



D5.4 Pilots operation and monitoring II

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

Related WP	WP5	Document Reference	D5.4
Related Deliverable(s)	D5.3	Dissemination Level (*)	PU
Lead Participant	FILL	Lead Author	Verena Stanzl
Contributors	I2CAT, HOLO, ATOS, ICCS, ENG, INTRA, INNOV, IMI	Reviewers	Nikos Kapsoulis (INNOV)
			Ioannis Sarris, Olga Segou (INTRA)

Keywords:
Pilots Operation, Pilots Monitoring, Metrics, Experimentation

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
Verena Stanzl	FILL
Michael Gillhofer	FILL
August Betzler	i2CAT
Estela Carmona	i2CAT
Adrian Pino	i2CAT
Nour Fendri	HOLO
Carina Pamminger	HOLO
Alexander Werlberger	HOLO
Francesco Iadanza	ENG

Document History			
Version	Date	Change editors	Changes
0.1	01/04/2022	Verena Stanzl (FILL)	Initial structure of the document, table of contents
0.2	29/04/2022	Verena Stanzl (FILL), August Betzler (i2CAT), Estela Carmona (i2CAT), Adrian Pino (i2CAT)	Section 5 added, inputs for UC2 included
0.3	29/04/2022	Verena Stanzl (FILL), Michael Gillhofer (FILL)	Section 6 added, inputs for UC3 included
0.4	02/05/2022	Verena Stanzl (FILL), Nour Fendri (HOLO), Carina Pamminger (HOLO)	Sections 4.1 and 4.3 added, inputs for UC1 included
0.5	02/05/2022	Verena Stanzl (FILL)	Section 2 added
0.6	03/05/2022	Verena Stanzl (FILL)	Section 3 added
0.7	04/05/2022	Verena Stanzl (FILL), Estela Carmona (i2CAT), Francesco Iadanza (ENG)	Revised Section 3 based on partner's inputs
0.8	04/05/2022	Verena Stanzl (FILL)	Section 1 (Introduction), Section 7 (Conclusions) added
0.9	05/05/2022	Verena Stanzl (FILL)	Executive Summary added
0.10	05/05/2022	Verena Stanzl (FILL)	Revised Section 6.3, with small updates in the motivation of the scenario and detailed description
0.11	05/05/2022	Verena Stanzl (FILL), Nour Fendri (HOLO), Alexander Werlberger (HOLO)	Added Section 4.2, inputs for UC1 included. Small formatting fixes in numbering of headings.

Document name:	D5.4 Pilots operation and monitoring II	Page:	2 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

0.12	05/05/2022	Verena Stanzl (FILL)	Added tables showing updates and progress of integration endpoints for UCs in the dedicated integration sections.
0.13	05/05/2022	Verena Stanzl (FILL), Carina Pamminer (HOLO), Nour Fendri (HOLO)	Revised Section 4.3.1.4 considering inputs provided by HOLO
0.14	09/05/2022	Verena Stanzl (FILL), Carina Pamminer (HOLO), Nour Fendri (HOLO), Alexander Werlberger (HOLO)	Added Section 4.4, revised Introduction of Section 4 and revised Section 4.3.1.1 based on UC1 inputs. Abbreviations updated, References added, Formatting checked. Guidelines removed.
0.15	09/05/2022	Verena Stanzl (FILL)	Version for internal review.
0.16	18/05/2022	Verena Stanzl (FILL), Nikos Kapsoulis (INNOV), Ioannis Sarris (INTRA)	Feedback from internal review integrated
0.17	30/05/2022	Verena Stanzl (FILL)	Clear version for quality review
0.18	30/05/2022	Carmen San Román (ATOS)	Quality assurance check
1.0	31/05/2022	Lara López (ATOS)	FINAL VERSION TO BE SUBMITTED

