



D3.3 QoS and SLA assessment and negotiation tools I

Document Identification			
Status	Final	Due Date	31/05/2021
Version	1.0	Submission Date	01/06/2021

Related WP	WP3	Document Reference	D3.3
Related Deliverable(s)	D2.3 Pledger Overall Architecture v1.0 D3.2 Edge/Cloud orchestration tools I	Dissemination Level (*)	PU
Lead Participant	ATOS	Lead Author	Antonio Castillo (ATOS)
Contributors	ATOS	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
Antonio Castillo	ATOS
Rui Sucasas	ATOS

Document History			
Version	Date	Change editors	Changes
0.1	09/04/2021	ATOS	Initial version
0.2	12/04/2021	ATOS	Work in progress versions for internal review
0.3	14/04/2021	ATOS	Work in progress versions for internal review
0.4	19/04/2021	ATOS	Work in progress versions for internal review
0.5	21/04/2021	ATOS	Work in progress versions for internal review
0.6	28/04/2021	ATOS	Work in progress versions for internal review
0.7	30/04/2021	ATOS	First version for partners review
	18/05/2021	Orfeas Voutyras (ICCS)	Internal review
0.8	19/05/2021	ATOS	Minor changes after reviewers' feedback
0.9	27/05/2021	Carmen San Román (ATOS)	Quality Assurance review
1.0	31/05/2021	Lara López (ATOS)	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Antonio Castillo (ATOS)	26/05/2021
Quality manager	Carmen San Román (ATOS)	27/05/2021
Project Coordinator	Lara López (ATOS)	31/05/2021

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	2 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status:
			Final

Table of Contents

Document Information	2
Table of Contents	3
List of Tables.....	4
List of Figures	5
List of Acronyms.....	6
Executive Summary	7
1 Introduction	8
1.1 Purpose of the document.....	8
1.2 Relation to other project work.....	8
1.3 Structure of the document.....	8
1.4 Glossary adopted in this document.....	8
2 Functional description	9
3 Technical description.....	12
3.1 Baseline technologies and dependencies	12
3.2 Components Architecture.....	12
3.3 Interfaces provided.....	13
3.4 Data models.....	14
4 Installation and usage guides.....	17
4.1 Requirements	17
4.2 Installation.....	17
4.3 Usage.....	19
4.4 Licenses.....	20
4.5 Source code repository.....	20
5 Demonstration	21
5.1 Scenario description.....	21
5.2 Validation and Verification.....	21
5.3 Demo.....	22
5.3.1 Scenario 1: SCALE OUT	22
5.3.2 Scenario 2: PLACEMENT	25
6 Conclusions and next steps.....	27
7 References	28
Annexes.....	29

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	3 of 29	
Reference:	D3.3	Dissemination:	PU	
	Version:	1.0	Status:	Final

List of Tables

Table 1: SLA subsystem Components (table extracted from D2.3 – Pledger Overall Architecture[1]) 10

Table 2 Baseline technologies used by SLA tool..... 12

Table 3 SLA operations with REST API..... 13

Table 4 SLA tool environment variables for configuration..... 18

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	4 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

List of Figures

<i>Figure 1: The Pledger Core Subsystems (image from D2.3 – Pledger Overall Architecture[1])</i>	9
<i>Figure 2: SLA subsystem Component Diagram and relation to other subsystems (image from D2.3 – Pledger Overall Architecture[1]).</i>	10
<i>Figure 3: SLA subsystem deployment diagram in a Kubernetes cluster</i>	13
<i>Figure 4 PLEDGER SLA subsystem swagger interface for REST API</i>	14
<i>Figure 5 PLEDGER Orchestrator swagger interface for REST API</i>	23
<i>Figure 6 PLEDGER SLA tool swagger interface for REST API</i>	24

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	5 of 29	
Reference:	D3.3	Dissemination:	PU	
	Version:	1.0	Status:	Final

List of Acronyms

Abbreviation / acronym	Description
CNCF	Cloud Native Computing Foundation
CRUD	Create, Read, Update and Delete basic operations of any entity.
DSS	Decisions Support System
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
FC	Functional Component
FG	Functional Group
HTML	HyperText Markup Language
IaaS	Infrastructure-as-a-Service
K8S	Kubernetes software. Container orchestrator software.
Mx	Project Month x
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
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 document acts as a **description report** accompanying Deliverable “D3.3 – QoS and SLA assessment and negotiation tools I”, one of the three **prototype** deliverables of Work Package “WP3 – Performance, QoS and orchestration mechanisms”, and more specifically, that of Task “T3.3 – QoS and SLA Assessment and Negotiation tools”. This is the first version of the deliverable, with an updated and final version to be provided in M30.

The primary audience of this document consists of the members of the consortium that participate in the design and development of the components and modules of the Pledger pilots as well as of the Pledger core system to facilitate future integration activities. Additionally, this document could be of interest for stakeholders willing to adapt the Pledger solutions in the context of SLA Assessment and Negotiation Tools, especially to the ones that are expected to use or extend the results of Pledger in the future.

T3.3 focuses on the QoS Assessment Entity and its link to other components of the toolkits. It will provide functionalities to create, query, evaluate and monitor of SLAs based on definition of guaranteed levels and metrics about observed performance of the infrastructure and to alert in case of any SLA violations. It will also provide an interface to extract QoS information gathered from UCs experiments.

Following the above activities of the Task, this deliverable is focused on the Pledger **SLA subsystem**, one of the core subsystems of Pledger. The subsystem is 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”* as mentioned in D2.3 – Pledger Overall Architecture[1].

This deliverable explains the successful implementation of the first iteration of the PLEDGER SLA subsystem based in the reference architecture proposed in D2.3 PLEDGER Overall Architecture. This implementation includes the main components of the SLA subsystem with the functionality obtained for this first iteration.

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	7 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

1 Introduction

1.1 Purpose of the document

This document is a description report of deliverable D3.3 “QoS and SLA assessment and negotiation tools”. The specific deliverable is a prototype tool which is the outcome of the work carried out in T3.3 during the first iteration of WP3. This document is a short-accompanied description report of this demonstrator.

1.2 Relation to other project work

This document describes the links to other components of the toolkits (such as SLA Monitoring or the Edge/Cloud Orchestrator tools). It is also based on stakeholder information from WP2 (such as Requirements Analysis and Overall Architecture).

1.3 Structure of the document

This document is structured in 6 major chapters

Chapter 2 presents the functional description of the tool.

Chapter 3 presents the technical description of the tool (data model, interfaces, and component architecture).

Chapter 4 presents the installation and usage guide of the tool.

Chapter 5 presents de Demonstration of the tool with the description of the validation scenarios.

Chapter 6 presents the conclusions and next steps for this first iteration of the tool.

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 project.
- ▶ **Pledger core subsystems.** The subsystems that belong to the Pledger core system.
- ▶ **Quality of Service.** 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.

