



D3.6 QoS and SLA assessment and negotiation tools II

Document Identification			
Status	Final	Due Date	31/05/2022
Version	1.0	Submission Date	31/05/2022

Related WP	WP3	Document Reference	D3.6
Related Deliverable(s)	D2.3 Pledger Overall Architecture v1.0 D3.3 QoS and SLA assessment and negotiation tools I D3.5 Edge/Cloud orchestration tools II	Dissemination Level (*)	PU
Lead Participant	ATOS	Lead Author	Roi Sucasas (ATOS)
Contributors	ATOS, i2CAT	Reviewers	Orfeas Voutyras (ICCS)
			Carina Pamminger (HOLO)

Keywords:
QoS, SLA, metric, KPI, agreement, guarantee, demo

This document is issued within the frame and for the purpose of the PLEDGER project. This project has received funding from the European Union's Horizon2020 Framework Programme under Grant Agreement No. 871536. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

The dissemination of this document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains. This deliverable is subject to final acceptance by the European Commission.

This document and its content are the property of the PLEDGER Consortium. The content of all or parts of this document can be used and distributed provided that the PLEDGER project and the document are properly referenced.

Each PLEDGER Partner may use this document in conformity with the PLEDGER Consortium Grant Agreement provisions.

(*) Dissemination level: **PU**: Public, fully open, e.g. web

Document Information

List of Contributors	
Name	Partner
Roi Sucasas	ATOS
Estela Carmona Cejudo	i2CAT

Document History			
Version	Date	Change editors	Changes
0.1	13/04/2022	ATOS	Initial version
0.2	04/05/2022	ATOS	i2CAT contribution in section 3
0.3	05/05/2022	ATOS	Initial draft
0.4	10/05/2022	ATOS	Ready for internal review
0.5	26/05/2022	ICCS	Added comments from the internal reviewer
0.6	27/05/2022	HOLO	Added comments from the internal reviewer
0.7	30/05/2022	ATOS	Quality check
1.0	31/05/2022	ATOS	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Roi Sucasas (ATOS)	26/05/2022
Quality manager	Carmen San Román (ATOS)	30/05/2022
Project Coordinator	Lara López (ATOS)	31/05/2022

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	2 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status: Final

Table of Contents

Document Information	2
Table of Contents	3
List of Tables.....	5
List of Figures	6
List of Acronyms.....	7
Executive Summary	8
1 Introduction	9
1.1 Purpose of the document.....	9
1.2 Relation to other project work.....	9
1.3 Structure of the document	10
1.4 Glossary adopted in this document	10
2 Functional description	11
2.1 Mayor Changes in final prototype.....	12
2.2 Interaction with other modules.....	13
3 Technical description.....	15
3.1 Baseline technologies and dependencies	15
3.2 Internal Architecture	17
3.3 Sequence diagrams.....	21
3.3.1 SLA creation.....	21
3.3.2 SLA assessment.....	22
3.4 Deployment diagrams	22
3.5 Interfaces provided.....	23
3.5.1 REST API.....	23
3.5.2 Kafka Connector.....	26
3.5.3 Other SLAs subsystem components.....	26
3.6 Data models.....	26
3.7 Metrics and the way to collect them	27
4 Installation and usage guides.....	30
4.1 Requirements	30
4.1.1 Docker images.....	30
4.1.2 Repository code.....	30
4.1.3 External tools.....	30
4.2 Installation.....	30
4.2.1 Docker images.....	30
4.2.2 Repository code.....	31

Document name:	D3.6 QoS and SLA assessment and negotiation tools II			Page:	3 of 37
Reference:	D3.6	Dissemination:	PU	Version:	1.0
				Status:	Final

4.3 Usage.....	31
4.4 Licenses.....	32
4.5 Source code repository.....	32
5 Demonstration	33
5.1 Scenarios description	34
5.1.1 Pledger integration demo 1.....	34
5.1.2 Pledger integration demo 2.....	35
5.2 Validation and Verification.....	35
5.3 Demo.....	35
6 Conclusions	36
7 References	37

List of Tables

<i>Table 1: Baseline technologies used by SLA tool.....</i>	<i>15</i>
<i>Table 2: SLA subsystem Components, baseline technologies and Pledger dependencies</i>	<i>19</i>

Document name:	D3.6 QoS and SLA assessment and negotiation tools II				Page:	5 of 37	
Reference:	D3.6	Dissemination:	PU	Version:	1.0	Status:	Final

List of Figures

Figure 1: Task 3.3 and its relation to other Work Packages	9
Figure 2: Pledger Core Subsystems and WPs	11
Figure 3: SLAs subsystem and its relationship with others Pledger Core components	13
Figure 4: Another view of the SLAs subsystem and its relationship with others components	14
Figure 5: Dependencies between SLAs subsystem tools	17
Figure 6: SLAs subsystem Component Diagram	18
Figure 7: SLAs subsystem Component Diagram and baseline technologies	20
Figure 8: SLAs main workflows	21
Figure 9: SLAs subsystem in a Kubernetes cluster	23
Figure 10: SLA Lite swagger interface (I)	24
Figure 11: SLA Lite swagger interface (II)	25
Figure 12: SLA Lite swagger interface (III)	25
Figure 13: Metrics collection	28
Figure 14: Pledger core components integrated demo	33
Figure 15: SLAs – Blockchain integration demo	33
Figure 16: Components involved in Pledger integration demo#1	34
Figure 17: SLAs subsystem role in Pledger integration demo#3	35

List of Acronyms

Abbreviation / acronym	Description
CNCF	Cloud Native Computing Foundation
CRUD	Create, Read, Update and Delete basic operations of any entity.
DBMS	Database Management System
DSS	Decisions Support System
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
FC	Functional Component
FG	Functional Group
GUI	Graphical User Interface
HTML	HyperText Markup Language
IaaS	Infrastructure as a Service
K8S	Kubernetes software. Container orchestrator software.
Mx	Project Month x (e.g., M30 – Month 30)
MVP	Minimum Viable Product
PaaS	Platform as a Service
QoS	Quality of Service
REST	Representational State Transfer
SaaS	Software as a Service
SLA	Service Level Agreement
SLO	Service Level Objective
SPA	Single Page Application (web user interface)
SUC	System User Cases
Tx.y	Task y of WP x
UI	User Interface
WP	Work Package

Executive Summary

This deliverable accompanies the final prototype release of the Pledger's "QoS and SLA assessment and negotiation tools", which correspond to Task 3.3. It includes a full description of the results of this task at month 30 (M30) and the progress made since M18.

Task 3.3 is responsible for the QoS Assessment Entity, which includes the management and assessment of SLAs, and for giving support to other Pledger components in relation to this area. As part of this task, this deliverable focuses on the SLAs subsystem, which is the Pledger set of tools responsible for these functionalities.

The scope of this deliverable comprehends the following:

- ▶ A full description about the functional and technical aspects of the final version of the SLAs subsystem tools, based in the reference architecture proposed in D2.3 – Pledger Overall Architecture [1] and in the work presented in D3.3 – QoS and SLA assessment and negotiation tools I [2].
- ▶ Links to the source code, an installation manual and technical documentation of the different software components that compose this final prototype.
- ▶ The demonstration and validation scenarios of the tool, and the final conclusions.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II			Page:	8 of 37	
Reference:	D3.6	Dissemination:	PU	Version:	1.0	Status: Final

1 Introduction

1.1 Purpose of the document

This document is a description report of deliverable D3.6 “QoS and SLA assessment and negotiation tools II”, which accompanies the final prototype tool demonstrator.

Following the requirements, specifications and architecture presented in previous deliverables (D2.2 – Pledger Requirements Analysis [3] and D2.3 – Pledger Overall Architecture [1]), and also following the initial prototype of the QoS and SLA assessment and negotiation tools presented in M18 in the deliverable D3.3 – QoS and SLA assessment and negotiation tools I [2], the objective of this deliverable is to present and describe the final results of the work done in Pledger in relation with Task 3.3.

The previous deliverable, D3.3 [2], already described most of the functionalities and technical requirements and components of this tools. This deliverable extends this information by summarizing all these previous technical aspects, and by detailing all the improvements made during this last period.

1.2 Relation to other project work

This document includes the work done in WP3, Task 3.3, during M18-M30. Apart from the requirements and architecture defined in WP2, other inputs to Task 3.3 come from the integration with the other subsystems developed in WP3 and WP4, and from the integration with the Use Cases (WP5).

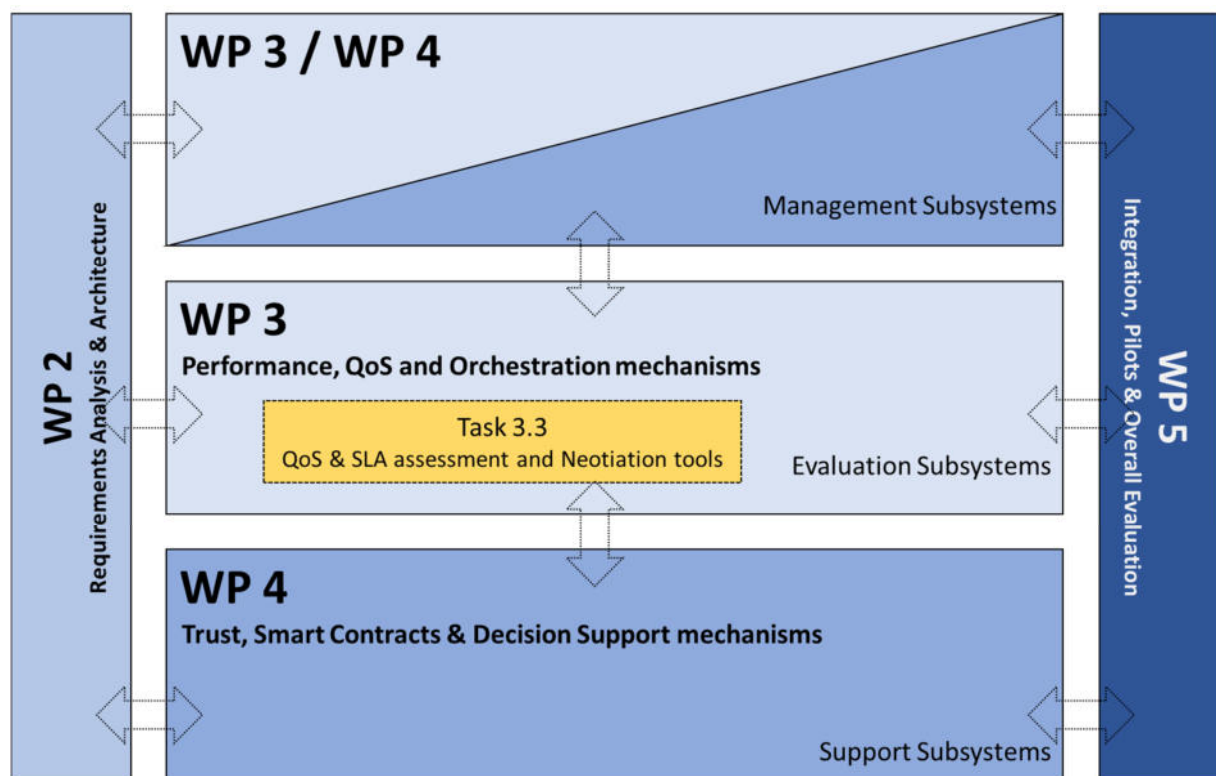


Figure 1: Task 3.3 and its relation to other Work Packages

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	9 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status:
			Final

1.3 Structure of the document

The present document is structured as follows:

- ▶ **Section 2** presents the functional description of the released final version of the QoS and SLA assessment and negotiation tools, and the work done since previous the version.
- ▶ **Section 3** presents the technical description of this final release. This includes the components architecture and the baseline technologies used in the final solution.
- ▶ **Section 4** presents an update of the installation and usage guide of the released tools, including the links to the code repositories.
- ▶ **Section 5** presents the demonstration of the tool with the description of the validation scenarios.
- ▶ **Section 6** presents the final conclusions.

