



## D5.6 Pledger integrated demonstrator II

Document Identification			
Status	Final	Due Date	31/08/2022
Version	1.8	Submission Date	01/09/2022

Related WP	WP5	Document Reference	D5.6
Related Deliverable(s)	D5.2 “Pledger integrated demonstrator I” D5.4 “Pilots operations and monitoring II” D3.5 “Edge/Cloud orchestration tools II”	Dissemination Level (*)	PU
Lead Participant	INTRA	Lead Author	Panos Matzakos (INTRA) Ioannis Sarris (INTRA) Olga Segou (INTRA)
Contributors	ATOS, ENG, i2CAT, ICCS, INNOV, FILL	Reviewers	Nikos Kapsoulis (INNOV) Estela Carmona (i2CAT)

### Keywords:

System integration, CI/CD, Pledger Core System, Microservices, Demonstrators

#### Disclaimer for Deliverables with dissemination level PUBLIC

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
Panos Matzakos	INTRA
Ioannis Sarris	INTRA
Olga Segou	INTRA
Estela Carmona	i2CAT
Adrián Pino	i2CAT
Antonio Castillo	ATOS
Roi Sucasas	ATOS
Gabriele Giammatteo	ENG
Francesco Iadanza	ENG
Alexandros Psychas	ICCS
Nikos Kapsoulis	INNOV
Verena Stanzl	FILL

Document History			
Version	Date	Change editors	Changes
0.1	06/05/2021	Ioannis Sarris (INTRA)	Initial ToC
0.2	24/06/2022	Antonio Castillo (ATOS)	Updated section 2.2.1 integration endpoint status table and provided inputs in section 3.1
0.3	30/06/2022	Gabriele Giammatteo (ENG)	Updated section 2.2.1 integration endpoint status table and provided inputs in section 3.2
0.4	04/07/2022	Estela Carmona (i2CAT)	Updated section 2.2.1 with analytic description of the updated integration endpoints and provided inputs in section 3.4
0.5	06/07/2022	Alexandros Psychas (ICCS)	Updated section 3.2
0.6	07/07/2022	Nikos Kapsoulis (INNOV)	Provided inputs in section 3.3
0.7	08/07/2022	Panos Matzakos (INTRA)	New draft version containing the received contributions from partners
0.8	21/07/2022	Panos Matzakos (INTRA), Francesco Iadanza (ENG), Nikos Kapsoulis (INNOV)	Added Executive summary, Introduction and Conclusion sections; Updates in sections 3.1, section 3.3.2
0.9	25/07/2022	Nikos Kapsoulis (INNOV) Roi Sucasas (ATOS) Verena Stanzl (FILL)	Updates in section 2.1 for Pledger integration plan; Updates in section 3.3.
1.0	26/07/2022	Panos Matzakos (INTRA)	Version ready for internal review
1.1	28/07/2022	Estela Carmona (i2CAT)	Reviewed version from i2CAT
1.2	01/08/2022	Nikos Kapsoulis (INNOV)	Reviewed version from INNOV
1.3	02/08/2022	Panos Matzakos (INTRA)	Version after addressing internal review comments

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	2 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b>
			Final

Document History			
Version	Date	Change editors	Changes
1.4	23/08/2022	Panos Matzakos (INTRA)	Final version after addressing second round of internal review comments
1.5	26/08/2022	Carmen San Roman (ATOS)	Quality assurance review
1.6	31/08/2022	Lara Lopez Muñiz (ATOS)	Reviewed version
1.7	01/09/2022	Panos Matzakos (INTRA)	Version after addressing the received comments
1.8	01/09/2022	Lara Lopez Muñiz (ATOS)	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Panos Matzakos (INTRA)	23/08/2022
Quality manager	Carmen San Roman Alonso (ATOS)	26/08/2022
Project Coordinator	Lara Lopez Muñiz (ATOS)	31/08/2022

## Table of contents

---

Document information .....	2
Table of contents .....	4
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 .....	9
2 Pledger core system integration updates .....	10
1.4 Updated Pledger integration plan.....	10
1.4.1 Pledger integrated demonstrator II .....	13
1.5 Integration matrix updates.....	14
1.5.1 Integration endpoints updates .....	16
3 Pledger core system demonstrators .....	22
1.6 Offloading in the edge-cloud continuum using SLAs.....	22
1.6.1 General description.....	22
1.6.2 Architecture .....	22
1.6.3 Scenario .....	23
1.7 Infrastructure benchmarking and application profiling.....	25
1.7.1 General description.....	25
1.7.2 Architecture .....	26
1.7.3 Scenario .....	27
1.8 Blockchain SLA management.....	28
1.8.1 General description.....	28
1.8.2 Architecture .....	29
1.8.3 Scenario .....	29
1.9 End to end slicing.....	31
1.9.1 General description.....	31
1.9.2 Architecture .....	33
1.9.3 Scenario .....	34
4 Conclusions .....	36
5 References .....	37

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	4 of 37	
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.8	<b>Status:</b>	Final

## List of tables

---

<i>Table 1: Pledger integration plan.....</i>	<i>11</i>
<i>Table 2 Pledger core system integration matrix.....</i>	<i>15</i>
<i>Table 3: Pledger core system integration endpoints.....</i>	<i>16</i>
<i>Table 4: Orchestrator – PLEDGER SOE Framework integration endpoint description.....</i>	<i>19</i>
<i>Table 5: PLEDGER SOE Framework - PLEDGER RAN Controller integration endpoint description.....</i>	<i>21</i>

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	5 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

## List of figures

---

<i>Figure 1: DSS interactions with Pledger core components</i> .....	23
<i>Figure 2: Demo steps and components involved</i> .....	24
<i>Figure 3: Benchmarking suite and app profiler integration architecture</i> .....	26
<i>Figure 4: Application performance assessment operation architecture</i> .....	26
<i>Figure 5: Infrastructure benchmarking and application profiling integration demo flow</i> .....	27
<i>Figure 6: Blockchain SLA management setup</i> .....	29
<i>Figure 7: SLASC bridge blockchain SLA management procedure</i> .....	30
<i>Figure 8: End-to-end network slice with IEEE 802.11p radio elements</i> .....	31
<i>Figure 9: Deployment of two independent end-to-end network slices with 5G new radio elements</i> .....	32
<i>Figure 10: Required integrations among Pledger components for the deployment of network slices based on IEEE 802.11p radio nodes</i> .....	33
<i>Figure 11: Required integrations among Pledger components for the deployment of 5G network slices</i> .....	34

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	6 of 37	
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.8	<b>Status:</b>	Final

## List of acronyms

Abbreviation / acronym	Description
API	Application Programming Interface
App	Application
CD	Continuous Deployment
CI	Continuous Integration
DB	Data Base
DLT	Distributed Ledger Technology
DSS	Decisions Support System
GPU	Graphics Processing Unit
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IDPS	Intrusion Detection and Privacy System
JDBC	Java Data Base Connectivity
K8s	Kubernetes
MANO	Management and Orchestration
ML	Machine Learning
OSM	Open Source MANO
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
SaaS	Software as a Service
SLA	Service Level Agreement
SLASC Bridge	SLA to Smart Contract Bridge
SOE	Slicing and Orchestration Engine
T&R Engine	Trust and Reputation Engine
UI	User Interface
UC	Use Case
VLAN	Virtual Local Area Network
VM	Virtual Machine

## Executive summary

---

Task 5.2 “Integration and Demonstrators setup” handles the integration activities that need to be carried out in order to produce the Pledger integrated demonstrators.

The deliverable D5.2 - Pledger integrated demonstrator I [1] described the integration methodology and activities that were carried out towards the first release of the Pledger integrated demonstrator. These activities covered, among others, the continuous integration / continuous development (CI/CD) pipeline deployment and the analytic description and implementation status of the integration endpoints between software components/services that form the end-to-end Pledger core system. The current deliverable describes the updates on the integration tasks that follow the defined methodology. These tasks cover the updates on the implementation status and functional descriptions of the integration endpoints towards the second and final release of the Pledger integrated demonstrator. Furthermore, the updates on the plan and status for the integration of Pledger Core System with the project use cases are also highlighted. In the same direction, four selected demonstrators of the Pledger core system are described in detail, to highlight the readiness of some key features of the platform that are also being exploited from one or more project use cases.

The first demo is on the offloading from Cloud/Edge infrastructures using SLAs. It demonstrates how an application (App) can run in Pledger system while satisfying quality of service (QoS) requirements (registered in a service level agreement (SLA) object) and keeping the App on the edge as much as possible, thus balancing between saturating the edge computation capacity and guarantying the promised SLAs and applying Decision Support System (DSS) optimizations.

The second demo is on the infrastructure benchmarking and application profiling. It demonstrates the interoperability among multiple components of the Pledger core system to maximize an application’s performance by deploying it in the most suitable infrastructure node, with respect to its resource capabilities and the application requirements according to the resource usage profile created by the App Profiler.

The third demo is on the Blockchain SLA Management. It demonstrates the corresponding SLA configuration workflow relevant to the creation and deployment of the respective smart contract structures. In this context, the interoperability between the candidate structural entities is required to provide the business intelligence of the SLA contractual terms into the corresponding blockchain equivalents.

The fourth demo is on the orchestration and deployment of end-to-end network slices using different radio access technologies over the Pledger platform. It showcases two different scenarios for network slicing based on IEEE 802.11p and 5G new radio, using different virtualization technologies while providing connectivity to the end users.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	8 of 37				
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	<b>Version:</b>	1.8	<b>Status:</b>	Final

# 1 Introduction

---

## 1.1 Purpose of the document

---

This deliverable presents the work performed during the second and final iteration of Task 5.2 (T5.2) “Integration and Demonstrators setup”. As reported in D5.2 [1], the first iteration released in M24 introduced the main components of the Pledger core, as well as the methodology for the integration activities that lead to the first release of the end-to-end prototype of the Pledger Core system. This methodology covered the definition of a CI/CD pipeline; the definition of common integration patterns based on the use of widely adopted approaches (e.g., microservices deployed in a cluster environment, REST APIs and publish/subscribe interfacing techniques); a full list and analytic presentation of integration endpoints that describe the interoperability process and status among bilateral software components.

The purpose of this document is to present the updates performed towards the second release of the Pledger Core System and integrated demonstrator, following the highlighted methodology. More specifically, these updates cover:

- ▶ The updates on the integration plan, mainly with respect to the integration of the Pledger core system with the project use cases.
- ▶ The updates concerning the implementation status of the defined integration endpoints.
- ▶ The analytic description of selected demonstrators that showcase the functionality of some key features of the Pledger Core System.

## 1.2 Relation to other project work

---

This deliverable builds on deliverable D5.2 [1] reporting the previous iteration of the Pledger integrated demonstrator, where the integration plan, the integration methodology and the analytic description of the integration endpoints are presented in detail. Moreover, D5.4 – Pilots operation and monitoring II [2] describes the essential features of the Pledger platform for its integration with the different project use cases. Some of these features are covered in the dedicated demonstrators of the Pledger core system which are described in section 3. Finally, the functionality of some components participating in the demonstrators (section 3) has been described in the corresponding demonstrations section (section 5) of D3.5 “Edge/Cloud orchestration tools II” [5] regarding the Pledger Orchestration and Configuration Subsystem.

