infrastructures Apps DSS Account
Create a new Sla
View Edit Delete

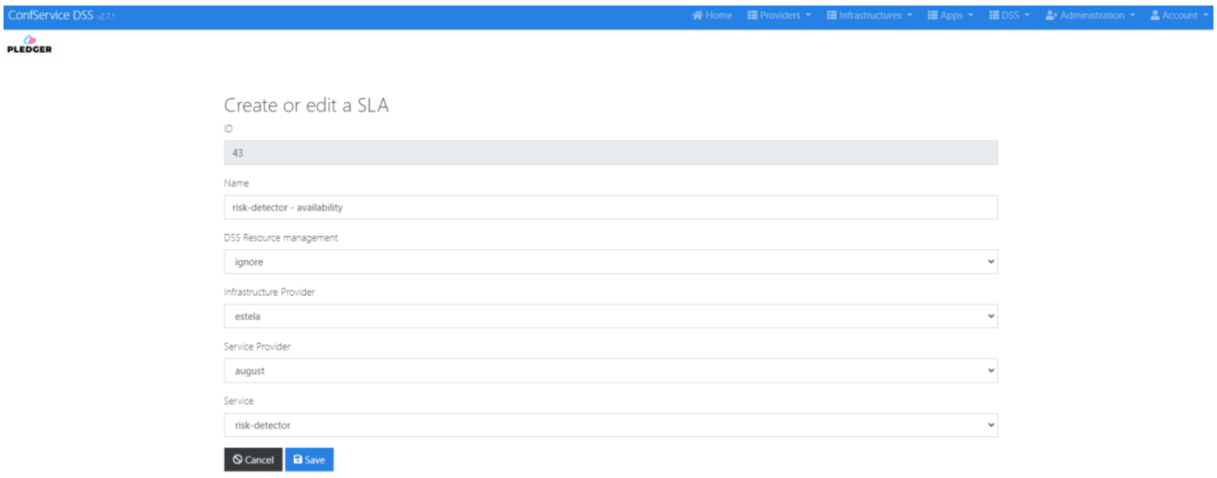
Slas

ID	Name	Type	Creation	Expiration	Service	Infrastructure Provider	Service Provider	
1	SP[august] - IP[estela]	INT	May 20, 2021, 1:52:32 PM		risk-detector	estela	august	View Edit Delete
2	SP[august] - IP[francesco]	EXT	May 20, 2021, 2:05:02 PM		risk-detector	francesco	august	View Edit Delete

Showing 1 - 2 of 2 items.

pledger-project.eu

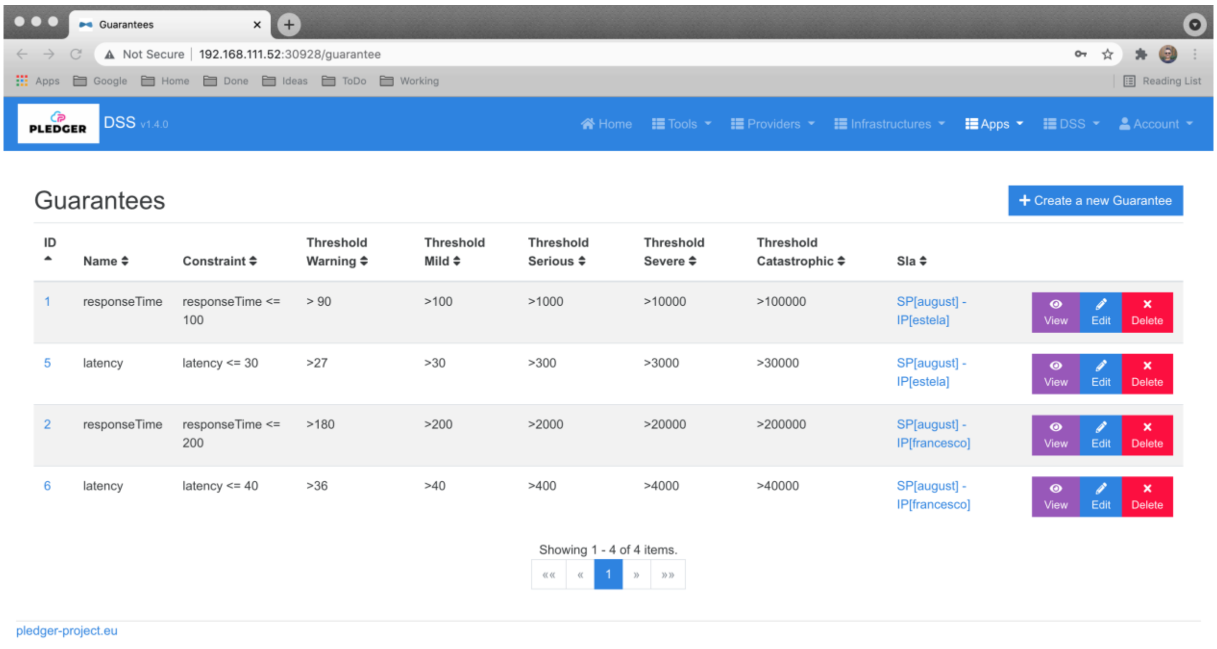
Figure 31: ConfService: SLA definition


ConfService DSS v2.7.1
Home Providers Infrastructures Apps DSS Administration Account
PLEDGER
Cancel
Save

Create or edit a SLA

ID	43
Name	risk-detector - availability
DSS Resource management	ignore
Infrastructure Provider	estela
Service Provider	august
Service	risk-detector

Figure 32: ConfService: SLA definition details



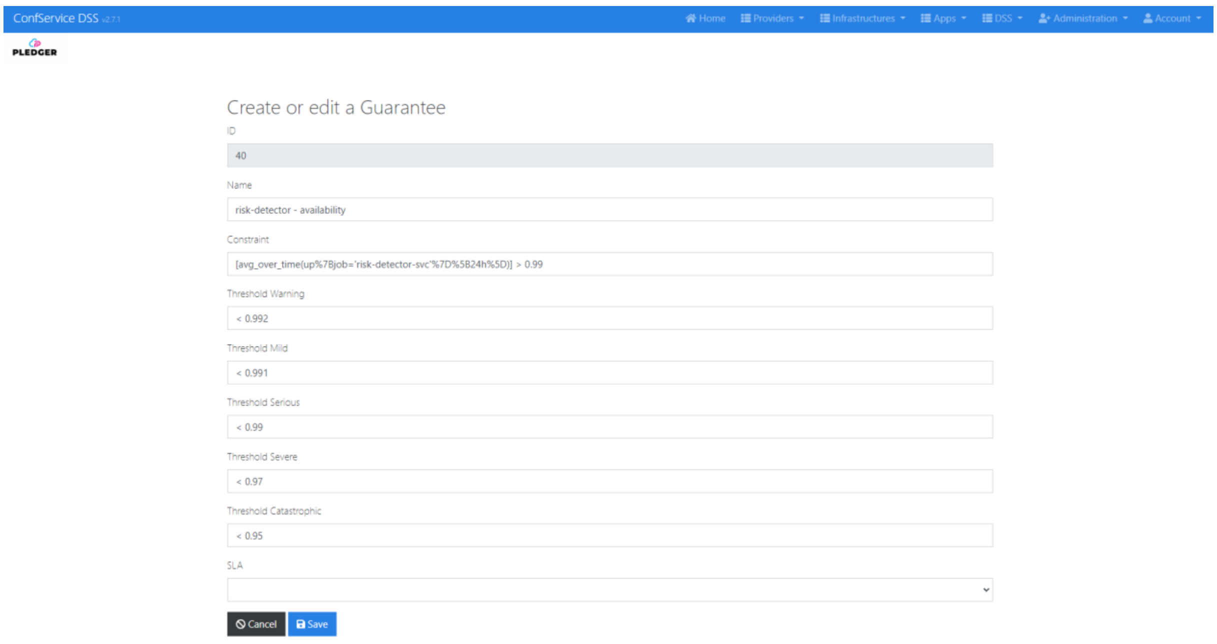
Guarantees

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]
5	latency	latency <= 30	>27	>30	>300	>3000	>30000	SP[august] - IP[estela]
2	responseTime	responseTime <= 200	>180	>200	>2000	>20000	>200000	SP[august] - IP[francesco]
6	latency	latency <= 40	>36	>40	>400	>4000	>40000	SP[august] - IP[francesco]

Showing 1 - 4 of 4 items.

pledger-project.eu

Figure 33: ConfService: Guarantees definition



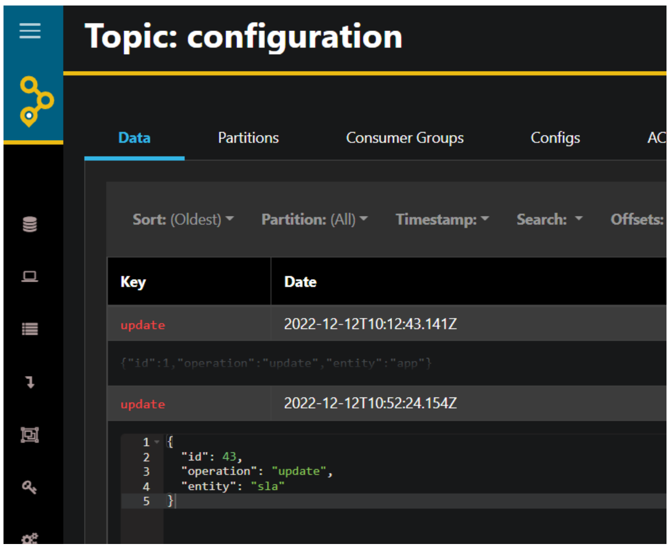
Create or edit a Guarantee

ID	40
Name	risk-detector - availability
Constraint	[avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] > 0.99
Threshold Warning	< 0.992
Threshold Mild	< 0.991
Threshold Serious	< 0.99
Threshold Severe	< 0.97
Threshold Catastrophic	< 0.95
SLA	

Figure 34: ConfService: Guarantees definition details

2.5.2 A SLA template, including metrics, thresholds and severity level is created.

After the SLA and guarantees have been created in the ConfService application, the SLA Framework gets a message from Kafka (Figure 35**Error! Reference source not found.**).



Key	Date
update	2022-12-12T10:12:43.141Z

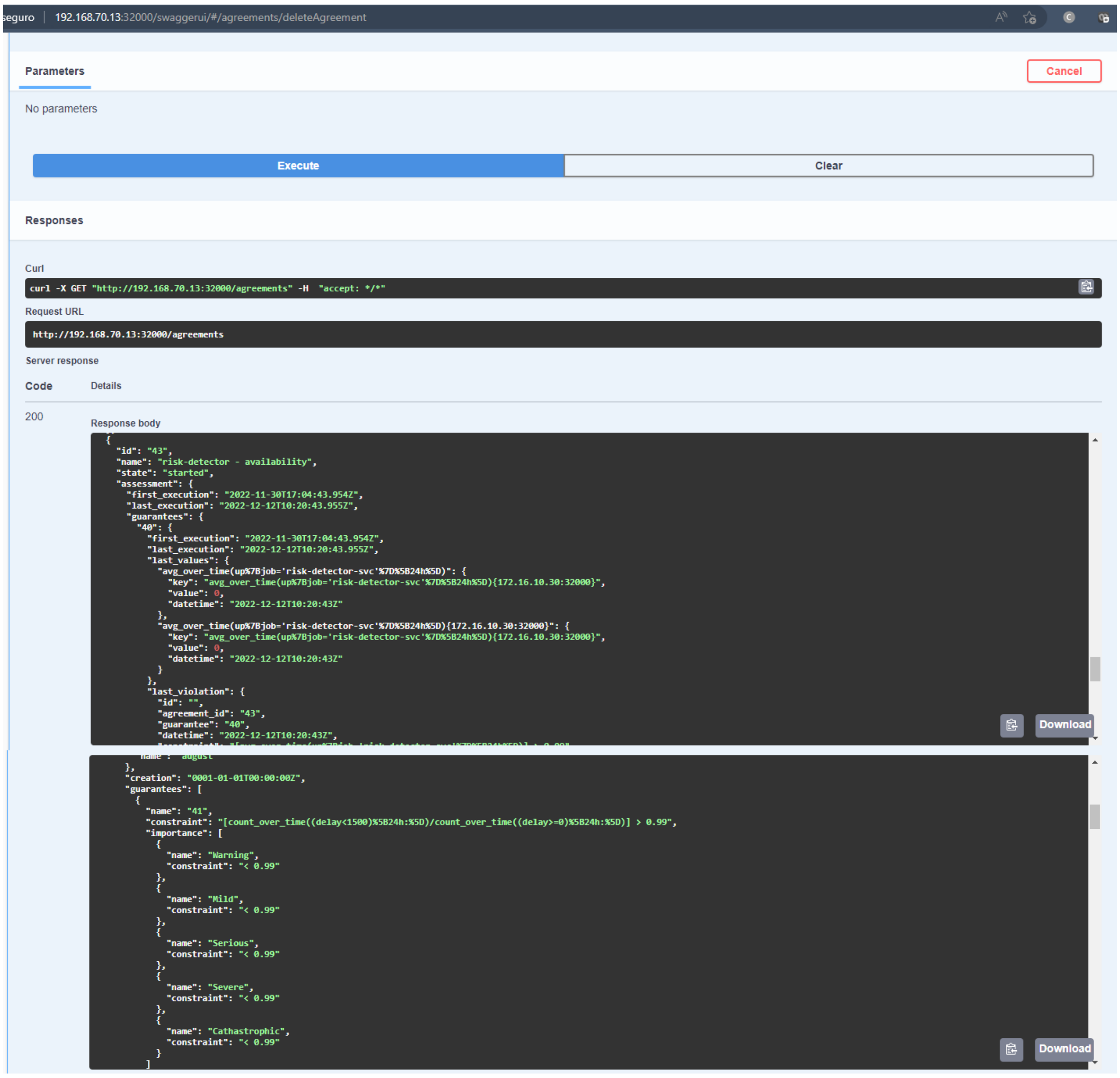
```
{"id":1,"operation":"update","entity":"app"}
```

Key	Date
update	2022-12-12T10:52:24.154Z

```
{
  "id": 43,
  "operation": "update",
  "entity": "sla"
}
```

Figure 35: Kafka: SLA creation message

After reading this message the SLA Framework creates the correspondent SLA, and starts the periodic assessment of it. Next figure (Figure 36**Error! Reference source not found.**) shows the SLA created and stored by the SLA Lite.


seguro | 192.168.70.13:32000/swaggerui/#/agreements/deleteAgreement
Parameters
Cancel
Execute
Clear
Responses
Download
Download

No parameters

Curl

```
curl -X GET "http://192.168.70.13:32000/agreements" -H  "accept: */*"
```

Request URL

```
http://192.168.70.13:32000/agreements
```

Server response

Code Details

200 Response body

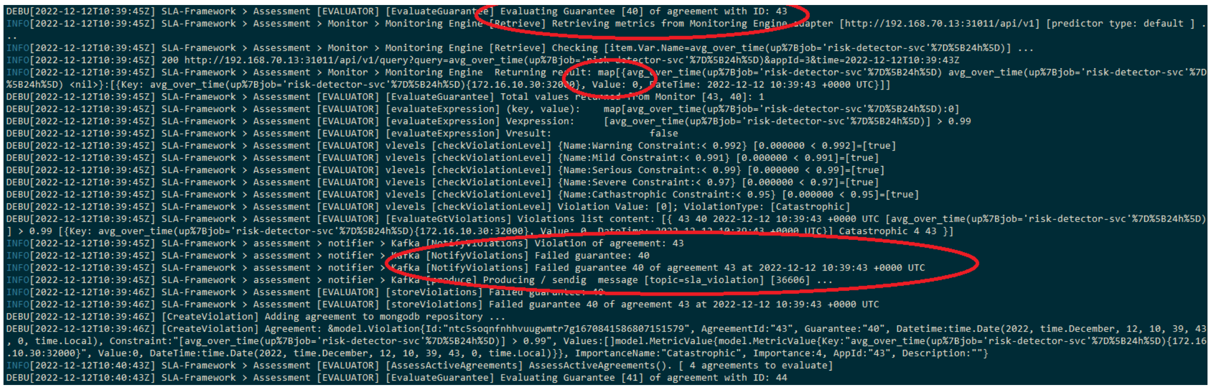
```
  {
    "id": "43",
    "name": "risk-detector - availability",
    "state": "started",
    "assessment": {
      "first_execution": "2022-11-30T17:04:43.954Z",
      "last_execution": "2022-12-12T10:20:43.955Z",
      "guarantees": {
        "40": {
          "first_execution": "2022-11-30T17:04:43.954Z",
          "last_execution": "2022-12-12T10:20:43.955Z",
          "last_values": {
            "avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)": {
              "key": "avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D){172.16.10.30:32000}",
              "value": 0,
              "datetime": "2022-12-12T10:20:43Z"
            },
            "avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D){172.16.10.30:32000}": {
              "key": "avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D){172.16.10.30:32000}",
              "value": 0,
              "datetime": "2022-12-12T10:20:43Z"
            }
          },
          "last_violation": {
            "id": "",
            "agreement_id": "43",
            "guarantee": "40",
            "datetime": "2022-12-12T10:20:43Z",
```

```
        "name": "august"
      },
      "creation": "0001-01-01T00:00:00Z",
      "guarantees": [
        {
          "name": "41",
          "constraint": "[count_over_time((delay<1500)%5B24h:%5D)/count_over_time((delay>=0)%5B24h:%5D)] > 0.99",
          "importance": [
            {
              "name": "Warning",
              "constraint": "< 0.99"
            },
            {
              "name": "Mild",
              "constraint": "< 0.99"
            },
            {
              "name": "Serious",
              "constraint": "< 0.99"
            },
            {
              "name": "Severe",
              "constraint": "< 0.99"
            },
            {
              "name": "Cathastrophic",
              "constraint": "< 0.99"
            }
          ]
```

Figure 36: SLA Framework Swagger UI: SLA

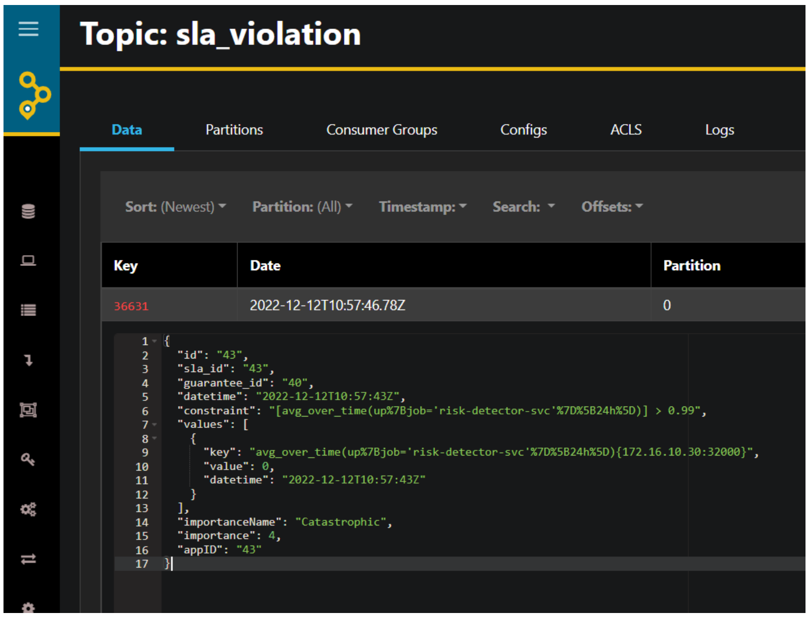
2.5.3 The template is used by the DSS to receive SLA violations