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

Table of Contents

Document Information	2
Table of Contents	4
List of Tables.....	6
List of Figures	7
List of Acronyms.....	8
Executive Summary	10
1 Introduction	11
1.1 Purpose of the document.....	11
1.2 Relation to other project work.....	11
1.3 Structure of the document.....	11
2 Overview of the work performed in this period (M24-M30)	12
3 Overview Pledger components.....	13
4 Use Case 1: Mixed Reality applications on the edge	15
4.1 Integration with Pledger.....	16
4.1.1 Integration with Orchestrator	17
4.1.2 Integration with SLA Lite and DSS	18
4.1.3 Updated Table with summarized Integration Endpoints.....	18
4.2 Metrics and Pilot KPIs	19
4.3 Scenarios performed in M24-M30.....	20
4.3.1 Automated upgrade of VM to enable faster loading times.....	20
4.4 Scenarios planned until end of project.....	27
4.4.1 Integration of benchmarking results to ease the decision for the ideal VM	27
4.4.2 Increasing the QoE considering AR resolution.....	28
5 Use Case 2: Edge infrastructure for enhancing safety vulnerable road users.....	29
5.1 Integration with Pledger.....	30
5.2 Metrics and Pilot KPIs	32
5.3 Scenarios performed in M24-M30.....	34
5.3.1 End-to-end integration with ConfService, E2CO and SOE for the deployment of a compute chunk and network service.....	34
5.3.2 Integration with the SLA and DSS for application scale-up upon delay violations.....	38
5.4 Scenarios planned until end of project.....	41
5.4.1 End-to-end integration for the deployment of a compute + radio chunk and network service	41
5.4.2 Integration with the DLT for the secure management of application data.....	42
6 Use Case 3: Manufacturing the data mining on the edge	43
6.1 Integration with Pledger.....	43

Document name:	D5.4 Pilots operation and monitoring II	Page:	4 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status: Final

6.2 Metrics and Pilot KPIs	46
6.2.1 Metrics to monitor the functionality of the UC applications	46
6.2.2 Metrics to monitor the performance of the UC applications	46
6.2.3 SLAs/KPIs to determine the performance of the UC	46
6.3 Scenarios performed in M24-M30	47
6.3.1 End-to-End integration with service deployment, monitoring and offloading to the cloud in case of an SLA violation	47
6.4 Scenarios planned until end of project	50
6.4.1 Integration with the DLT for the secure management of sensitive information about parts produced	50
6.4.2 Smart contracts	50
7 Conclusions	51
8 References	52

Document name:	D5.4 Pilots operation and monitoring II	Page:	5 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status: Final

List of Tables

<i>Table 1: Status integration endpoints UC1</i>	18
<i>Table 2: Status integration endpoints UC2</i>	32
<i>Table 3: Status integration endpoints UC3</i>	44

Document name:	D5.4 Pilots operation and monitoring II			Page:	6 of 53		
Reference:	D5.4	Dissemination:	PU	Version:	1.0	Status:	Final

List of Figures

Figure 1: Connected Pledger infrastructure	14
Figure 2: UC1 set-up and connectivity with Pledger's core infrastructure	16
Figure 3: API Interface	17
Figure 4: Connecting to the ENG cluster using VPN	20
Figure 5: Running the API interface using the terminal	21
Figure 6: Creating the SLA	21
Figure 7: Service creation and initial configuration	21
Figure 8: Setup DSS for scaling	22
Figure 9: Starting the application	22
Figure 10: Orchestrator launching the initial VM	22
Figure 11: Connect Server to Client	23
Figure 12: Importing a 3D model	23
Figure 13: Selecting the file to be imported	24
Figure 14: File loading cube	24
Figure 15: SLA violation	25
Figure 16: Violation status change	25
Figure 17: Scaling event triggered after receiving the SLA violation	26
Figure 18: The upgraded VM is launched	26
Figure 19: UC2 cluster setup and connectivity with Pledger's core infrastructure	29
Figure 20: UC2 street layout visualization	30
Figure 21: UC2's RDNS and V2X software stack deployment	31
Figure 22: Configuration of compute chunk parameters through the ConfService	35
Figure 23: SOE's compute chunk API endpoint along with the necessary payload	36
Figure 24: Service management through ConfService	36
Figure 25: UC2's setup with ConfService, E2CO and SOE	37
Figure 26: SLA selection	39
Figure 27: DSS action selection	39
Figure 28: Initial service resource assignment	40
Figure 29: UC2's set-up with SLA and DSS	41
Figure 30: UC3 set-up and connectivity with Pledger's core infrastructure	43
Figure 31: Configuration for service deployment and environment variables for UC3	45
Figure 32: App deployment option for the media-consumption service in UC3	48
Figure 33: SLA guarantees of UC3	48
Figure 34: Grafana dashboard of UC3 for monitoring the infrastructure and performance of the service	49
Figure 35: The DSS recommends the offload to the cloud. The terminal shows the application running in the cloud	49