## 1.3 Structure of the document

---

This document is structured as follows:

- ▶ **Introduction (present section)** presents the document scope and relation to the project work.
- ▶ **Section 2** describes the updates on the integration process of the Pledger core system, capturing the updates according to the integration plan, as well as the specific integration endpoints description and implementation status updates.
- ▶ **Section 3** describes in detail the four Pledger Core System demonstrators that were implemented to showcase key features of the platform.
- ▶ **Section 4** highlights the main outcomes of this document, that finalize the work of T5.2.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	9 of 37				
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	<b>Version:</b>	1.8	<b>Status:</b>	Final

## 2 Pledger core system integration updates

---

The Pledger core system was planned to be delivered in two releases: the first one in M24 (November 2021) and the final one in M33 (August 2022). As described in D5.2 [1], the effort was to provide a common integration framework that enables interoperability among various software components, which originate from different developer teams, are written in different programming languages etc. The approach to achieve this is based on the use of micro-services that have been appropriately extended to interact with each other through widely adopted interface methods (e.g., REST APIs, publish/subscribe mechanisms) within the K8s production cluster of the Pledger Core System. The objective was to provide in the end the capability to perform efficient end-to-end tests and different sorts of demonstrations related with the integration of the three different project use cases with the Pledger Core system. Furthermore, an appropriate CI/CD framework was adopted and delivered through the first release of the platform to enable automation in building, testing and deploying new software component versions, thus bridging the gap between development and daily testing operations. In D5.2 [1], the methodology of the integration activities, the analytic description of the bilateral integration endpoints and their integration status, as well as the timeline towards the final release of the Pledger core system were described in detail.

As described in section 1.1, in the current document the updates of this integration process towards the final release of the Pledger Core system are provided. In the current section, we provide:

- ▶ The updated plan for the Pledger core system integration
- ▶ The updates on the interoperability status of the existing integration endpoints compared to what was reported in D5.2 [1].
- ▶ The description of new implemented bilateral endpoints, compared to what was reported in D5.2 [1], following the evolution of the platform towards its final release.

### 2.1 Updated Pledger integration plan

---

Table 1 presents the whole Pledger integration plan. Having successfully delivered the expected outcomes for the first Pledger release (M24), the focus of the current deliverable is on the reporting period from M24 to M33 that indicates the second and final release of the Pledger core system and the integrated demonstrator. As highlighted in Table 1, this covers the full integration of the components/endpoints whose first versions were reported in D5.2 [1]. Compared to the first release, some of the integration endpoints which were only partially fulfilled are now finalized in the second release. For example, in UC1 the Orchestrator can now automatically trigger the launch of the application; in UC2 the Orchestrator is now integrated with the SOE framework for the deployment of network slices and instantiation of network services. A detailed description of the implementation status and updates for the individual integration endpoints is provided in section 2.2.1. Furthermore, the updates of the Pledger integrated demonstrator cover the final integration of the Pledger Core system with the Pledger use cases which are summarized in Table 1 and described in detail in the deliverable D5.4 – Pilots Operation and Monitoring II [2], as part of T5.3.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II			<b>Page:</b>	10 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	<b>Version:</b>	1.8
				<b>Status:</b>	Final

Table 1: Pledger integration plan

Iteration	Integration activities	Components	Partners	Date
<b>Init</b>	Installation of CI/CD Platform		INTRA	M14
<b>1<sup>st</sup> Pledger Release</b>	Main development phase of the software components of the Pledger Core System with their unit tests and their bilateral integration tests prepared for the 1 <sup>st</sup> release		All technical partners involved in WP3, WP4 and WP5	M6-M24
	Installation of StreamHandler Platform	StreamHandler	INTRA	M15
	Integration of CI/CD with the K8s Cluster of the Pledger Core System		INTRA, ENG	M16
	Integration of StreamHandler with K8s Cluster of Pledger Core System	DSS, StreamHandler	INTRA, ENG	M16
	First version of the development of the consumers/producers and REST endpoints needed for integrating the Pledger Core System	All software components of PLEDGER Core System	ATOS, ENG, ICCS, INNOV, i2CAT, INTRA	M16-M24
<b>Pledger Integrated Demonstrator I</b> <i>First Version</i>	First Release of the Pledger Core System Demonstrator partially integrated with the UC1, UC2 and UC3. Integration with UCs is still in progress and will be released on M26. The majority of the Pledger Core System components is fully integrated.			M24
<i>1<sup>st</sup> Integration with UC1</i>	First integrated release of the Pledger Core System with UC1	<b>PLEDGER Core System components</b> i. SaaS/IaaS Monitoring Engine ii. DSS (one-way, metrics consumption without additional action) iii. StreamHandler iv. Orchestrator (partially: orchestrator is integrated with local hypervisor of UC1 for the moment) <b>UC1 components</b>	ATOS, HOLO, ENG, INTRA	M16-M26

Iteration	Integration activities	Components	Partners	Date
		<ul style="list-style-type: none"> <li>i. VM Configuration</li> <li>ii. Client Unity App</li> <li>iii. Server Unity App</li> <li>iv. GPU</li> </ul>		
<i>1<sup>st</sup> Integration with UC2</i>	First integrated release of the Pledger Core System with UC2	<b>PLEDGER Core System components</b> <ul style="list-style-type: none"> <li>i. SaaS/IaaS Monitoring Engine</li> <li>ii. DSS</li> <li>iii. Orchestrator (integration without SOE)</li> </ul> <b>UC2 components</b> <ul style="list-style-type: none"> <li>i. Barcelona Infrastructure Compute</li> </ul>	ATOS, ENG, i2CAT	M16-M26
<i>1<sup>st</sup> Integration with UC3</i>	First integrated release of the Pledger Core System with UC3	<b>PLEDGER Core System components</b> <ul style="list-style-type: none"> <li>i. SaaS/IaaS Monitoring Engine</li> <li>ii. Orchestrator</li> <li>iii. DSS</li> <li>iv. StreamHandler</li> </ul> <b>UC3 components</b> <ul style="list-style-type: none"> <li>i. Basic Analytics</li> <li>ii. Message Broker</li> <li>iii. Edge Server</li> </ul>	ATOS, FILL, INTRA	M16-M26
<b>2<sup>nd</sup> Pledger Release</b>	Second implementation phase of the Pledger Core System, covering the full integration of its software components, and enabling the full integration of Pledger Core System with UC1, UC2 and UC3 (T5.3)		All technical partners involved in WP3, WP4 and WP5	M24-M33
<i>2<sup>nd</sup> integration with UC1</i>	Final integrated release of the Pledger Core System with UC1	<b>PLEDGER Core System components</b> <ul style="list-style-type: none"> <li>i. SaaS/IaaS Monitoring Engine</li> <li>ii. DSS StreamHandler</li> <li>iii. Orchestrator</li> <li>iv. DLT</li> </ul> <b>UC1 components</b> <ul style="list-style-type: none"> <li>i. VM Configuration</li> <li>ii. Client Unity App</li> <li>iii. Server Unity App</li> <li>iv. GPU</li> </ul>	All technical partners involved in WP3, WP4 and WP5	M33

Iteration	Integration activities	Components	Partners	Date
<i>2<sup>nd</sup> integration with UC2</i>	Final integrated release of the Pledger Core System with UC2	<b>PLEDGER Core System components</b> <ol style="list-style-type: none"> <li>i. SaaS/IaaS Monitoring Engine</li> <li>ii. DSS</li> <li>iii. Orchestrator</li> <li>iv. ConfService</li> <li>v. SOE and RAN Controller</li> </ol> <b>UC2 components</b> Barcelona Infrastructure Compute	ATOS, ENG, i2CAT	M33
<i>2<sup>nd</sup> integration with UC3</i>	Final integrated release of the Pledger Core System with UC3	<b>PLEDGER Core System components</b> <ol style="list-style-type: none"> <li>i. SaaS/IaaS Monitoring Engine</li> <li>ii. Orchestrator</li> <li>iii. DSS</li> <li>iv. StreamHandler</li> <li>v. DLT</li> <li>vi. Benchmarking</li> <li>vii. App Profiler</li> </ol> <b>UC3 components</b> <ol style="list-style-type: none"> <li>i. Basic Analytics</li> <li>ii. Advanced Analytics</li> <li>iii. Message Broker</li> <li>iv. Edge Server</li> </ol>	All technical partners involved in WP3, WP4 and WP5	M33
<b>Pledger Integrated Demonstrator II</b> <i>Final Version</i>	The outcome of the second implementation phase of the Pledger Core System (2 <sup>nd</sup> release) including the final integration of its components and showcasing the availability of key platform features which are exploited by the project's use cases			M33

### 2.1.1 Pledger integrated demonstrator II

The second release of the Pledger integrated demonstrator runs from M24 to M33 and comprises the following points:

- ▶ Finalize the implementation and testing of the integration endpoints which were partially developed and/or tested in the first release of the Pledger Core system, that is hosted in the production K8s cluster.
- ▶ Update specification, implementation and testing of integration endpoints that were defined in the first release of Pledger Core system (section 2.2.1).
- ▶ Second integration of the Pledger Core System with the extended UC1, UC2 and UC3, as described in detail in D5.4 – Pilots operations and monitoring II [2].

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	13 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

## 2.2 Integration matrix updates

---

In Table 2, the updated integration matrix among the bilateral endpoints is shown. D5.2 [1] describes in detail the integration methodology and the presentation logic of the integration matrix as an upper triangular square matrix. The only update compared to D5.2 [1] is that the SLA Negotiator component is removed, according to the updated Pledger architecture description with respect to the SLAs subsystem described in the deliverable D2.3 – PLEDGER overall architecture [3] (section 4.3.4).

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II				<b>Page:</b>	14 of 37	
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	<b>Version:</b>	1.8	<b>Status:</b>	Final

Table 2 Pledger core system integration matrix

Integration points	1. Orchestrator	2. DSS	3. SaaS & IaaS Monitoring Engine	4. Configuration DB	5. App Profiler	6. UI Configuration Dashboard	7. Benchmarking Scheduler	8. Metrics DB (Historical)	9. Benchmarking Creator	10. SLA Manager	11. SLA Notifier	12. SLA Monitoring	13. SLASC Bridge	14. IDPS	15. Anomalies Detection Reasoner	16. Rules/Cases/Schemas	17. T&R Engine	18. Slicing & Orchestrator Engine	19. RAN Controller
1. Orchestrator		1.2	1.3	1.4														1.18	
2. DSS			2.3	2.4				2.8			2.11								
3. SaaS & IaaS Monitoring Engine												3.12							
4. Configuration DB					4.5	4.6	4.7			4.10									
5. App Profiler									5.9										
6. UI Configuration Dashboard																			
7. Benchmarking Scheduler																			
8. Metrics DB (Historical)																			
9. Benchmarking Creator																			
10. SLA Manager																			
11. SLA Notifier													11.13				11.17		
12. SLA Monitoring																			
13. SLASC Bridge																			
14. IDPS																			
15. Anomalies Detection Reasoner																			
16. Rules Cases/Schemas																			
17. T&R Engine																			
18. Slicing & Orchestrator Engine																			18.19
19. RAN Controller																			

## 2.2.1 Integration endpoints updates

Table 3 highlights the updates on the status of the integration endpoints (2<sup>nd</sup> iteration) compared to the status of the 1<sup>st</sup> iteration of the Pledger core system.