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	8 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

2 Functional description

This chapter introduces the main functionalities and functional components of Pledger developed within Task “T3.3 – QoS and SLA Assessment and Negotiation tools” and provides useful information related to the development of the corresponding Pledger **SLA subsystem**, one of the Pledger Core Subsystems (Figure 1), as they have been identified in Deliverable “D2.3 Pledger Overall Architecture”, which acts as the roadmap for all development and integration tasks of the project.

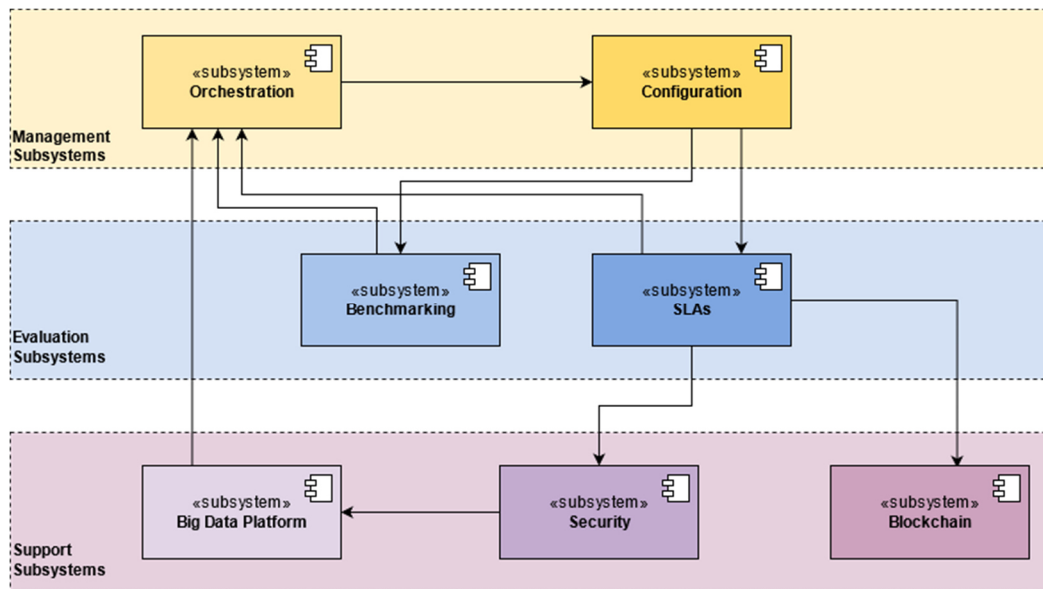


Figure 1: The Pledger Core Subsystems (image from D2.3 – Pledger Overall Architecture[1])

As one of the **Evaluation Subsystems** of Pledger, the QoS Assessment entity will be the tool responsible for reinforcing the Quality of Service (QoS) agreed between the provider of the infrastructure (IaaS/PaaS) and the consumer or service developer (SaaS).

This component assures performance and availability for software components captured as KPI or quantifiable metrics measurements within an SLA agreement. The QoS aspects are extracted from the SLA agreement and used as performance guarantees for the infrastructure used by the applications.

From the architecture point of view, the SLA 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 of the QoS agreed.

In the regular flow of the PLEDGER system, the IaaS consumer declares an expected application QoS and the system shows infrastructure of providers that could fulfill this QoS. This QoS is stored as guarantees in an SLA entity. When the SLA is agreed, the SLA subsystem starts an endless loop to evaluate if the threshold values associated to a metric inside a guarantee is met. In case of a value is not met, the SLA subsystem sends an alert violation to the rest of the PLEDGER system to take any predefined action (scaling, placement, etc.).

The instance values of selected metrics are provided by external monitoring tools and stored by metric collectors inside time-series databases like Prometheus. The SLA subsystem can define plugins to connect to different metric collector to make an ongoing evaluation.

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	9 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

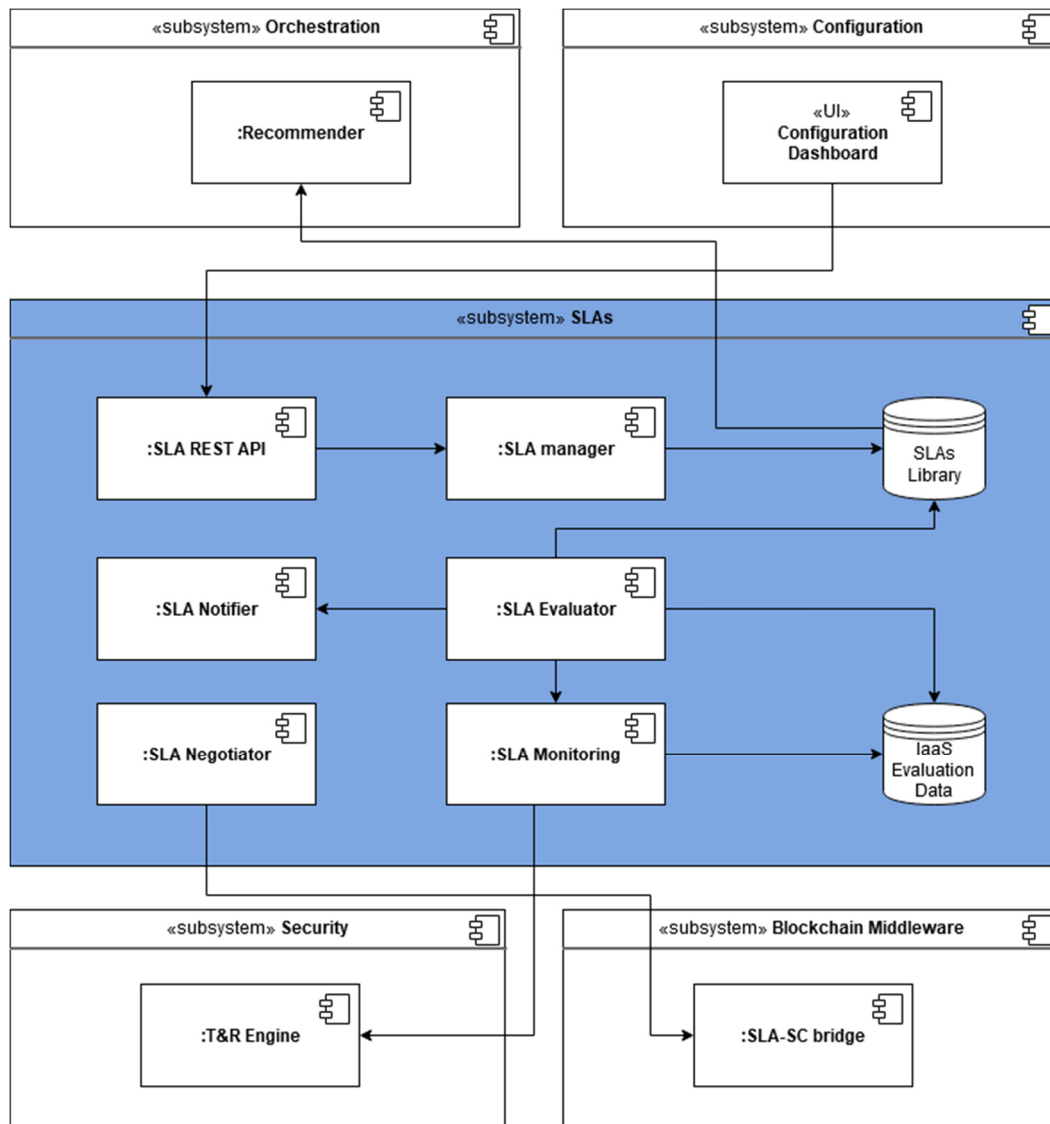


Figure 2: SLA subsystem Component Diagram and relation to other subsystems (image from D2.3 – Pledger Overall Architecture[1]).