List of Acronyms

Abbreviation / acronym	Description
API	Application Programming Interface
AR	Augmented Reality
CAD	Computer-Aided Design
CAM	Connected and Automated Mobility
CIDR	Classless Inter-Domain Routing
CPU	Central Processing Unit
DC	Data Center
DLT	Distributed Ledger Technology
DSS	Decision Support System
Dx.y	Deliverable number y belonging to WP x
E2CO	Edge-to-Cloud-Orchestration
GB	GigaByte
GPU	Graphics Processing Unit
HL2	HoloLens2
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISAR	Interactive Streaming for Augmented Reality
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MB	MegaByte
MQTT	Message Queuing Telemetry Transfer
NB	NorthBound
NS	Network Service
OBU	On Board Unit
OS	Operating System
PC	Personal Computer
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RDNS	Risk Detection and Notification System
RSU	Road Side Unit
SDK	Software Development Kit
SLA	Service Level Agreement
SOE	Slicing and Orchestration Engine
UC	Use Case
UCX	Use Case number X
UI	User Interface

Abbreviation / acronym	Description
V2X	Vehicle-to-Everything
vCPU	Virtual CPU
VM	Virtual Machine
VRU	Vulnerable Road User
WebRTC	Web Real-Time Communication
WP	Work Package

Executive Summary

Task T5.3 “Pilots operation and monitoring” handles the activities for the on-site preparations of the demonstrators deployment and pilots set-up to validate the Pledger use cases.

In the first deliverable of this task, D5.3 [1], the infrastructure set-up for the use cases (UCs) was described as well as the first experimentation scenarios outlined. In the second iteration of this task, these scenarios were implemented and performed, and the achieved results and observations are documented in this deliverable.

The scenarios presented in this document, validate the functionality of both the individual Pledger components and the Pledger Core system in terms of integration. Specifically, the reservation of dedicated computation and network resources, the orchestration of different services, the monitoring of their performance and their associated Service Level Agreements (SLAs), as well as the automated execution of performance optimization based on decision-making algorithms are demonstrated. To enable these scenarios and define the metrics and SLAs to monitor, discussion on the metrics have been intensified and documented as well.

UC1 presents a scenario to launch a Virtual Machine (VM) and starting an Augmented Reality (AR) application. Based on a metric monitoring the loading time of files, the VM gets upgraded in case the loading time exceeds a threshold impairing the Quality of Experience (QoE) of the user.

UC2 shows a scenario to reserve dedicated computational and network resources before deploying the actual Risk-Detection-Notification-System (RDNS). Furthermore, computational resources are increased in case the monitored end-to-end delay exceeds a certain threshold preventing alert notifications not reaching the Vulnerable Road Users (VRUs) in time and therefore averting a risky situation on street.

UC3 demonstrates a scenario to deploy an application determining the air-consumption of the processing machine on the edge. In the UC, other applications with stronger dependencies on the edge are developed and priorities are given to them. In case resources get limited on the edge, due to fluctuating resource consumption of these components, the air-consumption application is offloaded to the cloud to balance these peaks in the resource consumption. After the decrease of the resources used and having resources available again for the offloaded application, it is re-deployed on the edge.

Due to the current state of integration, not all scenarios could be implemented in a fully end-to-end manner. In the next iteration of this task, the remaining integration endpoints will be implemented to complete the end-to-end scenario and to complement the overall picture.

Furthermore, in the next iteration, the definition of Pilot Key Performance Indicators (KPIs) and metrics for QoE will be intensified to enable the comprehensive evaluation of Pledger.

Document name:	D5.4 Pilots operation and monitoring II	Page:	10 of 53				
Reference:	D5.4	Dissemination:	PU	Version:	1.0	Status:	Final

1 Introduction

1.1 Purpose of the document

This deliverable presents the work performed during the second iteration of Task T5.3 “Pilots operation and monitoring”. In Pledger, three pilots are set-up, namely “Mixed Reality applications on the Edge” lead by HOLO, “Edge infrastructure for enhancing safety vulnerable road users” lead by i2CAT and “Manufacturing the data mining on edge” lead by FILL.

The task consists of three iterations. The first being the set-up of the pilots as well as definitions of experiments for the first pilot phase documented in the Deliverable D5.3 [1]. In the second iteration, which is reported in this document, these first scenarios have been implemented and performed in the pilots. Furthermore, progress was made with the integration of the various components enabling the scenarios and building the basis for future ones. The scenarios reported in this deliverable focus on the service orchestration and monitoring the performance as well as the associated decision-making to improve the performance. Furthermore, metrics and pilot KPIs have been discussed and additional scenarios to be implemented until the end of the project are outlined.