Whenever an SLA violation is received, the DSS stores and reports them in the UI. These are used by the optimization algorithm, later explained in the following sections. Figure 37 shows an example of SLA violation received by the DSS and shown in its UI.



```
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] [EvaluateGuarantee] Evaluating Guarantee [40] of agreement with ID: 43
INFO[2022-12-12T10:39:45Z] SLA-Framework > Assessment > Monitor > Monitoring Engine [Retrieve] Retrieving metrics from Monitoring Engine adapter [http://192.168.70.13:31011/api/v1] [predictor type: default ] .
..
INFO[2022-12-12T10:39:45Z] SLA-Framework > Assessment > Monitor > Monitoring Engine [Retrieve] Checking [item.Var.Name=avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] ...
INFO[2022-12-12T10:39:45Z] 200 http://192.168.70.13:31011/api/v1/query?query=avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)&appId=3&time=2022-12-12T10:39:43Z
INFO[2022-12-12T10:39:45Z] SLA-Framework > Assessment > Monitor > Monitoring Engine  Returning result: map[{avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D) avg_over_time(up%7Bjob='risk-detector-svc'%7D
%5B24h%5D) <nil>}:[{Key: avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D){172.16.10.30:32000}, Value: 0, DateTime: 2022-12-12 10:39:43 +0000 UTC}]]
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] [EvaluateGuarantee] Total values returned from Monitor [43, 40]: 1
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] [evaluateExpression] (key, value):    map[avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D):0]
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] [evaluateExpression] Vexpression:     [avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] > 0.99
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] [evaluateExpression] Vresult:                 false
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] vlevels [checkViolationLevel] {Name:Warning Constraint:< 0.992} [0.000000 < 0.992]=[true]
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] vlevels [checkViolationLevel] {Name:Mild Constraint:< 0.991} [0.000000 < 0.991]=[true]
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] vlevels [checkViolationLevel] {Name:Serious Constraint:< 0.99} [0.000000 < 0.99]=[true]
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] vlevels [checkViolationLevel] {Name:Severe Constraint:< 0.97} [0.000000 < 0.97]=[true]
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] vlevels [checkViolationLevel] {Name:Cathastrophic Constraint:< 0.95} [0.000000 < 0.95]=[true]
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] vlevels [checkViolationLevel] Violation Value: [0]; ViolationType: [Catastrophic]
DEBU[2022-12-12T10:39:45Z] SLA-Framework > Assessment [EVALUATOR] [EvaluateGtViolations] Violations list content: [{ 43 40 2022-12-12 10:39:43 +0000 UTC [avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)
] > 0.99 [{Key: avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D){172.16.10.30:32000}, Value: 0, DateTime: 2022-12-12 10:39:43 +0000 UTC}] Catastrophic 4 43 }]
INFO[2022-12-12T10:39:45Z] SLA-Framework > assessment > notifier > Kafka [NotifyViolations] Violation of agreement: 43
INFO[2022-12-12T10:39:45Z] SLA-Framework > assessment > notifier > Kafka [NotifyViolations] Failed guarantee: 40
INFO[2022-12-12T10:39:45Z] SLA-Framework > assessment > notifier > Kafka [NotifyViolations] Failed guarantee 40 of agreement 43 at 2022-12-12 10:39:43 +0000 UTC
INFO[2022-12-12T10:39:45Z] SLA-Framework > assessment > notifier > Kafka [produce] Producing / sendig  message [topic=sla_violation] [36606] ...
INFO[2022-12-12T10:39:46Z] SLA-Framework > Assessment [EVALUATOR] [storeViolations] Failed guarantee: 40
INFO[2022-12-12T10:39:46Z] SLA-Framework > Assessment [EVALUATOR] [storeViolations] Failed guarantee 40 of agreement 43 at 2022-12-12 10:39:43 +0000 UTC
DEBU[2022-12-12T10:39:46Z] [CreateViolation] Adding agreement to mongodb repository ...
DEBU[2022-12-12T10:39:46Z] [CreateViolation] Agreement: &model.Violation{Id:"ntc5soqnfnhhvuugwmtr7g1670841586807151579", AgreementId:"43", Guarantee:"40", Datetime:time.Date(2022, time.December, 12, 10, 39, 43
, 0, time.Local), Constraint:"[avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] > 0.99", Values:[]model.MetricValue{model.MetricValue{Key:"avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D){172.16
.10.30:32000}", Value:0, DateTime:time.Date(2022, time.December, 12, 10, 39, 43, 0, time.Local)}}, ImportanceName:"Catastrophic", Importance:4, AppId:"43", Description:""}
INFO[2022-12-12T10:40:43Z] SLA-Framework > Assessment [EVALUATOR] [AssessActiveAgreements] AssessActiveAgreements(). [ 4 agreements to evaluate]
DEBU[2022-12-12T10:40:43Z] SLA-Framework > Assessment [EVALUATOR] [EvaluateGuarantee] Evaluating Guarantee [41] of agreement with ID: 44
```

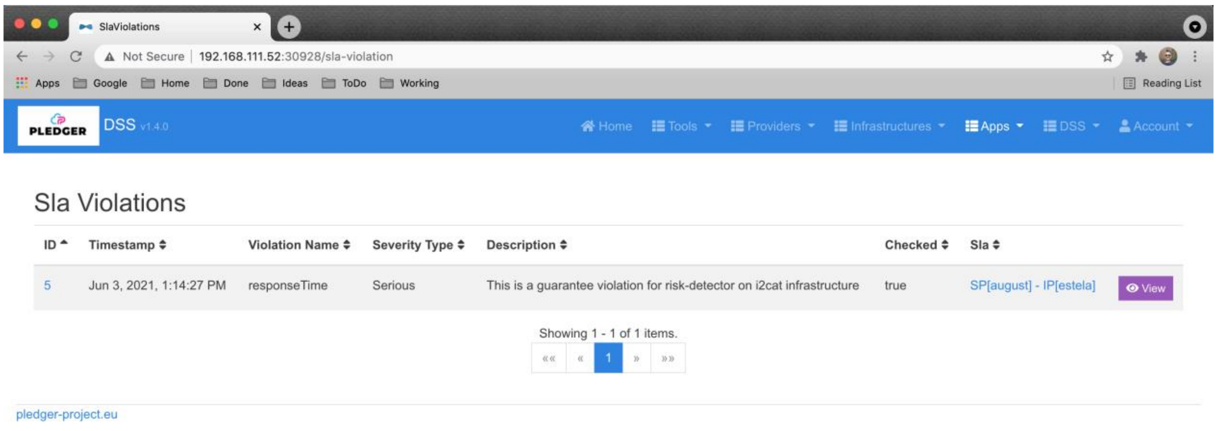
Figure 37: SLA Framework logs: SLA violation



Key	Date	Partition
36631	2022-12-12T10:57:46.78Z	0

```
{
  "id": "43",
  "sla_id": "43",
  "guarantee_id": "40",
  "datetime": "2022-12-12T10:57:43Z",
  "constraint": "[avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] > 0.99",
  "values": [
    {
      "key": "avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D){172.16.10.30:32000}",
      "value": 0,
      "datetime": "2022-12-12T10:57:43Z"
    }
  ],
  "importanceName": "Catastrophic",
  "importance": 4,
  "appID": "43"
}
```

Figure 38: Kafka: SLA violation message



Sla Violations

ID	Timestamp	Violation Name	Severity Type	Description	Checked	Sla	
5	Jun 3, 2021, 1:14:27 PM	responseTime	Serious	This is a guarantee violation for risk-detector on i2cat infrastructure	true	SP[august] - IP[estela]	View

Showing 1 - 1 of 1 items.

pledger-project.eu

Figure 39: DSS: SLA violations received

2.6 Deployment configuration

The **objective** of this section is to show how the DSS deployment configuration works and is used to enable the optimization algorithms. The components involved are:

- ConfService,
- DSS,
- Orchestrator,
- Benchmarking+AppProfiler,
- MonitoringEngine,
- StreamHandler

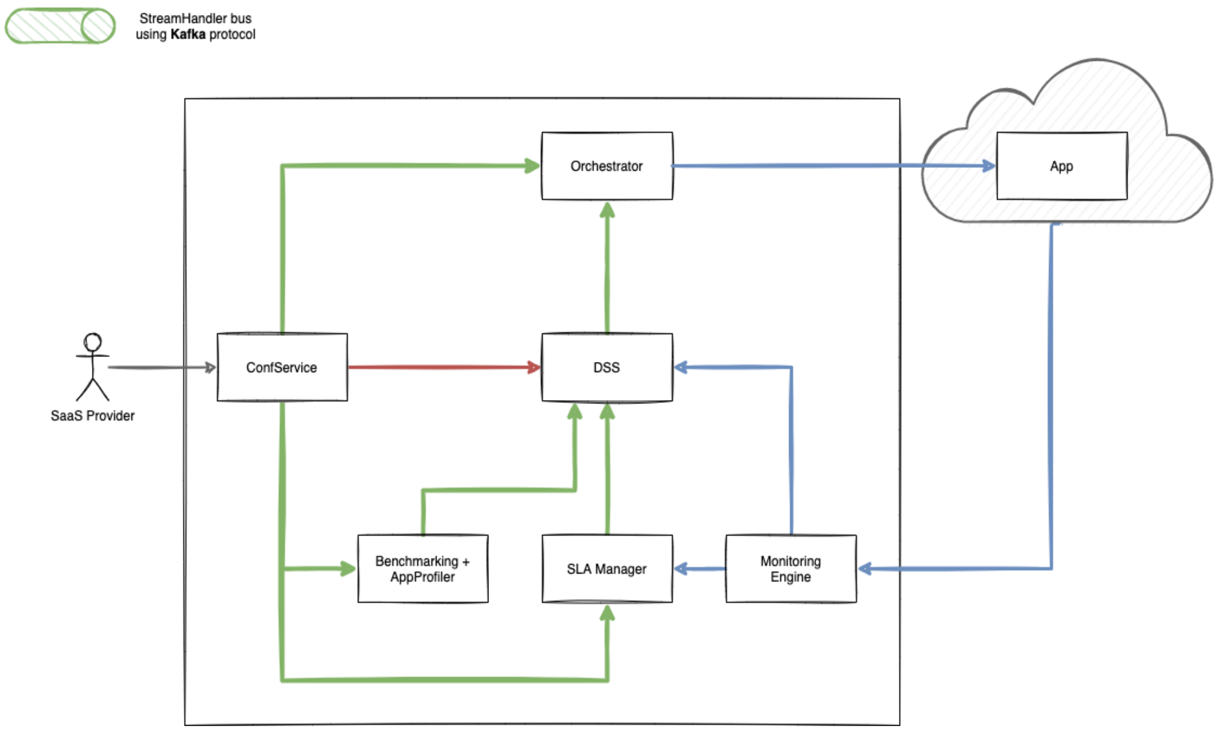
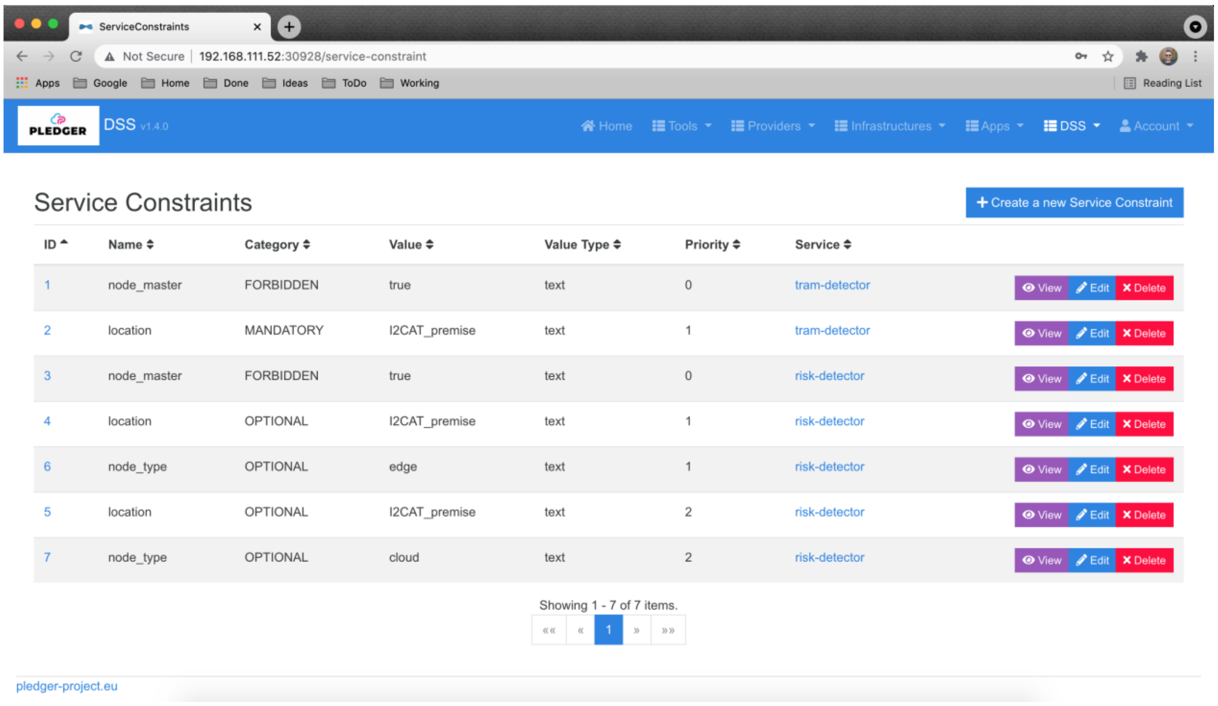


Figure 40: DSS main interactions with core components.

2.6.1 DSS analyses all the information provided and develops the most suitable deployment scheme.

Figure 41 shows the configuration of deployment constraints from the Service provider and Figure 42 the resulting prioritized deployment options that will be used by the DSS. This way, the DSS takes into consideration the service provider preferences about the infrastructures and nodes where to instantiate DApps, while keeping additional space for optimization within the remaining equivalent options using the optimization algorithm configured, as shown in Figure 43.

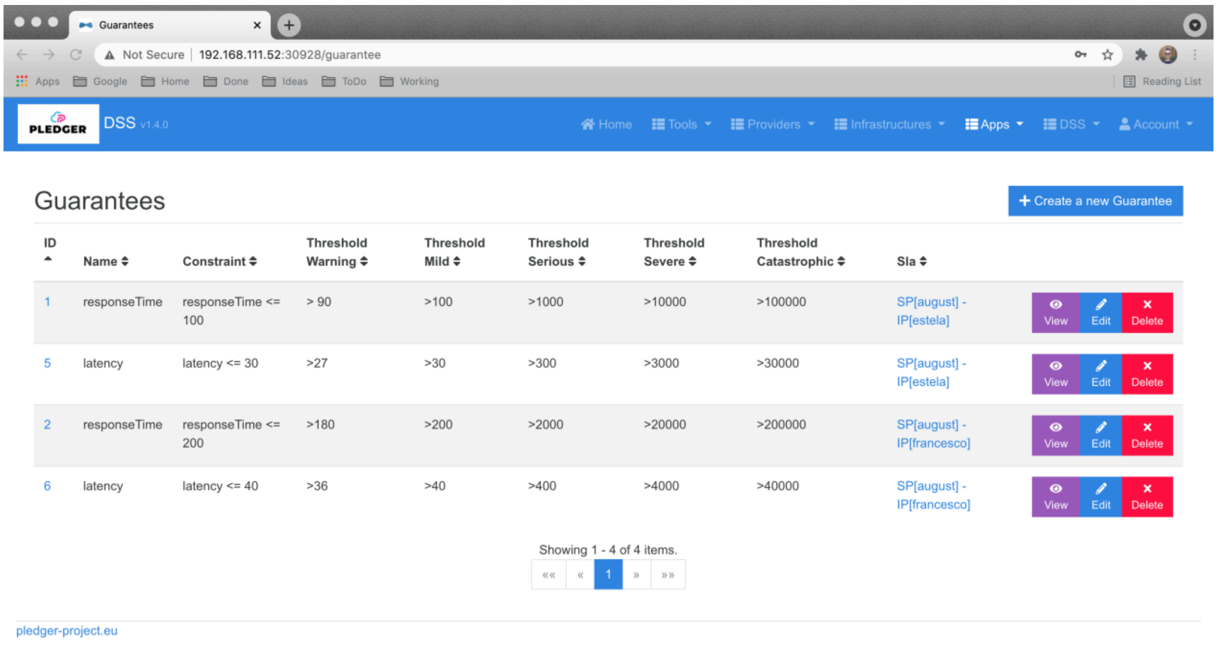

ServiceConstraints
Not Secure | 192.168.111.52:30928/service-constraint
Apps Google Home Done Ideas ToDo Working
Reading List
PLEDGER DSS v1.4.0
Home Tools Providers Infrastructures Apps DSS Account
Create a new Service Constraint
View Edit Delete
pledger-project.eu

Service Constraints

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

Showing 1 - 7 of 7 items.