Table 3: Pledger core system integration endpoints

Integration Endpoint	Components	Responsible	Data Type & Protocol	Publisher – Subscriber (if any)	Status 1 <sup>st</sup> iteration (M24)	Status 2 <sup>nd</sup> iteration (M33)
1.2	Orchestrator - DSS	ATOS, ENG	<b>Data Type:</b> JSON <b>Protocol:</b> Kafka via StreamHandler	<b>Topic(s):</b> - deployment - deployment-feedback	Testing	<b>Done</b> A feedback message has been added to send back to the DSS the result of the operation with a severity and a text message for the possible error.
1.3	Orchestrator – SaaS/IaaS Monitoring Engine	ATOS	<b>Data Type:</b> Text/plain <b>Protocol:</b> HTTPS	N/A	In Progress	<b>Done</b>
1.4	Orchestrator – Configuration DB	ATOS, ENG	<b>Data Type:</b> JSON <b>Protocol:</b> Kafka via StreamHandler	<b>Topic(s):</b> - configuration	Done	<b>Done</b>
1.18	Orchestrator – PLEDGER SOE Framework	ATOS, i2CAT	<b>Data Type:</b> JSON <b>Protocol:</b> REST API HTTPs	N/A	In progress	Pending a minor modification and testing at the time of writing. No risks foreseen.
2.3	DSS – SaaS/IaaS Monitoring Engine	ATOS, ENG	<b>Data Type:</b> JSON <b>Protocol:</b> HTTPs (either REST calls to Prometheus or Kafka via StreamHandler)	<b>Topic(s):</b> - monitoring	In progress	<b>Done</b>

Integration Endpoint	Components	Responsible	Data Type & Protocol	Publisher – Subscriber (if any)	Status 1 <sup>st</sup> iteration (M24)	Status 2 <sup>nd</sup> iteration (M33)
2.4	DSS - Configuration DB	ENG	<b>Data Type:</b> N/A <b>Protocol:</b> JDBC (SQL)	N/A	In progress	<b>Done</b>
2.8	DSS – Metrics DB	ENG	<b>Data Type:</b> JSON <b>Protocol:</b> Kafka via StreamHandler	<b>Topic(s):</b> - benchmarking	In progress	<b>Done</b>
2.11	DSS – SLA notifier	ATOS, ENG	<b>Data Type:</b> JSON <b>Protocol:</b> Kafka via StreamHandler	<b>Topic(s):</b> - sla_violation	Done	<b>Done</b>
3.12	SaaS/IaaS Monitoring Engine – SLA Monitoring	ATOS	<b>Data Type:</b> Text/plain <b>Protocol:</b> HTTPS	N/A	Done	<b>Done</b>
4.5	Configuration DB – App Profiler	ENG, ICCS	<b>Data Type:</b> JSON <b>Protocol:</b> - HTTPs (Kafka via StreamHandler) - REST API	<b>Topic(s):</b> - app_profile	In progress	<b>Done</b>
4.6	Configuration DB – UI Configuration Dashboard	ENG	<b>Data Type:</b> N/A <b>Protocol:</b> JDBC (SQL)	N/A	Done	<b>Done</b>
4.7	Configuration DB - Benchmarking Scheduler	ENG	<b>Data Type:</b> JSON <b>Protocol:</b> Kafka via StreamHandler	<b>Topic(s):</b> - configuration	Done	<b>Done</b>
4.10	Configuration DB - SLA Manager	ATOS, ENG	<b>Data Type:</b> JSON <b>Protocol:</b> Kafka via StreamHandler	<b>Topic(s):</b> - configuration	Done	<b>Done</b>

Integration Endpoint	Components	Responsible	Data Type & Protocol	Publisher – Subscriber (if any)	Status 1 <sup>st</sup> iteration (M24)	Status 2 <sup>nd</sup> iteration (M33)
5.9	App Profiler - Benchmarking Creator	ICCS, ENG	<b>Data Type:</b> JSON <b>Protocol:</b> REST API HTTPs	N/A	In progress	<b>Done</b>
11.13	SLA Notifier – SLA-SC Bridge	ATOS, INNOV	<b>Data Type:</b> JSON <b>Protocol:</b> Kafka via StreamHandler	<b>Topic(s):</b> - sla_contract - sla_violation	In progress	<b>Done</b>
11.17	SLA Notifier - T&R Engine	ATOS, ICCS	<b>Data Type:</b> JSON <b>Protocol:</b> Kafka via StreamHandler	<b>Topic(s):</b> - sla_violation	Not started	<b>Done</b> SLA Notifier reused from other integrations (pub/sub pattern)
18.19	PLEDGER SOE Framework - PLEDGER RAN Controller	i2CAT	<b>Data Type:</b> JSON <b>Protocol:</b> HTTPs (REST API)	N/A	In progress	<b>In progress</b> Pending last integrations for IEEE 802.11p slicing (expected end of M34)

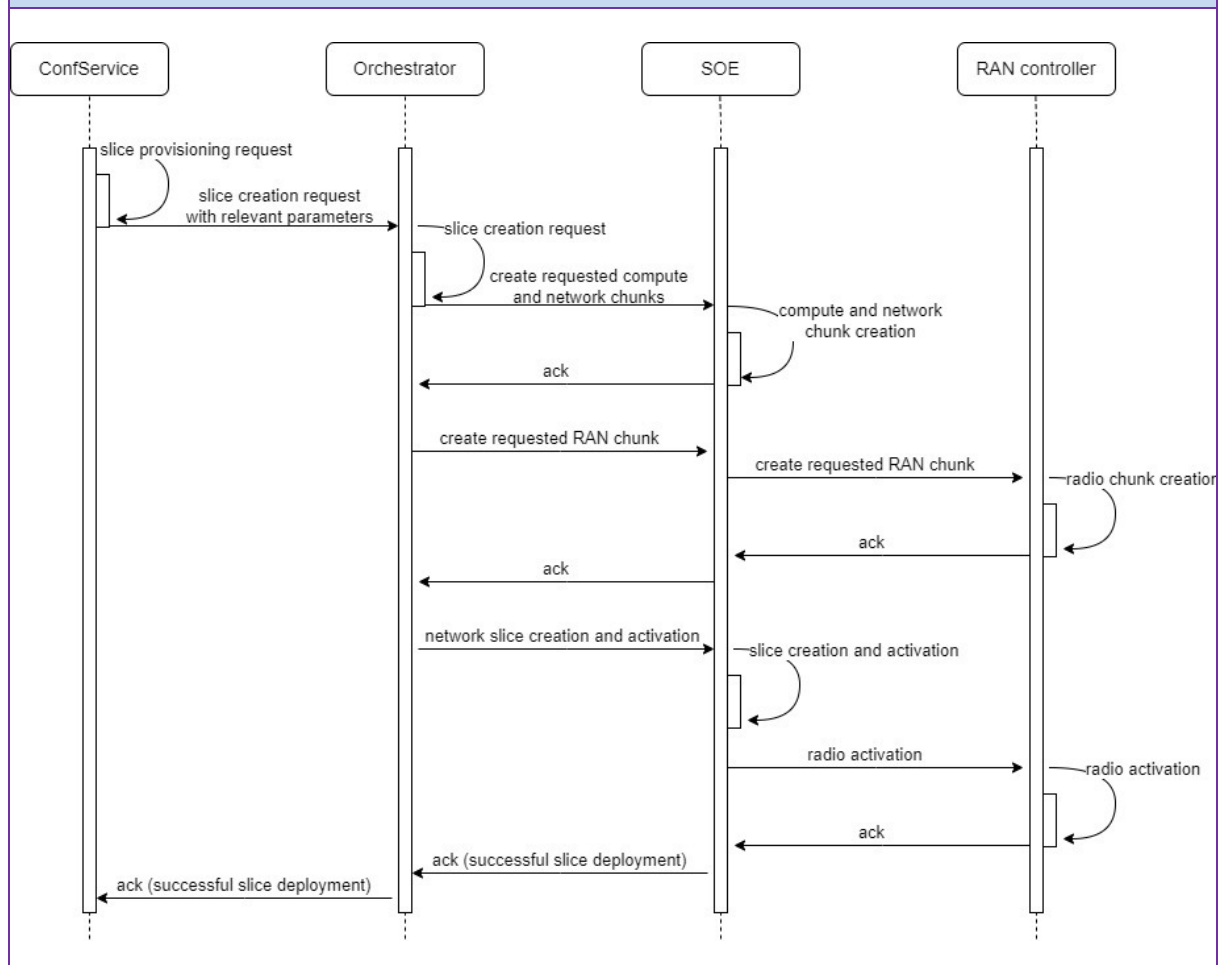
In the following tables, a detailed description is provided for the integration endpoints whose specification and implementation has been updated in the second release of the Pledger Core system compared to the first release.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	18 of 37
<b>Reference:</b>	D5.6 <b>Dissemination:</b> PU	<b>Version:</b>	1.8 <b>Status:</b> Final

Table 4: Orchestrator – PLEDGER SOE Framework integration endpoint description

Integration Point Description	
<b>Identifier</b>	1.18 Orchestrator – PLEDGER SOE Framework
<b>Responsible</b>	ATOS, i2CAT
<b>Integration Point Purpose</b>	Request the instantiation of an end-to-end network slice with either 5G or IEEE 802.11p radio nodes