1.2 Relation to other project work

This deliverable builds on the deliverable D5.3 [1] reporting the previous iteration, where the set-up of infrastructure and the integration endpoints, which all UCs have in common, are detailed.

The task focuses on the integration of the UCs with the Pledger Core system as well as configuration and execution of different scenarios. The integration of the different Pledger components is part of Task T5.2 “Integration and Demonstrators setup) and detailed in the associated deliverable D5.2 [3]. The implementation of application-specific modules for each UC is handled in Task T5.1 and documented in D5.1 [4]. The implementation is still work in progress and the second release is planned for August 2022. The extension of the functionality of the UC enables the definition of further scenarios to be implemented and evaluated during the last iteration of the task.

1.3 Structure of the document

This document is structured in 7 major chapters

Chapter 2 presents an overview of the work performed in this iteration to place it in its overall context.

Chapter 3 recaps the involved Pledger components and their functionality for the UCs.

Chapters 4, 5 and 6 present the work performed for UC1, UC2 and UC3 respectively. A short overview of the pilot is given, followed by describing the integration with the Pledger Core system. Furthermore, metrics and KPIs are discussed. The scenarios performed in this iteration are detailed and further scenarios for the remaining time of the project are outlined.

Chapter 7 closes this deliverable with a conclusion.

Document name:	D5.4 Pilots operation and monitoring II	Page:	11 of 53				
Reference:	D5.4	Dissemination:	PU	Version:	1.0	Status:	Final

2 Overview of the work performed in this period (M24-M30)

The main efforts during M25 and M30 (December 2021 and May 2022, respectively) regarding pilots operation consist of four aspects:

- ▶ Preparation and implementation of experimentation scenarios
- ▶ Integration work of UCs with Pledger components
- ▶ Set-up of configurations and preparation of demo material
- ▶ Definition of next steps

To enable the UCs to use components provided by Pledger, the set-up of the infrastructure was performed and the connectivity to the system hosted on the infrastructure provided by partner ENG has been established via OpenVPN [5]. These steps have been established during the first iteration of the task and were documented in the Deliverable D5.3 [1]. Furthermore, first integration endpoints between UC components and Pledger components have been implemented (e.g., integration with Monitoring Engine, Orchestrator, StreamHandler) and the first experimentation scenarios have been outlined.

In this iteration, this work has been intensified by detailing these experimentation scenarios and working on the specific integration endpoint to enable their execution. For this purpose, a common description template to develop the storyboard and to specify the particular steps, was used in all UCs. During this activity the UC leaders extracted the necessary actions to successfully implement the scenarios, which also enabled the monitoring of these defined activities on a weekly basis.

Based on this template, four scenarios (one for UC1 and UC3 each, two for UC2) have been implemented. All UCs implemented a scenario for optimizing resources by either scaling applications on the edge or offloading applications to the cloud. The triggering of the optimization was established by defining associated SLAs based on the application metrics defined in D5.3 and their monitoring. In case of a violation, the defined optimization was recommended and performed in an automated way. The details of configuring and implementation of these optimization scenarios are given in the UC dedicated Sections 4.3, 0 and 6.3, as well as the individual benefits Pledgers offers to the UCs in these scenarios are detailed. In the UC2 another scenario to benefit from Pledger's cloud-native network slicing capabilities was implemented. This enables the reservation of dedicated computing and network resources before deploying the actual application. Both scenarios have been performed separately because of the current state of implementation and integration of the involved components.

The implementation of the scenarios required efforts in integration work, especially for service orchestration, metrics provisioning and SLA configuration, configuration of the resource optimization strategy and reservation of computational resources. In these activities, all partners contributing to the task supported with configuration, testing and execution of the scenarios. Furthermore, they were involved in resolving any technical issues arising during the testing phase.

The implementation of the scenarios was finalized with the preparation of demo material in the form of videos and screenshots supporting the reporting work done in this document.