Figure 41: DSS: ServiceConstraints, setup by the SP, to express deployment preferences

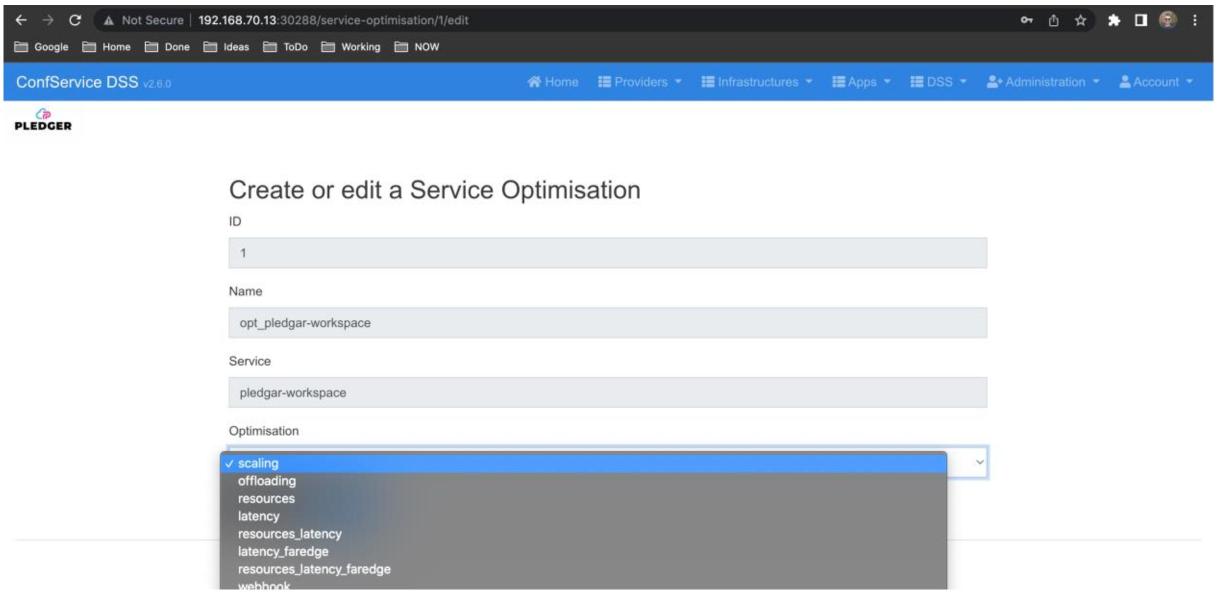

Guarantees
Not Secure | 192.168.111.52:30928/guarantee
Apps Google Home Done Ideas ToDo Working
Reading List
PLEDGER DSS v1.4.0
Home Tools Providers Infrastructures Apps DSS Account
Create a new Guarantee
View Edit Delete
pledger-project.eu

Guarantees

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]
5	latency	latency <= 30	>27	>30	>300	>3000	>30000	SP[august] - IP[estela]
2	responseTime	responseTime <= 200	>180	>200	>2000	>20000	>200000	SP[august] - IP[francesco]
6	latency	latency <= 40	>36	>40	>400	>4000	>40000	SP[august] - IP[francesco]

Showing 1 - 4 of 4 items.

Figure 42: DSS deployment options for a specific App based on the ServiceConstraints



Create or edit a Service Optimisation

ID: 1

Name: opt_pledgar-workspace

Service: pledgar-workspace

Optimisation: ✓ scaling

Figure 43: DSS optimisation algorithm configuration

Table 2 reports the DSS deployment options demo on PLEDGER YouTube channel.

Title	Link
ConfService_#4: SP setting App deployment options	https://www.youtube.com/watch?v=D7LZbcHfiXk
DSS_#1: DSS using deployment options to privilege edge over cloud and scaling down	https://www.youtube.com/watch?v=gJecLrZAoNw

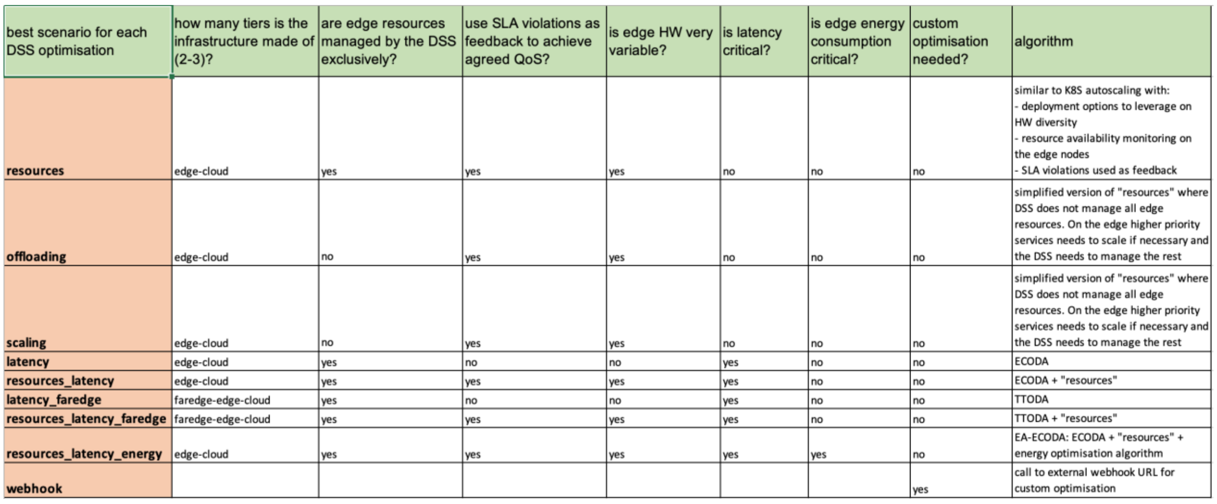
Table 2: DSS deployment options demo

2.6.2 DSS computes the best node and notifies the orchestrator to start the deployment of the application

Depending on the optimisation algorithm configured, the DSS computes the action to take according to the resources available, the node-to-node latency, etc. from the MonitoringEngine, as well as the Becnhmarking and AppProfiler data, the SLA violations, then communicates with the Orchestrator.

The optimisation algorithms available and their scenarios are described in the DSS documentation on Gitlab[3] and a comparison is reported in Figure 44.

[3] https://gitlab.com/pledger/public/confservicedss/-/blob/master/doc/optimisations/optimisation_scenarios.md



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?	algorithm
resources	edge-cloud	yes	yes	yes	no	no	no	similar to K8S autoscaling with: - deployment options to leverage on HW diversity - resource availability monitoring on the edge nodes - SLA violations used as feedback
offloading	edge-cloud	no	yes	yes	no	no	no	simplified version of "resources" where DSS does not manage all edge resources. On the edge higher priority services needs to scale if necessary and the DSS needs to manage the rest
scaling	edge-cloud	no	yes	yes	no	no	no	simplified version of "resources" where DSS does not manage all edge resources. On the edge higher priority services needs to scale if necessary and the DSS needs to manage the rest
latency	edge-cloud	yes	no	no	yes	no	no	ECODA
resources_latency	edge-cloud	yes	yes	yes	yes	no	no	ECODA + "resources"
latency_faredge	faredge-edge-cloud	yes	no	no	yes	no	no	TTODA
resources_latency_faredge	faredge-edge-cloud	yes	yes	yes	yes	no	no	TTODA + "resources"
resources_latency_energy	edge-cloud	yes	yes	yes	yes	yes	no	EA-ECODA: ECODA + "resources" + energy optimisation algorithm
webhook							yes	call to external webhook URL for custom optimisation

Figure 44: DSS optimisation algorithm scenarios

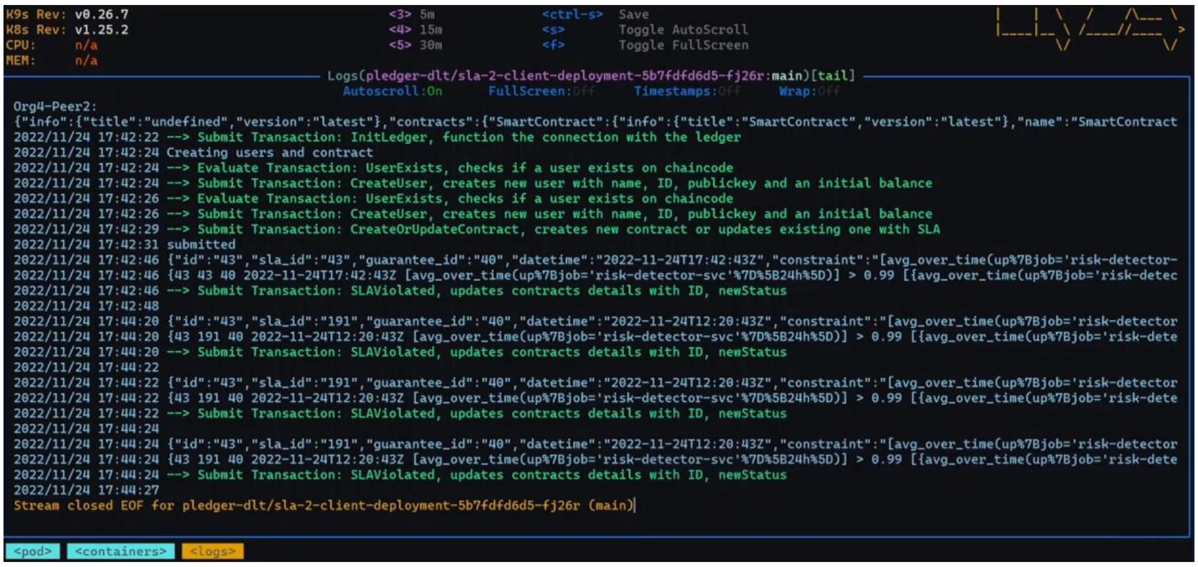
2.7 Generating smart contracts

After the SLA template is created from the corresponding system UI (2.5.2), the blockchain framework receives the data. The Pledger DLT hosts the SLASC Bridge core component that is responsible for the smart contract generation. The entire procedure along with the penalties described later in 2.13 unfolds within a private and confidential blockchain environment of the network. In the following figures, the sequence of the data flow is depicted with each action described. In simple terms, the main actions that occur are as follows:

- SLA configuration or creation
- Smart contract equivalent creation, deployment, and submission on the DLT
- Smart contract accessibility on the DLT

2.7.1Once the SLA template is created, the blockchain framework is notified.

Once SLA template is created in the corresponding system UI, the SLA application sends the data with a secure way to the blockchain network. In the meantime, the blockchain components are constantly receiving notifications as depicted in Figure 45.



```
K9s Rev: v0.26.7                                   <3> 5m          <ctrl-s>  Save
K8s Rev: v1.25.2                                   <4> 15m         <s>       Toggle AutoScroll
CPU:      n/a                                      <5> 30m         <f>       Toggle FullScreen
MEM:      n/a
                          Logs(pledger-dlt/sla-2-client-deployment-5b7fdfd6d5-fj26r:main)[tail]
                             Autoscroll:On      FullScreen:Off      Timestamps:Off      Wrap:Off
 Org4-Peer2:
 {"info":{"title":"undefined","version":"latest"},"contracts":{"SmartContract":{"info":{"title":"SmartContract","version":"latest"},"name":"SmartContract
 2022/11/24 17:42:22 --> Submit Transaction: InitLedger, function the connection with the ledger
 2022/11/24 17:42:24 Creating users and contract
 2022/11/24 17:42:24 --> Evaluate Transaction: UserExists, checks if a user exists on chaincode
 2022/11/24 17:42:24 --> Submit Transaction: CreateUser, creates new user with name, ID, publickey and an initial balance
 2022/11/24 17:42:26 --> Evaluate Transaction: UserExists, checks if a user exists on chaincode
 2022/11/24 17:42:26 --> Submit Transaction: CreateUser, creates new user with name, ID, publickey and an initial balance
 2022/11/24 17:42:29 --> Submit Transaction: CreateOrUpdateContract, creates new contract or updates existing one with SLA
 2022/11/24 17:42:31 submitted
 2022/11/24 17:42:46 {"id":"43","sla_id":"43","guarantee_id":"40","datetime":"2022-11-24T17:42:43Z","constraint":"[avg_over_time(up%7Bjob='risk-detector-
 2022/11/24 17:42:46 {43 43 40 2022-11-24T17:42:43Z [avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] > 0.99 [{avg_over_time(up%7Bjob='risk-detec
 2022/11/24 17:42:46 --> Submit Transaction: SLAViolated, updates contracts details with ID, newStatus
 2022/11/24 17:42:48
 2022/11/24 17:44:20 {"id":"43","sla_id":"191","guarantee_id":"40","datetime":"2022-11-24T12:20:43Z","constraint":"[avg_over_time(up%7Bjob='risk-detector
 2022/11/24 17:44:20 {43 191 40 2022-11-24T12:20:43Z [avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] > 0.99 [{avg_over_time(up%7Bjob='risk-dete
 2022/11/24 17:44:20 --> Submit Transaction: SLAViolated, updates contracts details with ID, newStatus
 2022/11/24 17:44:22
 2022/11/24 17:44:22 {"id":"43","sla_id":"191","guarantee_id":"40","datetime":"2022-11-24T12:20:43Z","constraint":"[avg_over_time(up%7Bjob='risk-detector
 2022/11/24 17:44:22 {43 191 40 2022-11-24T12:20:43Z [avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] > 0.99 [{avg_over_time(up%7Bjob='risk-dete
 2022/11/24 17:44:22 --> Submit Transaction: SLAViolated, updates contracts details with ID, newStatus
 2022/11/24 17:44:24
 2022/11/24 17:44:24 {"id":"43","sla_id":"191","guarantee_id":"40","datetime":"2022-11-24T12:20:43Z","constraint":"[avg_over_time(up%7Bjob='risk-detector
 2022/11/24 17:44:24 {43 191 40 2022-11-24T12:20:43Z [avg_over_time(up%7Bjob='risk-detector-svc'%7D%5B24h%5D)] > 0.99 [{avg_over_time(up%7Bjob='risk-dete
 2022/11/24 17:44:24 --> Submit Transaction: SLAViolated, updates contracts details with ID, newStatus
 2022/11/24 17:44:27
 Stream closed EOF for pledger-dlt/sla-2-client-deployment-5b7fdfd6d5-fj26r (main)

 <pod>  <containers>  <logs>
```

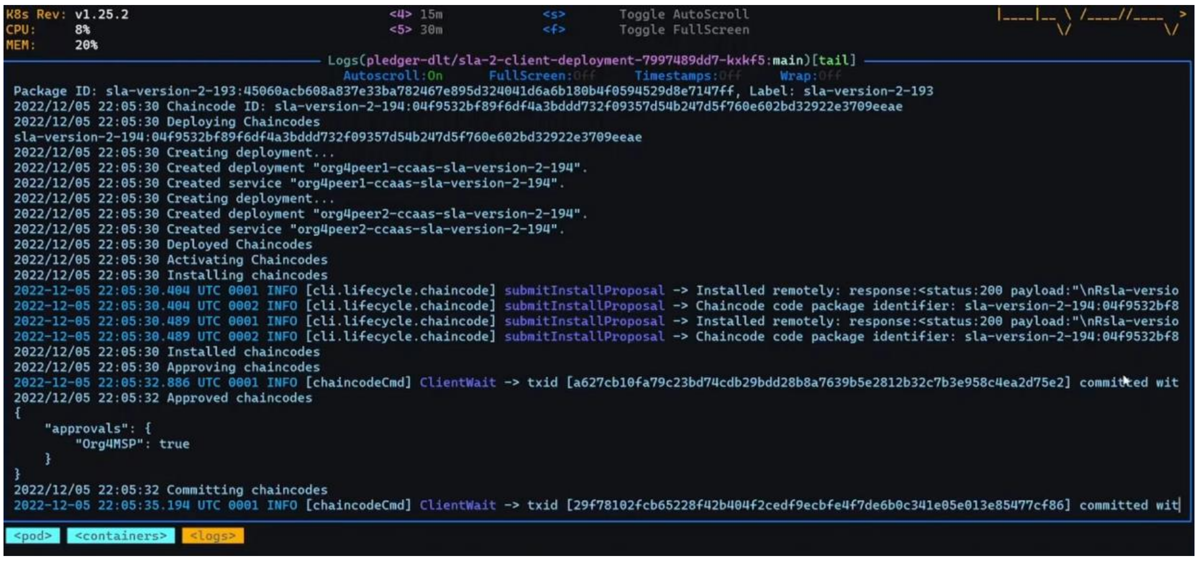
Figure 45: Blockchain receiving notifications.

2.7.2 SLA-SC gathers the SLA guarantees and converts them into a smart contract between the service provider and the infrastructure provider.

SLASC Bridge initiates the process of SLA to Smart Contract by completing the following workflow:

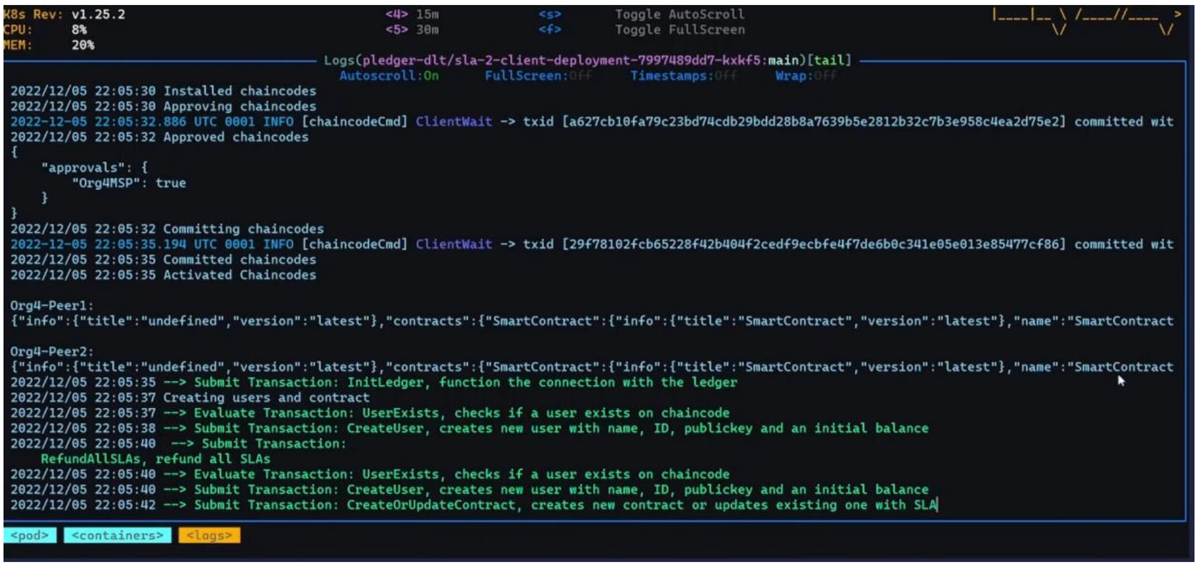
- creating the smart contract equivalent
- deploying the smart contract on the blockchain
- submitting the SLA logic on the ledger