1.4 Glossary adopted in this document

- ▶ **Pledger core system.** The system that will be the “minimum viable product” (MVP) of Pledger, the system that will be freely available to users and other projects and initiatives. It is the exploitable part of the project and it contains all the main features that will deem Pledger successful as a project¹.
- ▶ **Pledger core subsystems.** The subsystems that belong to the Pledger core system, like the Orchestration and SLAs subsystems.
- ▶ **Monitoring Engine.** The Monitoring Engine is a Pledger component that is part of the Orchestration subsystem, and it is responsible for gathering metrics from multiple collectors deployed in the Cloud and Edge.
- ▶ **SLA Lite.** This application is the main component of the SLAs subsystem. It is the tool responsible for the management and assessment of the SLAs, and for the interaction of this subsystem with other subsystems and external tools.
- ▶ **SLA Monitoring subcomponent.** This subcomponent is part of the SLA Lite tool, and it is responsible for the connection to external metrics collectors, like Prometheus or the Monitoring Engine.
- ▶ **Quality of Service.** Also known as QoS, it defines the expected availability and performance of the infrastructure measures by the continuous checking of metrics or KPIs values provide by the infrastructure provider. This expected availability and performance can be extracted from the agreement or SLA between the infrastructure provider (IaaS/PaaS) and the consumer of the infrastructure (service provider of SaaS).
- ▶ **SLA guarantees.** Group of different metrics names and thresholds values agreed between the provider and the consumer that constitute an SLA and represent the quantifiable QoS expected of the infrastructure associated to the application.
- ▶ **SLA violation.** A message alert and track record generated when the system detects that a guarantee in an SLA is not met. This alert will be sent to other components of the system to take required actions.
- ▶ **Severity:** Categories of the grade of the violation of one guarantee, e.g. Warning, Mild, Severe, Critical, etc.
- ▶ **Penalties.** Compensation agreed between the provider and the consumer in case of any violation of the QoS defined in the SLA for the infrastructure associated to the application.

¹ Note, that the SOE and RAN controller framework are proprietary assets and will not be made freely available.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	10 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status: Final

2 Functional description

This section summarizes the main functionalities of the Pledger core's **SLAs subsystem** already described in previous deliverable in relation to Task “T3.3 QoS and SLA Assessment and Negotiation tools”, and it also introduces and describes the new functionalities and improvements added to the final prototype.

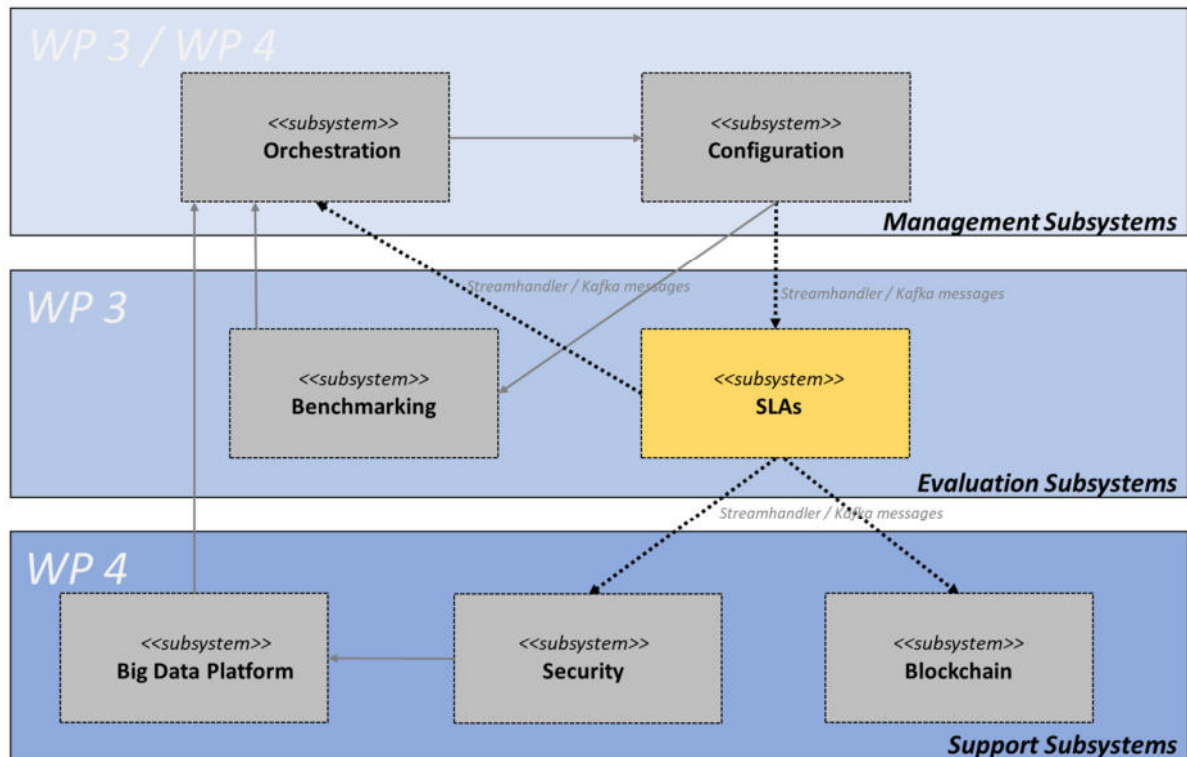


Figure 2: Pledger Core Subsystems and WPs

The main objective of the SLAs subsystem is to give support to other Pledger components (i.e., Orchestration, Security and Blockchain subsystems) by performing the management and evaluation of the SLAs used by the services or applications running in the Pledger platform. The management and evaluation of these SLAs includes their continuous assessment, the provisioning and management of the repositories for these SLAs and the results of their evaluation, including the violations, and the notification about violations and SLA creations to other Pledger components. Apart from the notifications, to give support to the Pledger platform applications, the SLAs subsystem is also responsible for providing them an interface to access all the information (i.e., evaluations and violations) stored and processed by this subsystem.

The first release of the QoS and SLA assessment and negotiation components, which included a set of applications (SLA Lite application, Swagger User Interface, GUI application), delivered in M18, provided the implementation of the following functionalities:

- ▶ SLAs management: creation, modification and storage of the SLAs handled by Pledger. This included the generation of agreements based on templates and based on events received from Kafka. To do this management not only a REST API was exposed by the SLA Lite application, but also a Swagger UI (OpenApi v2) and a GUI application done in React JS.
- ▶ Continuous assessment of applications SLAs running in different environments. The SLAs subsystem relied on the information gathered by external monitoring tools (i.e., Prometheus) to do this assessment. The main component of this subsystem, the SLA Lite implements a set of plugins that

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	11 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status:
			Final

are used to connect to external tools, like Prometheus, to obtain all the information about the metrics (defined in the guarantees / QoS definitions of the SLAs) that need to be monitored.

- Notifications of violations to other Pledger components via Kafka.

2.1 Mayor Changes in final prototype

This final prototype includes new functionalities and improvements with respect to the previous version.

Management of historical data

The first mayor update is the management of historical data about violations and evaluations results. The SLA Lite application stores now all the evaluations and violations of the evaluated metrics. These evaluations and violations include the information about the applications that are being monitored and linked to these SLAs, and it also includes the information about the infrastructures where this application is running.

To access this information, the SLA Lite REST API has been extended to include new operations. These operations can be used now by external tools or other Pledger components to get a more detailed information about the SLAs evaluations and violations.

Monitoring Engine plugin

Other improvements introduced during these last months is the integration with the Monitoring Engine component. The first version of the SLA Lite application included a plugin to connect the SLAs subsystem with Prometheus. Prometheus was used to collect all metrics from different sources, including network metrics as explained in Section 3.7, and the SLA Lite application connected to this collector to get the metrics values and perform then the assessment of the SLAs. This new version connects to the Monitoring Engine by implementing another plugin, which makes the process of gathering metrics transparent. This Monitoring Engine is not only responsible for gathering these metrics, but it can also connect to multiple metrics collectors. This way, the SLA Lite application can do the assessment of SLAs using multiple sources with just one plugin connection.

Integration with more Pledger components via Kafka

During this period, the integration with the T&R Engine and the SLA-SC Bridge components was implemented. The SLA Lite now sends notifications to them via the Stream Handler.

Other improvements

Finally, other minor changes have been made during this period. These changes include, on one hand the upgrade of the Swagger UI application from OpenApi version2 to OpenApi version 3. This new version includes new improvements and an easier way to define the methods and parameters exposed by the application, including a more comprehensive security definition. The other Graphical User Interface React JS application has also been improved with the addition of the new functionalities, like the management of historical data.

Apart from these changes in the user interface applications, the data models were improved and extended to enable the management of historical data and the relation between SLAs and the applications and the infrastructures where these applications are running.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II			Page:	12 of 37	
Reference:	D3.6	Dissemination:	PU	Version:	1.0	Status: Final

2.2 Interaction with other modules

The final version of the SLAs subsystem is connected to the following Pledger components:

- ▶ Configuration subsystem (via Kafka)
- ▶ Monitoring Engine (via REST API)
- ▶ DSS / Recommender (via Kafka)
- ▶ SLA-SC Bridge (via Kafka)
- ▶ T&R Engine (via Kafka)

This SLAs subsystem is continuously listening for events coming from the **Configuration subsystem** to create new SLAs or to update the guarantees of these SLAs. Every time an SLA is created by the SLAs subsystem, it publishes an event in Kafka to announce this new contract creation. The target consumer of these events is the **SLA-SC Bridge** component.

Then, to do the assessment of these SLAs, the SLAs subsystem connects to the **Monitoring Engine** to get the guarantees metrics values. If a violation is being detected, the SLAs subsystem publishes an event on Kafka to notify other components about this violation. The target consumers of these events are the **DSS / Recommender** and the **T&R Engine** components.