More specifically, the subsystem comprises the following main Functional Components (FC):

Table 1: SLA subsystem Components (table extracted from D2.3 – Pledger Overall Architecture[1])

ID	Component	Functionality
FC.4.1	SLA REST API	The REST API module exposes all the methods needed to create and manage the SLAs, the SLA templates and the metrics associated to the applications running in the system. This module acts a facade and constitutes the entry point for other Pledger components and subsystems.
FC.4.2	SLA Manager	The SLA Manager component processes all the requests from the REST API. It offers two main features. On one hand, it is responsible for generating and storing the SLAs of the applications based on the QoS constraints defined by

		IaaS providers. On the other hand, this component handles the metrics used later by the Evaluator to check these SLAs.
FC.4.3	SLAs library	Both SLAs and associated metrics required for the SLA evaluation phase are stored in the SLAs library.
FC.4.4	SLA Monitoring	The SLA Monitoring component connects the SLA subsystem with a set of external monitoring tools by means of a set of plug-ins/ monitoring collectors (e.g. Prometheus) to get the metrics/ information about the applications and infrastructures behaviour.
FC.4.5	SLA Evaluator	The SLA Evaluator is the component responsible for the assessment of SLAs by checking that values obtained from monitoring metrics comply with the rules defined in the SLA template that correspond to a certain SLA stored in the system. First, it gets the metrics values from the monitor connectors (monitor module), and then it evaluates these metrics against the rules defined in the SLA applicable for a certain application or service.
FC.4.6	IaaS Evaluation Data	Both the monitoring and evaluator components collect the resultant information in the IaaS Evaluation DB.
FC.4.7	SLA Notifier	If the Evaluator detects a violation, it makes use of the SLA Notifier to spread this information to other Pledger components (e.g. the DSS). This component connects the SLA subsystem to other Pledger subsystems.
FC.4.8	SLA Negotiator	Through this component a SaaS reviews and accepts an SLA, ensuring there are no misunderstandings regarding the promised services and prices. In certain cases, modification of the available SLAs may be possible, facilitating real negotiation between IaaS providers and SLAs providers.

This deliverable will be focusing on FC.4.1-FC.4.7, while FC.4.8 will be presented as part of the second iteration of this tool.

3 Technical description

3.1 Baseline technologies and dependencies

The following baseline technologies are used within this component:

Table 2 Baseline technologies used by SLA tool

Name	Description	Version
SLA Manager	The SLA Manager is a framework that manages service-level agreements between service providers and consumers. It is being developed by ATOS under an open source license (Apache License 2.0), and it has been used in other H2020 projects.	latest
Prometheus [1]	Prometheus is an open source monitoring and metric collector system. This tool is licensed under Apache License 2.0 .	Prometheus Operator Stack
Kubernetes [1]	Kubernetes is an open source orchestration software for deploying, scaling, and management of containerized applications. This tool is licensed under Apache License 2.0. Kubernetes is now managed by the Cloud Native Computing Foundation (CNCF).	Vanilla 1.19+
Grafana [1]	Grafana is a data visualization tool to make dashboards with metrics data from Prometheus.	latest
Docker [1]	Container engine for containerized apps running in the edge	1.18+
Kafka [1]	Apache Kafka is a message broker software with event streaming functionalities.	latest
MongoDB [1]	MongoDB is a NoSQL database engine that stores data in JSON format (JSON documents). This an open source software licensed under Server Side Public License (SSPL).	latest

3.2 Components Architecture

The main functional components of this subsystem are described in the deliverable D2.3 Pledger Overall Architecture and showed in Table 1.

Most of the functional components are implemented in a single application (main module) coded in Golang language and containerized in one container (slalite-app). The datastores are implemented in MongoDB and are containerized in another container (slalite-db). The User Interface (UI) for this iteration in implemented in a single page application (SPA) with HTML and React JavaScript framework and containerized in another container (slalite-ui). The UI calls the REST API of the SLA main module and the agreements and violations are storage in the database.

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	12 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

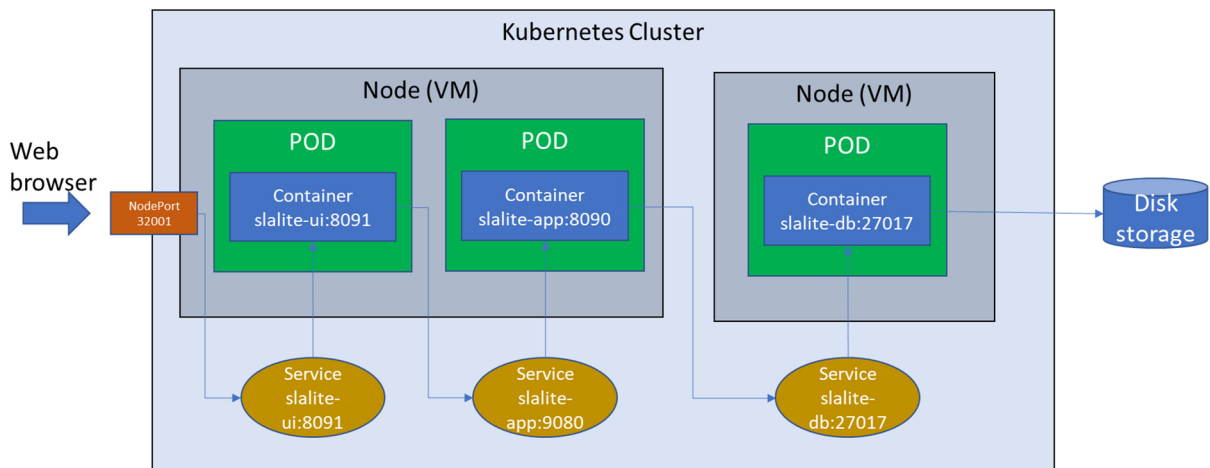


Figure 3: SLA subsystem deployment diagram in a Kubernetes cluster

Every container is deployed in a separated Pod and each Pod could be in the same node or in different nodes in the cluster.