In coordination with all partners, the next steps have been defined. To enable the project evaluation and quantification of Pledger results and benefits, pilot KPIs are key. In this iteration, the UC leaders started to identify first metrics to be further developed into KPIs in the early phase of the next iteration. The identified metrics and ideas are described in Sections 4.2, 5.2 and 6.2. Beside finalizing the remaining points to provide full end-to-end scenarios, the focus of pilot operation in the next iteration will be on implementation of smart contracts including the discussed metrics and KPIs.

Document name:	D5.4 Pilots operation and monitoring II			Page:	12 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

3 Overview Pledger components

Before describing the details of the different UCs, a short recap and overview of the Pledger components involved in the different scenarios is given. The architecture of the Pledger Core system, as well as the details for the functionalities of the components are given in the deliverable D2.3 [6]. The details for the individual components and subsystems are given in the associated deliverables of WP3 and WP4. The integration and interaction between the different components are part of Task T5.2 and documented in its corresponding deliverables.

The following list summarizes the main components mentioned in this document and a short recap of their core functionality:

- ▶ Edge-to-Cloud Orchestrator (E2CO)¹: management of information related to runtime environment specifications, service orchestration (start, stop, update of applications)
- ▶ Benchmarking subsystem: provisioning of performance data of configured infrastructures
- ▶ App Profiler: Manual or automatic profiling of applications
- ▶ Decision Support System (DSS): provisioning of suggestions related to the app orchestration to optimize performance
- ▶ SLA Lite: Creation, management and evaluation of the SLAs associated to the applications running and notification to other components in case of SLA violations
- ▶ ConfService: User Interface (UI) for infrastructure and application configuration
- ▶ Distributed Ledger Technology (DLT): Pledger’s blockchain network for supporting the respective smart contracts and transactional operations
- ▶ Slicing and Orchestration Engine (SOE) Framework and Radio Access Network (RAN) controller: configuration and management of end-to-end network slices including both radio, network and compute resources, allowing the deployment and orchestration of services on top of these slices
- ▶ Monitoring Engine: collection of cluster infrastructure and application metrics
- ▶ StreamHandler: a high-performance distributed streaming platform, backbone for integration of core components, used by UC1 and UC2 for integration with Monitoring Engine and for data transfer for offloading in UC3

The infrastructure set-up for the different pilots and also the infrastructure provided by ENG hosting the Pledger core components has been described in detail in the former version of this deliverable in D5.3 [1]. The connectivity of the different infrastructures is established by OpenVPN connections to abilitate full visibility among the different Pledger infrastructures and allow the communication for the control plane and the data plane.

Within multiple options available, the consortium chose to start from the needs and constrains raised by each of the infrastructure owners and use an approach which could be easily adopted in the future by potential additional partners. In particular, on UC3 there was a hard requirement to avoid exposure on public Internet, so we opted for a site-to-site OpenVPN [5] configuration configured with reverse tunnel connection so that OpenVPN servers are publicly exposed only on ENG and i2CAT.

With such configuration in place, PODs running on each Kubernetes [2] infrastructures (ENG and i2CAT), containers (FILL) and processes (HOLO) are visible to each other facilitating the distribution of applications over the different infrastructures according to the DSS decisions and the capabilities of the E2CO to configure micro services connectivity. To complement this setup, Pledger also used StreamHandler to allow service to service communication whenever there are no real-time communication requirements.

Figure 1 shows the available infrastructure and their connection showing the core system hosted at the ENG-infra and the different UC infrastructures. The green nodes show the infrastructure of the partners

¹ Both terms, E2CO and Orchestrator are used interchangeable.

Document name:	D5.4 Pilots operation and monitoring II			Page:	13 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

4 Use Case 1: Mixed Reality applications on the edge

UC1's objective is to improve the capabilities of Augmented Reality (AR) solutions by combining them with edge computing technologies and load allocation and optimisation tools from the PLEDGER components to provide high-level services in industrial environments. The industrial environment in which the pilots for this use case will be run is the one provided by FILL in UC3.

The UC will explore on three case studies that integrate machine data into Mixed Reality interfaces and optimizing them via edge computing technology:

- Fast Prototyping
- Remote Support
- Training

All these case studies have a high demand for the visualisation of 3D Computer-Aided Design (CAD) Models as holograms. However, the number of polygons, the size, and the quality of the model represents a challenge for the limited computational power of the smart glasses. Therefore, outsourcing these computations to an external source like an edge device coupled with the adequate resource orchestration, dramatically boosts the performance, enabling up to 50 times larger data visualisation and resulting in a high level of detail and resolution.