**Sequence diagram**



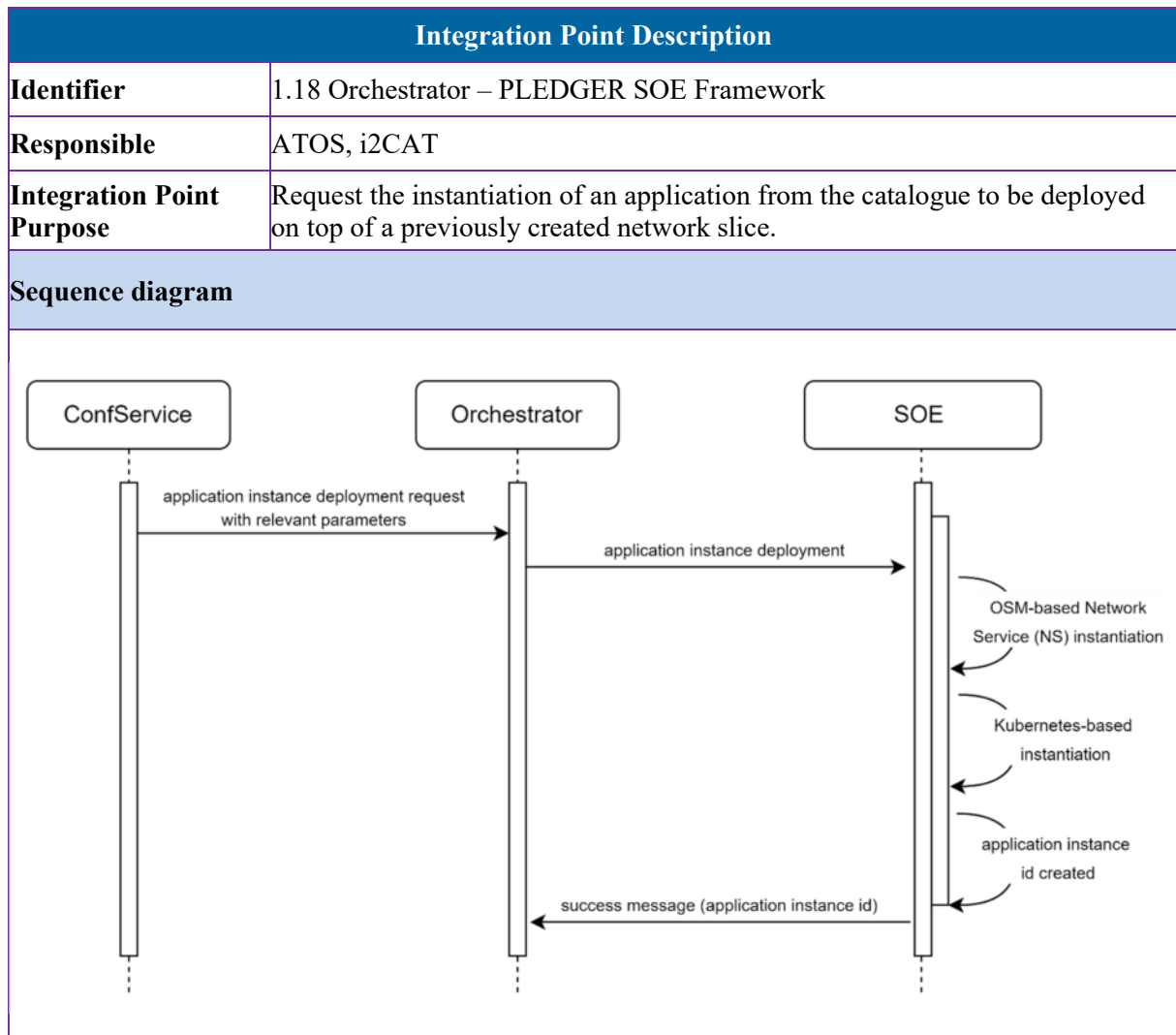
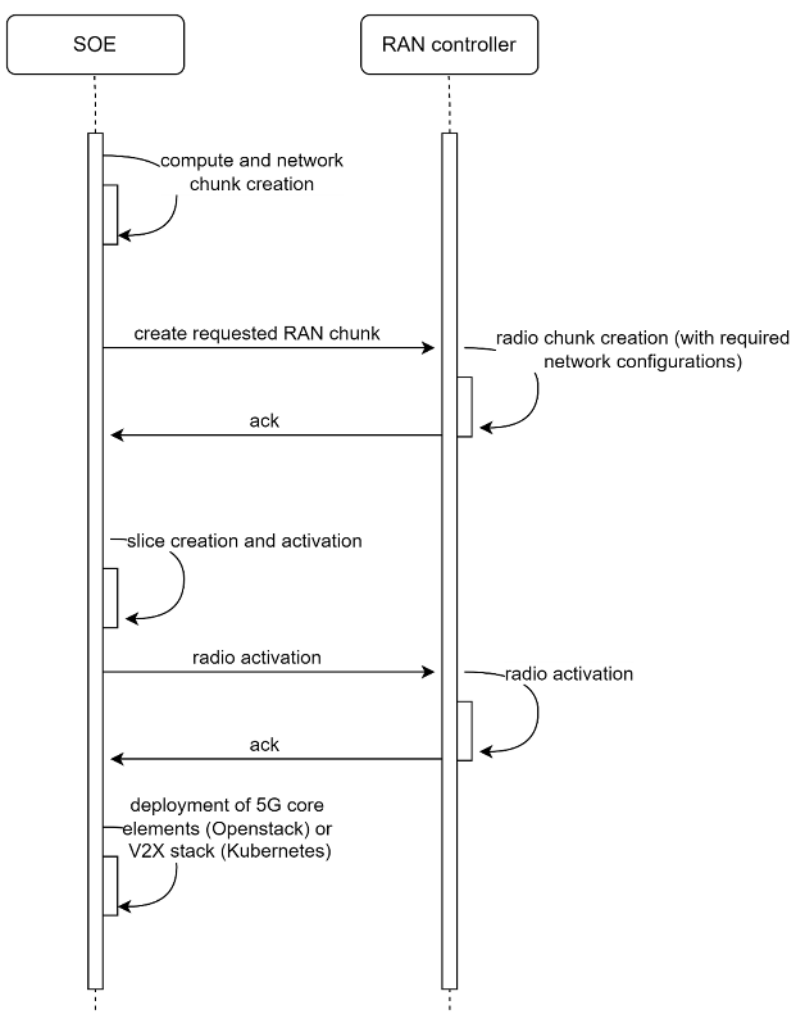


Table 5: PLEDGER SOE Framework - PLEDGER RAN Controller integration endpoint description

Integration Point Description	
<b>Identifier</b>	18.19 PLEDGER SOE Framework - PLEDGER RAN Controller
<b>Responsible</b>	i2CAT
<b>Integration Point Purpose</b>	Necessary interactions for the configuration of radio elements and their inclusion into a network slice
<b>Sequence diagram</b>	
 <pre> sequenceDiagram     participant SOE     participant RAN_controller as RAN controller      Note over SOE: compute and network chunk creation     SOE-&gt;&gt;RAN_controller: create requested RAN chunk     Note over RAN_controller: radio chunk creation (with required network configurations)     RAN_controller--&gt;&gt;SOE: ack     Note over SOE: slice creation and activation     SOE-&gt;&gt;RAN_controller: radio activation     Note over RAN_controller: radio activation     RAN_controller--&gt;&gt;SOE: ack     Note over SOE: deployment of 5G core elements (Openstack) or V2X stack (Kubernetes)     </pre>	

## 3 Pledger core system demonstrators

---

In this section, the four selected Pledger Core System demonstrators are described in detail. The aim is to showcase the availability of some key features of the platform and the interoperability among the corresponding software components/services. Those features are also important for the full integration of the Pledger Core system with one or more of the project use cases.

### 3.1 Offloading in the edge-cloud continuum using SLAs

---

This integration demo has been presented during the second technical review in March 2022 and described in D3.5 (section 5.1) [5] released in M30.

In this section we focus more on the core components interactions, the data exchanged and the flow with the main steps involved.

#### 3.1.1 General description

The general goal of this demo is to show how service providers make an App to run and fulfil some QoS (registered in an SLA object) while keeping the Apps on the edge as much as possible, saturating the edge computation capacity while guarantying the promised SLAs and applying DSS optimizations, such as resource usage and latency.

In this demo, the integration and interoperability among different Pledger components is demonstrated, in particular:

- ▶ ConfService,
- ▶ DSS,
- ▶ SLA Manager,
- ▶ Orchestrator,
- ▶ Monitoring Engine,
- ▶ StreamHandler.

We also mention the integration among DSS, AppProfiler and Benchmarking, used by the DSS during the offloading decisions, further explained in a separate integration demo in section 3.2 to simplify the presentation.

As a result, the interoperability among the above-mentioned components gives Pledger the ability to orchestrate Apps on virtually any kind of infrastructure, scaling and offloading Apps to achieve agreed SLAs and based on the service providers' preferences to guide the DSS decisions so that some nodes are prioritized over others (e.g., to keep computation on the edge as much as possible) and benefit from the several optimization algorithms the DSS provides, described in D4.6.

To complement the interactions, AppProfiler and Benchmarking integration allows the DSS to choose, among deployment options with the same priority, the node that provides the best performance possible for a specific App.

#### 3.1.2 Architecture

Figure 1 shows the components involved in this demo, with connections having different colors to highlight the communication protocol used:

- ▶ JDBC (red arrows)
- ▶ REST (blue arrows)
- ▶ Async messaging using the StreamHandler (green arrows)

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	22 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

It also shows the main data exchanged, with the ConfService coordinating the configuration notification to the interested components, and the DSS coordinating the deployment activities triggered by the monitoring data and the SLA violations received.

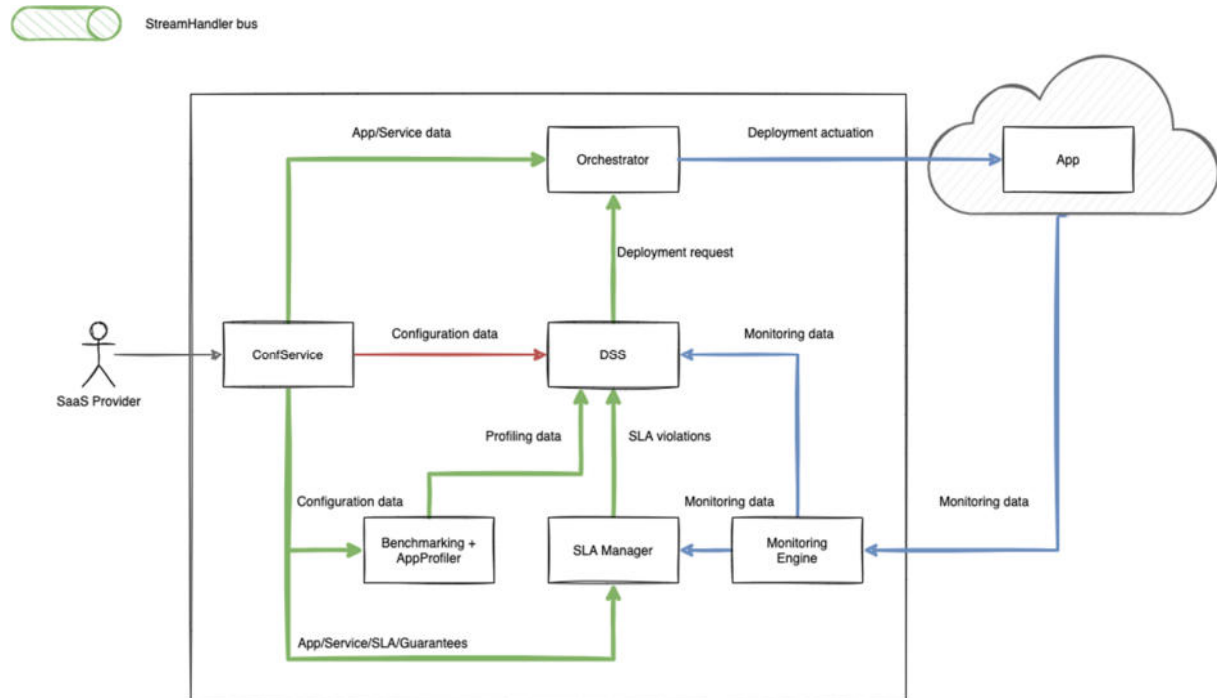


Figure 1: DSS interactions with Pledger core components

### 3.1.3 Scenario

The demo shows a test app configured, started, monitored, offloaded from edge to cloud, and back to the edge when the DSS reacts to SLA warnings and violations with some scaling/offloading actions and runtime adaptations reducing the SLA violations. The demo scenario is composed of separate operations:

**Operation 1:** the service provider checks the Infrastructure and App configuration and sets the deployment options priorities to privilege edge over cloud or one infrastructure (or node) to another. Here the components involved are ConfService, SLA Manager and Orchestrator.