Figure 3 shows the dependencies of the SLAs subsystem with the other main Pledger subsystems:

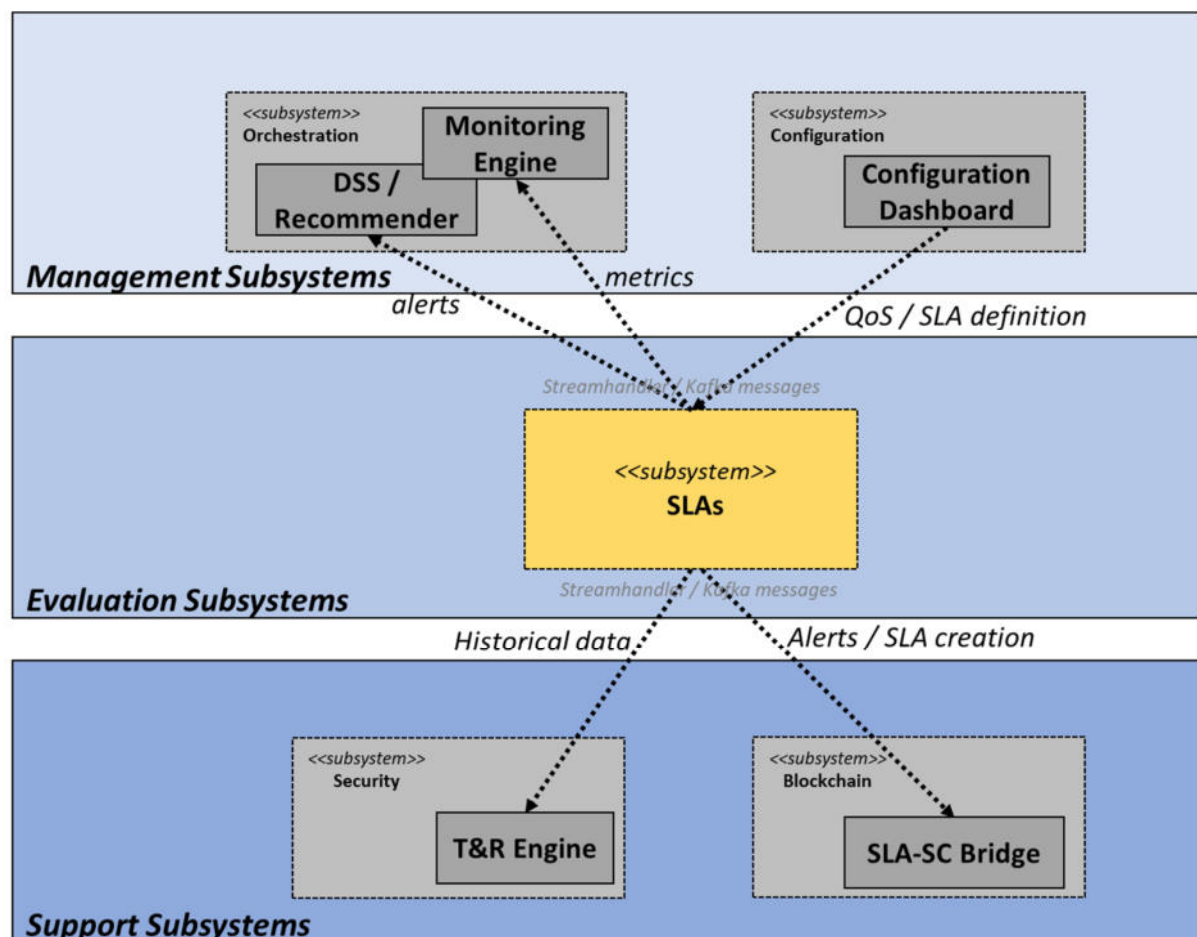


Figure 3: SLAs subsystem and its relationship with others Pledger Core components

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	13 of 37
Reference:	D3.6	Dissemination:	PU
Version:	1.0	Status:	Final

Figure 4 illustrates a more detailed view of these dependencies:

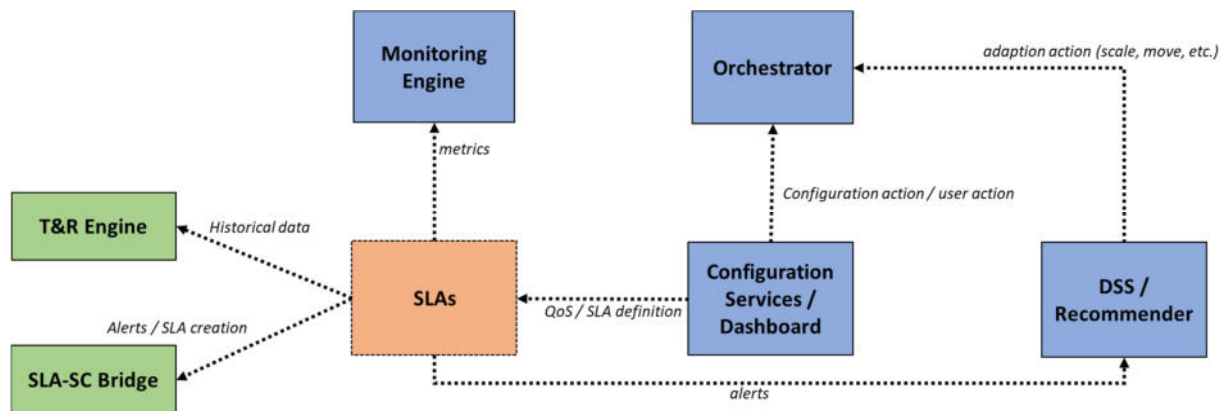


Figure 4: Another view of the SLAs subsystem and its relationship with others components

3 Technical description

This section describes the technical aspects of the final version of the SLAs subsystem. First, this section introduces an updated list of the baseline technologies that compose or are part of this subsystem. Then, the updated internal architecture will be described, including the relationship between the internal components and the baseline technologies. Subsection 3.3 presents two sequence diagrams, which describe the two main workflows of the SLAs subsystem. Next two subsections present the deployment diagram and the interfaces provided by this subsystem. And finally, the last sections present the new data models used by the SLAs subsystem, and a description about how metrics are collected.

3.1 Baseline technologies and dependencies

The following table presents an updated list of the baseline technologies used within this SLAs subsystem, and how they are used (this table updates the baseline technologies presented in D3.3 [2]):

Table 1: Baseline technologies used by SLA tool

Name	Description and role	Version	License	Owner
SLA Lite	The SLA Lite is the main component of the SLAs subsystem. It is a framework, done in Golang, that manages service-level agreements between service providers and consumers. This component has a set of plugins used to connect to Prometheus instances and the Monitoring Engine.	-	Apache 2.0	ATOS
Monitoring Engine	The Monitoring Engine is a Pledger component, done in Golang, that is part of the Orchestration subsystem. It is responsible for gathering all the metrics defined in the SLAs. This tool connects to Kafka and Prometheus instances to get the metrics.	-	Apache 2.0	ATOS
Swagger [4]	Open-source suite of developer tools for describing, defining and creating APIs. It is used by the SLA Lite application to create and manage a REST API user interface. The result is the Swagger UI application.	OpenApi version 3.0.3	Apache 2.0	SmartBear Software
Prometheus [5]	Prometheus is an open-source monitoring and metric collector system. In this final version, Prometheus is one of the collectors used by the Monitoring Engine.	Prometheus Operator Stack	Apache 2.0	SoundCloud, CNCF
Grafana [6]	Grafana is an open-source data visualization tool. It is connected to the Prometheus instances to offer a visualization of the metrics monitored by the system.	latest	AGPLv3	Grafana Labs

Name	Description and role	Version	License	Owner
Kafka [7]	Apache Kafka is a message broker software with event streaming functionalities. This is the official Stream Handler used in the Pledger platform to make possible the communication between the different components.	latest	Apache 2.0	Apache Software Foundation
MongoDB [8]	MongoDB is a NoSQL database engine that stores data in JSON format (JSON documents). It is used by the SLA Lite component as the internal persistent database. It stores the information about SLAs, violations, and evaluations.	Community Edition	SSPL	MongoDB, Inc.
Golang [9]	Golang is an open-source, compiled, and statically typed programming language designed by Google. The SLA Lite (and the Monitoring Engine) application has been written in this programming language. Golang is a high-performing, readable, and efficient, that allow programs to be containerized into very small images, easy to move, deploy and run in the Edge and Cloud.	1.15.1	3-clause BSD + patent grant	Google
React JS [10]	ReactJS) is a free and open-source front-end JavaScript library for building user interfaces. The GUI application has been written with this library.	16.13.1	MIT	Meta
Docker [11]	Docker is an open-source containerization platform that enables developers to package applications into containers to run that code in any environment. SLAs subsystem applications have been containerized using Docker.	latest	Apache 2.0	Docker, Inc.

The following diagram (Figure 5) shows the relations and dependencies between the QoS and SLA Assessment and Negotiation tools and all the baseline technologies described in the previous table:

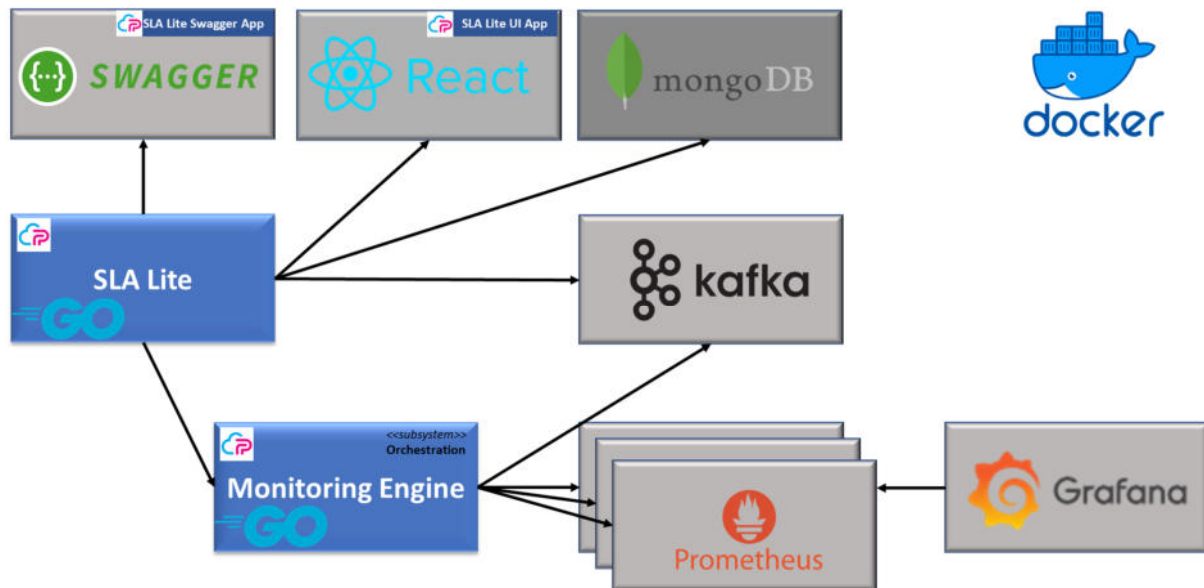


Figure 5: Dependencies between SLAs subsystem tools

3.2 Internal Architecture

This section presents the final version of the functional components of the SLAs subsystem and their interaction with other Pledger core components and subsystems. This subsection is an update of the functional components presented in deliverable D2.3 [1] and in deliverable D3.3 [2].