3.3 Interfaces provided

This section describes the REST interfaces provided by SLA subsystem. Interfaces provided by the other tools or platforms like, for example, Kubernetes or Prometheus, can be found in their respective web sites and documentations.

Table 3 SLA operations with REST API

Operation	Method	URI	Description
LIST	GET	/agreements	Returns the list of all SLA entities in the datastore
READ	GET	/agreements/{id}	Returns the details of an SLA agreement with the ID requested.
DELETE	DELETE	/agreements/{id}	Delete an SLA with the id requested.
CREATE	POST	/agreements	Create an SLA (a JSON file with the SLA details must be provided)
UPDATE	PUT	/agreements/{id}	Update an SLA with the ID requested.
START	PUT	/agreements/{id}	Start the evaluation of an SLA (resume).
STOP	PUT	/agreements/{id}	Stop the evaluation of an SLA (pause).
TERMINATE	PUT	/agreements/{id}	End the evaluation of an SLA forever.

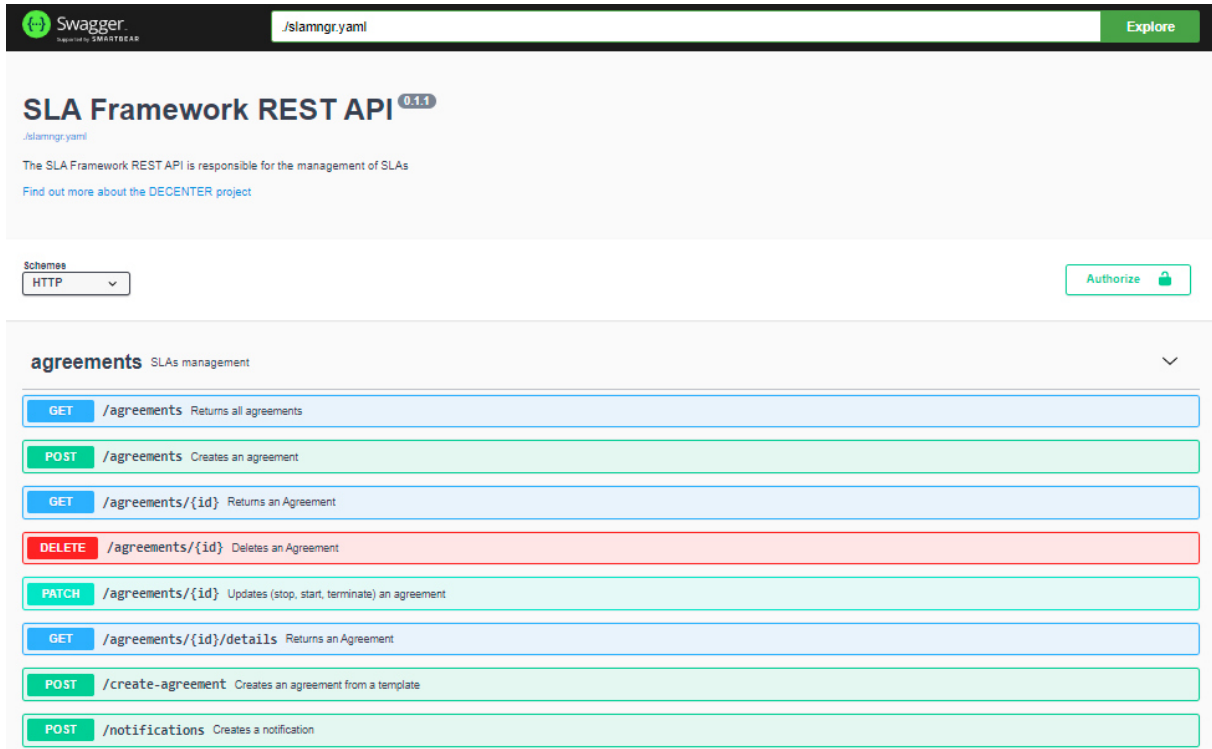


Figure 4 PLEDGER SLA subsystem swagger interface for REST API

3.4 Data models

An agreement has three different states (lifecycle):

- ▶ **STARTED**: Valid SLA that will be evaluated until expiration date. When an SLA is created, this is the first state. A change to STARTED state form STOPPED is a resume operation.
- ▶ **STOPPED**: Valid SLA that will not be evaluated until it changes to STARTED state and expiration date is a date in the future. It is pause operation.
- ▶ **TERMINATED**: SLA ended and archived for audit logs.

The agreement description (aka SLA) is defined in JSON format with the following schema:

```
{
  "id": "unique id in the system",
  "name": "an-agreement-name",
  "state": "started",
  "details": {
    "id": " unique id in the system",
    "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] comparison_op threshold",

```

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	14 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status:
			Final

```

    "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"
    }
    ]
}

```

A violation message is sent to the system and storage as a JSON format with the following schema:

```

{
    "id": "internal autogenerated id as primary key in the database",
    "agreement_id": "id of the agreement",
    "guarantee": "id of the guarantee violated. An SLA has a group of guarantees",
    "datetime": "timestamp of the violation detection",
    "constraint": "metric name + eval_operation + threshold",
    "values": "metric values that met threshold condition",
    "appid": "id of the app associated to this SLA",
    "importanceName": "severity of the violation: Mild, Serious, Severe,
Catastrophic",
    "importance": "severity of the violation in numeric format, e.g. Mild is 0"
    "description": "human readable message for this violation",
}

```

This is an example of a violation alert message:

```
{
  "id": "tnzothug57duue1k5aks9m1618332570950442900",
  "agreement_id": "tnzothug57duue1k5aks9m1618332570950442900",
  "guarantee": "scrape_duration_seconds",
  "datetime": "2021-04-13T17:27:06Z",
  "constraint": "scrape_duration_seconds \u003c 0.04",
  "values": [{
    "key": "scrape_duration_seconds{10.244.0.8:3000}",
    "value": 2.8476827,
    "datetime": "2021-04-13T17:27:06Z"
  }],
  "importanceName": "Serious",
  "importance": 1
  "appid": "tnzothug57duue1k5aks9m1618332570950442900"
  "description": "This is a violation of the agreement of app X"
}
```

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	16 of 29				
Reference:	D3.3	Dissemination:	PU	Version:	1.0	Status:	Final

4 Installation and usage guides

4.1 Requirements

The following software components and credentials must be in place to install the SLA subsystem:

- ▶ Kubernetes cluster ver. 1.19+ (this cluster will be the broker cluster)
- ▶ Credentials of K8S cluster with permission to deploy applications.
- ▶ Persistence storage for MongoDB in K8S cluster.
- ▶ Kafka client certificate KeyStore and passwords as secrets object in K8S cluster.
- ▶ Kafka endpoints as IP address of Kafka servers.
- ▶ The container images of the subsystem in a container image repository. It could be the image repository of the project (JFROG) in the following url: 116.203.2.204:443/plgregistry
- ▶ A monitoring system or metric collector (Prometheus) in the K8S broker cluster.