The accompanying figures showcase each stage of the workflow from the blockchain point of view. During the entire process, the user accounts are identified if they exist or newly created otherwise.



```
K8s Rev: v1.25.2                              <4> 15m          <s>      Toggle AutoScroll
CPU:     8%                                   <5> 30m          <f>      Toggle FullScreen
MEM:     20%
                         Logs(pledger-dlt/sla-2-client-deployment-7997489dd7-kxkf5:main)[tail]
                              Autoscroll:On      FullScreen:Off      Timestamps:Off      Wrap:Off
Package ID: sla-version-2-193:45060acb608a837e33ba782467e895d324041d6a6b180b4f0594529d8e7147ff, Label: sla-version-2-193
2022/12/05 22:05:30 Chaincode ID: sla-version-2-194:04f9532bf89f6df4a3bddd732f09357d54b247d5f760e602bd32922e3709eeae
2022/12/05 22:05:30 Deploying Chaincodes
sla-version-2-194:04f9532bf89f6df4a3bddd732f09357d54b247d5f760e602bd32922e3709eeae
2022/12/05 22:05:30 Creating deployment...
2022/12/05 22:05:30 Created deployment "org4peer1-ccaas-sla-version-2-194".
2022/12/05 22:05:30 Created service "org4peer1-ccaas-sla-version-2-194".
2022/12/05 22:05:30 Creating deployment...
2022/12/05 22:05:30 Created deployment "org4peer2-ccaas-sla-version-2-194".
2022/12/05 22:05:30 Created service "org4peer2-ccaas-sla-version-2-194".
2022/12/05 22:05:30 Deployed Chaincodes
2022/12/05 22:05:30 Activating Chaincodes
2022/12/05 22:05:30 Installing chaincodes
2022-12-05 22:05:30.404 UTC 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nRsla-versio
2022-12-05 22:05:30.404 UTC 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: sla-version-2-194:04f9532bf8
2022-12-05 22:05:30.489 UTC 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nRsla-versio
2022-12-05 22:05:30.489 UTC 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: sla-version-2-194:04f9532bf8
2022/12/05 22:05:30 Installed chaincodes
2022/12/05 22:05:30 Approving chaincodes
2022-12-05 22:05:32.886 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [a627cb10fa79c23bd74cdb29bdd28b8a7639b5e2812b32c7b3e958c4ea2d75e2] committed wit
2022/12/05 22:05:32 Approved chaincodes
{
    "approvals": {
        "Org4MSP": true
    }
}
2022/12/05 22:05:32 Committing chaincodes
2022-12-05 22:05:35.194 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [29f78102fcb65228f42b404f2cedf9ecbfe4f7de6b0c341e05e013e85477cf86] committed wit
<pod> <containers> <logs>
```

Figure 46: Blockchain creates the smart contract

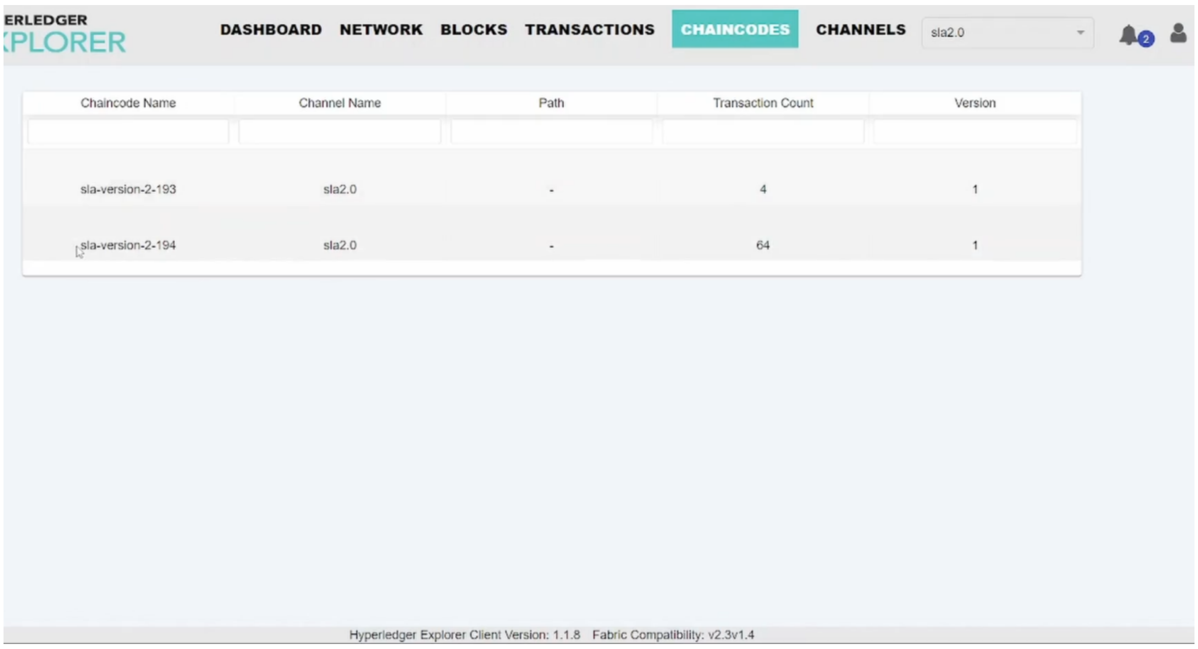


```
K8s Rev: v1.25.2                              <4> 15m          <s>      Toggle AutoScroll
CPU:     8%                                   <5> 30m          <f>      Toggle FullScreen
MEM:     20%
                         Logs(pledger-dlt/sla-2-client-deployment-7997489dd7-kxkf5:main)[tail]
                              Autoscroll:On      FullScreen:Off      Timestamps:Off      Wrap:Off
2022/12/05 22:05:30 Installed chaincodes
2022/12/05 22:05:30 Approving chaincodes
2022-12-05 22:05:32.886 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [a627cb10fa79c23bd74cdb29bdd28b8a7639b5e2812b32c7b3e958c4ea2d75e2] committed wit
2022/12/05 22:05:32 Approved chaincodes
{
    "approvals": {
        "Org4MSP": true
    }
}
2022/12/05 22:05:32 Committing chaincodes
2022-12-05 22:05:35.194 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [29f78102fcb65228f42b404f2cedf9ecbfe4f7de6b0c341e05e013e85477cf86] committed wit
2022/12/05 22:05:35 Committed chaincodes
2022/12/05 22:05:35 Activated Chaincodes

Org4-Peer1:
{"info":{"title":"undefined","version":"latest"},"contracts":{"SmartContract":{"info":{"title":"SmartContract","version":"latest"},"name":"SmartContract

Org4-Peer2:
{"info":{"title":"undefined","version":"latest"},"contracts":{"SmartContract":{"info":{"title":"SmartContract","version":"latest"},"name":"SmartContract
2022/12/05 22:05:35 --> Submit Transaction: InitLedger, function the connection with the ledger
2022/12/05 22:05:37 Creating users and contract
2022/12/05 22:05:37 --> Evaluate Transaction: UserExists, checks if a user exists on chaincode
2022/12/05 22:05:38 --> Submit Transaction: CreateUser, creates new user with name, ID, publickey and an initial balance
2022/12/05 22:05:40  --> Submit Transaction:
    RefundAllSLAs, refund all SLAs
2022/12/05 22:05:40 --> Evaluate Transaction: UserExists, checks if a user exists on chaincode
2022/12/05 22:05:40 --> Submit Transaction: CreateUser, creates new user with name, ID, publickey and an initial balance
2022/12/05 22:05:42 --> Submit Transaction: CreateOrUpdateContract, creates new contract or updates existing one with SLA
<pod> <containers> <logs>
```

Figure 47: Blockchain deploys and submits the smart contract

2.7.3 Smart contract is accessible in a readable manner by the two parties.

The smart contract equivalent is accessible to the dedicated users that are participating on the created SLA. The contractual terms are submitted on the blockchain, while the new smart contract is triggered in sync with the necessary operations that pertain to the SLA configuration. The automation of the refunding mechanisms are explained later in 2.13.



Chaincode Name	Channel Name	Path	Transaction Count	Version
sla-version-2-193	sla2.0	-	4	1
sla-version-2-194	sla2.0	-	64	1

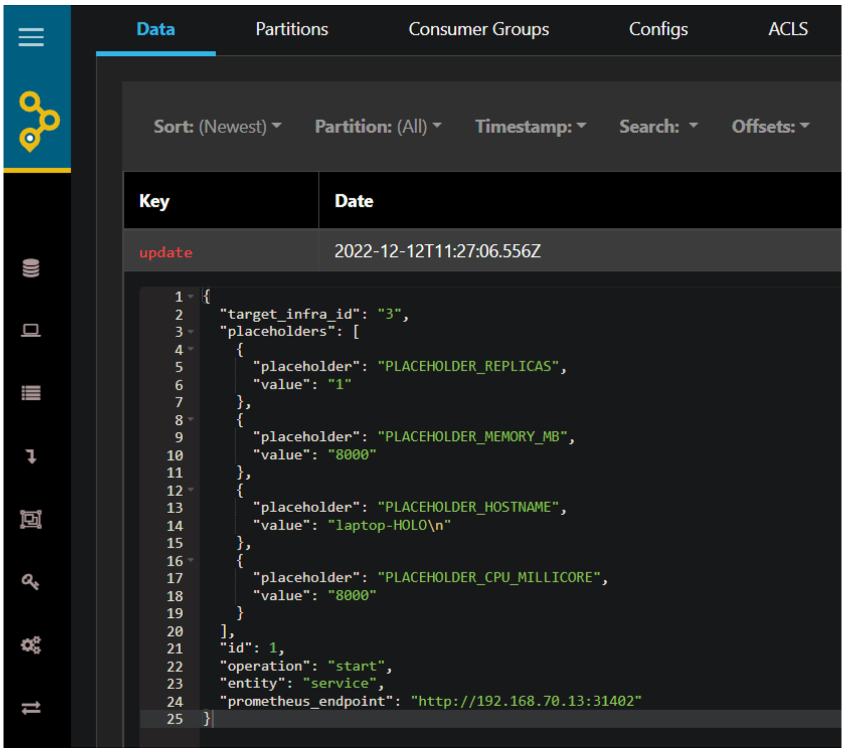
Hyperledger Explorer Client Version: 1.1.8 Fabric Compatibility: v2.3v1.4

Figure 48: Smart contract accessible by parties

2.8 Application deployment

2.8.1The orchestrator receives the notification from the DSS for deploying an application.

The E2CO application reads the deploying message from Kafka (Figure 49**Error! Reference source not found.**) and connects to the required infrastructure to deploy there the application.



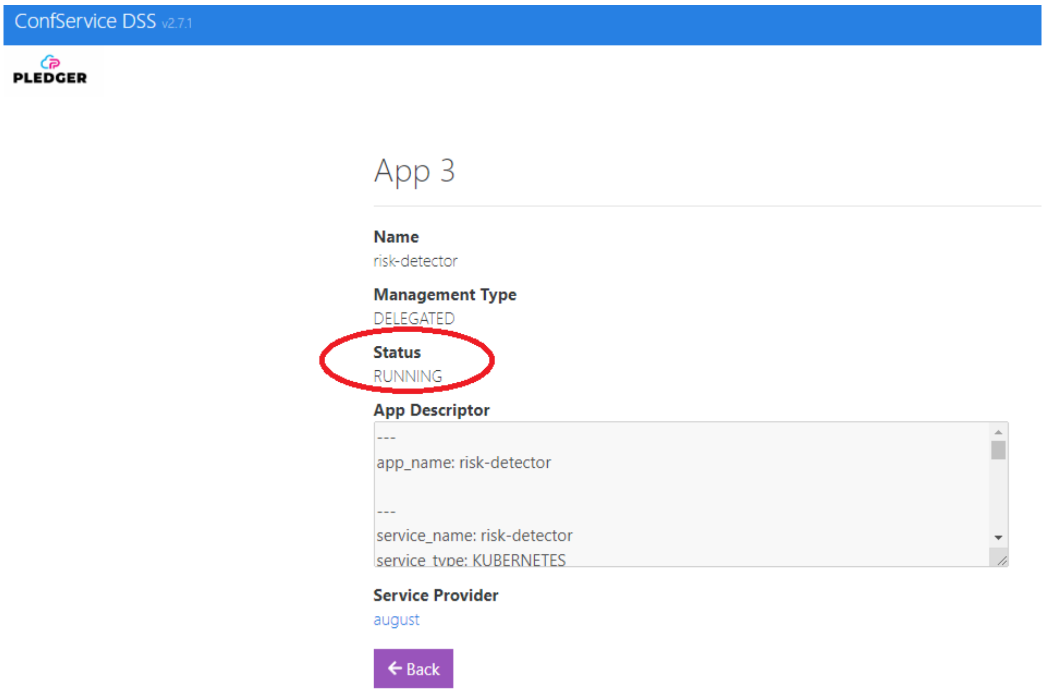
Key	Date
update	2022-12-12T11:27:06.556Z

```
{
  "target_infra_id": "3",
  "placeholders": [
    {
      "placeholder": "PLACEHOLDER_REPLICAS",
      "value": "1"
    },
    {
      "placeholder": "PLACEHOLDER_MEMORY_MB",
      "value": "8000"
    },
    {
      "placeholder": "PLACEHOLDER_HOSTNAME",
      "value": "laptop-HOLO\n"
    },
    {
      "placeholder": "PLACEHOLDER_CPU_MILLICORE",
      "value": "8000"
    }
  ],
  "id": 1,
  "operation": "start",
  "entity": "service",
  "prometheus_endpoint": "http://192.168.70.13:31402"
}
```

Figure 49: Kafka: App deployment message

2.8.2 Orchestrator deploys the application in the selected node and starts it.

The status of the application is shown in the ConfService application (Figure 50).



App 3

Name
risk-detector

Management Type
DELEGATED

Status
RUNNING

App Descriptor

```
---
app_name: risk-detector

---
service_name: risk-detector
service_type: KUBERNETES
```

Service Provider
august

Figure 50: ConfService: App running

2.8.3 The Monitoring Engine starts automatically to monitor the application health.

Once the application has been successfully deployed and the SLA created, the SLA Framework starts the periodic evaluation of the existing SLAs (Figure 37). To do that it connects to the Monitoring Engine, which is the component responsible for gathering the metrics values from the different infrastructures.

2.9 E2E slicing

The objective of this section is to show how the deployment of end-to-end slices is performed in Pledger. In a nutshell, the deployment of network slices consists of the reservation and isolation of compute, network, and radio elements. From the infrastructure provider's perspective, network slicing allows for the partitioning of a physical infrastructure; each partition or network slice can then be rented to an interested party or service provider. Network slices can be created under the service provider's request with designated capacities, to cater for the QoS requirements of the specific services to be deployed.

After the network slice is created and activated by the infrastructure provider, services (including network services) can then be deployed over the end-to-end network slice.

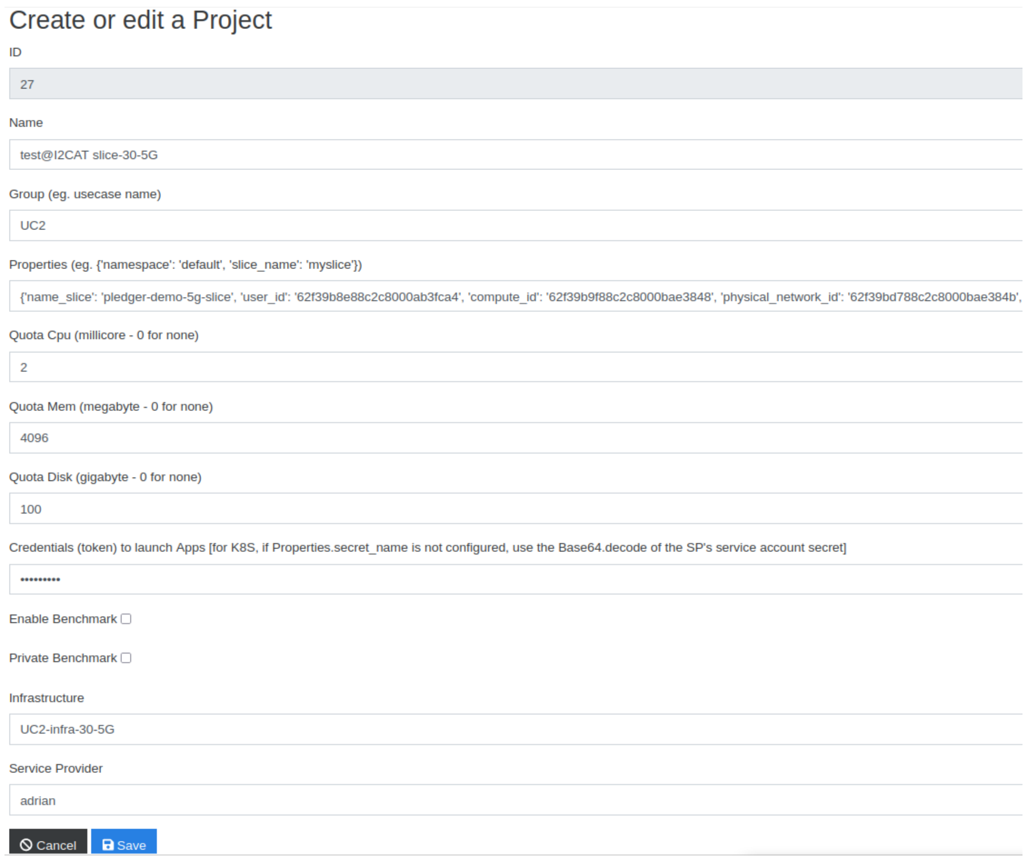
Note that Pledger supports the deployment of two types of network slices that vary in their composition depending on the radio access technology of choice:

- 5G network slices, where the underlying compute infrastructure is based on Openstack and one single, isolated radio cell is allocated to each network slice.
- Cloud native network slices, where the underlying compute infrastructure is based on Kubernetes, and the radio elements are based on the IEEE 802.11p standard.

In both types of slices, in addition to providing the desired radio access technology, a slice request also includes the desired computing resources to host the application and services to be deployed by the service provider.

2.9.1 The infrastructure provider starts the network slice creation process using the ConfService.

In those cases where a network slice needs to be in place, the infrastructure administrator is responsible for its deployment. Note that, prior to the slice deployment phase, an agreement must exist between the infrastructure and service providers regarding the requirements of the network slice in terms of resources and location of elements comprising the slice. After such an agreement has been reached, Pledger is used for the deployment of the network slice. Figure 51 shows the user dashboard from which the infrastructure administrator can set up the relevant slice parameters. Figure 52 shows the ConfService's *Projects* dashboard, from which network slices can be managed (i.e., configured, provisioned and unprovisioned).