**Operation 2:** the service provider starts the App. The components involved are ConfService and Orchestrator.

**Operation 3:** the DSS gets the metrics from the Monitoring Engine and the SLA violations from SLA Manager, gets the configuration, the deployment options and the optimization algorithm selected from the ConfService. Next, according to these, the DSS gets the best node to run Apps from the Benchmarking and AppProfiler, then computes the next deployment plan.

**Operation 4:** the DSS commands the Orchestrator to apply the new deployment plan.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	23 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

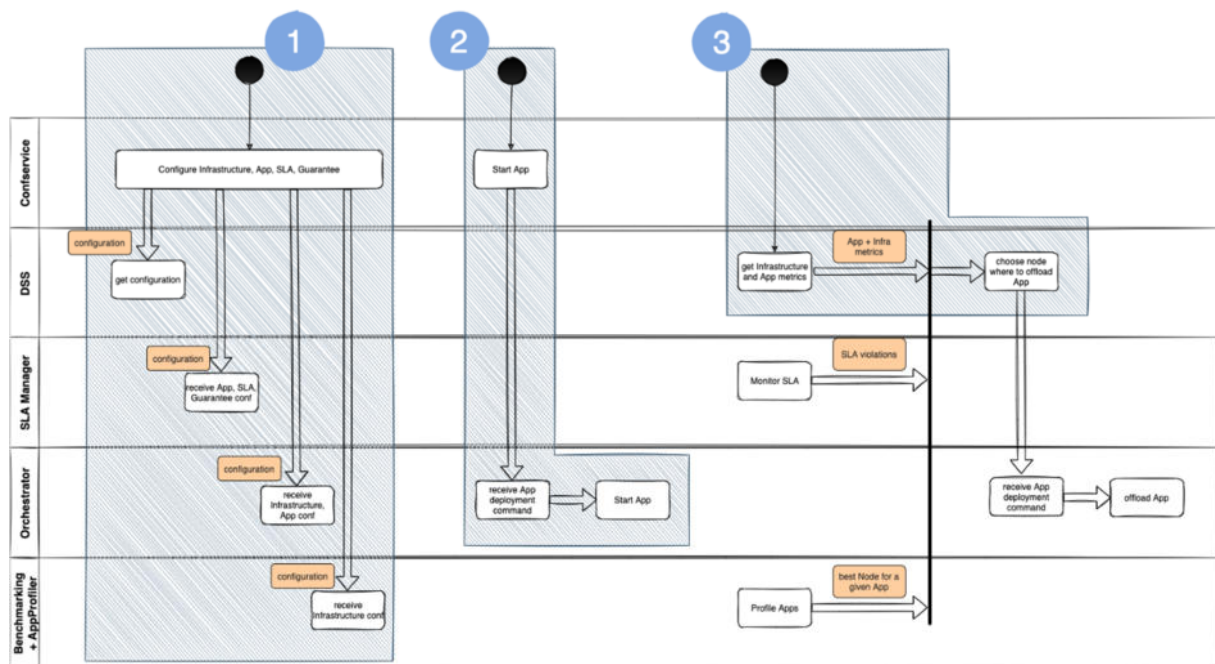


Figure 2: Demo steps and components involved

As depicted in Figure 2, there are multiple steps and components involved for every operation. As far as **Operation 1** is concerned, the steps are the following:

- ▶ **Step1.1**, configure the infrastructures needed for the edge (in the demo, a VM with docker engine) and the cloud (in the demo, ENG K8S cluster) and the service provider preferences to keep an app on the edge as primary option as much as possible.

The main data shown in this step is about infrastructures configuration and their nodes with different properties (HW capabilities, location, etc.) to allow the service provider to express her/his preferences.

- ▶ **Step1.2**, configure an App with one service, SLA agreement with guarantees (expected QoS) and deployment options.

The main data shown in this step is about the App and the Services it is made of, the SLAs associated to the App and the different Guarantees, meaning the metrics name, threshold and associated severity.

- ▶ **Step1.3**, Orchestrator, SLA Manager, Benchmarking and AppProfiler receives configuration updates. In the demo, Orchestrator and SLA manager updates (infrastructures, applications and SLA agreements created on the fly) are shown via swagger API web UIs (Orchestrator, SLA Manager components).

The main data shown in this step is about the internal data model representation of SLA Manager and Orchestrator components, shared by the ConfService through the StreamHandler.

For **Operation 2**, the steps are the following:

- ▶ **Step2.1**, an app is started on ConfService. The main data shown in this step is about the ConfService sending a message via StreamHandler to the Orchestrator to start the App.
- ▶ **Step2.2**, the Orchestrator deploys the App on the edge (as a docker container) and starts it.

The main data shown in this step is about the Orchestrator receiving the message from the StreamHandler to start the App.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	24 of 37
<b>Reference:</b>	D5.6 <b>Dissemination:</b> PU	<b>Version:</b>	1.8
		<b>Status:</b>	Final

For **Operation 3**, the steps are the following:

- ▶ **Step3.1**, the app is monitored, and an SLA violation is received when some load is artificially generated to trespass the threshold defined by the SLA agreement (Monitoring Engine, SLA manager, DSS components).

The main data shown in this step is about the SLA violation received by the DSS about the App metrics and violated thresholds.

- ▶ **Step3.2**, the DSS chooses the best benchmark tool for a service using label matching set manually or by the AppProfiler (Benchmarking, AppProfiler, DSS components).

The main data shown in this step is about the matching between a Service and a Benchmark. This is done through labels in the ConfService that can be manually set or populated by the AppProfiler through messages using the StreamHandler

- ▶ **Step3.3**, the DSS chooses the node (from a K8S cluster in the cloud) with the highest score for the selected benchmark tool for offloading the app from the edge to the cloud (DSS).

The main data shown here is about the DSS UI where messages are produced to announce the DSS offloading decision and its actuation.

- ▶ **Step3.4**, the DSS commands the Orchestrator to apply the deployment plan.

The main data shown here is about the Apps offloaded from edge to cloud based on the message sent by the DSS to the Orchestrator.

To conclude, as **Operation 3** is executed at regular intervals, in the demo a final step is shown with the DSS not receiving SLA violations for some time, edge nodes notifying there is space to host Apps and the DSS triggering the offload back to the edge based on the preferences configured by the service provider. So, the flow is basically repeated, but with a different action taken by the DSS and executed by the Orchestrator.

The complete video of the integrated demo can be accessed in the official project YouTube channel: <https://www.youtube.com/watch?v=6aRlhWTi2k8>

## 3.2 Infrastructure benchmarking and application profiling

### 3.2.1 General description

In this demo, the integration and interoperability among various components of Pledger is demonstrated. More specifically, the Components involved in this demo are:

- ▶ Confservice
- ▶ Benchsuite
- ▶ DSS
- ▶ App Profiler

All these components work together in an effort to maximize an application's performance by deploying it in the best infrastructure node. As far as the best node is concerned, there are two major identifiers, the first one is the resource capabilities based on the performance metrics derived from the benchmark tests performed on it. The second major identifier are the application needs, based on the resource usage profile created by the App Profiler. As for the Confservice and DSS, the first component is responsible for the configuration and the access to the infrastructure, and the later for the decision on the deployment based on the identifiers produced by the other two components.

Interoperability among all of these components can enable an infrastructure environment to structure and automate the evaluation of the resources and applications, creating in the process knowledge and decision mechanisms for the deployment and governance of the different applications.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	25 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

### 3.2.2 Architecture

The objective of this integration architecture is to demonstrate the interoperability and functionalities provided by the AppProfiler and Benchmarking Suite components. These two components are highly integrated with other components in the Pledger system in order to provide a seamless data exchange within the system and offer a better user experience.

In the first phase of the integrated demo architecture (Figure 3), the performance data is produced and the AppProfiler model is trained. In this phase, the Benchmarking Suite interacts with the ConfService (to get registered infrastructure access data), the target infrastructure (to start the execution of the tests), the AppProfiler (to notify that a test is being executed) and the DSS (to upload the performance data computed from the test execution). The AppProfiler interacts with the target infrastructure in order to monitor the execution of the test and collect data needed to train its model.

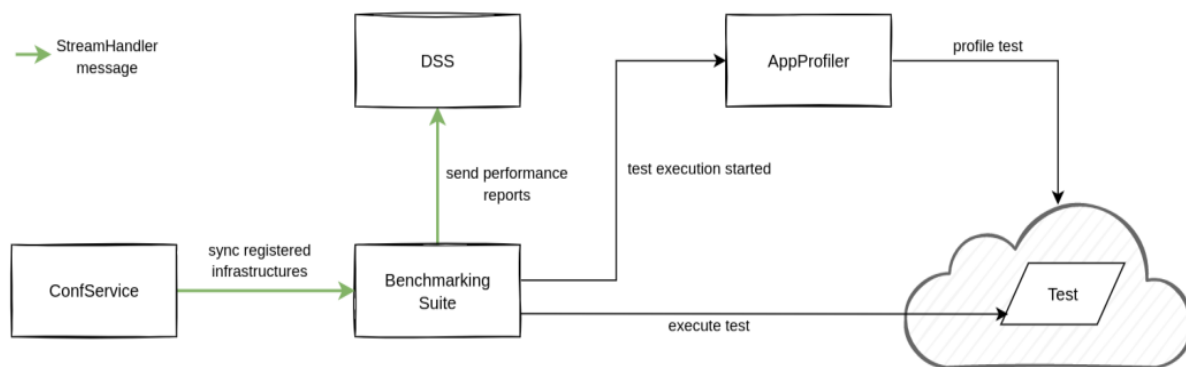


Figure 3: Benchmarking suite and app profiler integration architecture

In the second phase (Figure 4), a new app is deployed for the first time and deployment decisions are taken. In this phase, the Benchmarking Suite has not an active role, since the DSS already has all the performance data needed. The AppProfiler interacts with the ConfService (to receive the first deployment location of a new app), the target infrastructure (to monitor the execution of the app and compute its profile) and the DSS (to send the calculated profile for the application). Finally, the DSS interacts with the Orchestrator to adjust/move the deployment of the app.