Figure 6 shows the final components diagram, which includes the Monitoring Engine introduced in this final version, the SLA UI applications (in green), and some updates related with the management of historical data about SLA evaluations and violations.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	17 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status:
			Final

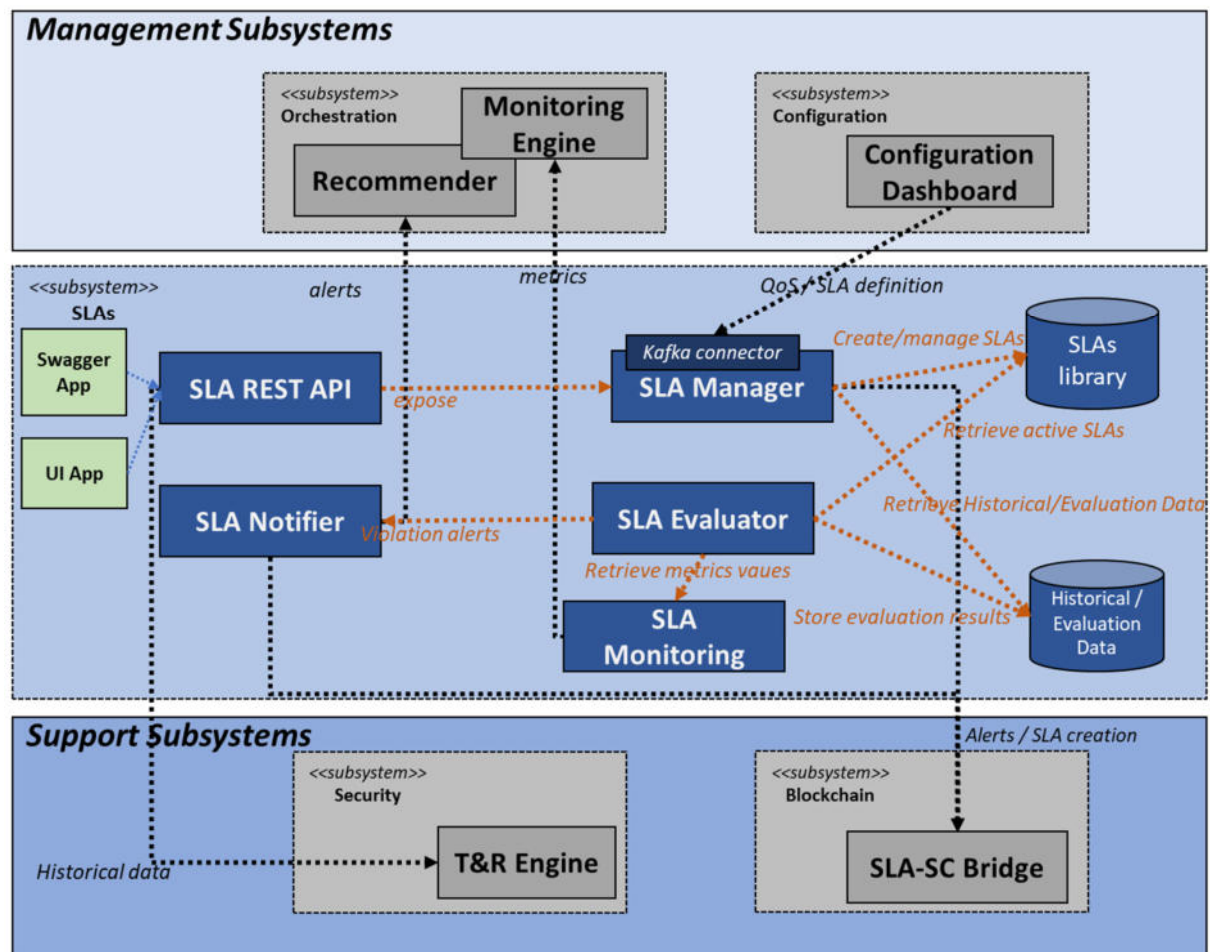


Figure 6: SLAs subsystem Component Diagram

Previous version of this deliverable already included the UI applications (e.g., Swagger App and UI App), but they were not included in the components diagram. Both applications make use of the REST API exposed by the SLA Lite application (SLA REST API component).

This new version also includes the connection between the Kafka connector and the Configuration subsystem, and the connection between the SLA REST API and the T&R Engine as part of the historical data (evaluation results and violations) management.

The REST API, SLA Manager, Notifier, Evaluator, and Monitoring components are part of the SLA Lite application. Pledger requests come from components like the T&R Engine or the Configuration subsystem, which make use of the REST API and the Kafka Connector (SLA Manager) to perform the correspondent actions.

As it was mentioned earlier, the SLA Monitoring connects to the Monitoring Engine, which is now the responsible for gathering the metrics used to do the assessment of the SLAs. The results of these evaluations, including the violations, are stored in the Historical and Evaluation database.

Table 2 presents these internal functional components, the baseline technologies used to implement and build them, the Pledger components that interact with these functional components, and which Docker containerized applications contain these functional components:

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	18 of 37
Reference:	D3.6	Dissemination:	PU
Version:	1.0	Status:	Final

Table 2: SLA subsystem Components, baseline technologies and Pledger dependencies

Functional component	Description	Baseline tools	Pledger dependencies	Containerized (Docker) app
SLA REST API	REST API module responsible for exposing the methods needed to create and manage the SLAs.	Golang	T&R Engine, Swagger App, UI App	SLA Lite
SLA Manager	The SLA Manager module processes all REST API requests and Kafka events. It also creates the SLAs, and notify other components about the creation of these SLAs.	Golang, Kafka, MongoDB	Configuration Service, SLA-SC Bridge	SLA Lite
SLAs library	SLAs and other internal elements needed to manage them are stored in this library.	MongoDB	-	MongoDB
SLA Monitoring	The SLA Monitoring is responsible for gathering all the required metrics from external monitoring tools. It connects to the Monitoring Engine via a plugin, the same way it can connect directly to Prometheus instances.	Golang, Prometheus	Monitoring Engine	SLA Lite
SLA Evaluator	This component performs the assessment of the SLAs. The results of these evaluations are stored in the Historical / Evaluation database.	Golang, MongoDB	-	SLA Lite
IaaS Historical / Evaluation Data	SLA evaluations and violations are stored in this database.	MongoDB	-	MongoDB
SLA Notifier	The SLA Notifier is responsible for sending violations notifications to external components via Kafka.	Golang, Kafka	DSS Recommender / SLA-SC Bridge	SLA Lite
Swagger App	REST API User Interface done with Swagger.	Swagger	SLA REST API	SLA Lite
UI App	Graphical User Interface used to manage and visualize the SLAs.	React JS	SLA REST API	UI App

The SLA Negotiator functional component has finally been discarded. Part of the functionality defined for this component has been moved to the SLA-SC Bridge component. And the other part consists in publishing via Kafka all the information about the creation of new SLAs.

Figure 7 shows the SLAs subsystem components diagram including the baseline technologies described before:

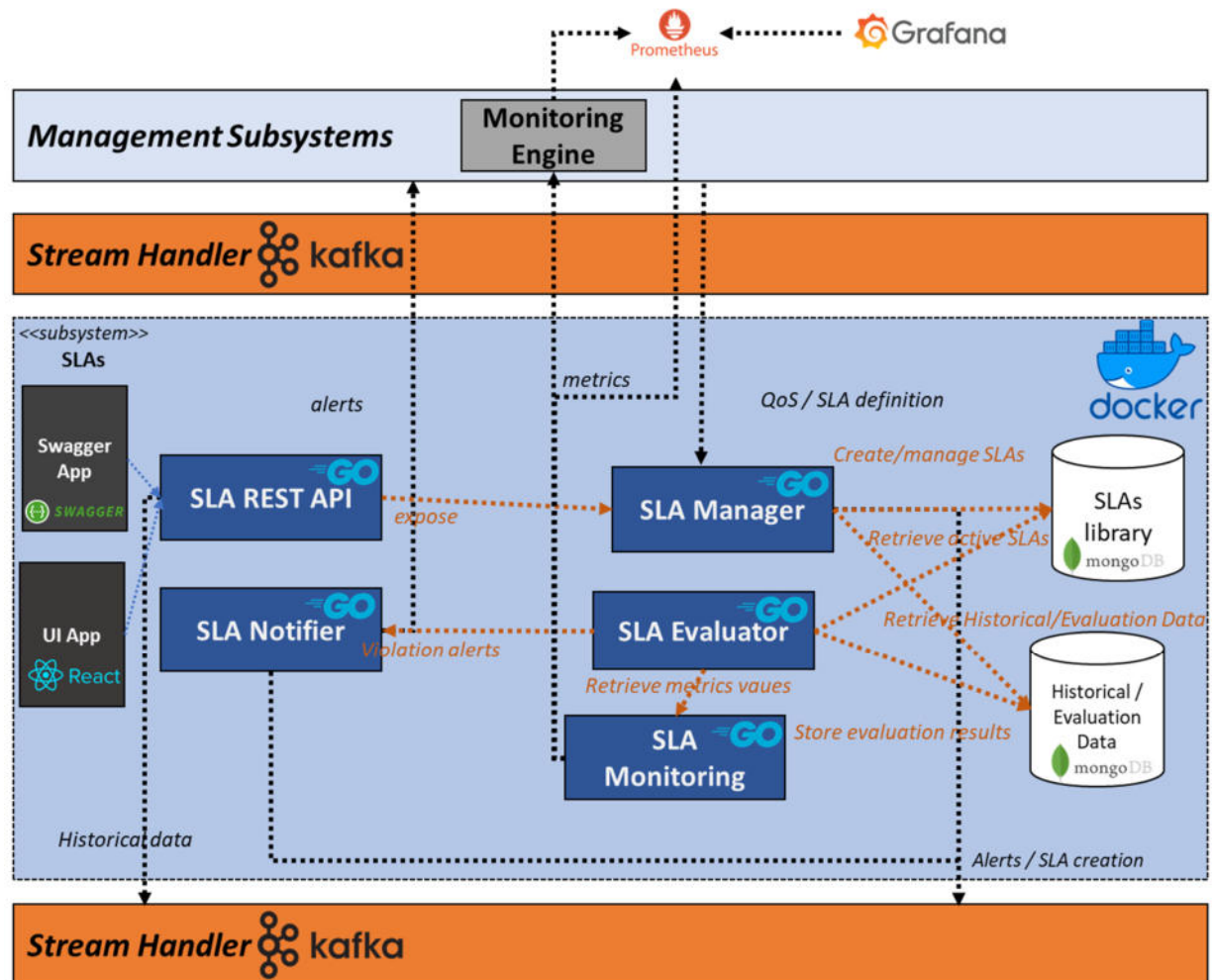


Figure 7: SLAs subsystem Component Diagram and baseline technologies

The REST API, Manager, Notifier, Evaluator, and Monitoring components compose the SLA Lite application, and they have been written in Golang programming language. Some of these components also use Kafka and MongoDB libraries to implement the required connections. Then, this SLA Lite application has been containerized as a Docker image in the SLA Application together with the Swagger Application.

Finally, the databases are included in another Docker container specific for MongoDB, and the React JS UI application in another one. The deployment diagram of these application is described in “Deployment diagrams” section.