Create or edit a Project

ID

27

Name

test@I2CAT slice-30-5G

Group (eg. usecase name)

UC2

Properties (eg. {'namespace': 'default', 'slice_name': 'myslice'})

{'name_slice': 'pledger-demo-5g-slice', 'user_id': '62f39b8e88c2c8000ab3fca4', 'compute_id': '62f39b9f88c2c8000bae3848', 'physical_network_id': '62f39bd788c2c8000bae384b',

Quota Cpu (millicore - 0 for none)

2

Quota Mem (megabyte - 0 for none)

4096

Quota Disk (gigabyte - 0 for none)

100

Credentials (token) to launch Apps [for K8S, if Properties.secret_name is not configured, use the Base64.decode of the SP's service account secret]

•••••••••

Enable Benchmark ☐

Private Benchmark ☐

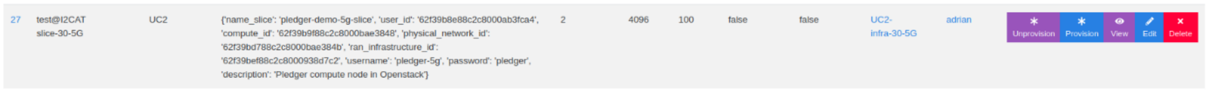
Infrastructure

UC2-infra-30-5G

Service Provider

adrian

Figure 51: End-to-end slice configuration from the ConfService



27	test@I2CAT slice-30-5G	UC2	{'name_slice': 'pledger-demo-5g-slice', 'user_id': '62f39b8e88c2c8000ab3fca4', 'compute_id': '62f39b9f88c2c8000bae3848', 'physical_network_id': '62f39bd788c2c8000bae384b', 'ran_infrastructure_id': '62f39bef88c2c8000938d7c2', 'username': 'pledger-5g', 'password': 'pledger', 'description': 'Pledger compute node in Openstack'}	2	4096	100	false	false	UC2-infra-30-5G	adrian	Unprovision Provision View Edit Delete

Figure 52: End-to-end slice management from the ConfService

After the infrastructure administrator initiates the provisioning of the network slice, the ConfService parses the configuration parameters to the Orchestrator, which in turn sends the required flow of API calls to SOE's northbound interface.

2.9.2 The SOE Framework and the RAN Controller execute the required actions for the creation and activation of the network slice.

The SOE Framework is responsible for the deployment and activation of network slices. Network slice creation and activation requests are sent to the SOE Framework by the Orchestrator, as a set of API calls, in order to execute the following actions (in the presented order):

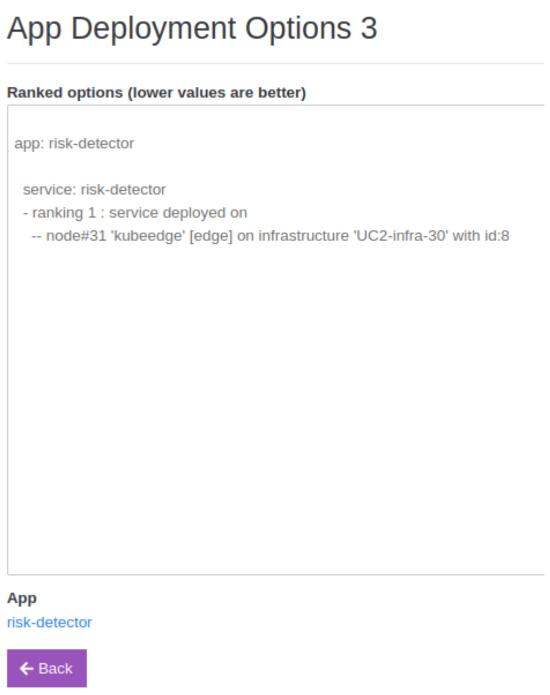
- Compute, network, and radio partitions (i.e., chunks) creation. The SOE Framework receives one API call from the Orchestrator for the creation of each chunk, and it returns an acknowledgement to the Orchestrator after the successful execution of each action. Note that the RAN Controller is the responsible entity for the creation of radio chunks and, therefore, there exists an interaction between the SOE Framework and the RAN Controller during the radio chunk creation phase.

- Creation of the network slice as a composition of compute, network, and radio chunks. The SOE Framework receives one single API call from the Orchestrator for the creation of the network slice, and it returns an acknowledgement to the Orchestrator after the network slice creation phase is completed.
- Network slice activation, including the deployment of required software elements (i.e., the 5G core for 5G network slices, or a V2X stack for network slices with IEEE 802.11p radio elements.) The SOE Framework receives one single API call from the Orchestrator for the activation of the network slice, and it returns an acknowledgement to the Orchestrator after the network slice activation phase is completed.

2.9.3 Once the slice is activated, the service provider can instantiate a (network) service through the ConfService.

Service providers can instantiate services, as well as network services, over a network slice. In both cases, the deployment is triggered by the service provider from the ConfService dashboard. However, the instantiation of network services and non-network services require different deployment flows.

On one hand, network services are instantiated over a network slice through SOE, which in turn interacts with OSM. In this manner, network services in Pledger are aligned with ETSI NFV standards. On the other hand, non-network services (such as the risk detection and notification service in UC2) are directly deployed over the underlying compute infrastructure. Therefore, prior to deployment, the service provider must manually select the target infrastructure, be it SOE-based (for the deployment of network services) or Kubernetes-based (for the deployment of other services); Figure 53 shows the *App Deployment Options* menu from the ConfService, where the service provider can manually enter the target infrastructure for their service.


App Deployment Options 3

Ranked options (lower values are better)

```
app: risk-detector

  service: risk-detector
  - ranking 1 : service deployed on
    -- node#31 'kubeedge' [edge] on infrastructure 'UC2-infra-30' with id:8
```

App
risk-detector

Figure 53: App deployment options

After the target infrastructure has been selected, the service provider needs to execute a two-step process in order to deploy their application. These two steps are common for the deployment of both network and other (non-network) services, and can be summarized as follows:

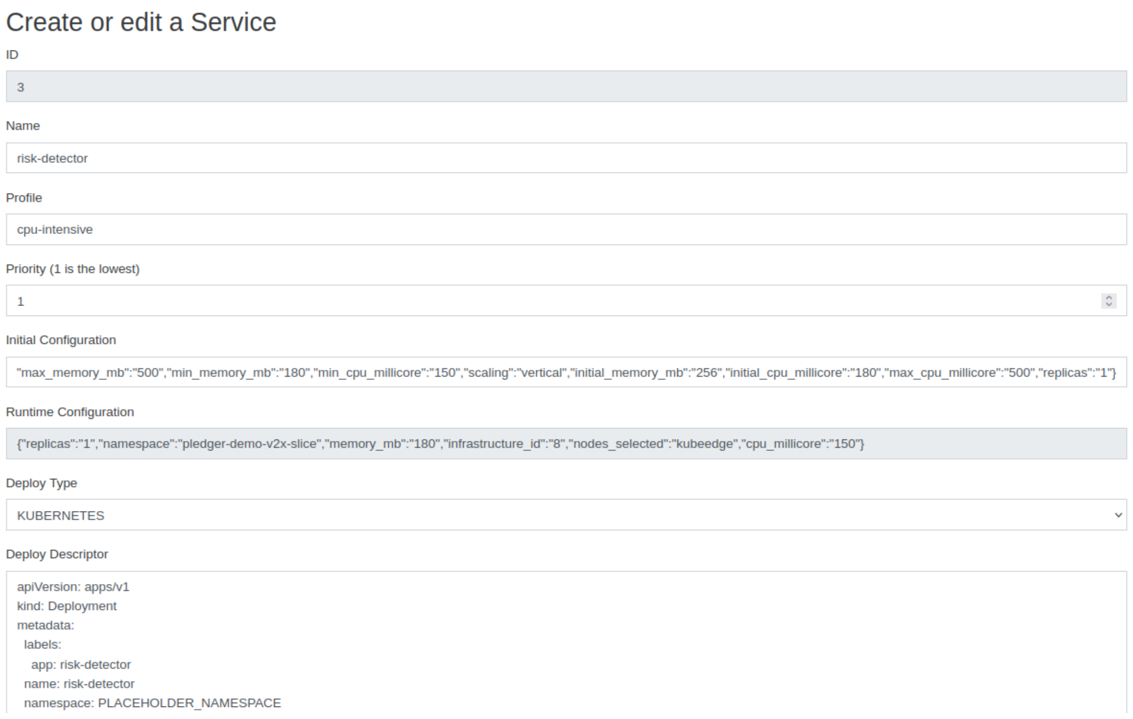
- Service creation and configuration: during this first step, the service provider inputs the service descriptor and relevant configuration parameters on the ConfService's *Service* dashboard, as shown in Figure 54.
- Application deployment: during this second step, the service provider manages the start and stop of an application composed of microservices through the ConfService's *App* dashboard, as represented in Figure 55.

The flow associated to the deployment of network services is as follows:

- After the service provider requests the deployment of the network service through the ConfService, a request is sent to the Orchestrator, which in turn sends an API call to the SOE Framework in order to request the deployment of the network service.
- The SOE Framework parses the deployment request, with relevant parameters to OSM, which encapsulates the service into a ETSI NFV-compliant network service. OSM then deploys the network service over the underlying Kubernetes infrastructure.

The flow associated to the deployment of other (non-network) services is as follows:

- After the service provider requests the deployment of the application through the ConfService, a request is sent to the Orchestrator, which communicates directly with the underlying Kubernetes infrastructure, and the application is then deployed.



Create or edit a Service

ID
3

Name
risk-detector

Profile
cpu-intensive

Priority (1 is the lowest)
1

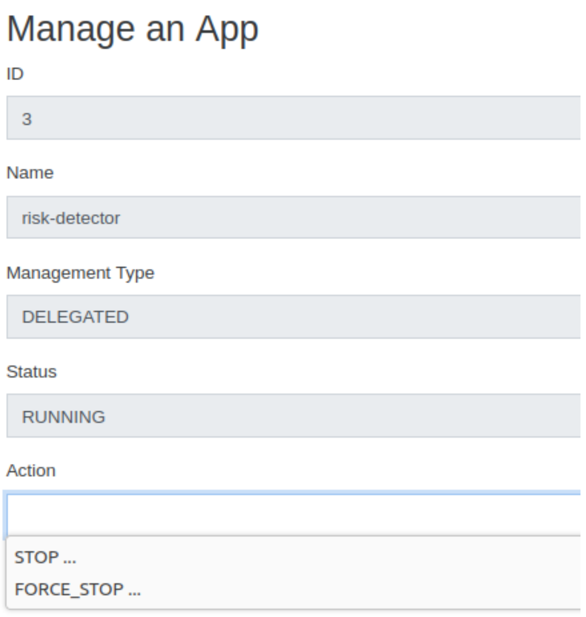
Initial Configuration
"max_memory_mb":"500","min_memory_mb":"180","min_cpu_millicore":"150","scaling":"vertical","initial_memory_mb":"256","initial_cpu_millicore":"180","max_cpu_millicore":"500","replicas":"1"}

Runtime Configuration
{"replicas":"1","namespace":"pledger-demo-v2x-slice","memory_mb":"180","infrastructure_id":"8","nodes_selected":"kubeedge","cpu_millicore":"150"}

Deploy Type
KUBERNETES

Deploy Descriptor
```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: risk-detector
  name: risk-detector
  namespace: PLACEHOLDER_NAMESPACE
```

Figure 54: Service creation and configuration



Manage an App

ID

3

Name

risk-detector

Management Type

DELEGATED

Status

RUNNING

Action

STOP ...

FORCE_STOP ...

Figure 55: Application management

2.10 Security configurations

2.10.1 An infrastructure provider can use PLEDGER IDPS for managing virtualized intrusion detection

In the context of Pledger project, a distributed cybersecurity network streaming platform has been implemented aiming to offer real time intrusion detection (IDS) and network security monitoring (NSM). The **business objectives** that drive this solution are to:

- Increase security, privacy and trust from the Edge to the Cloud
- Detect network threats and security anomalies and provide prompt alerts
- Provide sophisticated data analytics of the network traffic through a user-friendly UI

The architecture of the cybersecurity solution can be found in the following figure:

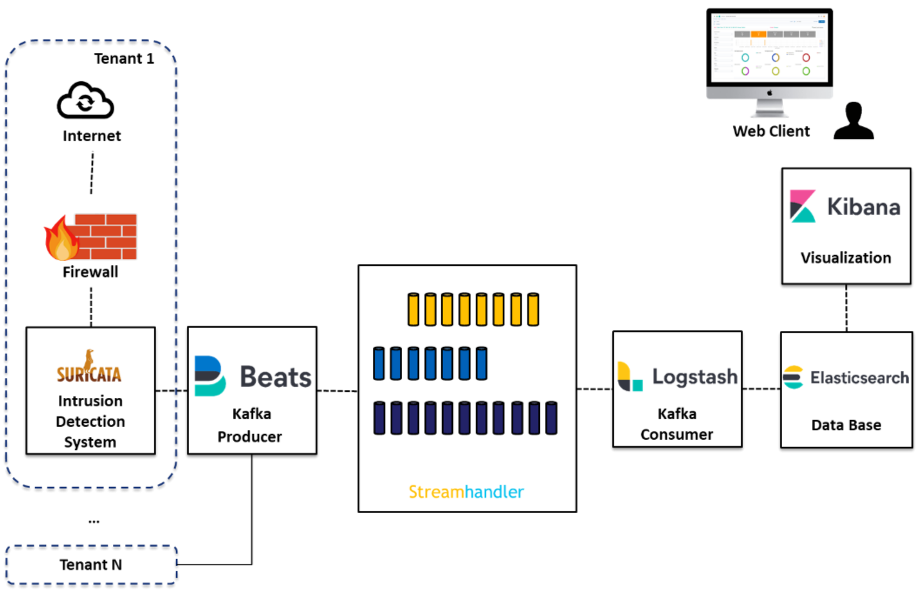


Figure 56: Distributed Cybersecurity platform architecture.

Our solution consists of the following software components:

- **Suricata**, which is a very popular open source, mature, fast, multi-threaded and robust network threat detection engine. The Suricata engine is capable of real time intrusion detection (IDS), inline intrusion prevention (IPS), network security monitoring (NSM) and offline log (in pcap format) processing. It inspects the network traffic using a powerful signature language to match on known threats, policy violations and malicious behavior. Moreover, it has a powerful Lua scripting support for detection of complex threats. With standard input and output formats like YAML and JSON integration with tools like existing SIEMs, Splunk, Logstash/Elasticsearch, Kibana and other frameworks, become effortless. Suricata constitutes the heart of the Cybersecurity solution as it acts as an IDS and NSM application.
- **Filebeat**, which is part of the ELK stack, belongs to Beats collection of lightweight data shippers for forwarding and centralizing log data in a distributed environment. Installed as an agent on your servers, Filebeat monitors the log files or locations that you specify, collects log events, and forwards them either to Elasticsearch or Logstash for indexing. In current solution, Filebeat acts as a Kafka Producer that collects JSON data generated by Suricata and forwards them to StreamHandler (Message Broker)

- **StreamHandler**, as already mentioned is a data streaming and analytics platform based on Apache Kafka offering:
  - Real-time monitoring and event-processing
  - Distributed messaging system
  - High fault-tolerance
  - Elasticity - High scalability
- **Logstash**, which is part of the ELK stack, is used to dynamically ingest, transform and ship data using data processing pipelines (input-filter-output). It can be used to ingest data of all shapes, sizes or sources, parse and transform data on the fly and route data to a great variety of outputs.
  In current solution, Logstash acts as a Kafka Consumer that is responsible for:
  - Collecting data from StreamHandler (Message Broker).
  - Decoding, parsing, formatting and enriching of data.
  - Forwarding data to Elasticsearch
- **Elasticsearch**, which is part of the ELK Stack, is an open-source, distributed, RESTful search and analytics engine that centrally stores your data for lightning-fast search, fine tuned relevancy, and powerful analytics that scale with ease. In current solution, Elasticsearch acts as a time series Data Base, where all network traffic from all distributed servers is centrally persisted.
- **Kibana**, which is part of the ELK Stack, is used for data visualization using custom visualization panels & dashboards. It has built-in support for authentication and authorization and a great variety of tools for interacting with Elasticsearch.

This architecture is specifically designed to support a decentralized approach in the deployment of cybersecurity appliances on the edge (virtual Intrusion Detection, virtual Honeypot, virtual Deep Packet Inspection etc.). It then allows the project to aggregate all the network logs and threat information into a SIEM-like interface based on the ELK Stack. The use of the Kafka-based Streamhandler platform ensures that high throughput can be achieved, thus facilitating the data ingestion process from multiple instances across the Cloud-Edge continuum. This solution is designed to be able to aggregate data from multiple compute tenants/slices, as dedicated security services run in each slice can stream their results through Kafka providing an operational picture across the infrastructure that allows threats to be afficiently addressed by the Pledger provider.

2.10.1.1 IDS Modes

Pledger examines and implements two different IDS approaches, that both add significant value to gain clear insight about potential network threats and therefore help security analysts to apply efficient mitigation actions. Both approaches are compliant with the decentralised architecture, although the deployment and configuration of the IDS differs in order to provide two distinct functionalities.

Threat Detection mode

In the Thread Detection mode, as it can be seen in the following figure, the IDS component analyzes only the traffic that is accepted by the applied Firewall rules. In this way, administrators can detect any severe security vulnerabilities that have to be addressed. The IDS acts as part of the perimeter defenses and identifies threats within the cloud infrastructure.

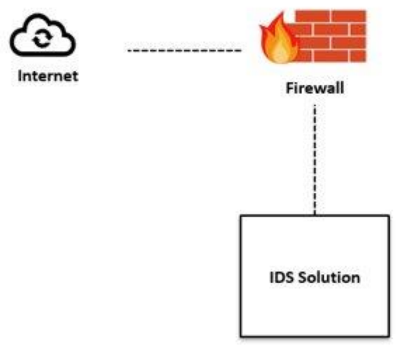


Figure 57: Threat Detection Mode – IDS

Honeypot Mode

In computer security terms, a cyber honeypot is a sacrificial computer system that is intended to attract cyberattacks. It mimics a target for hackers, and uses their intrusion attempts to gain information about cybercriminals and the way they are operating or to distract them from other targets.

In the Honeypot mode, IDS follows a different deployment and configuration in order to log the threats and is attached directly to the network interface(s), before any firewall rules are applied, as it can be seen in the following figure. In this way, IDS can be configured properly to capture all the external network traffic, including network threats and attacks, so that they can be studied in order to improve the applied security policies.

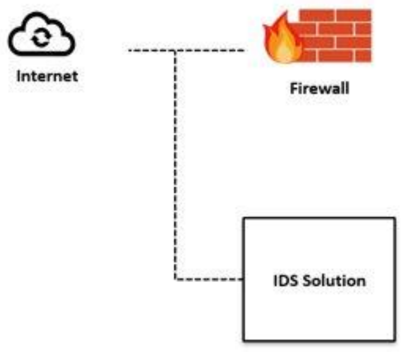


Figure 58: Honeypot Mode – IDS

Usage information

The user may connect to the ELK stack and visualise the aggregated data directly as shown in the figure below. The threat screen provides more information on the individual threats. A geographical indication of the origin of offending IP addresses is also available. As seen in these screens, users may select to filter the information shown on screen per IDS instance (threat detection, or honeypode mode), service, protocol, country, alert, threat severity etc.