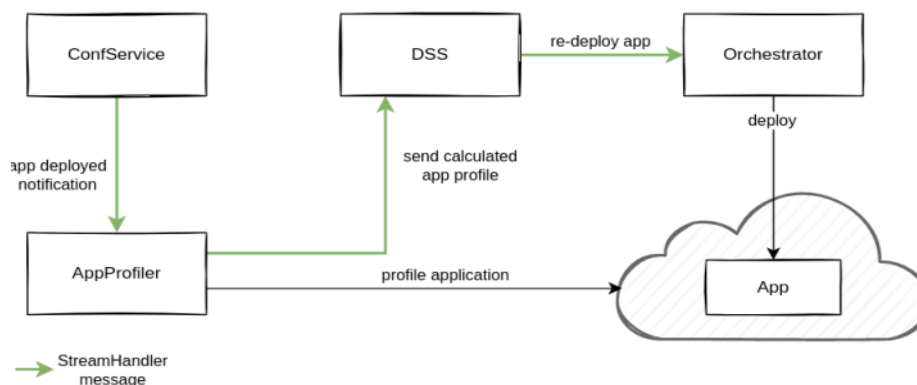


Figure 4: Application performance assessment operation architecture

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	26 of 37	
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.8	<b>Status:</b>	Final

In both Figure 3 and Figure 4, the interactions realized using the StreamHandler messaging middleware component are highlighted in green. The other interactions are implemented using HTTP REST API calls.

### 3.2.3 Scenario

As described in the introductory section of this chapter, the objective of this integrated demo is to demonstrate how the App Profiler and the Benchmarking Suite are able to assess the infrastructure and applications in order to select the best node for deployment. In order to achieve that, there were three distinct operations that needed to be performed:

**Operation 1:** Run a set of application-level benchmarks that simulate different types of applications (web servers, databases, streaming servers) and measure their performance on different nodes.

**Operation 2:** Profile the application to deploy and match it with one of the executed benchmarks.

**Operation 3:** Suggest the deployment of the application in the node where the matched benchmark had the best performance.

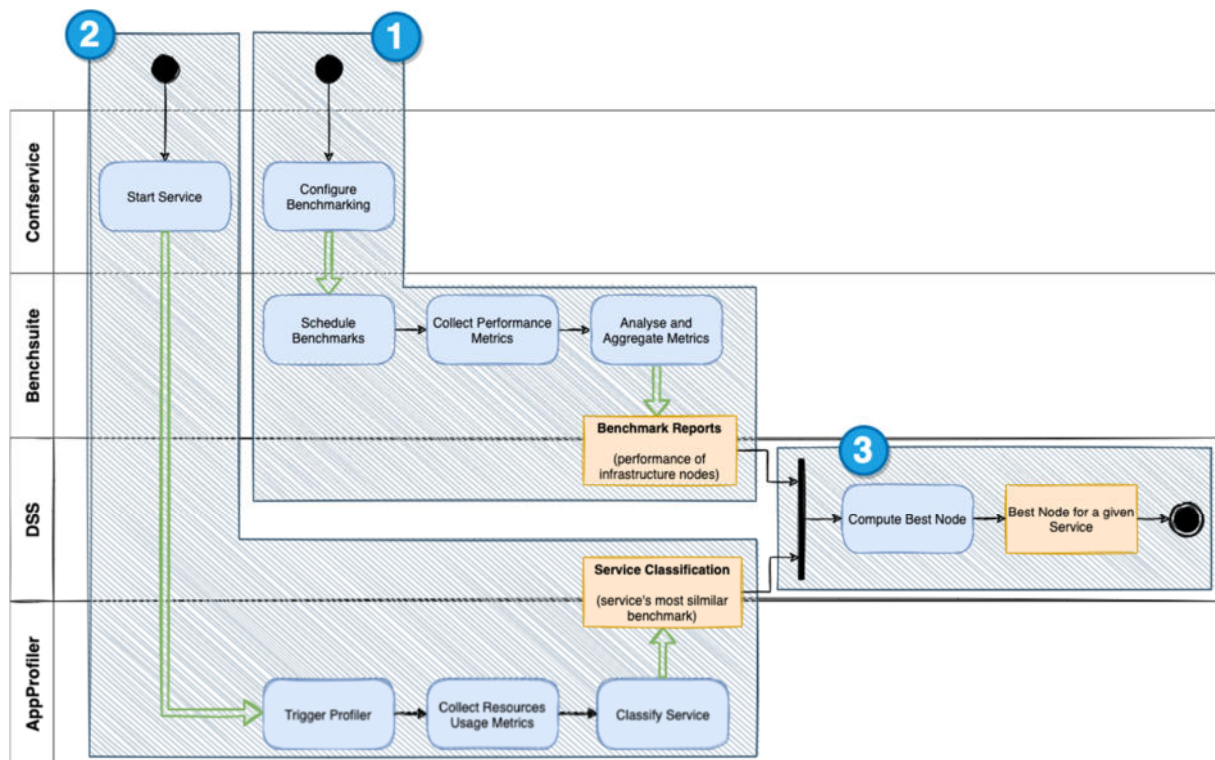


Figure 5: Infrastructure benchmarking and application profiling integration demo flow

As depicted in Figure 5, there are multiple steps and components involved for every operation. As far as **Operation 1** is concerned, the steps are the following:

- ▶ **Step1:** benchmarking of an infrastructure is enabled by a Service or Infrastructure Provider in the Confservice. A notification is sent to the Benchmarking Suite through the StreamHandler;
- ▶ **Step 2:** the Benchmarking Suite receives the notification and starts the execution of a set of default (but customizable) benchmark tests on the target infrastructure in a completely automatic way;
- ▶ **Step 3:** the Benchmarking Suite collects performance data during the execution of tests and parses the output of the tests execution to extract more performance metrics;

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	27 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

- ▶ **Step 4:** Metrics collected are stored in the Benchmarking Suite and they are analyzed by the analytics module, creating higher level and aggregated metrics;
- ▶ **Step 5:** the Benchmarking Suite sends performance reports for the target infrastructure through the StreamHandler. The reports are received and stored by the DSS.

The steps for Operation 2 that regards App Profiler take place as follows (as mentioned also in D3.5 [5]):

- ▶ **Step 1** of the process to automatically profile the application is the actual deployment of the application or service in the infrastructure. As depicted in the image below, the App Profiler is informed about the deployment of the application from the deployment engine and starts the process of profiling.
- ▶ **Step 2**, after obtaining the deployment information about the application, the App Profiler can trigger the operation of collection of resource usage data.
- ▶ **Step 3**, App Profiler collects the resource usage data and composes the profile vector that will be later used for the classification process.

In the final **step 4** the App Profiler takes the profile vector created in the previous process and classifies the application using the ML Models created in the training phase. Classification results are then sent to the DSS via the Stream Handler Component.

After the success of the two previous the last **Operation 3** takes place. DSS takes the accumulated data from the Benchmarking Suite and the App Profiler and makes the deployment decision. It is important to mention that the gathering of benchmark profiles and the training of the algorithms for the App Profiler are not shown on this flow because they are background operations that are performed outside the scope of this demo. The complete video of the integrated demo can be accessed in the official project YouTube channel: <https://www.youtube.com/watch?v=oJGZCa8SUUI>

### 3.3 Blockchain SLA management

---

#### 3.3.1 General description

In this demo, the Blockchain SLA Management with regards to the SLA configurations management on the ledger is delivered. The following core components are involved:

- ▶ ConfService
- ▶ SLA Lite
- ▶ SLASC Bridge
- ▶ DLT

This demonstration represents the corresponding SLA configuration workflow that takes place inside Pledger in relation to the creation and deployment of the relevant smart contract structures. The relevant interoperability between the candidate structural entities is necessary to provide the business intelligence of the SLA contractual terms into the corresponding blockchain equivalents. Particularly, the SLASC Bridge capabilities and formulation strive to match its reliant interactions with the established components outside of the blockchain. As a result, in terms of SLA settings, the deployed interoperability framework is mostly set up on the SLASC Bridge. SLASC Bridge constitutes the major entry point for the represented SLA contractual terms on the ledger while being implemented as an architectural layer. The related integrated SLA application emits the SLA configuration parameters, and consequently SLASC Bridge receives the corresponding data.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	28 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

### 3.3.2 Architecture

The main scenario begins with an SLA being configured in the ConfService, continues with the creation of the SLA by the SLA Lite application, then with the SLASC Bridge depiction of the SLA contractual terms equivalents, and concludes with the deployment of the corresponding smart contract structures on the DLT network. A dedicated notification is provided to the SLASC Bridge for each SLA creation that is completed by the SLA Lite through the StreamHandler. When starting the process of creating the related smart contract structures, SLASC Bridge consumes the configured SLA parameters that come from the ConfService and are part of the notifications transmitted between the two components.

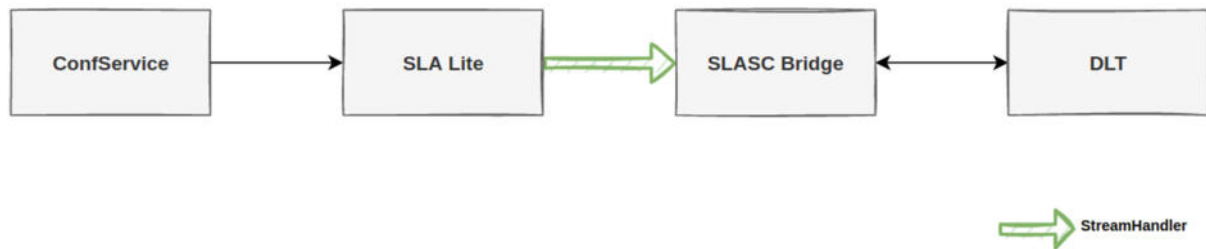


Figure 6: Blockchain SLA management setup

For each received SLA configuration, SLASC Bridge defines a new structure constituting the smart contract equivalent. The final stage involves the submission of the smart contract equivalent to the ledger. Smart contract equivalents of SLA configurations are created as part of the SLASC Bridge process and are prepared to be submitted on the ledger as a decentralized representation of SLA parametrization. Additionally, the refunding mechanism that is implemented on the DLT automatically includes the received SLA configurations. The equivalent smart contract structures are then posted to the blockchain network as the final step. By deploying their representative models on the ledger, the SLA consolidates the parameters' actuation. The entire workflow is focused on the easy integration of the economic model implemented on the DLT with the manifestation of the SLA configuration and its smart contract equivalent inside the trusted network.

### 3.3.3 Scenario

Regarding SLA configuration on the ledger and how SLA parameters and violations are managed on the blockchain, the SLASC Bridge component covers two key objectives, the "SLA to Smart Contract Process" and the "SLA Violation Handling". First of all, the "SLA to Smart Contract Process" manages the SLA contractual terms and results in the production of the corresponding blockchain equivalents. Then, SLASC Bridge materializes the SLA contractual terms on the blockchain. In particular, an SLA configuration defined in the ConfService is converted into its blockchain equivalent smart contract structure, which includes and specifies the business intelligence of the SLA (contractual terms and behaviour). Secondly, and in a sequence the "SLA Violation Handling" occurs. The latter leads to Violations Handled where SLASC Bridge handles the emitted SLA violations on the blockchain network. Specifically, every breach of the promised services activates the specific compensation system on the ledger, and the violated contractual terms of an SLA configuration are tracked on the blockchain in accordance with its relevant smart contract equivalent.