To ensure the enhancement of user experience and functionality of the three case studies, UC1 implements its own remote rendering solution - Interactive Streaming for Augmented Reality (ISAR) alongside PledgAR Workspace which is a feature rich application providing useful tools for each of the case studies.

ISAR Software Development Kit (SDK) [7] is a cross-platform remote rendering solution that enables streaming of AR applications, visualise, and interact with high-polygon content by making use of edge computing resources.

PledgAR Workspace is Unity application that runs on an edge resource and comes with a variety of features, e.g. Hand Collision Detection, CAD Model Visualization, Multi-Media Player and more enhancing the Training, Fast Prototyping, and Remote Support scenarios. The implementation of these features is performed in task T5.1 and documented in the associated deliverables.

Infrastructure

The required infrastructure to support this UC is a desktop Personal Computer (PC) using Ubuntu 18.06 Operating System (OS), which runs the Virtual Machine (VM) and is connected to the Pledger Core system, as shown in Figure 2. Several Hypervisors have been tested and finally Qemu KVM [12] was chosen. The machine consists of 2 GPUs, one that will be used by the host OS and the other to be bound to the VMs ran by Qemu KVM. The port 9999 must be open for the connection between Client application (installed on HoloLens 2 (HL2) [8] and Server application (running on the VM). The HL2 provides an Octa-core CPU and 4GB RAM [8]. PledgAR Workspace will run on the VM, while a client application will run on the HL2 establishing a bi-directional data channel. The chosen network must support the Wifi protocol and should ideally allow at least a 20Mbps connection.

Document name:	D5.4 Pilots operation and monitoring II			Page:	15 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

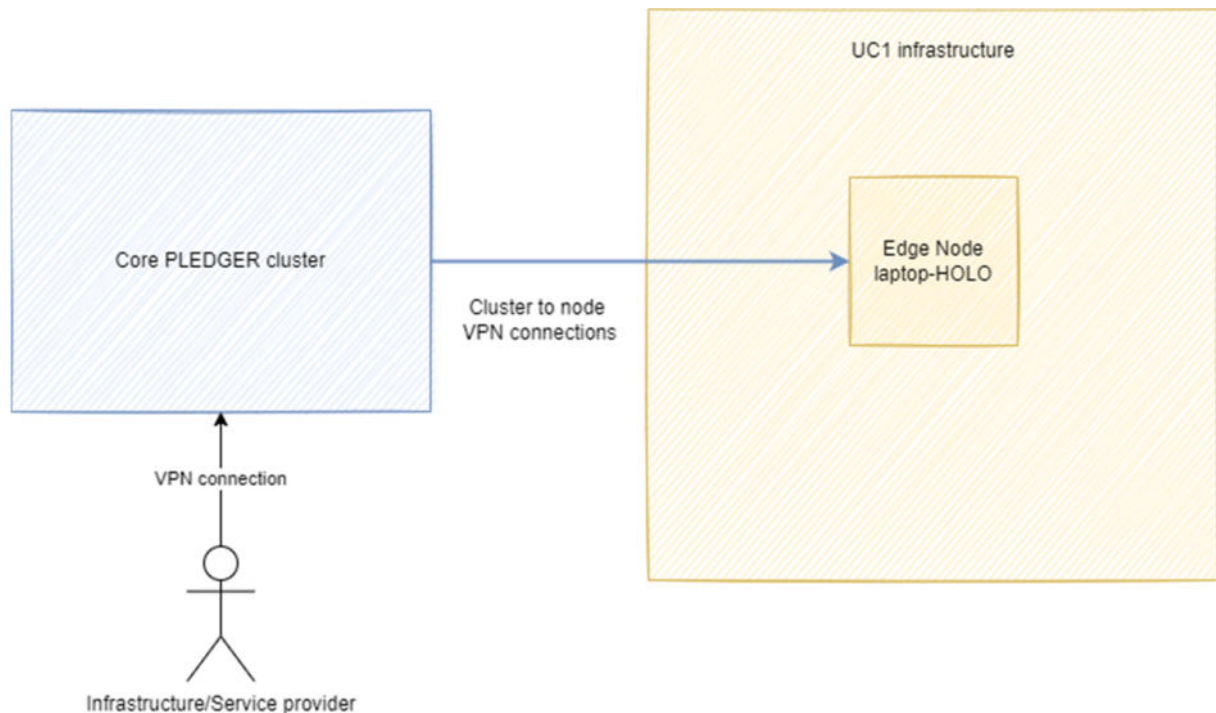


Figure 2: UC1 set-up and connectivity with Pledger's core infrastructure

4.1 Integration with Pledger

Pledger is responsible for providing vital functionality to the UC1 infrastructure:

- ▶ Resource orchestration
- ▶ Application monitoring and decision making
- ▶ Application security enhancement

The UC is a Unity application providing tools for three case studies. The application is deployed on a VM and streamed back to the HL2 using ISAR.