4.2 Installation

The first step is to download the source code form the repository:

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

After then you need to build the docker image of the UI and the core tier:

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

To install the SLA subsystem, you need the command line tool **kubectl** from K8S. It is also possible to install this subsystem in the testbed of the project with CI/CD tools (Jenkins pipelines).

To customize the installation to your environment you can use “kustomizer” tool and the corresponding yaml files. Some examples of this customization can be seen in the subdirectory “overlays” of the project.

For example, the command to install this subsystem in the testbed is the following:

```
kubectl apply -k overlays/testbed
```

For more details of manifest files (yaml) can be found in <https://gitlab.com/pledger/sla-framework/-/tree/master/base> and <https://gitlab.com/pledger/sla-framework/-/tree/master/overlays/testbed>.

There are several environment variables to configure before the installation of the subsystem. These vars are declared in the manifest files of the subsystem (yaml files). This is the relation of vars:

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	17 of 29				
Reference:	D3.3	Dissemination:	PU	Version:	1.0	Status:	Final

Table 4 SLA tool environment variables for configuration

Name	Description	Default	Mandatory	Example
SLA_REPOSITORY	DB engine type	:memory:	No	mongodb
SLA_DATABASE	DB instance	Slalite	No	slalite
SLA_DB_USER	DB user		No	user
SLA_DB_PASS	DB password		No	password
SLA_CONNECTION	DB url connection string		No	mongodb://\$(SLA_DB_USER):\$(SLA_DB_PASS)@slalite-db:27017
SLA_ADAPTER	Metric collector type		Yes	prometheus
SLA_PROMETHEUSURL	Prometheus url endpoint		No	http://kube-prometheus-stack-prometheus.monitoring:9090
SLA_NOTIFIER	Notifier type	Logging	No	kafka2
SLA_KAFKA_ENDPOINT	Kafka servers' addresses		No	static.180.8.203.116.clients.your-server.de:9093
SLA_KAFKA_NOTIFIER_TOPIC	Kafka topic for produce sla violation alerts		No	sla_violation
SLA_KAFKA_AGREEMENT_TOPIC	Kafka topic for consume agreements info		No	sla_agreement
SLA_KAFKA_KEYSTORE_PASSWORD	Kafka KeyStore password for client certificates		No	password
SLA_KAFKA_KEY_PASSWORD	Kafka private key password for client certificates		No	password
SLA_KAFKA_TRUSTSTORE_PASSWORD	Kafka truststore password for server public certificates (CA)		No	password
SLA_KAFKA_KEYSTORE_LOCATION	Kafka KeyStore location in container filesystem		No	/var/kafka_keystore/kafka.client.keystore.p12
SLA_KAFKA_TRUSTSTORE_LOCATION	Kafka truststore location in container filesystem		No	/var/kafka_truststore/kafka.client.truststore.pem

4.3 Usage

We can manage the subsystem via REST API calls or by means of the UI component (web console). For the list of API calls you can find the swagger documentation at an annex of this document and in the following URL of the UI component: <https://cluster-external-ip:slalite-ui-svc-port/sla#/> (e.g. <http://localhost:32001/sla#/>).

The main operations for this subsystem are the lifecycle management of SLAs (Create, Read, Update and Delete (CRUD) operations). Every SLA represent an agreement between the provider and the consumer of Pledger system.

This an example of agreement:

```
{
  "id": "2018-000234",
  "name": "an-agreement-name",
  "details": {
    "id": "2018-000234",
    "type": "agreement",
    "name": "an-agreement-name",
    "provider": { "id": "a-provider", "name": "A provider" },
    "client": { "id": "a-client", "name": "A client" },
    "creation": "2018-01-16T17:09:45Z",
    "expiration": "2019-01-17T17:09:45Z",
    "guarantees": [ {
      "name": "TestGuarantee",
      "constraint": "[execution_time] < 100"
    } ]
  }
}
```

We can also invoke the REST API via swagger HTML interface published in this URL: <https://cluster-external-ip:slalite-ui-svc-port/swaggerui/#/> (e.g. <http://localhost:32000/swaggerui/#/>). We can see an example in the demonstration chapter of this document.

Examples of SLA management via REST API and curl tool:

Add an agreement (agreement below is stopped):

```
curl -k -X POST -d @resources/samples/agreement.json
http://localhost:8090/agreements
```

Change agreement state:

```
curl -k http://localhost:8090/agreements/a02 -X PATCH -
d '{"state": "started"}'
```

Get all agreements and an agreement by id:

```
curl -k http://localhost:8090/agreements
curl -k http://localhost:8090/agreements/a02
```

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	19 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

4.4 Licenses

This SLA subsystem has an Apache license type version 2 for research and academic purposes. The final license is been evaluated.

4.5 Source code repository

The source code repository for this subsystem is the source code repository of the project hosted in GitLab.com in private repositories.

The URL of the repository of this subsystem is this: <https://gitlab.com/pledger/sla-framework> This URL directs to a private GitLab repository. Depend on the decision of final license this code could be publish in a public repository.

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	20 of 29				
Reference:	D3.3	Dissemination:	PU	Version:	1.0	Status:	Final

5 Demonstration

5.1 Scenario description

Scenario 1:

Objectives: Demonstrate configuration of infrastructure and applications (resources and QoS). Deployment of applications in the infrastructure. Check QoS and take remediation action at runtime.

Script:

1. Add a cluster infra configuration
2. Add an app deployment config with QoS params and actions
3. Deploy the app with the Orchestrator in the new cluster
4. Check QoS threshold (metric) from Monitoring with SLA tool
5. Send a violation alert from SLA to message middleware
6. Orchestrator read violation alert from middleware and execute an action (*scale out*)

Scenario 2:

Objectives: Demonstrate placement of applications between Edge and Cloud as a runtime adaptation.

Script:

1. Add cluster and edge infra configuration
2. Add an app deployment config with QoS params and actions
3. Deploy the app with the Orchestrator in the target location
4. Check QoS threshold (metric) from Monitoring with SLA tool
5. Send a violation alert from SLA to message middleware
6. Orchestrator read violation alert from middleware and execute an action (*placement*)

5.2 Validation and Verification

This first iteration of SLA subsystem addresses the MVP requirements for this first iteration of the tool. The System User Cases (SUC) that were implemented in this iteration were the following:

- ▶ SUC.27 Create SLA template
- ▶ SUC.28 Update/delete SLA template
- ▶ SUC.29 Monitor compliance with SLAs
- ▶ SUC.31 Accept SLA
- ▶ SUC.24 Check reliability (of IaaS)

These SUCs were defined in D2.3 PLEDGER Overall Architecture.