There exist multiple steps and components that are involved during each one of the two objectives of SLASC Bridge, i.e. "SLA to Smart Contract Process" and "SLA Violation Handling", as depicted in Figure 7.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	29 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

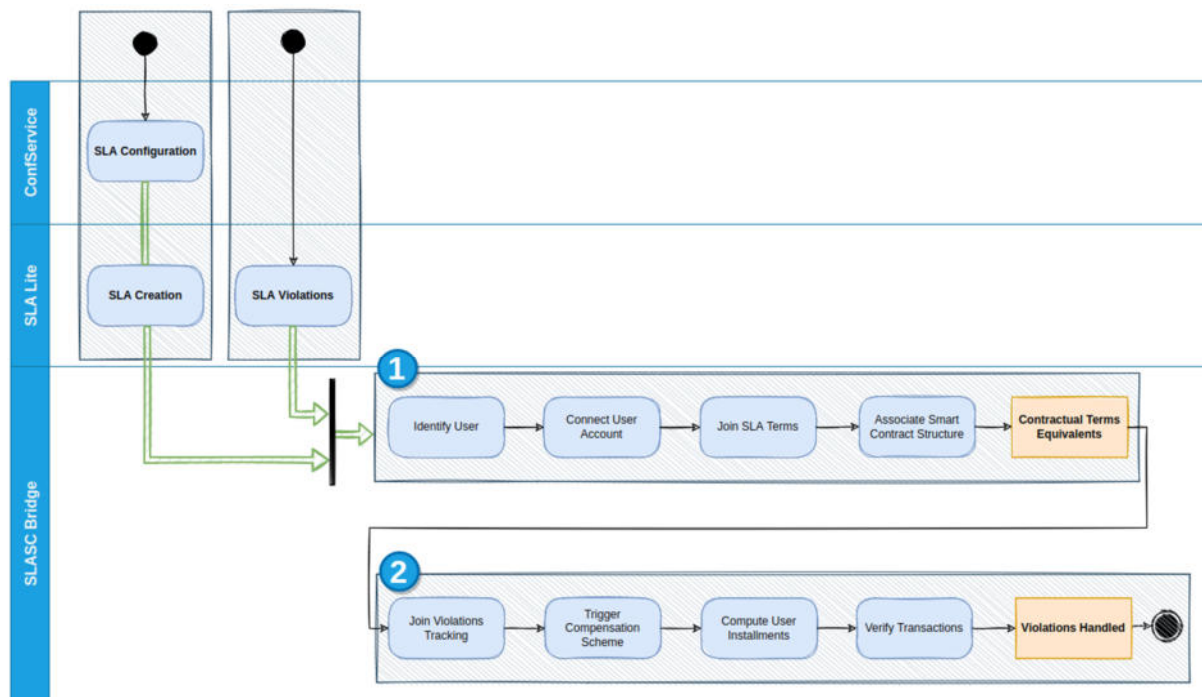


Figure 7: SLASC bridge blockchain SLA management procedure

In order to create the smart contract equivalent of an SLA configuration, the following steps occur during the "SLA to Smart Contract Process":

- ▶ **Step 1:** The SLA contractual terms (SLA configuration) are sent to the blockchain network from the ConfService through the SLA Lite.
- ▶ **Step 2:** SLASC Bridge identifies the respective information for the participating users
- ▶ **Step 3:** SLASC Bridge connects the user accounts on the blockchain. When an account does not exist yet, this step is responsible for creating a new account for the new user.
- ▶ **Step 4:** SLASC Bridge encapsulates the SLA contractual terms of the SLA configuration with the involved users, and then manages the respective user accounts on the blockchain, in order to prepare for the smart contract structure creation.
- ▶ **Step 5:** SLASC Bridge associates the related SLA configuration on a new smart contract structure that defines the dedicated contractual terms on the ledger.
- ▶ **Step 6:** In the final step of the "SLA to Smart Contract Process", the contractual terms equivalent for the SLA configuration is created (smart contract equivalent structure).

In order to handle the violations of an SLA configuration on the blockchain network, the following steps take place during the "SLA Violations Handling":

- ▶ **Step 7:** SLA violations are announced to the blockchain and the ConfService components from the SLA Lite application.
- ▶ **Step 8:** SLASC Bridge retrieves and triggers the equivalent on chain smart contract structure for the specific SLA configuration.
- ▶ **Step 9:** SLASC Bridge jointly triggers the violations tracking on the contractual terms on the ledger. In this step, SLASC Bridge enables the capability of detecting SLA violations for the corresponding smart contract structure of the initial SLA configuration.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	30 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

- ▶ **Step 10:** In this step, the refunding scheme that is deployed on the blockchain network is triggered in order to be able to provide the corresponding compensations for the SLA violation that occurred.
- ▶ **Step 11:** SLASC Bridge uses the refunding scheme that is deployed on the ledger in order to compute the user instalments to their accounts according to the incurred SLA violation.
- ▶ **Step 12:** SLASC Bridge performs verification of the corresponding transactions on the blockchain in order to finalize the violations handling.
- ▶ **Step 13:** On transaction finality, the scenario of "SLA Violations Handling" completes its life-cycle. The video of the demonstrator for SLASC Bridge is found in the official YouTube Channel of Pledger: <https://www.youtube.com/watch?v=v5eRSUmCNfk>.

## 3.4 End to end slicing

### 3.4.1 General description

Through the SOE and RAN controller components, the Pledger platform can orchestrate network slices and network connectivity over Kubernetes- and OpenStack-based infrastructures. Specifically, it can provision network slices on demand, deploy cloud-native network functions (CNFs) and virtual network functions (VNFs) and set-up radio-access connectivity. The purpose of this demo is to showcase:

- ▶ The performed integrations for the deployment of an end-to-end network slice based on the IEEE 802.11p radio protocol, and the deployment of a network service over such slice. A top-level representation of this demonstration is depicted in Figure 8, including an end-to-end network slice with virtual Kubernetes-based computational resources and two IEEE 802.11p radio nodes. A containerized network service is deployed as a Kubernetes instance, for which SOE leverages the use of OSM.
- ▶ The performed integrations for the deployment of two end-to-end network slices based on 5G new radio, replicating a scenario where two mobile network operators would deploy their own network elements over a shared infrastructure. A top-level representation of this demonstration is provided in Figure 9, where two isolated end-to-end network slices are deployed over an OpenStack-based infrastructure, with 5G radio elements. The required 5G core elements run as dedicated virtual machines on each network slice. End-to-end connectivity is demonstrated through the mobile network ID of two user equipment, each connected to a different slice (i.e., to a different operator's network).

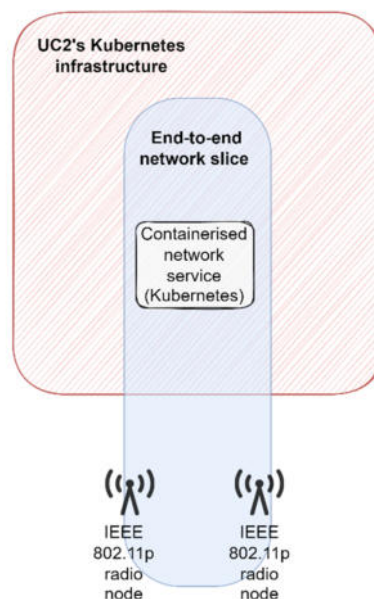


Figure 8: End-to-end network slice with IEEE 802.11p radio elements

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	31 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

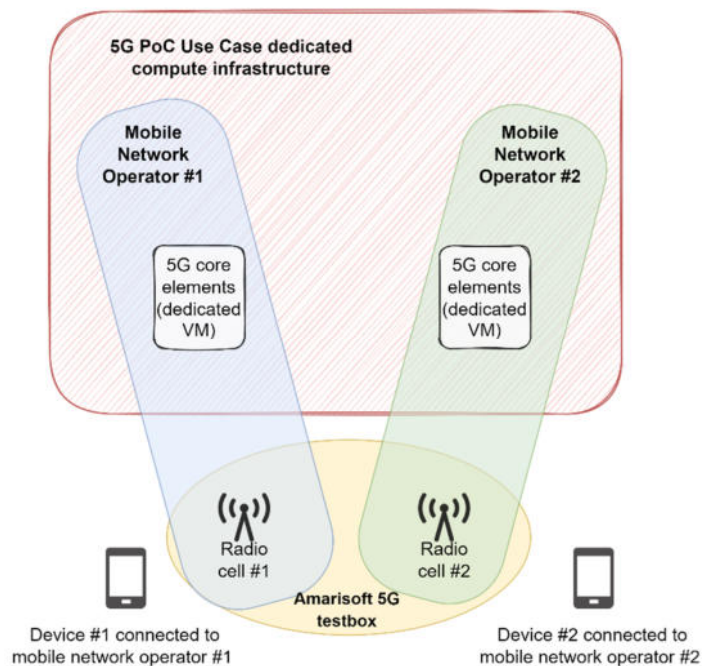


Figure 9: Deployment of two independent end-to-end network slices with 5G new radio elements

The motivation for this demo is to prove that the Pledger platform can support the deployment of end-to-end network slices based on different radio technologies, as well as different virtualization technologies over diverse underlying infrastructure types, while providing connectivity to the end users.

By using Pledger, infrastructure providers can deploy network slices from a simple-to-use user interface, in a manner that the underlying infrastructure is transparent to users. The ConfService is used by the infrastructure provider to enter parameters of interest, such as the computational capabilities of the network slice to be created, and the infrastructure over which the network slice is to be deployed. After the deployment of a network slice is accomplished, service providers can use the ConfService to deploy their desired services over that slice. The goal of this demo is to showcase the specific flows that occur among the ConfService, Orchestrator, SOE and RAN controller when a request for the creation of a network slice is issued, along with the flows that are required for the deployment of a network service on top of a network slice.

Our deployment-driven approach to network slicing [4] is also showcased in the demo. In contrast with service-driven and resource-driven approaches, our deployment-driven approach consists of a decoupling between resource sharing and runtime constructs, where service creation and instantiation can be facilitated by a network function virtualization orchestrator (OSM, in our case). The implementation of the slice resource management is performed by SOE outside of a network function virtualization Orchestrator (NFVO), for greater flexibility. In addition, services deployed on the slice are not restricted to be modelled in a specific way. In other words, SOE is not bound to a specific data model for service instantiation; therefore, services can be modelled according to relevant standards, such as 3GPP and ETSI. This deployment-driven approach allows for the reutilisation of network service and virtual network function descriptors among several slice instances, providing a lower complexity<sup>1</sup> than service-driven and resource-driven approaches, especially as more network slices are simultaneously deployed over the same infrastructure [4].