Although ISAR solves the difficulties related to the limited computational prowess of the HL2, the versatility of the edge device remains a challenge that is solved through Pledger. Primarily, linking the Hypervisor with the Orchestrator component provided by Pledger gives the service provider control over the VMs (Start, Stop, and Scale), as well as, adding the missing flexibility and adaptability to them.

Additionally, hardware and application related data is sent to the Monitoring Engine that gets consumed by the SLA Lite to detect any SLA violations. This helps the DSS analyze the state of the PledgAR Workspace application running on the VM and make recommendations to the Orchestrator based on the provided SLAs. This coupling of the Orchestrator and the Recommender gives the infrastructure more automation and judgement while enhancing the Quality of Experience (QoE).

Finally, ISAR, the custom remote rendering solution is based on Web Real-Time Communication (WebRTC). The technology allows devices to establish peer-to-peer direct connections with each other that allows for real-time communications. However, for this direct link to be formed, WebRTC requires a crucial step known as signalling. The signalling phase is obligatory to exchange information for authorization as well as rules and paths for the Peer-To-Peer communication phases. As already described in the deliverable D4.1[9], this data very sensitive and although the media channels created by WebRTC are secure, the signalling phase is not. To solve this, the Whisper solution deployed in the context of Pledger blockchain features will secure the WebRTC's handshake between the peers.

Document name:	D5.4 Pilots operation and monitoring II	Page:	16 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

In conclusion, UC1 requires three separate integrations with Pledger, which will be explained each in the following subsections.

4.1.1 Integration with Orchestrator

To ensure a successful integration with the Orchestrator, the virtualised environment on top of which the PledgAR Workspace is running is required to have a Windows Operating System (OS) with a Graphics Processing Unit (GPU) passthrough.

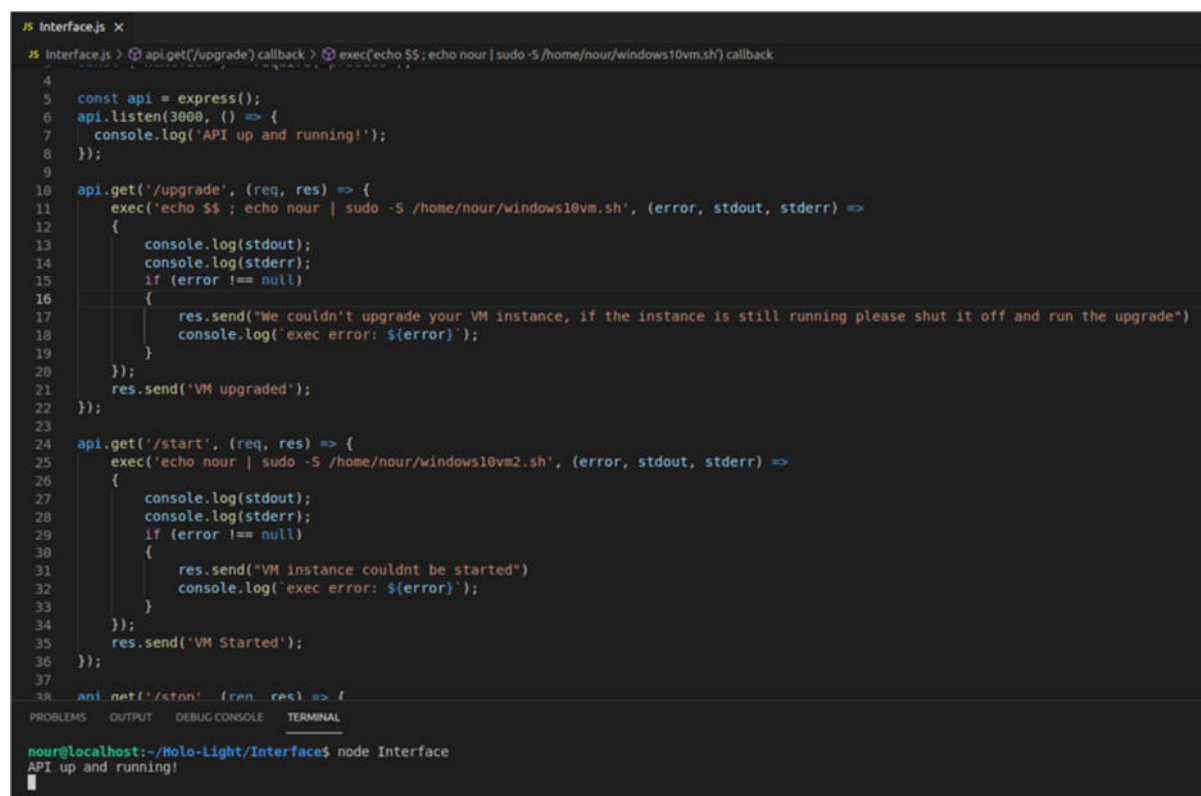
The application was tested over multiple hypervisors (e.g., VMWare [10], VirtualBox [11]), however none provided the desired results as GPU passthrough was not possible on Type-2 Virtual Machines.