For the verification of this first iteration we present a demonstration with two different and integrated scenarios to show the implementation of the MVP's SUCs (at least the first version of them) and with the evidences in video recording format.

For the validation of the European Commission (EC) of the work done we made this document to help in the process of validating of this first iteration of the SLA tool describing the work done, conclusion and next steps.

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	21 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

5.3 Demo

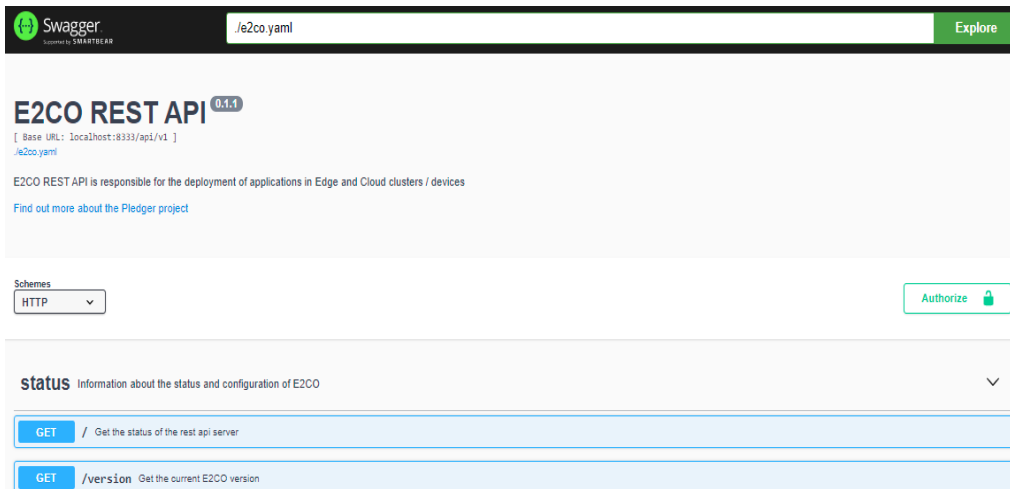
5.3.1 Scenario 1: SCALE OUT

1. Add a cluster infra configuration

The description file for add a new infra cluster of Kubernetes for this demo scenario is the following:

```
{
  "id": "mycluster",
  "description": "Pledger Testbed",
  "type": "k8s",
  "defaultNamespace": "core",
  "restAPIEndPoint": "https://kubernetes.default:443",
  "connectionToken": "eyJh... sYWA",
  "slaliteEndPoint": "http://slalite-app:9080",
  "prometheusEndPoint": ""
}
```

We can add this new infra configuration via API REST call using the swagger interface of the orchestrator as shown below (http://<Pledger_cluster_IP>:31000/swaggerui/#/ime/addOrchestrator):



Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	22 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

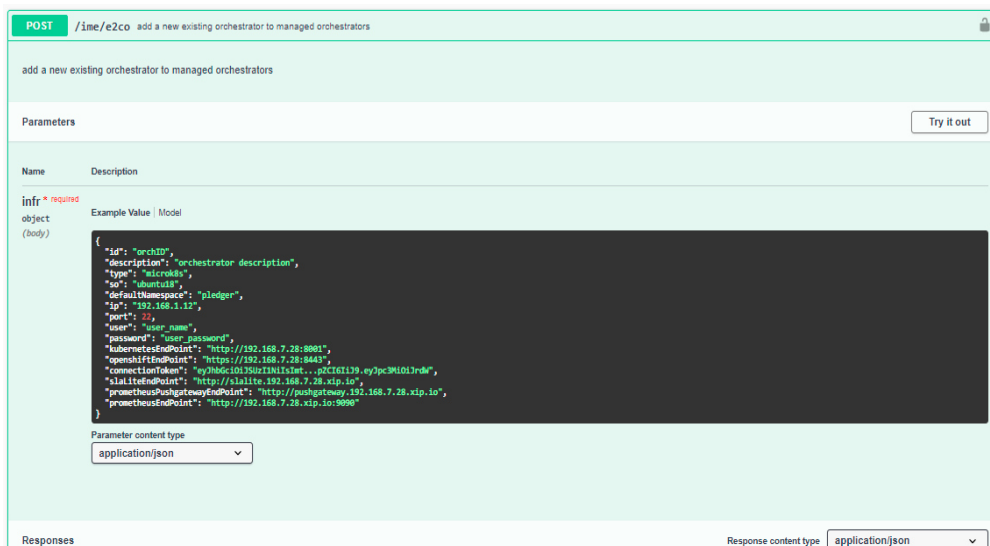


Figure 5 PLEDGER Orchestrator swagger interface for REST API

We can also use the Orchestrator UI or another tool that can make REST API calls.

2. Add an app deployment config with QoS params and actions

To deploy a new application, we use a description file like the following:

```
{
  "name": "myapp",
  "namespace": "core",
  "idE2cOrchestrator": "mycluster",
  "qos": [{
    "name": "scrape_duration_seconds",
    "constraint": "scrape_duration_seconds < 0.4",
    "actions": [{
      "type": "SCALE OUT",
      "value": "1",
      "unit": ""
    }]
  }],
  "replicas": 1,
  "maxReplicas": 3,
  "containers": [
    {
      "name": "nginx",
      "image": "nginx",
      "ports": [
        {
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "TCP"
        }
      ]
    }
  ]
}
```

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	23 of 29	
Reference:	D3.3	Dissemination:	PU	
	Version:	1.0	Status:	Final

As usual, we can create this app deployment using the Orchestrator REST API with Orchestrator UI, swagger HTML interface or others external tool.

The highlight point of this JSON file for SLA subsystem is the part of quality of service (qos). Here the consumer defines the requirements to fulfill at runtime for the infrastructure expressed by means of metrics and thresholds.

For this demo we add a rule-based actions (actions) to execute when the agreement is not met (scale out) to allow the system to recover the QoS expected.

3. Deploy the app with the Orchestrator in the new cluster

The Orchestrator subsystem deploys the app in the Kubernetes cluster and send an API rest call to the SLA subsystem to create the SLA agreement of the new app with the QoS defined by the consumer.