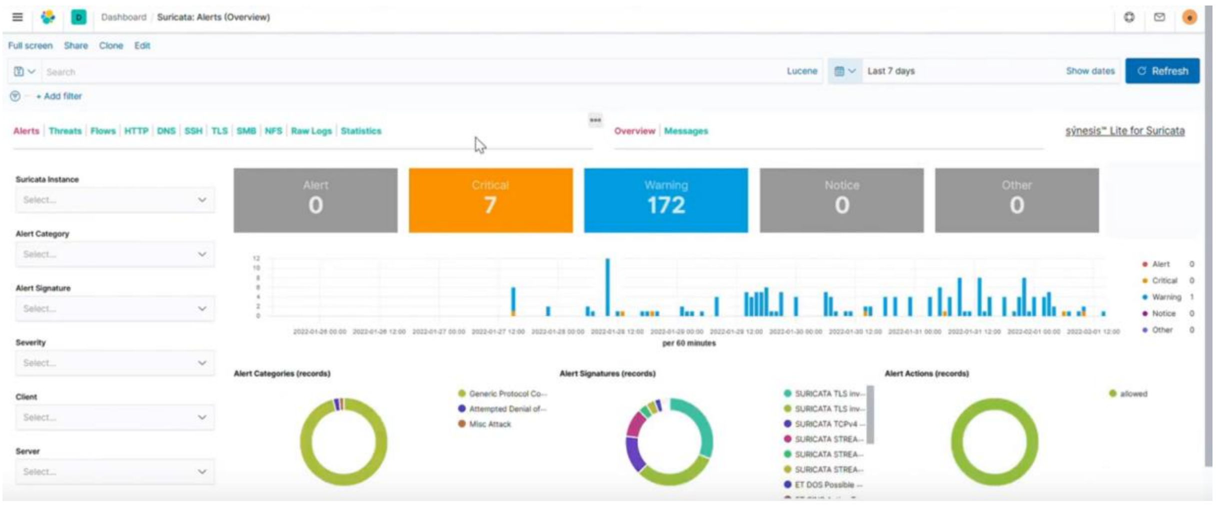
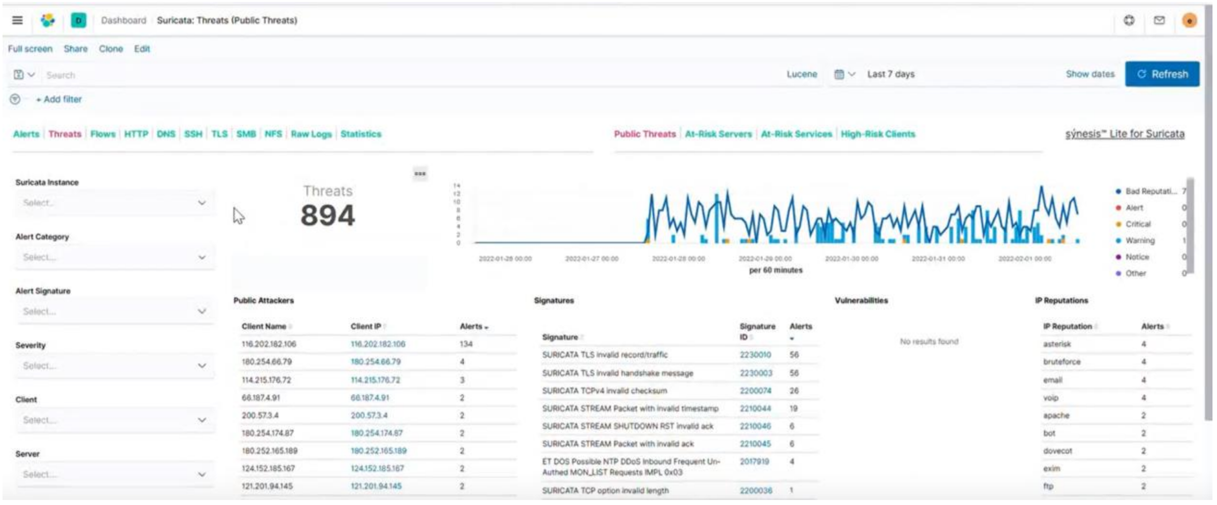


Figure 59: Main alerts page



Public Attackers

Client Name	Client IP	Alerts
116.202.182.106	116.202.182.106	134
180.254.66.79	180.254.66.79	4
114.215.176.72	114.215.176.72	3
66.187.4.91	66.187.4.91	2
200.57.3.4	200.57.3.4	2
180.254.174.87	180.254.174.87	2
180.252.165.189	180.252.165.189	2
124.152.185.167	124.152.185.167	2
121.201.94.145	121.201.94.145	2

Signatures

Signature	Signature ID	Alerts
SURICATA TLS invalid record/traffic	2230010	56
SURICATA TLS invalid handshake message	2230003	56
SURICATA TCPv4 invalid checksum	2200074	26
SURICATA STREAM Packet with invalid timestamp	2210044	19
SURICATA STREAM SHUTDOWN RST invalid ack	2210046	6
SURICATA STREAM Packet with invalid ack	2210045	6
ET DOS Possible NTP DDoS Inbound Frequent Un-Authed MON_LIST Requests IMPL 0x03	2017919	4
SURICATA TCP option invalid length	2200036	1

Vulnerabilities

No results found

IP Reputations

IP Reputation	Alerts
asterisk	4
bruteforce	4
email	4
voip	4
apache	2
bot	2
dovecot	2
exim	2
ftp	2

Figure 60: Threats page

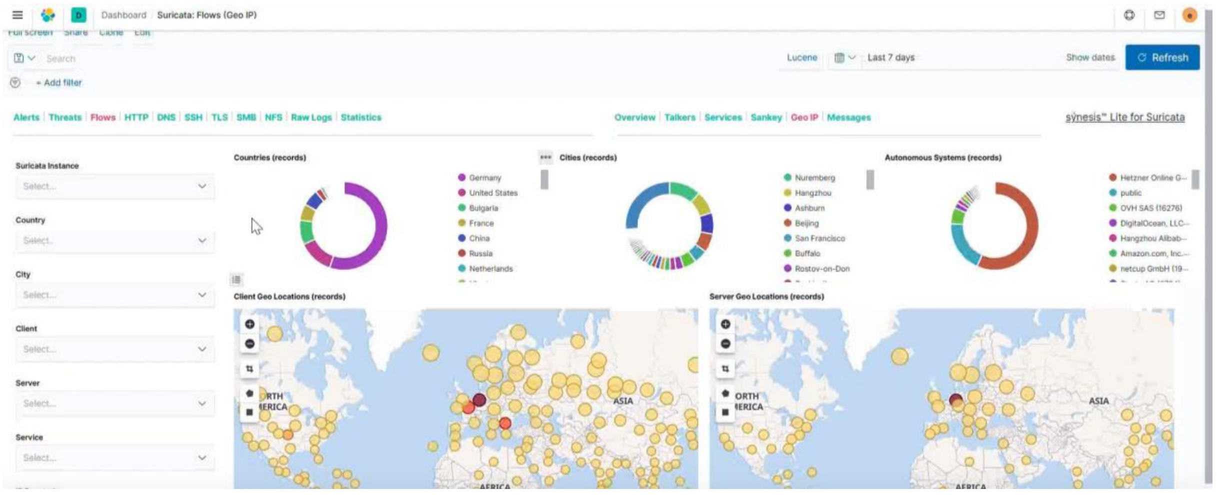


Figure 61: Flows (geographical origin)

2.10.1.2 Deployent and configuration

The components of the distributed cybersecurity network streaming platform are containerized, so their deployment is controlled through **docker-compose** configuration files, controlling the definition of the different services, the network among them and their associated volumes. The possibility of instantiating the application with or without an ELK stack at the same host machine is offered.

It is also straightforward to configure the operation mode of the IDS instance (threat detection, or honeypot mode) by modifying an environment file accordingly. For the deployment in threat detection mode, appropriate firewall rules should also be applied (for e.g., through linux iptables) at the corresponding host where the IDS instance is deployed. Combining an IDS instance in threat detection with one in honeypot mode (sharing a common ELK instance) can be used to compare and assess the performance of the IDS system, by indicating which types of threats are blocked by the firewall in threat detection mode, and which ones can still pass the firewall. This was one of the subjects of the corresponding IDS demonstration that is available in PLEDGER YouTube channel here.

2.11 SLA violation

2.11.1 Monitoring engine gathers and stores data about an application health.

The Monitoring Engine is used by the SLA Framework to connect to the correspondent infrastructure monitor (i.e. Prometheus instances deployed in the different infrastructures) to gather the metrics values defined in the SLAs guarantees

2.11.2 The data is analysed by the SLA Manager to identify if there's any breach in the application guarantees.

The SLAs evaluation is done every minute, and when it detects a violation (see figure Figure 37) it sends a notification to other PLEDGER components via Kafka.

2.11.3 If there is any violation detected, the SLA Manager sends this information to the Blockchain Framework and the DSS to take any corrective action.

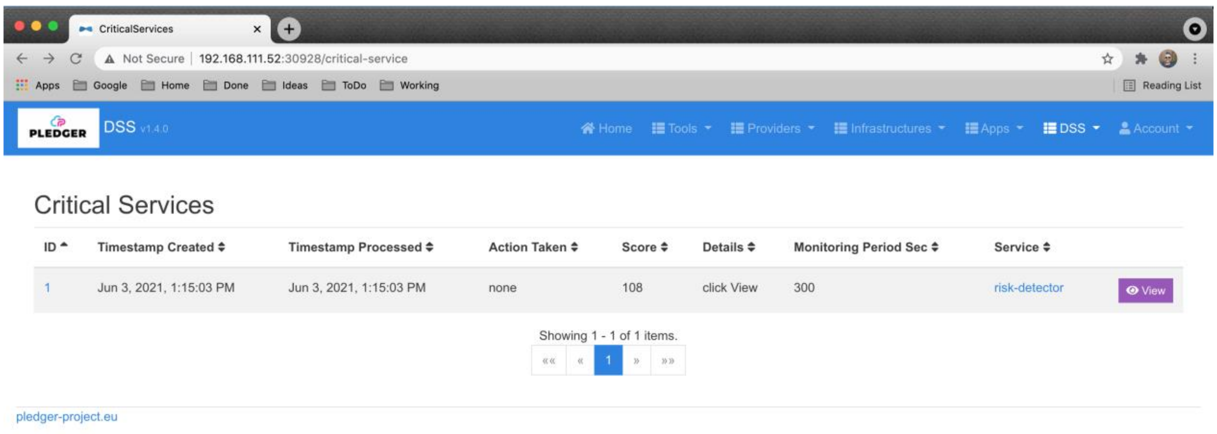
When a violation is detected, the SLA Framework sends a message to Kafka (see figure Figure 38). This message can be read from other PLEDGER components, like the Blockchain Framework and the DSS.

2.12 Application redeployment

The objective of this section is to show how the DSS continuously monitor the system and command new actions to the orchestrator. This section is the same as 2.6.1, with the DSS continuously monitoring the data coming from SLA Lite, Monitoring Engine etc. and applying re-deployment strategies.

2.12.1 Whenever an SLA violation occurs and the DSS is notified, it takes the decision of redeploying an application in order not to lose QoS.

With the continuous monitoring done by the DSS, QoS is maintained continuously offloading, staling and adjusting resources allocated to the DApps. As an example, Figure 62 shows a service, considered critical, as a SLA violation has been received and offloading or scaling is required; similarly, Figure 63 shows a service considered "steady" as no SLA violations are received for some time, so a reduction of resources can be done, eventually leading to offloading back to the edge.


CriticalServices
Not Secure | 192.168.111.52:30928/critical-service
Apps Google Home Done Ideas ToDo Working
Reading List
PLEDGER
DSS v1.4.0
Home Tools Providers Infrastructures Apps DSS Account

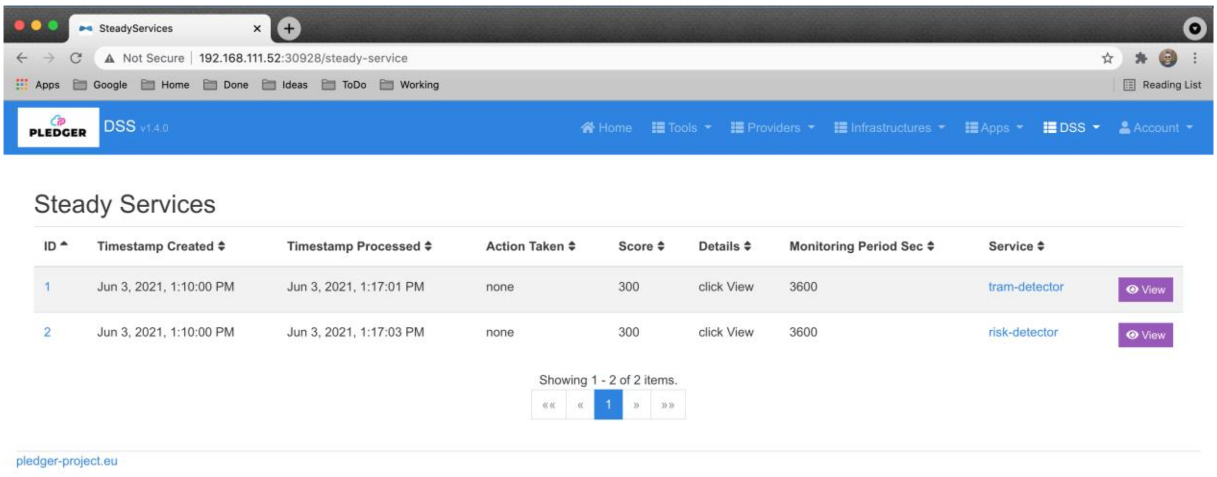
Critical Services

ID	Timestamp Created	Timestamp Processed	Action Taken	Score	Details	Monitoring Period Sec	Service	
1	Jun 3, 2021, 1:15:03 PM	Jun 3, 2021, 1:15:03 PM	none	108	click View	300	risk-detector	View

Showing 1 - 1 of 1 items.

pledger-project.eu

Figure 62: DSS: critical service found, that might require offload


SteadyServices
Not Secure | 192.168.111.52:30928/steady-service
Apps Google Home Done Ideas ToDo Working
Reading List
PLEDGER
DSS v1.4.0
Home Tools Providers Infrastructures Apps DSS Account

Steady Services

ID	Timestamp Created	Timestamp Processed	Action Taken	Score	Details	Monitoring Period Sec	Service	
1	Jun 3, 2021, 1:10:00 PM	Jun 3, 2021, 1:17:01 PM	none	300	click View	3600	tram-detector	View
2	Jun 3, 2021, 1:10:00 PM	Jun 3, 2021, 1:17:03 PM	none	300	click View	3600	risk-detector	View

Showing 1 - 2 of 2 items.

pledger-project.eu

Figure 63: DSS: steady service found, that might require a scale down

For more information, Table 3 reports the DSS optimization demos available on PLEDGER YouTube channel.

Title	Link
DSS_#2: DSS increasing resources and offloading to the cloud and back to the edge	https://www.youtube.com/watch?v=vfohE_d6XfA
DSS_#3: “DSS installation from source code	https://www.youtube.com/watch?v=qpyJC9fU7aw
DSS_#4: “KinD cloud-edge K8S environment setup and DSS manual operations	https://www.youtube.com/watch?v=haUBihzrxdc
DSS_#5: “DSS scaling up/down based on SLA violations	https://www.youtube.com/watch?v=2q_dVuwUS9w
DSS_#6: “DSS scaling out based on SLA violations	https://www.youtube.com/watch?v=i5kQy4HOceA
DSS_#7: “DSS offloading to the cloud based on SLA violations	https://www.youtube.com/watch?v=q00eCdPXY2c
DSS_#8: “DSS optimising latency on cloud-edge K8S using ECODA	https://www.youtube.com/watch?v=360YbWvu7YU
DSS_#9: “DSS optimising latency on cloud-edge K8S using ECODA and SLA violations	https://www.youtube.com/watch?v=YEx__AI44h8
DSS_#10: “KinD cloud-edge-faredge K8S environment setup	https://www.youtube.com/watch?v=5OI8X9zq-bU
DSS_#11: “DSS optimising latency on cloud-edge-faredge K8S using TTODA	https://www.youtube.com/watch?v=6lR6IgEuwCo
DSS_#12: “DSS optimising latency on cloud-edge-faredge K8S using TTODA and SLA violations	https://www.youtube.com/watch?v=J7T4QODMotM
DSS_#13: “EA-ECODA optimisation	https://www.youtube.com/watch?v=VSCBOREAk2E

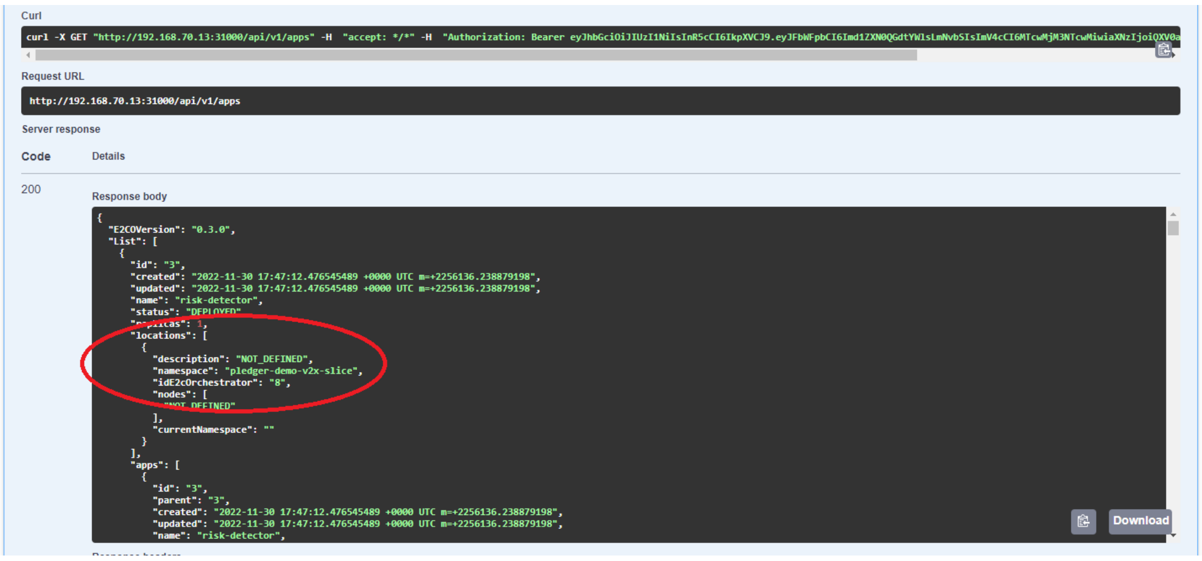
Table 3: DSS optimisation demos

2.12.2 DSS notifies the Orchestrator.

This section is the same as 2.8, with the DSS communicating a new action to the executed by the Orchestrator.

2.12.3 The Orchestrator stops the application, deploys it in a different node and starts it again.

E2CO application connects to the new infrastructure to deploy there the application. Then, it removes the application from the old infrastructure. The status of the application and the infrastructure where it is deployed and running is shown in the E2CO swagger UI (Figure 64) and also in the ConfService application.