3.3 Sequence diagrams

The following two sequence diagrams describe in more detail the main workflows of the SLAs subsystem, which include the interaction with other Pledger components, the creation of SLAs, how the evaluation and assessment process is done, and finally how the violations are sent to other components:

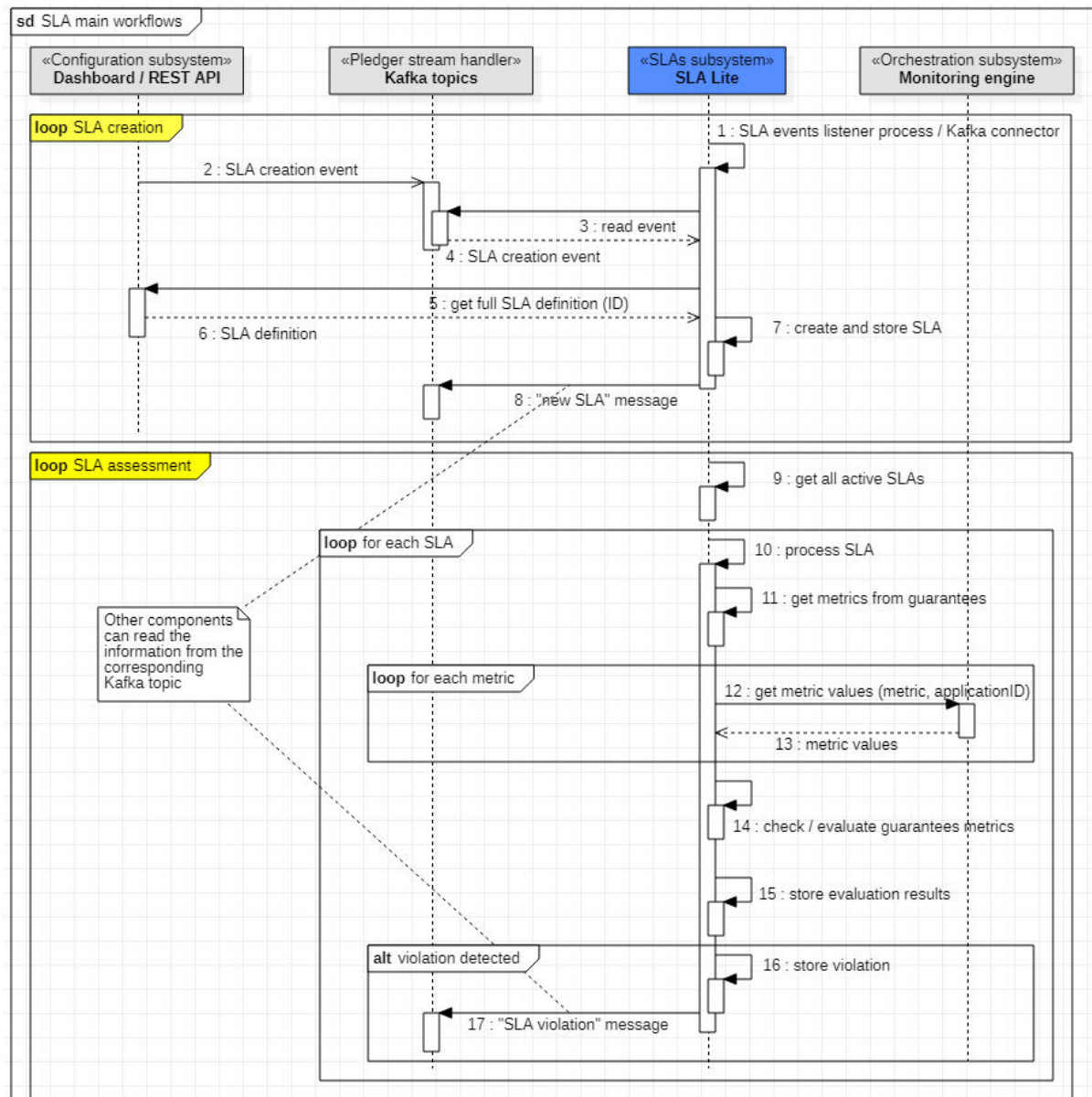


Figure 8: SLAs main workflows

3.3.1 SLA creation

This workflow (steps 1 to 8) describes how an SLA is created by the SLAs subsystem, and it shows the interaction between all the Pledger components involved in this task: the Stream Handler (Kafka), the Configuration Service Dashboard, and the Configuration Service REST API.

[1] First, the SLA Lite application starts the process that connects to Kafka / Pledger Streamhandler to receive messages from other components and to send messages to other components.

[2] The Configuration subsystem component sends to Kafka a “SLA creation event”.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	21 of 37
Reference:	D3.6	Dissemination:	PU
Version:	1.0	Status:	Final

[3-4] The SLA Lite Kafka connector receives the event.

[5] The SLA Lite process the event, extracts the type of operation (“SLA creation”) and the identifier (“ID”) of the object that stores all the information about the new SLA in the Configuration subsystem, and connects to the Configuration subsystem REST API to get this information.

[6] The SLA Lite receives all the information needed to create the SLA. This information includes the provider, the guarantees and QoS constraints, and the identifier of the application that is linked to the new SLA.

[7] The SLA Lite creates the SLA, sets the status to “active”, and stores it in the internal database. This SLA is represented as a JSON file with all the information.

[8] Finally, the SLA Lite uses the Kafka connector to send to other components (i.e., the Blockchain subsystem) the information about this new SLA.

3.3.2 SLA assessment

This workflow (steps 9 to 17) describes the evaluation and assessment task, and it shows the interaction between all the Pledger components involved in this task. The SLA Lite is continuously running a daemon process to evaluate all the active SLAs. This daemon process executes the following steps:

[9] First, the SLA Lite gets all the “active” SLAs stored in the subsystem

[10-11] Then, for each SLA, the SLA Lite daemon process extracts the guaranties and metrics defined in these SLAs.

[12] For each metric, the SLA Lite process connects to the Monitoring Engine component to get the values. This monitoring component gets the name of the metric and the identifier of the application linked to the processed SLA.

[13] The SLA Lite process gets the information of the requested metric.

[14] The correspondent guarantees are evaluated and checked.

[15] The information of this evaluation is stored in the internal historical database.

[16] If a violation is detected, the SLA Lite stores the violation in the internal historical database and ...

[17] ... sends the violation to Kafka so that other components (i.e., the Blockchain subsystem, Recommender) receive the information about this SLA violation.

3.4 Deployment diagrams

The deployment diagram of the final prototype of the “QoS and SLA Assessment and Negotiation” tools, already deployed in the Kubernetes Pledger testbed, is shown in the next picture (Figure 9). This deployment includes the SLA Lite application, the SLA Lite Swagger application, the SLA Lite Web UI, and finally, a MongoDB instance used to store all the internal data, which include the SLAs and the historical information.

All the applications have been containerized in three different images: SLA App, which includes the SLA Lite and the Swagger UI applications; the UI App (React JS application); and the internal database (MongoDB).

The following diagram shows this deployment in a Kubernetes cluster.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	22 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status: Final

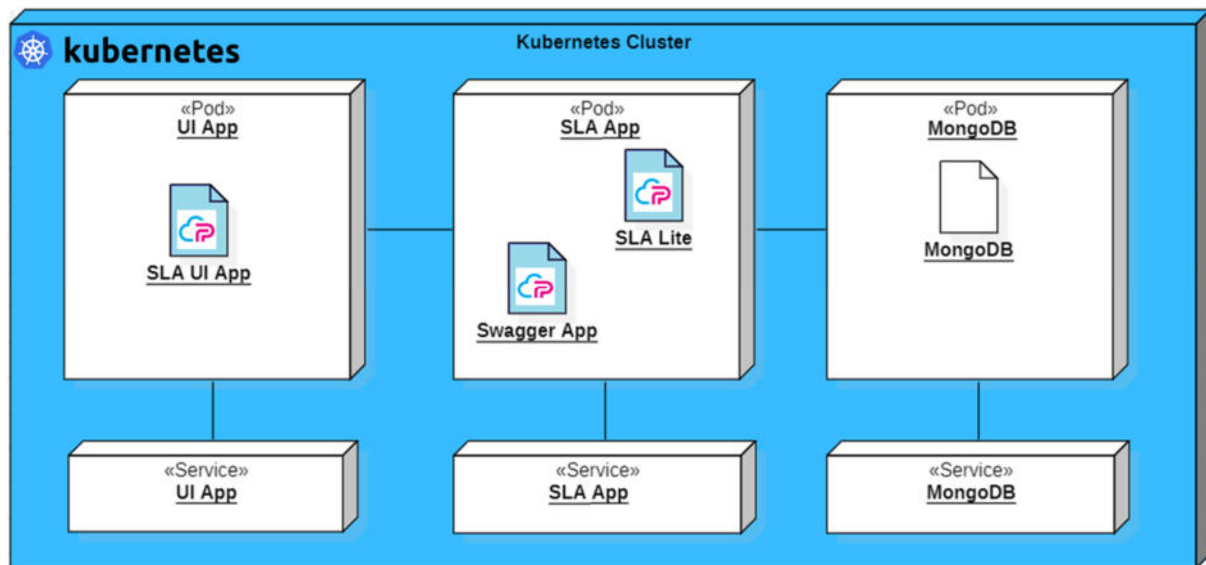


Figure 9: SLAs subsystem in a Kubernetes cluster

The three images are deployed in Kubernetes as “Deployment” objects². The result is three Pods (containers) running with instances of the applications. To connect these applications to other Pledger components or other external applications, we need to create three “Service” objects³, one for each Pod/Deployment object. More details about the installation can be found in “Installation and usage guides” section.

3.5 Interfaces provided

This section describes the interfaces provided by the SLA Lite application to other applications and Pledger components, and the available operations, including the new operations used to retrieve historical data about metrics and applications evaluations and violations. The SLA Lite offers two ways to access all these operations: a REST API and a Kafka connector. Both can be accessed and used by third party tools, like the other Pledger core components.

3.5.1 REST API

The SLA Lite application exposes a REST API with all the available operations, which are grouped in four categories:

3.5.1.1 Agreements

The following operations are the main ones used to create, manage, and remove the SLAs:

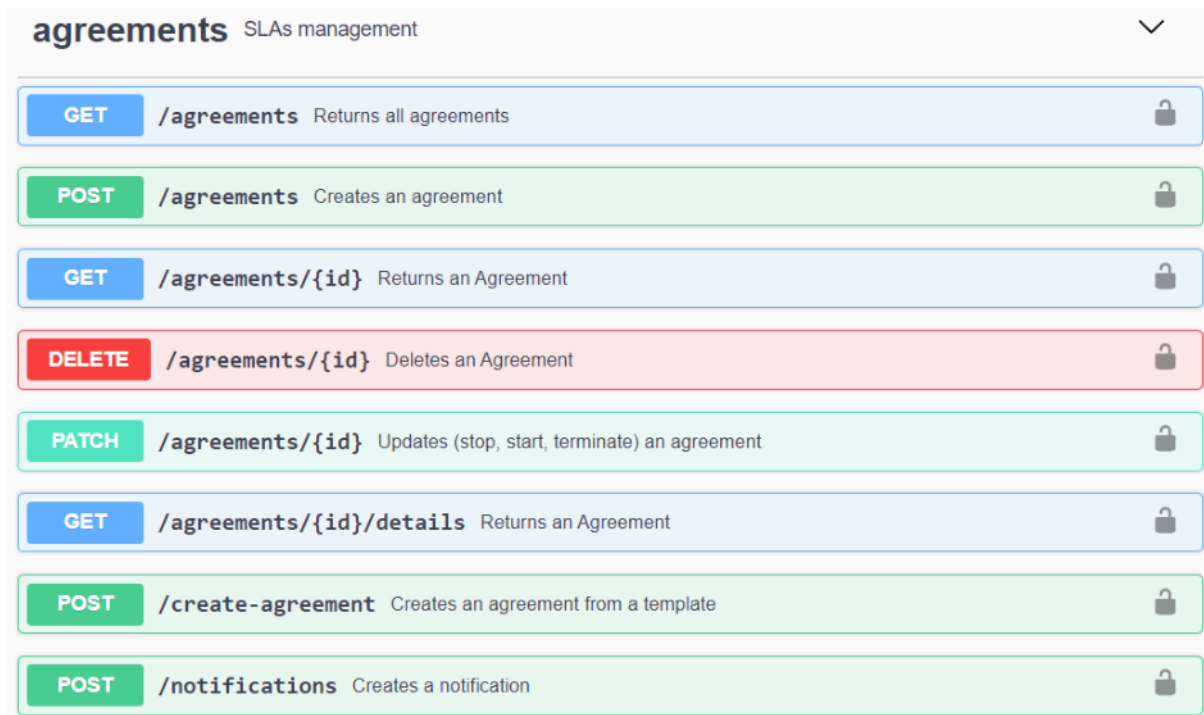
Method	URI	Description
GET	/agreements	Returns a list of all SLA entities stored in the internal database
POST	/agreements	Creates an SLA: a JSON file with the SLA details must be provided
GET	/agreements/{id}	Returns the details of an SLA
DELETE	/agreements/{id}	Deletes an SLA