We can reproduce this call using the swagger interface of the SLA like this example or with others external tools (URL http://<Pledger_cluster_IP>:32000/swaggerui/):

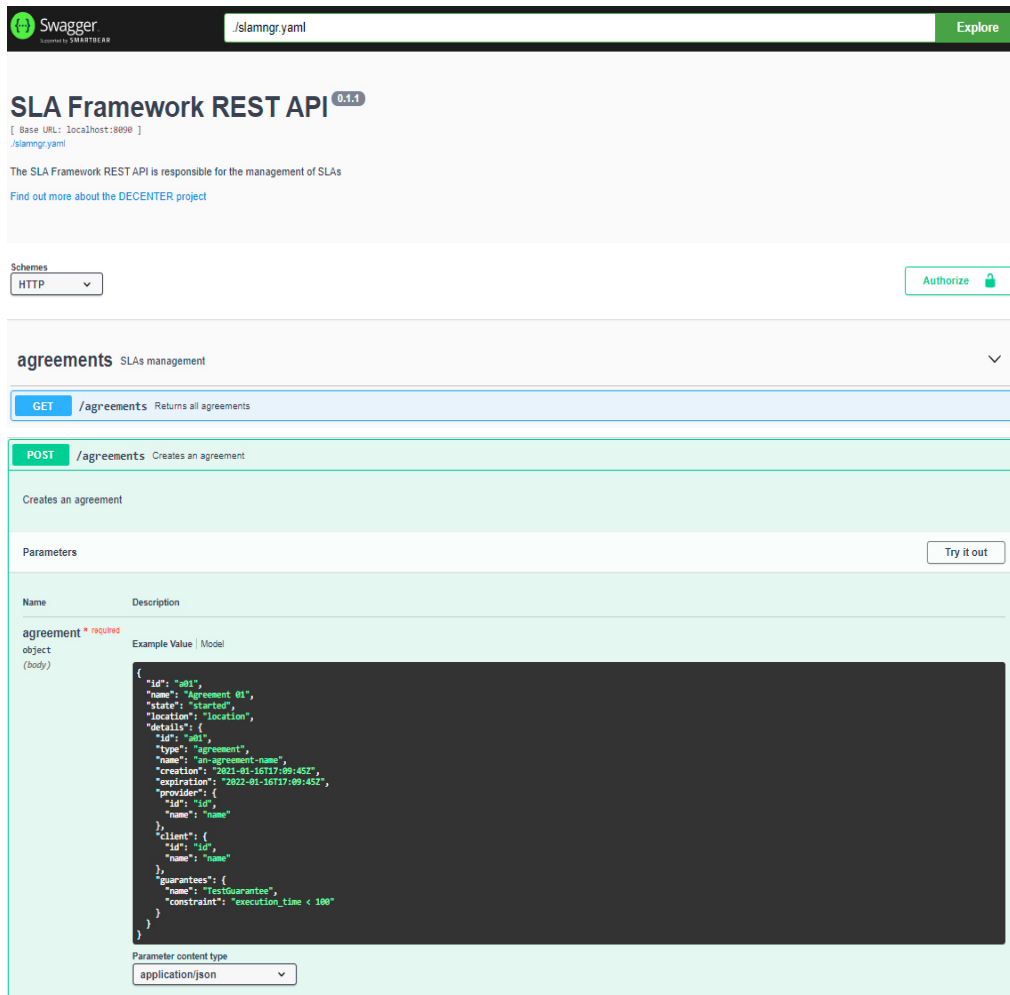


Figure 6 PLEDGER SLA tool swagger interface for REST API

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	24 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

4. Check QoS threshold (metric) from Monitoring with SLA tool

The SLA tool starts a monitoring loop to check the metrics values every sleep time parameter. When it detects a violation of the metric (threshold value reaches) it creates a violation alert with the details of the agreement violated.

5. Send a violation alert from SLA to message middleware

To notify the rest of the system about a violation event we use a message broker system (Kafka) as a core component of Pledger. The SLA tool publish the violation message with the topic “sla_violation” and any component subscribed to this topic could receive the notification.

6. Orchestrator read violation alert from middleware and execute an action (*scale out*)

For this first iteration of the Pledger tools the orchestrator will be subscribed to this alert and will implement an adaptation action based in the rules defined by the consumer. For this demo scenario, it will be a scale out of the number of instances of the app until the maxReplicas parameter (max instance per app in the cluster).

Video recording link in PLEDGER YouTube channel:

<https://www.youtube.com/channel/UCXV6V9rJ0ZvWhXeoWvDsArQ/videos%3A%2520PLEDGER%2520WP3%2520Demo%2520M18%2520Scenario%25201.mp4>

5.3.2 Scenario 2: PLACEMENT

1. Add cluster and edge infra configuration

This first step is like the first step of scenario 1 but in this case, we must define two infrastructure locations:

- ▶ EDGE: docker engine runtime
- ▶ CLOUD: Kubernetes cluster

The description file for the Edge is the following:

```
{
  "id": "mycluster2",
  "name": "Remote edge docker",
  "description": "Pledger remote testbed",
  "type": "docker",
  "so": "windows",
  "user": "remote_user (windows account)",
  "password": "remote_password",
  "restAPIEndPoint": "http://192.168.1.24:2375",
}
```

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	25 of 29	
Reference:	D3.3	Dissemination:	PU	
	Version:	1.0	Status:	Final

2. Add an app deployment config with QoS params and actions

The deployment config is like the scenario 1 but we implement a different action (placement) to move the app from the edge to the cloud (from mycluster2 to mycluster).

This is the QoS part defined by the consumer in the app descriptor:

```

"qos": [
  {
    "name": "scrape_duration_seconds",
    "constraint": "scrape_duration_seconds < 0.04",
    "importance": [
      {
        "Name": "Mild",
        "Constraint": " > 0.04"
      },
      {
        "Name": "Serious",
        "Constraint": " > 0.4"
      }
    ],
    "actions": [{
      "type": "MIGRATE",
      "value": "mycluster",
      "unit": ""
    }]
  }
]

```

3. Deploy the app with the Orchestrator in the target location

This step is like in scenario 1 but in this case the Orchestrator deploy the app in a remote site using Docker HTTP/S API.

4. Check QoS threshold (metric) from Monitoring with SLA tool

This step is like in scenario 1 but we could check the metric values in the metric collector of the PLEDGER cluster or in the remote site. For this iteration, we will use the metric collector of PLEDGER broker cluster (Prometheus based).

5. Send a violation alert from SLA to message middleware

This step is the same as in scenario 1.

6. Orchestrator read violation alert from middleware and execute an action (*placement*)

In this step, the Orchestrator undeploys the app from the remote edge (docker engine) and redeploys the app in the cloud (Kubernetes cluster).

Video recording link in PLEDGER YouTube channel:

<https://www.youtube.com/channel/UCXV6V9rJ0ZvWhXeoWvDsArQ/videos: PLEDGER WP3 Demo M18 Scenario 2.mp4>

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	26 of 29	
Reference:	D3.3	Dissemination:	PU	
	Version:	1.0	Status:	Final

6 Conclusions and next steps

This deliverable D3.3 reported the work done in WP3 Task T3.3 in M6-M18. The target milestone MS4 “First iteration of WP3 prototypes” has been achieved and documented in this deliverable as well as in D3.1 and D3.2. This will be a good start position for the next milestone MS7 “Second iteration of WP3 prototypes” in M33.