Curl

```
curl -X GET "http://192.168.70.13:31000/api/v1/apps" -H  "accept: */*" -H  "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJFbWFpbCI6Imd1ZXN0QGdtYWlsLmNvbSIsImV4cCI6MTcwMjM3NTcwMiwiaXNzIjoiQXV0a
```

Request URL

```
http://192.168.70.13:31000/api/v1/apps
```

Server response

Code Details

200

Response body

```
{
  "E2COVersion": "0.3.0",
  "List": [
    {
      "id": "3",
      "created": "2022-11-30 17:47:12.476545489 +0000 UTC m=+2256136.238879198",
      "updated": "2022-11-30 17:47:12.476545489 +0000 UTC m=+2256136.238879198",
      "name": "risk-detector",
      "status": "DEPLOYED",
      "replicas": 1,
      "locations": [
        {
          "description": "NOT_DEFINED",
          "namespace": "pledger-demo-v2x-slice",
          "idE2cOrchestrator": "8",
          "nodes": [
            "NOT_DEFINED"
          ],
          "currentNamespace": ""
        }
      ],
      "apps": [
        {
          "id": "3",
          "parent": "3",
          "created": "2022-11-30 17:47:12.476545489 +0000 UTC m=+2256136.238879198",
          "updated": "2022-11-30 17:47:12.476545489 +0000 UTC m=+2256136.238879198",
          "name": "risk-detector",
```

Figure 64: E2CO Swagger UI: Application status

2.13 Penalties

In case that an SLA is breached, the corresponding violation notification reaches the blockchain network and its deployed refunding mechanisms. Automatically, these mechanisms are triggered and the refunding processes unfolds with regards to the severity of the violation occurred. In the following figures, the different steps of the penalty workflow are depicted. In simple terms they consist of:

- Blockchain receiving violation notifications
- User balance impacted
- Violations stored securely on the DLT

2.13.1 The Blockchain Framework receives the same notification and based on the severity level, decides the penalty to be applied to the infrastructure provider.

As soon as violation notifications reach the DLT (similar instance in Figure 45), the refund mechanism are automatically triggered and the corresponding users balances are impacted according to the severity of the violation.

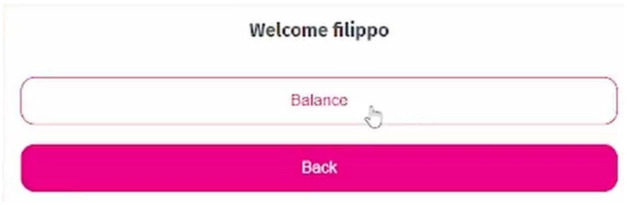


Figure 66: A logged in user checks their balance

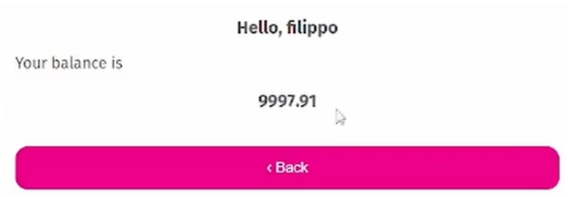
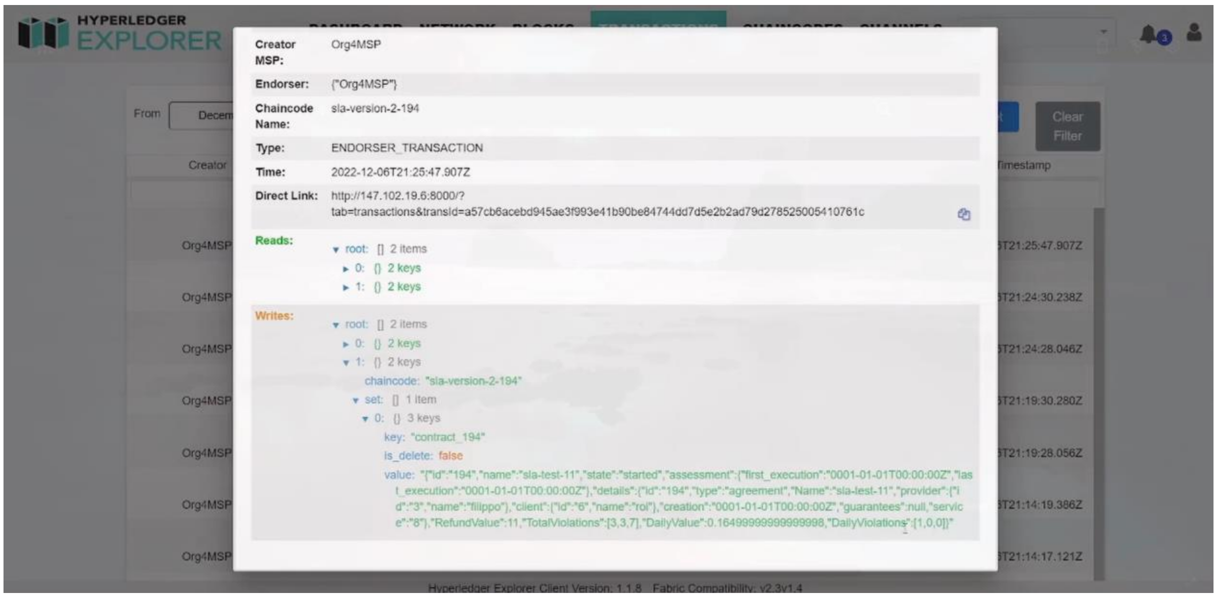


Figure 65: User balance has been impacted due to SLA breach.

2.13.2 This information is available within the smart contract.

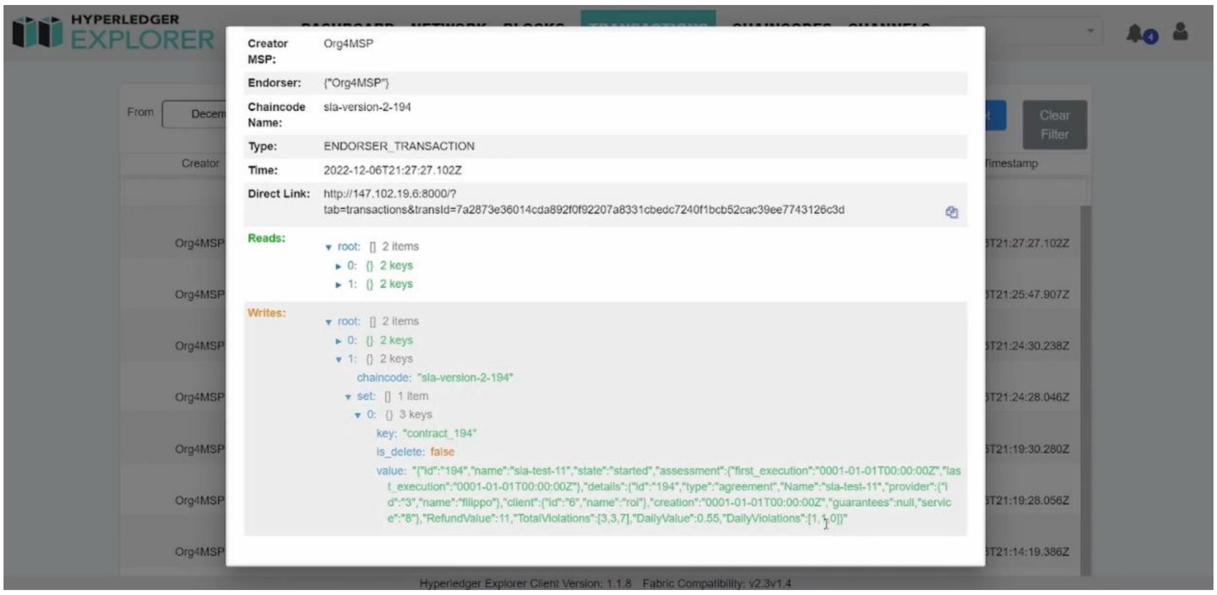
The corresponding result of the refund operations include the account balances, the user accounts addresses, the compensation amounts, and the related violation data. This result is securely stored on the DLT. The following figures showcase the storing on the chain of three (3) types of violations:

- Warning
- Serious
- Catastrophic



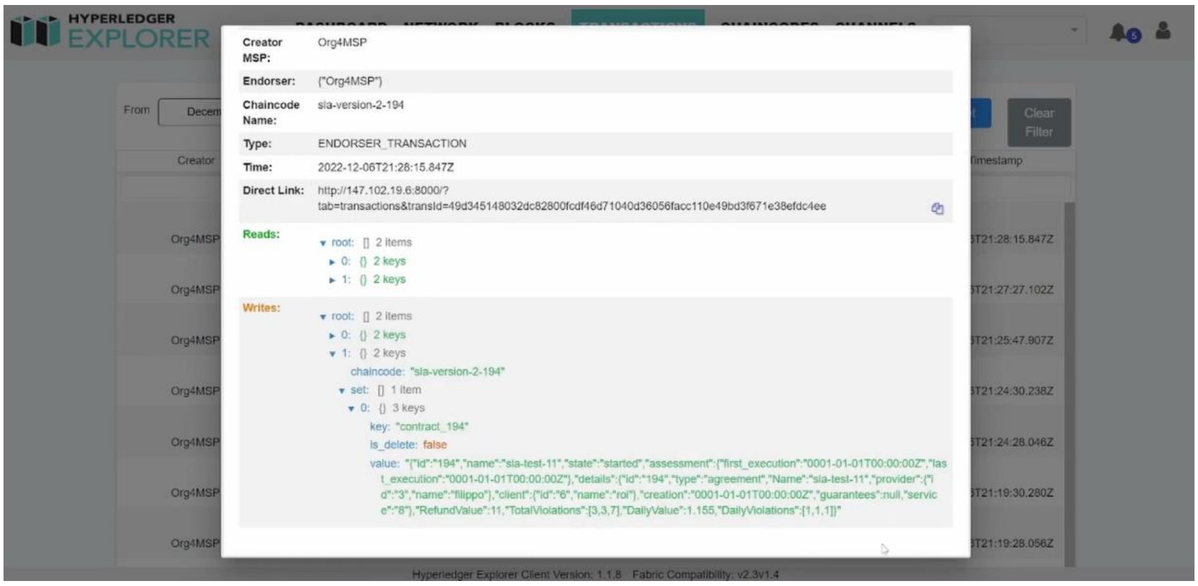
```
Creator MSP:    Org4MSP
Endorser:       {"Org4MSP"}
Chaincode Name: sla-version-2-194
Type:           ENDORSER_TRANSACTION
Time:           2022-12-06T21:25:47.907Z
Direct Link:    http://147.102.19.6:8000/?
                tab=transactions&transId=a57cb6acebd945ae3f993e41b90be84744dd7d5e2b2ad79d278525005410761c
Reads:          ▾ root: [] 2 items
                  ▸ 0: {} 2 keys
                  ▸ 1: {} 2 keys
Writes:         ▾ root: [] 2 items
                  ▸ 0: {} 2 keys
                  ▾ 1: {} 2 keys
                      chaincode: "sla-version-2-194"
                    ▾ set: [] 1 item
                      ▾ 0: {} 3 keys
                          key: "contract_194"
                          is_delete: false
                          value: "{"id":"194","name":"sla-test-11","state":"started","assessment":{"first_execution":"0001-01-01T00:00:00Z","las
                            t_execution":"0001-01-01T00:00:00Z"},"details":{"id":"194","type":"agreement","Name":"sla-test-11","provider":{"i
                            d":"3","name":"filippo"},"client":{"id":"6","name":"rol"},"creation":"0001-01-01T00:00:00Z","guarantees":null,"servic
                            e":"8"},"RefundValue":11,"TotalViolations":[3,3,7],"DailyValue":0.16499999999999998,"DailyViolations":[1,0,0]}"
```

Figure 67: A warning violation stored on the DLT



```
Creator MSP:    Org4MSP
Endorser:       {"Org4MSP"}
Chaincode Name: sla-version-2-194
Type:           ENDORSER_TRANSACTION
Time:           2022-12-06T21:27:27.102Z
Direct Link:    http://147.102.19.6:8000/?
                tab=transactions&transId=7a2873e36014cda892f0f92207a8331cbedc7240f1bcb52cac39ee7743126c3d
Reads:          ▾ root: [] 2 items
                  ▸ 0: {} 2 keys
                  ▸ 1: {} 2 keys
Writes:         ▾ root: [] 2 items
                  ▸ 0: {} 2 keys
                  ▾ 1: {} 2 keys
                      chaincode: "sla-version-2-194"
                    ▾ set: [] 1 item
                      ▾ 0: {} 3 keys
                          key: "contract_194"
                          is_delete: false
                          value: "{"id":"194","name":"sla-test-11","state":"started","assessment":{"first_execution":"0001-01-01T00:00:00Z","las
                            t_execution":"0001-01-01T00:00:00Z"},"details":{"id":"194","type":"agreement","Name":"sla-test-11","provider":{"i
                            d":"3","name":"filippo"},"client":{"id":"6","name":"rol"},"creation":"0001-01-01T00:00:00Z","guarantees":null,"servic
                            e":"8"},"RefundValue":11,"TotalViolations":[3,3,7],"DailyValue":0.55,"DailyViolations":[1,1,0]}"
```

Figure 68: A serious violation stored on the DLT



Creator MSP:	Org4MSP
Endorser:	("Org4MSP")
Chaincode Name:	sla-version-2-194
Type:	ENDORSER_TRANSACTION
Time:	2022-12-06T21:28:15.847Z
Direct Link:	http://147.102.19.6:8000/?tab=transactions&transId=49d345148032dc82800fcdf46d71040d36056facc110e49bd3f671e38efdc4ee

Reads:

```
▼ root: [] 2 items
  ▸ 0: {} 2 keys
  ▸ 1: {} 2 keys
```

Writes:

```
▼ root: [] 2 items
  ▸ 0: {} 2 keys
  ▼ 1: {} 2 keys
      chaincode: "sla-version-2-194"
    ▼ set: [] 1 item
      ▼ 0: {} 3 keys
          key: "contract_194"
          is_delete: false
          value: "{"id":"194","name":"sla-test-11","state":"started","assessment":{"first_execution":"0001-01-01T00:00:00Z","last_execution":"0001-01-01T00:00:00Z"},"details":{"id":"194","type":"agreement","Name":"sla-test-11","provider":{"id":"3","name":"filippo"},"client":{"id":"6","name":"roi"},"creation":"0001-01-01T00:00:00Z","guarantees":null,"service":"8"},"RefundValue":11,"TotalViolations":[3,3,7],"DailyValue":1.155,"DailyViolations":[1,1,1]}"
```

Hyperledger Explorer Client Version: 1.1.8 Fabric Compatibility: v2.3v1.4

Figure 69: A catastrophic violation stored on the DLT

3 Applicability examples

In order to fully grasp how PLEDGER can be deployed and what is the added value of this software, applicability examples are presented in this section. These applicability examples showcase how different components of the pledger solution are incorporated in specific Use case pipelines.

3.1 Edge infrastructure for enhancing safety vulnerable road users

The ever-growing popularity of micro-mobility transportation in cities has introduced new challenges related to the safeguarding of the safety of pedestrians and road users: these include the drafting of new legislation to accommodate the increasing number of scooters and bicycles, the adaptation of city infrastructures with dedicated lanes for micromobility users, etc.

The main goal of this use case is to reduce the number of accidents on the road, especially those related to the use of micro-mobility transportation in cities, by introducing the use of dedicated edge infrastructure to deploy a service that can help to increase the safety of vulnerable road users (VRUs) about risky situations. Applied to the specific use case scenario developed in Pledger, the so-called risk detection and notification system (RDNS) alerts VRUs of potential risks as they approach a tram station while pedestrians are entering or exiting the tram.

In this use case, information about the location of electric scooters is gathered via IEEE 802.11p radio communication. Vehicular data coming from on-board units (OBUs) attached to electric scooters is shared with the RDNS via road-side units (RSUs). Through the RSUs, OBUs can communicate with the RDNS, which runs in the compute infrastructure. Based on the scooters' location data, which is obtained from the OBUs via vehicular communications, the RDNS detects whether trams and scooters are simultaneously located by (or approaching) the vicinity of a tram station; the RDNS then sends a notification to electric scooter users, encouraging them to exercise precaution when navigating through a potentially crowded area as passengers enter and leave the tram. For that purpose, a custom gadget has been developed which sends out an audio-visual warning signal (buzzer + LED) to scooter riders.

In order to implement all required features for this use case, a number of integrations with Pledger components have been performed, as described below.

3.1.1 Infrastructure management: network slicing.

This use case relies on the use of a city-wide infrastructure, which is distributed and shared among different parties. In addition, this infrastructure is heterogeneous, as it includes radio nodes and compute nodes, which are managed differently. This increased complexity can be managed by performing infrastructure reservation through network slicing.

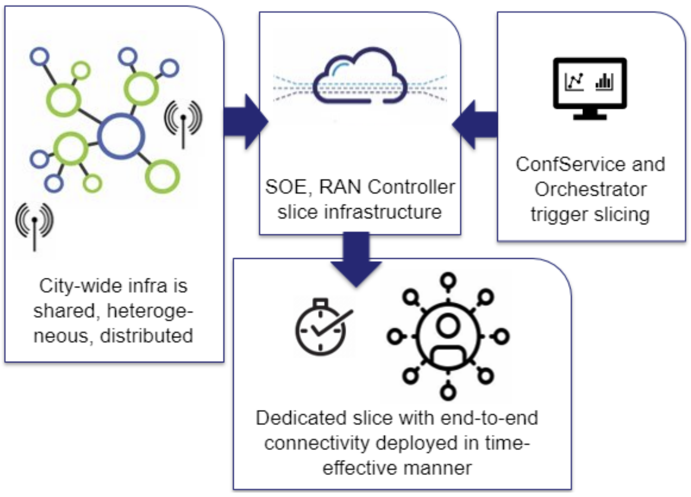

City-wide infra is shared, heteroge-neous, distributed
SOE, RAN Controller slice infrastructure
ConfService and Orchestrator trigger slicing
Dedicated slice with end-to-end connectivity deployed in time-effective manner

Figure 70: Network slicing workflow in the use case

In Pledger, dedicated network slices can be provisioned at the touch of a button, as represented in Figure 70. In Pledger, the modules involved in the process of network slicing are the SOE, RAN Controller, ConfService and Orchestrator. These modules perform the provisioning of a network slice over the use case's cloud native infrastructure, providing end-to-end connectivity and automated configuration of radio resources, as well as soft resource isolation. Slices are provisioned from the ConfService's dashboard at the touch of a button, as previously shown in Figure 51 and Figure 52; in this use case, however, vehicle-to-everything (V2X) slices with IEEE 802.11p radios are deployed instead of 5G slices.