² A Deployment provides declarative updates for Pods (Containers) and ReplicaSets. ([Deployments | Kubernetes](#))

³ A Service is an abstract way to expose an application running on a set of Pods as a network service. ([Service | Kubernetes](#))

Method	URI	Description
PATCH	/agreements/{id}	Create an SLA (a JSON file with the SLA details must be provided)
POST	/create-agreement	Creates an SLA from a template: a JSON file with the SLA template ID must be provided

The REST API can also be accessed via a Swagger User Interface which provide all the available operations:



agreements SLAs management		
GET	/agreements	Returns all agreements
POST	/agreements	Creates an agreement
GET	/agreements/{id}	Returns an Agreement
DELETE	/agreements/{id}	Deletes an Agreement
PATCH	/agreements/{id}	Updates (stop, start, terminate) an agreement
GET	/agreements/{id}/details	Returns an Agreement
POST	/create-agreement	Creates an agreement from a template
POST	/notifications	Creates a notification

Figure 10: SLA Lite swagger interface (I)

3.5.1.2 Providers and Templates

The providers and templates operations are used to create and manage SLA providers and the generic templates that can be used to create SLAs:

Method	URI	Description
GET	/providers	Returns a list of all SLA providers stored in the internal database
POST	/providers	Creates an SLA provider: a JSON file with the provider details must be provided
GET	/providers/{id}	Returns the details of an SLA provider
DELETE	/providers/{id}	Deletes an SLA provider
GET	/templates	Returns a list of all templates stored in the internal database
POST	/templates	Creates a new template: a JSON file with the template SLA details must be provided
GET	/templates/{id}	Returns the details of a template
DELETE	/templates/{id}	Deletes a template



Figure 11: SLA Lite swagger interface (II)

3.5.1.3 Historical Data

The new methods responsible for offering other components a complete information about historical data about SLA evaluations and violations are the following:

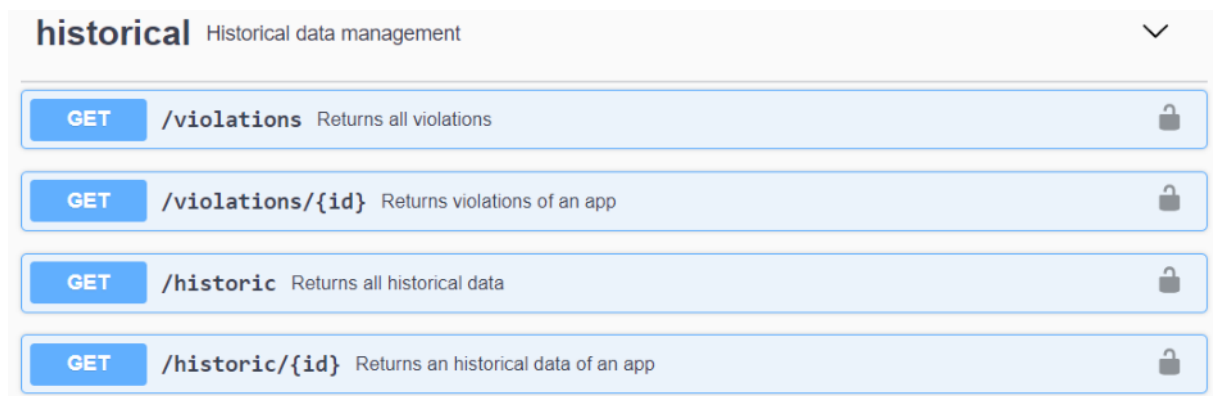


Figure 12: SLA Lite swagger interface (III)

Method	URI	Description
GET	/violations	Returns a list of all SLA violations stored in the internal database
GET	/violations/{id}	Returns a list of all violations that correspond to a specific application
GET	/historic	Returns a list of all SLAs evaluations stored in the internal database
GET	/historic/{id}	Returns a list of all evaluations that correspond to a specific application

3.5.2 Kafka Connector

The SLA Lite also provides a Kafka connector used to receive and send messages to other components or applications. This is used in Pledger to process events coming from the Configuration subsystem, and to send notifications about violations and SLAs creations to other components. This interface doesn't offer the same operations as the REST API interface. Only the main operations are handled, like the creation and removal of SLAs.

This Kafka connector is continuously listening to a specific Kafka topic where other components send all the events related to the SLAs subsystem. In the same way, the SLAs subsystem uses the Kafka connector to send events or messages to other Pledger components via other specific topics.

3.5.3 Other SLAs subsystem components

Interfaces provided by third party tools that are also part of this SLAs subsystem (Prometheus, Kafka, mongoDB, Grafana) can be found in their respective web sites and documentations.

3.6 Data models

As it was described in previous deliverable, D3.3, SLAs objects managed by the SLAs subsystem are defined as JSON objects. Some minor changes have been made in these models in order to enable the processing of historical data of violations and evaluations. In the case of the SLA definition, a reference to the application or service that is being monitored has been added (in red):

```
{
  "id": "<ID_SLA>",
  "name": "an-agreement-name",
  "state": "started",
  "details": {
    "id": "<ID_SLA>",
    "type": "agreement",
    "name": "an-agreement-name",
    "provider": { "id": "a-provider-id", "name": "A provider name" },
    "client": { "id": "a-client-id", "name": "A client name" },
    "creation": "creating date in YYYY-MM-ddThh:mm:ssZ format",
    "expiration": "expiration date in YYYY-MM-ddThh:mm:ssZ format ",
    "guarantees": [ {
      "name": "a guarantee name",
      "constraint": "[METRIC] [COMPARATOR] [THRESHOLD_VALUE]",
      "importance": [ {
        "name": "name of severity category",
        "Constraint": "comparison_op threshold"
      } ],
      "penalties": [ {
        "type": "type of penalty",
        "value": "value of penalty",
        "unit": "unit of penalty"
      } ],
      "actions": [ {
        "type": "type of action, e.g. SCALE OUT, MIGRATE",
        "value": "value for the action, e.g. number of replicas to scale",
        "unit": "unit of the value"
      } ]
    } ],
    "service": "<ID_APP>"
  }
}
```

Document name:	D3.6 QoS and SLA assessment and negotiation tools II			Page:	26 of 37
Reference:	D3.6	Dissemination:	PII	Version:	1.0
				Status:	Final

The violation model has also been updated to add a reference to the infrastructure where the monitored application or service is running. The following example shows the new version, with the new field in red.

```
{
  "id": "<ID_VIOLATION>",
  "agreement_id": "<ID_SLA>",
  "guarantee": "scrape_duration_seconds",
  "datetime": "2021-04-13T17:27:06Z",
  "constraint": "[METRIC] [COMPARATOR] [THRESHOLD_VALUE]",
  "values": [{
    "key": "scrape_duration_seconds{10.244.0.8:3000}",
    "value": 2.8476827,
    "datetime": "2021-04-13T17:27:06Z"
  }],
  "importanceName": "Serious",
  "importance": 1,
  "appid": "<ID_APP>",
  "infrid": "<ID_INFRA>",
  "description": "This is a violation of the agreement of app X"
}
```

Finally, one more model has been added to store the information of the SLA evaluations. The following JSON shows this new model:

```
{
  "id": "<ID_EVAL>",
  "agreement_id": "<ID_SLA>",
  "guarantee": "scrape_duration_seconds",
  "datetime": "2021-04-13T17:27:06Z",
  "constraint": "[METRIC] [COMPARATOR] [THRESHOLD_VALUE]",
  "values": [{
    "key": "scrape_duration_seconds{10.244.0.8:3000}",
    "value": 2.8476827,
    "datetime": "2021-04-13T17:27:06Z"
  }],
  "appid": "<ID_APP>",
  "infrid": "<ID_INFRA>"
}
```

3.7 Metrics and the way to collect them

In the previous version of the SLAs subsystem prototype, the SLA Lite component connected directly to Prometheus via a plugin or connector in order to get the metrics values needed to evaluate the guarantees of the SLAs. This new version has been improved by replacing this connection to Prometheus with a new connection to the Monitoring Engine component. On one side, this Monitoring Engine can connect to multiple metric collectors located in the Edge and Cloud, and on the other side it knows where to look for a specific metric that is linked to an application deployed in this Cloud-Edge environment.

These new functionalities are needed in order to support the collection of metrics from different sources (Edge and Cloud), and also to collect the metrics linked to applications that migrate from one location to a new one. All the logic that handles this metric gathering has been implemented in this new component. In the side of the SLA lite component the Prometheus plugin was replaced by a new plugin.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II			Page:	27 of 37
Reference:	D3.6	Dissemination:	PU	Version:	1.0
				Status:	Final

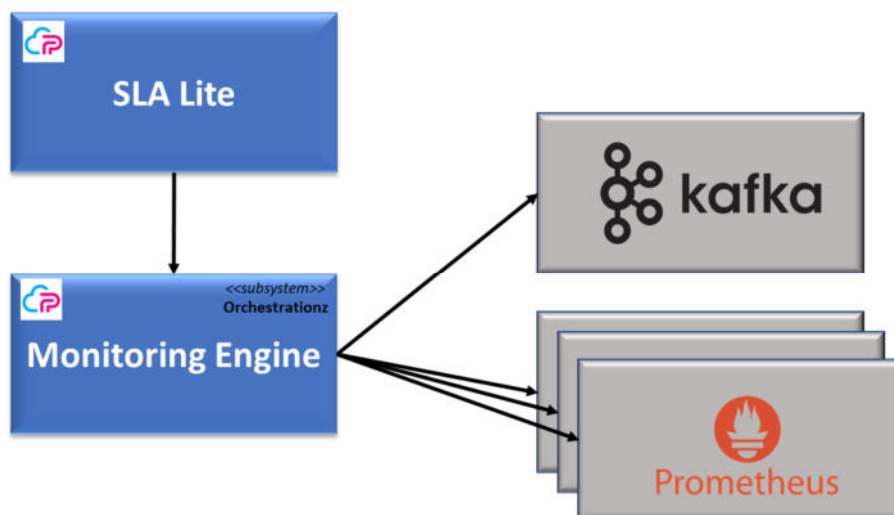


Figure 13: Metrics collection

Network metrics

Just like application metrics, network metrics can help to determine possible issues that might affect the performance or operational status of services hosted by Pledger, e.g., metrics like dropped packets or packet errors. In the case of network telemetry, the Prometheus component will need to configure exporters or agents to get metrics from the network. The extraction of network metrics will be done on a per use case basis, e.g., monitoring of slice-related metrics, but this is not in the scope of SLAs subsystem. Moreover, the use of software defined networking (SDN) to collect network telemetry has been dropped in favour of Prometheus for two main reasons. First of all, in distributed networks, each domain is managed by an independent SDN controller. However, since the use cases in Pledger are based on single administrative domains with no SDN devices, there are no clear advantages for using SDN in order to collect network telemetry. The second reason is that the use of Prometheus brings distinctive advantages to the project [14]: it is an open source tool that is considered the de facto standard metrics solution in the Cloud Native world, and it can be easily integrated with Kubernetes and OpenStack; it supports pull and push models; its metrics format is supported by a wide set of tools and services such as Grafana or GitLab; and it supports exporters which are used in exporting existing metrics from third-party databases, hardware, CI/CD tools, etc.