<sup>1</sup> Complexity is understood as the number of objects and relationships needed for the deployment of the network slices.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	32 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

### 3.4.2 Architecture

The deployment of end-to-end network slices in Pledger, based on either 5G new radio or IEEE 802.11p, requires the interaction among the ConfService, the Orchestrator, the SOE and the RAN controller.

The ConfService is the starting point where requests for the deployment of a network slice are placed. After a request, the ConfService passes on relevant configuration parameters to the Orchestrator, which in turn passes on those parameters to the SOE. After that, based on the received parameters, the SOE will execute some actions on the target infrastructure. These flows are explained in more detail in Section 3.4.3.

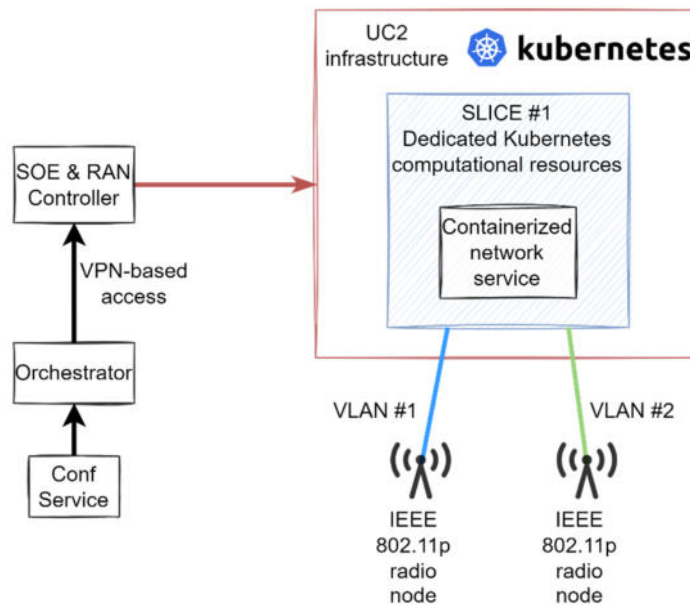


Figure 10: Required integrations among Pledger components for the deployment of network slices based on IEEE 802.11p radio nodes

Figure 10 represents the required integrations among core components for the deployment of an end-to-end network slice with two IEEE 802.11p radio nodes, over UC2’s Kubernetes infrastructure. Note that, in the integration represented in Figure 10 for network slices with IEEE 802.11p radio nodes, individual VLANs are used to provide each radio node with traffic isolation. In this way, incoming traffic can be mapped to a specific radio node, and thus to a specific location, which is a useful feature for many vehicular applications.

Similarly, Figure 11 represents the required integrations for the deployment of an end-to-end network slice with 5G new radio elements, over a dedicated OpenStack infrastructure. In the approach presented here, radio isolation is provided by allocating different radio channels to each network slice. Note that, in Pledger, the deployment of a network slice with 5G radio elements is provided as a proof of concept, showcasing how the platform can support this technology, even though it is not used in any of Pledger’s use cases.

Although the integration points and data flows are similar in the 5G and IEEE 802.11p network slicing scenarios, there are some differences between them, mainly related to the underlying infrastructure over which the network slice is to be deployed, and to the typology of the radio technology to be used. These differences are highlighted in the description of the involved flows in Section 3.4.3.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	33 of 37
<b>Reference:</b>	D5.6 <a href="#">Dissemination:</a> PU	<b>Version:</b>	1.8
		<b>Status:</b>	Final

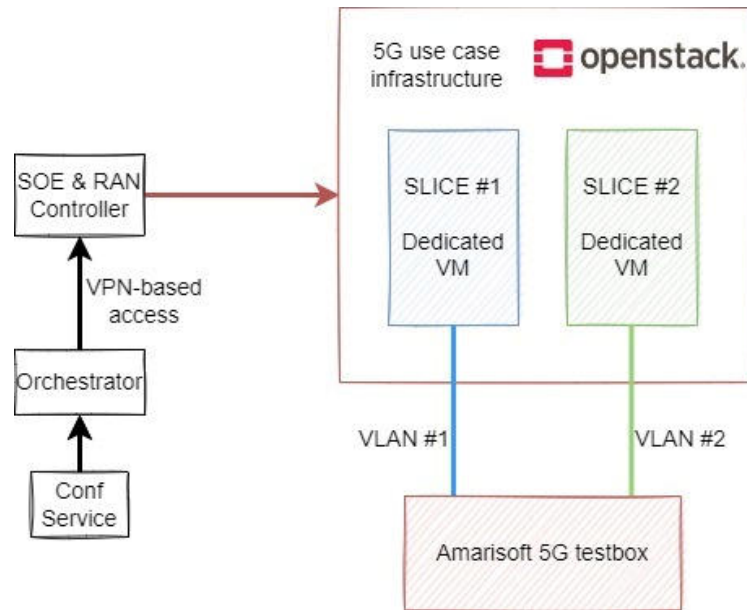


Figure 11: Required integrations among Pledger components for the deployment of 5G network slices

### 3.4.3 Scenario

This Section presents an overview and two demos of the flows and required calls for the deployment of end-to-end network slices in Pledger, with both 5G and IEEE 802.11p radio elements.

In order to provide the reader with the required background to follow the steps performed in the demos, a top-level overview of the flows involved is given below, prior to the presentation of the demos. These flows are generic to Pledger’s network slicing mechanisms and are applied independently of the underlying compute infrastructure and radio technology.

The definition of the flows involved is derived from our approach to network slicing, where an individual network slice is composed by compute, network, and radio chunks<sup>2</sup>, as previously introduced in the deliverable D3.5 Edge/Cloud orchestration tools II [5]. This approach to the composition of network slices requires the following common steps:

- ▶ **Step 1:** Compute chunk creation, where RAM and CPU requirements for the network slice are specified.
- ▶ **Step 2:** Network chunk creation, where the network slice’s virtual network is tagged, and certain required network elements, such as VLANs, are created and configured.
- ▶ **Step 3:** Radio chunk creation, where relevant radio parameters such as transmit power and channel bandwidth are configured.
- ▶ **Step 4:** Creation of a network slice as a collection of compute, network, and radio chunks.
- ▶ **Step 5:** Network slice activation, involving the activation of the radio elements. This involves the deployment of the 5G core elements, in the case of network slices with 5G nodes; or the deployment of a vehicular software stack, for slices with IEEE 802.11p nodes. After this stage, the network slice can be considered as created, activated, and deployed.

The creation of the network slice (steps 1-5 above) is started by the infrastructure provider on the ConfService’s user interface, through the creation and provisioning of a so-called project, where the user specifies the underlying infrastructure where the network slice is to be deployed, together with the values of the required network slice parameters. This provisioning request is then passed on to the

<sup>2</sup> A chunk is understood as a collection of resources.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	34 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b>
			Final

Orchestrator, along with the parameters specified by the user, through a unique Kafka message containing an associated JSON file.

Next, the Orchestrator performs the individual steps listed in steps 1-5 above. For this purpose, for each of the steps, the Orchestrator performs a POST call to the SOE's API interface, containing the relevant parameters gathered from the ConfService.

Note that steps 3, 4 and 5 (radio chunk creation, network chunk creation, network slice activation) require an additional interaction between the SOE and the RAN controller: upon reception of the relevant POST calls from the Orchestrator at the SOE, additional POST calls are sent from the SOE to the RAN controller, which perform the required radio and network configurations. Compute-related configurations, in contrast, are handled in full by the SOE.

After a network slice has been deployed, an application instance (i.e., a network service) can be instantiated on top of it. This requires a request from the service provider on the ConfService's user interface. After the request is generated, the ConfService issues a Kafka message with the relevant parameters related to the instance and the network slice over which it will be deployed. This message is intercepted by the Orchestrator, which then issues a POST call to the SOE, with all the information needed for the instantiation of the network service over a previously created network slice.

### **1. Demo: deployment of a Kubernetes-based network slice with IEEE 802.11p radio elements and instantiation of network service**

In this demo, an end-to-end network slice with two IEEE 802.11p radio elements and Kubernetes computational resources is deployed. Then, a network service is instantiated on top of the slice, leveraging the use of OSM. This demo is represented in Figure 8. At the time of writing, the preparation of a video demonstration was ongoing. This will be uploaded to Pledger's YouTube channel when finalised (expected by M35).

### **2. Demo: deployment of two isolated OpenStack-based network slices with 5G radio elements**

In this demo, two end-to-end network slices with isolated 5G radio cells are OpenStack computational resources are deployed. Then, one user equipment is connected to each network slice. It is verified that each user equipment's network identity matches that of the network it should be connected to. This demo is represented in Figure 9. At the time of writing, the preparation of a video demonstration was ongoing. This will be uploaded to Pledger's YouTube channel when finalised (expected by M34).

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	35 of 37
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.8	<b>Status:</b> Final

## 4 Conclusions

---

This work describes in detail the updates performed with respect to the integration activities of the Pledger Core system and the integrated demonstrator since its first release, and towards its second and final release. Thus, these updates are built on top of the already defined integration methodology using the deployed CI/CD platform and cover the progress up to the finalisation of the integration endpoints among the different software components. At this point, almost all the integration activities have been finalised, missing only the completion of the interoperation among the SOE framework and the Pledger RAN controller for slicing on top of IEEE 802.11p radio access technology. This is part of UC2 and is expected by the end of M34. Thus, the timeline defined in D5.2 [1] for the integration activities has been followed without any serious deviations from the initial plan.

The progress of the integration activities allowed for the second and final release of the Pledger Core system integrated demonstrator that conclude the work of T5.2. This progress allowed for the creation of several selected demonstrators that showcase critical platform features which are being used from one or more use cases. These demonstrators were presented analytically in section 3, together with the references to the corresponding YouTube videos. The preparation of the videos for the end-to-end slicing demonstrators is ongoing and will be published to Pledger's YouTube channel upon their completion.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	36 of 37				
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	<b>Version:</b>	1.8	<b>Status:</b>	Final

## 5 References

---

- [1] PLEDGER. D5.2 – Pledger integrated demonstrator I, Sarris, Ioannis. 2021, <http://www.Pledger-project.eu/content/deliverables>, retrieved on 19 July 2022.
- [2] PLEDGER. D5.4 – Pilots Operation and Monitoring II, Stanzl, Verena. 2022. <http://www.Pledger-project.eu/content/deliverables>.
- [3] PLEDGER. D2.3 – Pledger Overall Architecture, Voutyras, Orfefs. 2022. <http://www.Pledger-project.eu/content/deliverables>, retrieved on 26 August 2022.
- [4] A. Papageorgiou et. al., On 5G network slice modelling: Service-, resource-, or deployment-driven?, Computer Communications, 149, 2020, 232-240.
- [5] PLEDGER. D3.5 – Edge/Cloud orchestration tools II v1.0, Psychas, Alexandros. 2022, <http://www.Pledger-project.eu/content/deliverables>, retrieved on 25 August 2022.

<b>Document name:</b>	D5.6 Pledger integrated demonstrator II	<b>Page:</b>	37 of 37				
<b>Reference:</b>	D5.6	<b>Dissemination:</b>	PU	<b>Version:</b>	1.8	<b>Status:</b>	Final