Figure 71 captures the Pledger modules and dependencies that enable end-to-end network slicing. After the slice provisioning request is activated, the ConfService sends a slice creation request to the Orchestrator, along with a relevant set of configuration parameters, using Kafka. Next, the Orchestrator sends a flow of API calls to the SOE's northbound interface. The SOE creates a compute chunk and a network chunk over the underlying infrastructure, and it interacts with the RAN Controller for the creation of a radio chunk and the activation of the network slice. In this use case, the activation of the network slice requires the deployment of a containerized V2X software stack, i.e., a set of software components that enable communication between the road side units (which contain the radio elements) and the RDNS.

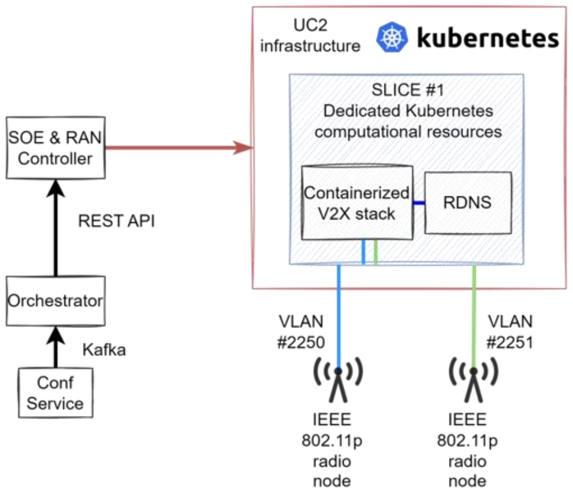


Figure 71: End-to-end V2X slicing - integration diagram

Through Pledger, the provisioning of network slices is a simple, seamless and very time-effective process: as a reference, it takes 61 seconds to provision a V2X network slice through Pledger, an improvement of over 11 times of the time required for an equivalent manual deployment. In addition, manual deployments must be performed by a highly skilled technician with a broad knowledge of networking topics; in contrast, no such expertise is required when provisioning network slices from Pledger.

3.1.2 Smart contracts

In this use case, the RDNS should ideally be always available, i.e. the pods that compose it need to be up and running and responsive, so that risky situations can always be detected and alerted to road users. However, on occasions, the availability of the application is not guaranteed due to external factors.

In Pledger, the application availability is constantly monitored by the Monitoring Engine. When an SLA violation occurs, the SLA Lite detects it and sends a violation alert that is fed to a smart contract (that has been established beforehand among interested parties) through Kafka. A suitable refund policy is defined and automatically applied, according to the severity of the SLA violation, and the resulting refund is added to the smart wallet's balance. In this use case, refunds are calculated and applied once a day; however, the implementation of refund policies in Pledger is very flexible and highly customizable. A top-level diagram of this functionality is represented in Figure 72.

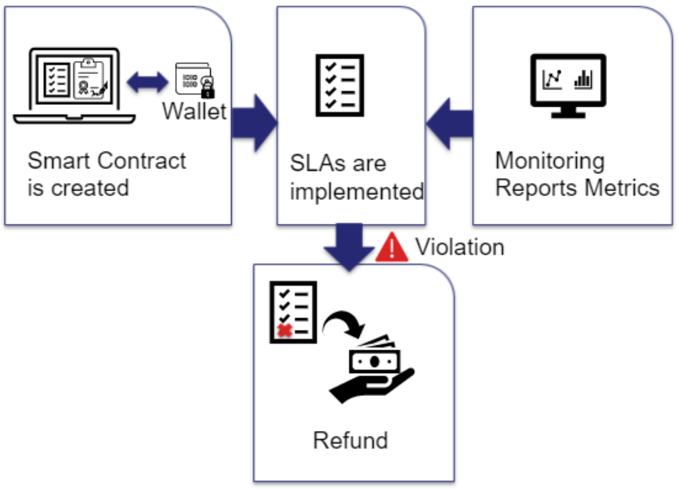


Figure 72: Pledger's smart contracts in road safety use case

Pledger's functionalities related to the issuing of smart contracts bring several advantages to this use case. Relevant refund-related metrics are automatically and continuously monitored in a manner that is transparent to the infrastructure and service providers. In addition, refunds are automatically calculated by the smart contracts according to some previously implemented refund policies, then added to the smart wallet's balance.

3.1.3 Sensitive information management with Pledger’s DLT

The RDNS generates user-related information containing the location of users at a certain instant, detected collision risk events, and corresponding timestamps. This information is documented in logs that must be securely handled. In Pledger, these logs are securely pushed to Kafka and retrieved by the DLT. By subscribing to the relevant Kafka topic, Pledger's DLT gathers RDNS logs, then securely stores them in the blockchain. Only authorized users have access to relevant and anonymized statistics. For example, authorized users can retrieve the number of risky situations that arose during a given time window. In this manner, data is securely stored, ensuring the users' privacy rights are protected. This functionality is represented in Figure 73.

The use of Pledger for the management of sensitive information brings several advantages to this use case. By storing sensitive data in the blockchain, the DLT ensures the protection of users' data related to the risk detection and notification service. In addition, this data is anonymized and can be accessed by authorized users only.

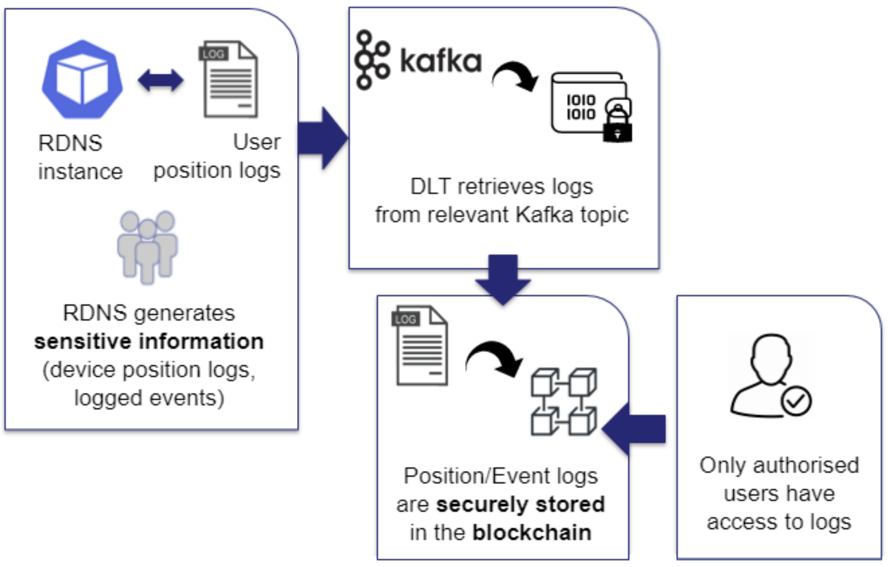

RDNS instance
User position logs
RDNS generates **sensitive information** (device position logs, logged events)
kafka
DLT retrieves logs from relevant Kafka topic
Position/Event logs are **securely stored** in the **blockchain**
Only authorised users have access to logs

Figure 73: Management of sensitive user information

3.1.4 Delay management with Pledger's DSS

Road hazard signaling applications rely on the timely execution and reception of awareness and notification messages. The maximum allowable round-trip delay is standardized by ETSI, such that users are notified of risky situations in a timely manner.

When the RDNS is running with a large number of end users, the computational resources assigned to the application need to be large enough in order to keep the delay low. In Pledger, the monitoring engine continuously monitors the RDNS delay. In addition, delay thresholds have been defined such that SLA violations are reported when these thresholds are crossed. Upon delay violations, the DSS performs a scaling up of the resources assigned to the risk detection and notification service, such that the delay returns to a lower value. In order to use resources efficiently, the DSS also performs a scaling down of the resources assigned to the application if no violations occur during a given period of time. This flow is represented in Figure 74.

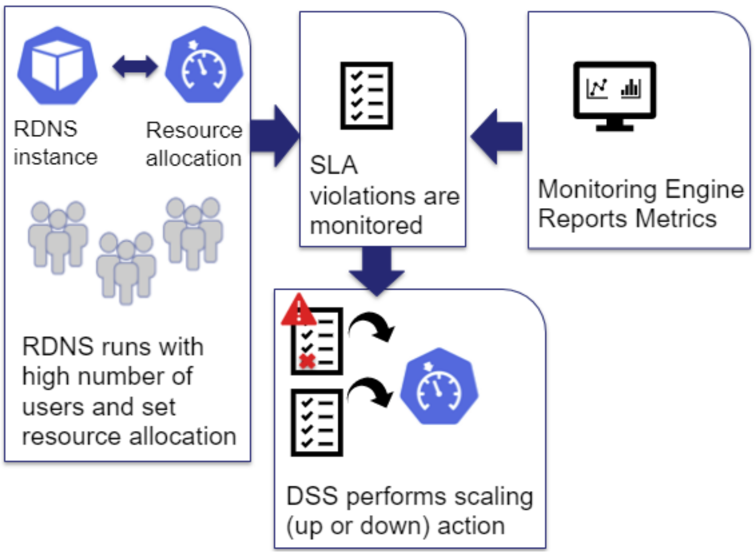

RDNS instance
Resource allocation
RDNS runs with high number of users and set resource allocation
SLA violations are monitored
Monitoring Engine Reports Metrics
DSS performs scaling (up or down) action

Figure 74: Management of a critical RDNS service in Pledger

The use of Pledger for the management of critical services brings several advantages to this use case. It guarantees the quality of service of the RDNS, ensuring that road safety application requirements are met while using the available computational resources in an efficient manner (for example, by liberating resources when the delay is consistently low). The resiliency of the application is also improved, by increasing allocated resources as needed.

The RDNS delay performance has been tested under three different scenarios. In the first scenario, the application runs with fixed, sometimes scarce, resources. In the second scenario, the application runs with increased (but still fixed) resources. In the third scenario, the DSS' scaling algorithm is applied, and the application runs with variable resources. It was demonstrated that the application of Pledger's DSS algorithms translates into a more consistent quality of service and user experience, as the delay variance is much lower than in the other two tested scenarios. The average resource utilisation is reduced, and the average delay is improved by a factor of 2.75 with respect to the second test scenario, and by a factor of 4.8 with respect to the first test scenario.

In summary, the dynamic resource allocation capabilities of Pledger enable the efficient use of computational resources. In addition, the application behavior is more consistent due to the smaller delay variance when Pledger is used. Moreover, the quality of service of the RDNS is guaranteed by increasing the resources to the application when the delay deteriorates. In this manner, the road safety application requirements are met, even when the computational resources are under stress.

3.2 Manufacturing the data mining on edge

The UC3 supports manufacturing experts to determine the process stability, enabling the improvement of it in the area of metal processing of highest accuracy. Using the developed analytical services, the expert is able to analyse the process and find deviations in the process in three regards:

- Stability of media consumed by the machine (pneumatic and electric)
- Stability of parts produced by the machine in terms of cycle time and quality
- Thermal stability

The determination of the thermal stability is of high interest for the machine operator. The (metal) components of the machine are extending due to thermal influence during the machine movements until they reach a plateau. A module was developed to estimate a thermal stability performance indicator and the subsequent estimation about the thermal condition of the machine. The result is an instruction to the machine about the further proceedings (continue warm-up, go into production mode), which is transferred directly to the Programmatic Logical Control (PLC) on which the machine reacts within the time of one controller cycle.

Furthermore, modules to evaluate the media-consumption and parts produced by the machine were developed to enable the analysis of the process for a manufacturing expert. They are deployed on the industrial PC, acting as an edge device. These PCs are very robust, with the drawback of limited resources.

Using Pledger, the limitations of an edge device regarding computational power can be overcome, by offloading applications to the cloud to ensure the QoS and associated SLAs of the applications. Furthermore, sensitive information obtained by the application, can be stored securely on the DLT giving access to authorized parties only.

3.2.1 Automated deployment and performance monitoring

FILL machines are located across the world, therefore services need to be orchestrated automatically and performance should be easy to monitor.

This task can be achieved by using Pledger and its orchestration and performance monitoring functionalities. Using ConfService the service to be orchestrated can be configured and with the push of a button it will be deployed by the Orchestrator automatically on the infrastructure of choice. As soon as the application is running, relevant metrics are extracted and pushed via StreamHandler to Monitoring Engine. The Service Provider can observe the performance on a dashboard right afterwards.

This process is illustrated in Figure 75.

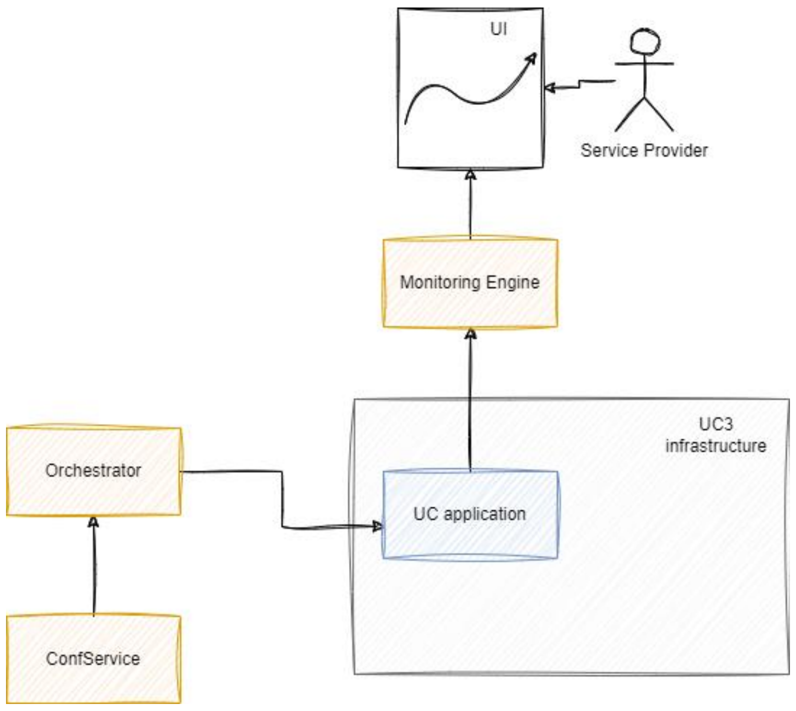


Figure 75: Automatic deployment and performance monitoring in UC3

3.2.2 Automated offloading of applications to the cloud based on decision making

During the manufacturing process, different analytical services analyze the running process. Depending on the complexity and involved sub-processes, the load can be different. The thermal-stability component is latency-critical as the machine needs to react on the results of the component and waiting times > 90ms decrease the QoE of the machine operator in the shopfloor. Lack of computational resources increase the calculation time for the service. To ensure enough resources to the component, another non-latency critical component can be offloaded to the cloud. Pledger supports this process by monitoring the performance of the thermal-stability component and the associated SLAs. In case of a SLA violation, the DSS triggers an offload of the media-consumption service to a suitable cloud infrastructure based on benchmarking and app profiling to ensure the QoE for the machine operator. Figure 76 shows this procedure.

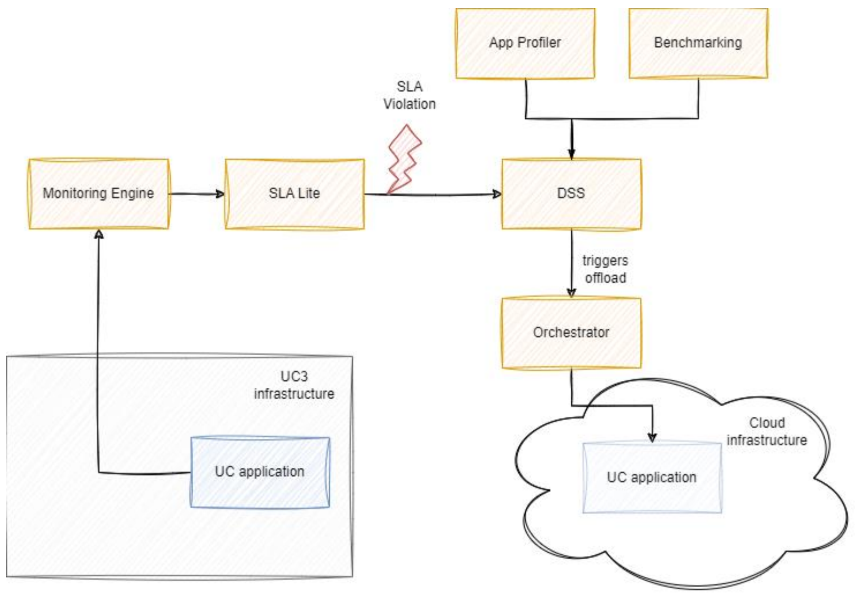


Figure 76: Automated offloading in UC3

3.2.3 Management of sensitive information

During the manufacturing of parts, detailed information is extracted to visualize the process to the manufacturing experts to enable them the analysis of the process. The machine builder is interested in some of the results, too, especially to improve the engineering and to explore new business models based on pay-per-use. However, this information is considered sensitive and should be shared with authorized parties only. Using the Pledger DLT, the authorized access to this information is ensured. The application stores the information on the DLT and an authorized user can retrieve the desired information on a UI. This process is illustrated in Figure 77.

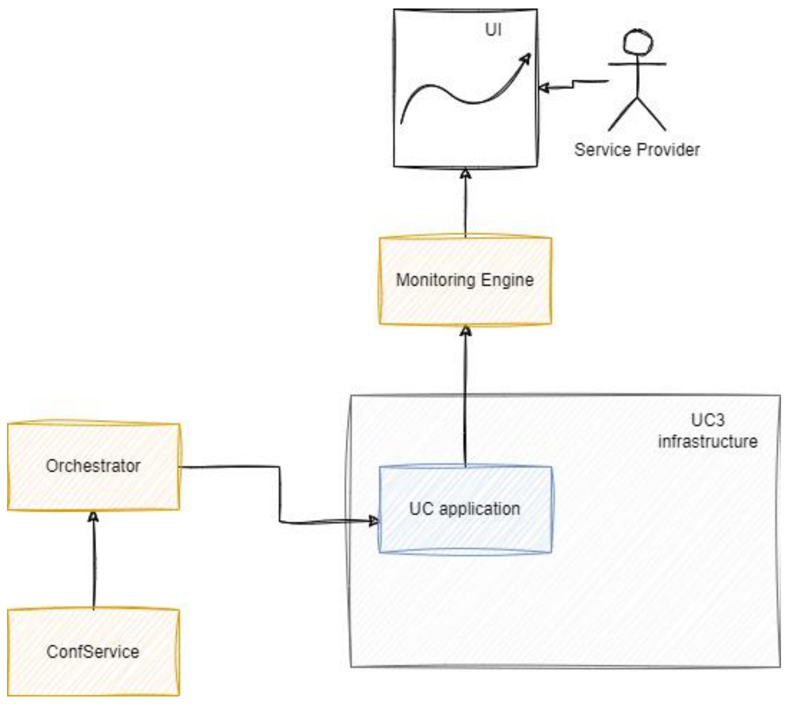


Figure 77: Management of sensitive information in UC3