For the collection of network metrics in a specific cluster, a Prometheus instance needs to be present that gathers the network metrics; this can also be integrated with that connects to the monitoring engine. The network metrics fed to this Prometheus instance can come from different sources and always depend on the specific infrastructure present in a use case, and also on the virtual infrastructure manager deployed (e.g., OpenStack, Kubernetes). . In Pledger, a cloud-native approach is followed, using Kubernetes as the virtual infrastructure manager and with all UCs using an underlying layer 2-connected infrastructure of one or several compute nodes as part of a single domain. Kubernetes measures a variety of network metrics on a per-pod, per-node and per-interface basis that can be exported to a Prometheus instance. Some of the metrics are of an informative nature, e.g. number of transmitted/received bytes of a pod. For example, the metric *container_network_receive_bytes_total* indicates the total number of received bytes at a given pod. Other metrics indicate network failures: number of dropped packets or transmission errors, for example, the metric *container_network_transmit_packets_dropped_total* indicates the number of dropped transmit packets from a given pod. Depending on the desired use, e.g. just keeping track of the network usage of the Kubernetes cluster or using the metrics to detect issues related to specific services, a relevant set of metrics from each type can be considered; this set of metrics can also be pushed to the monitoring engine, and relevant SLAs can be set accordingly.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	28 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status:
			Final

The set of metrics generated by Prometheus on Kubernetes-based infrastructures are very similar to those that can be gathered in multi-domain topologies where an SDN controller is used; for example, in an OpenStack-based architecture with an underlying SDN controller, a set of metrics equivalent to those describe above could be pushed to a Prometheus instance.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II			Page:	29 of 37	
Reference:	D3.6	Dissemination:	PU	Version:	1.0	Status: Final

4 Installation and usage guides

This section presents the requirements, installation, and usage guides of the final prototype of the QoS and SLA assessment and negotiation tools.

4.1 Requirements

4.1.1 Docker images

As the SLAs subsystem application has been containerized in three different Docker images (SLA Lite / SLA App, MongoDB, SLA UI Application) that can be downloaded from <https://hub.docker.com/u/atospledger>, this is the only requirement:

- ▶ **Docker, Kubernetes** or any other container orchestrator that supports the management of docker images. These are the links to the installation guides of these orchestrator tools:
 - <https://docs.docker.com/engine/install/>
 - <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

4.1.2 Repository code

The code of this applications is also available in Gitlab, in the following URL: <https://gitlab.com/pledger/public/sla-framework>

In this case, the requirements to run the main application, the SLA Lite application, are the following:

- ▶ Golang (<https://golang.org/doc/install>)
- ▶ Git (git-scm.com)

To run the SLA UI Application (React JS) using the available code, React and Node JS (<https://nodejs.org/en/>) are needed.

4.1.3 External tools

Although the SLA Lite application can be installed as a standalone application, with an internal memory database and other plugins for gathering the metrics or for notifying other components, the requirements for installing the external applications, mentioned in this document, which are also part of the SLAs subsystem, can be found in the following links:

- ▶ **Kafka:** <https://kafka.apache.org/quickstart>
- ▶ **MongoDB:** <https://docs.mongodb.com/manual/installation/>
- ▶ **Grafana:** <https://grafana.com/docs/grafana/latest/installation/>
- ▶ **Prometheus:** <https://prometheus.io/docs/prometheus/latest/installation/>

4.2 Installation

4.2.1 Docker images

Docker images can be found in Dockerhub, in the following URLs:

<https://hub.docker.com/r/atospledger/sla-framework>

<https://hub.docker.com/r/atospledger/sla-framework-ui>

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	30 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status: Final

To deploy and run the applications in a Docker environment run the following commands:

```
docker pull atospledger/sla-framework
docker run -ti -p 8090:8090 sla-framework
docker pull atospledger/sla-framework-ui
docker run -ti -p 9090:9090 sla-framework-ui
```

More information about the environment variables that can be used to customize the deployment can be found in the source code repository, in <https://gitlab.com/pledger/public/sla-framework>.

To deploy and run the applications in a Kubernetes environment, see the documentation that is part of the source code repository, where some deployment yaml examples can be found.

4.2.2 Repository code

To download the source code from gitlab repository execute the following command:

```
git clone https://gitlab.com/pledger/public/sla-framework.git
```

To run the application run the following command:

```
cd sla-framework
go build
go run main.go app.go
```

More information about the environment variables that can be found in the source code repository.

Finally, to build the docker images (Docker is required) from the source code execute the following commands:

```
cd sla-framework
docker build --rm -t slalite .
cd ../UI
docker build --rm -t slalite-ui .
cd ..
```

4.3 Usage

To use the application independently from the Pledger platform, there are three ways to do that:

- ▶ Swagger REST API application
- ▶ GUI application
- ▶ Command line tools like “curl”

Curl examples were already presented in the previous deliverable, and they can also be found in the source code repository. These examples include how to create an SLA, and how to manage them,

The best way to use this application, is the Swagger UI application that is launched together with the SLA Lite application. The URL to access this Swagger UI application is the following: [“http://<IP:PORT>/swaggerui/”](http://<IP:PORT>/swaggerui/)

This Swagger application contains all the necessary information about how to create and send the requests to the SLA Lite application to manage the SLAs or to get all historic data about evaluations and violations.

Finally, the UI application offers a different way to manage and visualize the SLAs. The URL to access this application is the following: [“http://<IP:PORT>/#/”](http://<IP:PORT>/#/)

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	31 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status: Final

4.4 Licenses

The core components of the SLAs subsystem developed in the context of Pledger, are the following: the SLA Lite application, the Swagger UI application, and the GUI application

All of them have an Apache license type version 2 for research and academic purposes.

Other tools that are part of the subsystem, like Prometheus and Kafka, are also licensed under Apache 2.0 license. MongoDB is licensed under SSPL, and Grafana under AGPLv3.

4.5 Source code repository

The source code of the SLAs subsystem is published in the following **GitLab.com** repository: <https://gitlab.com/pledger/public/sla-framework>.

Apart from the code, this repository contains more information, documents, and other resources needed to deploy the application in a Kubernetes cluster.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II			Page:	32 of 37	
Reference:	D3.6	Dissemination:	PU	Version:	1.0	Status: Final

5 Demonstration

The demonstration of the final version of the SLAs subsystem has been made as part of the Pledger core components integration tasks. There are two videos available with real examples about this integration in the Pledger official YouTube channel: [Pledger Project - YouTube](#)

► [Pledger integration demo#1 - YouTube](#)

► [Pledger integration demo#3 - YouTube](#)

The first video, “**Pledger integration demo#1**”, shows all the main Pledger core components working together to demonstrate a runtime adaptation (placement) scenario in the Edge-Cloud continuum.

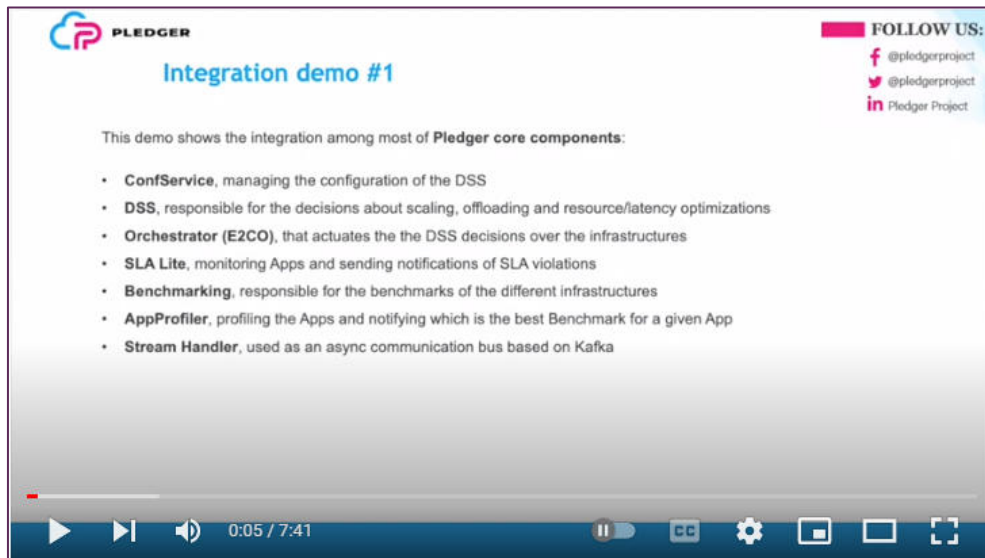


Figure 14: Pledger core components integrated demo

The second video, “**Pledger integration demo#3**”, shows the integration of the SLAs subsystem with the Blockchain subsystem.

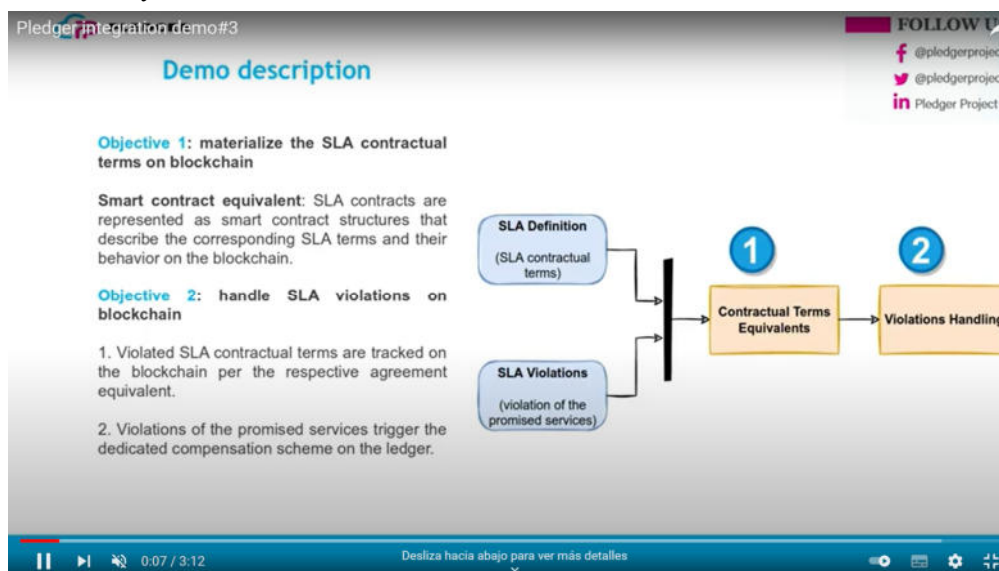


Figure 15: SLAs – Blockchain integration demo

Document name:	D3.6 QoS and SLA assessment and negotiation tools II			Page:	33 of 37
Reference:	D3.6	Dissemination:	PII	Version:	1.0
				Status:	Final

5.1 Scenarios description

5.1.1 Pledger integration demo 1

Objectives: To show a test application being configured, deployed, started, monitored, and offloaded from Edge to Cloud, and then back to Edge, to reduce SLA violations.