This first iteration of the SLA subsystem addresses the MVP requirements for the tool. The System User Cases (SUC) that were implemented in this iteration were the following:

- ▶ SUC.27 Create SLA template
- ▶ SUC.28 Update/delete SLA template
- ▶ SUC.29 Monitor compliance with SLAs
- ▶ SUC.31 Accept SLA
- ▶ SUC.24 Check reliability (of IaaS)

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

The progress done will be the basis for the next phase where the goal will be evaluating the Use Cases using the SLA subsystem presented in this deliverable. For this purpose, we will continue the development and improvement of the SLA subsystem.

The suggested next steps are the following:

- ▶ Creation of SLA from PLEDGER console/UI (Configuration Dashboard component). In this iteration, the creation is sent by the orchestrator when the app is deployed.
- ▶ More complex integration with Decision Support System (DSS) will be expected.
- ▶ Query tools for historical tracks of SLA monitoring and violation (e.g. REST API).
- ▶ Integration with Blockchain subsystem to create SLA agreements in a distributed ledger.
- ▶ Integration with remote metric collectors in the Cloud based in Prometheus if a Prometheus federation is not available. Integration with metric collectors in the edge without Prometheus (e.g. Docker engine metrics, node-exporter, etc.).
- ▶ SLA tool internal metrics in Prometheus format (e.g.: violation counters, uptime, etc.).
- ▶ QoS policies of the infrastructure expected performance expresses in TOSCA template format.

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	27 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final

7 References

- [1] PLEDGER. *D2.3 - Pledger Overall Architecture v1.0*. Voutyras Orfefs. 2020
- [2] Prometheus home page. <https://prometheus.io>, retrieved 2021-05-25
- [3] Kubernetes home page. <https://kubernetes.io>, retrieved 2021-05-25
- [4] Grafana home page. <https://grafana.com>, retrieved 2021-05-25
- [5] Docker home page. <https://www.docker.com>, retrieved 2021-05-25
- [6] Kafka home page. <https://apache.kafka.org>, retrieved 2021-05-25
- [7] MongoDB home page. <https://www.mongodb.com>, retrieved 2021-05-25

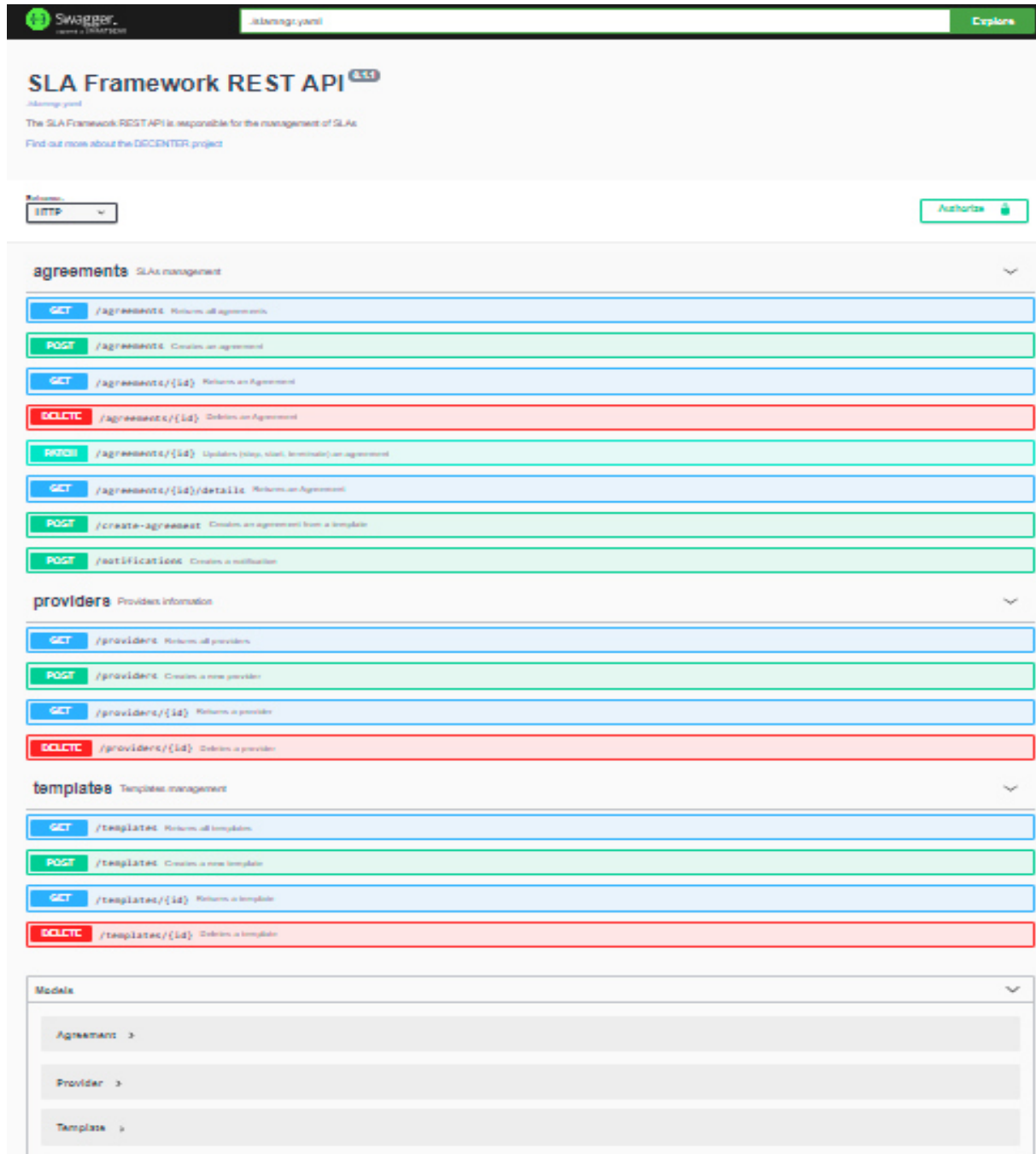
Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	28 of 29				
Reference:	D3.3	Dissemination:	PU	Version:	1.0	Status:	Final

Annexes

SLA subsystem REST API in Swagger format (yaml):



Package



Swagger `slafw.yml` [Explore](#)

SLA Framework REST API

The SLA Framework REST API is responsible for the management of SLAs. Find out more about the DOCCENTER project.

Reference: [Authorize](#)

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

providers Provides information

- GET** /providers Returns all providers
- POST** /providers Creates a new provider
- GET** /providers/{id} Returns a provider
- DELETE** /providers/{id} Deletes a provider

templates Templates management

- GET** /templates Returns all templates
- POST** /templates Creates a new template
- GET** /templates/{id} Returns a template
- DELETE** /templates/{id} Deletes a template

Models

- Agreement >
- Provider >
- Template >

Document name:	D3.3 QoS and SLA assessment and negotiation tools I	Page:	29 of 29
Reference:	D3.3	Dissemination:	PU
	Version:	1.0	Status: Final