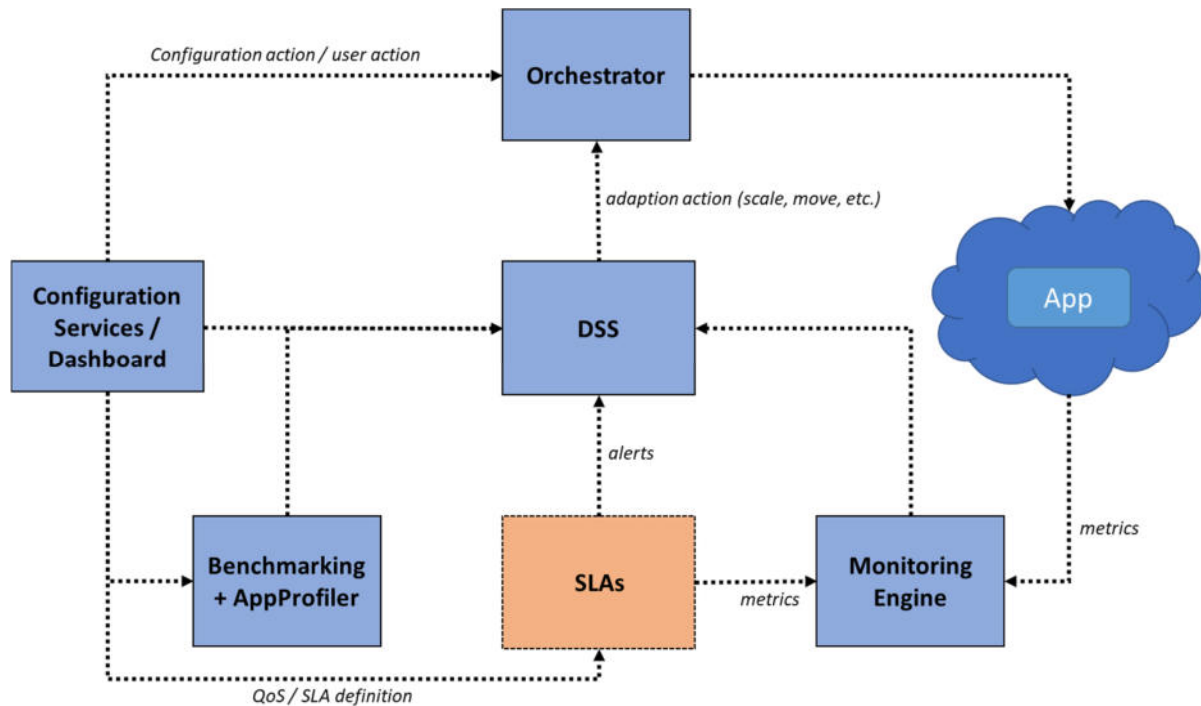


Figure 16: Components involved in Pledger integration demo#1

The components involved in this integration demo are the following:

- ▶ ConfService
- ▶ DSS, Orchestrator,
- ▶ SLA Lite
- ▶ Benchmarking
- ▶ AppProfiler
- ▶ Stream Handler (Kafka)

Here the SLA Lite component is responsible for creating and managing an SLAs used by the DSS to decide when and where to redeploy the application used in this demo.

Script:

1. Use the **ConfService** UI to show the configured infrastructures and service providers preferences.
2. Use the **ConfService** UI to configure an application, a service, the deployment options, and an SLA and its guarantees.
3. Show the **Orchestrator** (E2CO) and **SLA Lite** interfaces to see the configuration events received by the **ConfService**.
4. Show how the application is deployed and started on the Edge.
5. Show how the application is being monitored.
6. Show how an SLA violation is generated after some load is artificially generated.
7. Show how the **DSS** chooses the best **Benchmark** tool for a service using matching set manually or by the **AppProfiler**.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	34 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status:
			Final

8. Show how the DSS chooses the node with the highest score for the selected Benchmark tool.
9. Show how the application is migrated to the Cloud.
10. Show how the application continues being monitored.
11. After some minutes without any SLA violation, the DSS triggers an offload back to the Edge.
12. Show how the application is migrated back to the Edge.
13. Show configuration, deployment and SLA violations Kafka messages sent through the Stream Handler.

5.1.2 Pledger integration demo 2

Objectives: (1) Demonstrate the materialization of an SLA contractual terms on Blockchain, and (2) handle SLA violations on Blockchain.

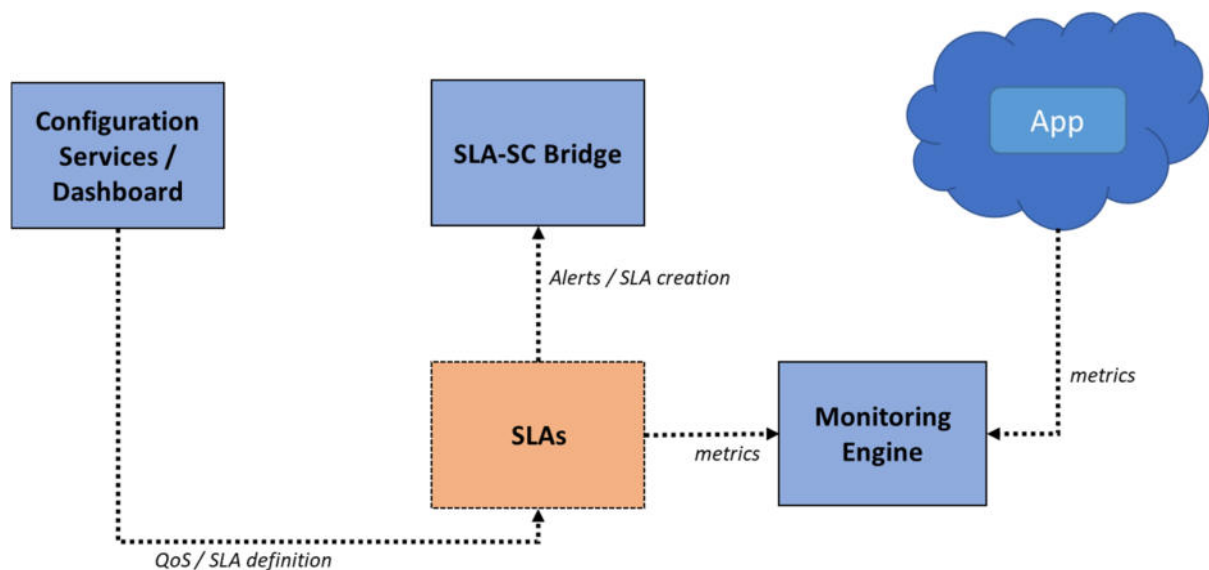


Figure 17: SLAs subsystem role in Pledger integration demo#3

Here the SLA Lite component is responsible not only for creating and managing the SLAs, but it also sends to the Blockchain subsystem the information about new SLAs and SLAs violations.

5.2 Validation and Verification

In these two scenarios we can see the integration of the Pledger core components, including the SLAs subsystem, which is responsible for creating and managing the SLAs, for the assessment of these SLAs, and for the notification to other components about the creation of new SLAs and SLA violation events. All this information is used by other Pledger components, like the DSS or the Blockchain subsystem to perform adaption actions with the applications, or to manage the SLAs in the Blockchain subsystem.

5.3 Demo

The two demo videos can be found in the Pledger official YouTube channel: [Pledger Project - YouTube](#)

- First integrated demo: [Pledger integration demo#1 - YouTube](#)
- Second integrated demo, named as 3 according to the WP order of presentations: [Pledger integration demo#3 - YouTube](#)

Document name:	D3.6 QoS and SLA assessment and negotiation tools II	Page:	35 of 37
Reference:	D3.6	Dissemination:	PU
	Version:	1.0	Status:
			Final

6 Conclusions

This deliverable reported the work done in WP3, in Task T3.3 in M18-M30. The target milestone MS7 “Second iteration of WP3 prototypes” has been successfully achieved and documented in this deliverable, as well as in D3.4 [12] and D3.5 [13].

This deliverable also presented the source code, guides, documentation, and the instructions to install and manage the SLA subsystem of PLEDGER, including demonstrations (videos).

During this period, we could finish some of the tasks that were missing in the previous release. This includes the tools for historical tracks of SLA monitoring and violation, the use of a monitoring tool that can collect metrics from multiple locations, the integration with Blockchain subsystem to create SLA agreements in a distributed ledger and also the integration with the Configuration Dashboard component, among other minor improvements.

The progress done in the last two milestones, MS4 and MS5, will pave the way for the next phase where the goal will be evaluating the Use Cases using the Pledger platform environment presented in these deliverables (D3.4, D3.5 and D3.6). For this purpose, we will continue the development and the improvement of the platform and in this particular case, the QoS and SLA assessment and negotiation tools.

Document name:	D3.6 QoS and SLA assessment and negotiation tools II				Page:	36 of 37
Reference:	D3.6	Dissemination:	PU	Version:	1.0	Status: Final

7 References

- [1] PLEDGER. D2.3 – Pledger Overall Architecture v1.0. Voutyras Orfefs. 2020. <http://pledger-project.eu/D2.3.pdf>. Retrieved 30/05/2021.
- [2] PLEDGER. D3.3 – QoS and SLA assessment and negotiation tools I, Castillo, Antonio. 2021. <http://pledger-project.eu/D3.3.pdf>. Retrieved 31/05/2022 .
- [3] PLEDGER. D2.2 – Pledger Requirements Analysis, Francesco Iadanza, Gabriele Giammatteo 2020. <http://pledger-project.eu/D2.2.pdf>. Retrieved 30/05/2021.
- [4] Swagger home page <https://swagger.io/>. Retrieved 30/05/2022.
- [5] Prometheus home page. <https://prometheus.io>. Retrieved 30/05/2022.
- [6] Grafana home page. <https://grafana.com>. Retrieved 30/05/2022.
- [7] Kafka home page. <https://apache.kafka.org>. Retrieved 30/05/2022.
- [8] MongoDB home page. <https://www.mongodb.com>. Retrieved 30/05/2022.
- [9] Golang home page <https://go.dev/>. Retrieved 30/05/2022.
- [10] React JS home page <https://reactjs.org/>. Retrieved 30/05/2022.
- [11] Docker home page. <https://www.docker.com>. Retrieved 30/05/2022.
- [12] PLEDGER. D3.4 – Performance Measurements and classification tools II, Giammatteo, Gabriele. 2022. <http://pledger-project.eu/content/deliverables>.
- [13] PLEDGER. D3.5 – Edge/Cloud orchestration tools II. , Psychas, Alexandros. 2021. <http://pledger-project.eu/content/deliverables>.
- [14] J. Perez-Romero et. al., “Monitoring and Analytics for the Optimisation of Cloud Enabled Small Cells,” 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2018.