To solve this challenge, customized VMs using Qemu KVM [12] on the Linux PC were developed. The described setup led to successful creation of VMs running a Windows host with GPU passthrough enabling the successful execution of the UC application.

Since Qemu does not provide an interface to control the VMs, a simplified Application Programming Interface (API) using a NodeJS web framework called Express.js was developed as shown in Figure 3.

The interface provides three main API calls:

- ▶ Start: Calls a script to run the VM
- ▶ Upgrade: Optionally shuts down the VM in case it is running and starts the upgraded version of it.
- ▶ Stop: Calls a script to stop the VM



```

4
5 const api = express();
6 api.listen(3000, () => {
7   console.log('API up and running!');
8 });
9
10 api.get('/upgrade', (req, res) => {
11   exec('echo $$ ; echo nour | sudo -S /home/nour/windows10vm.sh', (error, stdout, stderr) =>
12     {
13       console.log(stdout);
14       console.log(stderr);
15       if (error !== null)
16         {
17           res.send("We couldn't upgrade your VM instance, if the instance is still running please shut it off and run the upgrade")
18           console.log('exec error: $' + error);
19         }
20     });
21   res.send('VM upgraded');
22 });
23
24 api.get('/start', (req, res) => {
25   exec('echo nour | sudo -S /home/nour/windows10vm2.sh', (error, stdout, stderr) =>
26     {
27       console.log(stdout);
28       console.log(stderr);
29       if (error !== null)
30         {
31           res.send("VM instance couldnt be started")
32           console.log("exec error: $" + error);
33         }
34     });
35   res.send('VM Started');
36 });
37
38 api.get('/stop', (req, res) => {

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

nour@localhost:~/Holo-Light/Interface\$ node Interface
API up and running!

Figure 3: API Interface

Document name:	D5.4 Pilots operation and monitoring II	Page:	17 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

4.1.2 Integration with SLA Lite and DSS

For successful integration with SLA Lite and the DSS, metrics have to be provided to the Pledger Core system. This is done via Integration with the Monitoring Engine. As described in Deliverable D5.3 [1], one option to establish this integration is to publish them on a topic of StreamHandler², which is based on Apache Kafka [13].

As PledgAR Workspace is built using Unity, the adequate Confluent.Kafka library for the .NET framework [14] alongside the credentials provided for a successful integration was used for this purpose.

4.1.3 Updated Table with summarized Integration Endpoints

Table 1 in this section gives an update of the table presented in the deliverable D5.3, showing the status of the different integration endpoints.

It shows the components involved, the responsible partners for achieving the integration, data types and protocol used as well as information about publisher and subscribers (if applicable) and the status of integration at the time of writing this deliverable. An identifier is assigned in the schema X.Y.Z, where X is the number of the UC, Y is the core component (Orchestrator/E2CO – 1, DLT – 2, SaaS/IaaS – 3, SOE – 4, RAN – 5 and StreamHandler – 6), whereas Z the UC component: VM Configuration – 1, Client Utility Application – 2.

Table 1: Status integration endpoints UC1

Integration Endpoint	Components	Responsible	Data Type & Protocol	Publisher/Subscriber	Status M24	Status M30
1.1.1	Orchestrator – VM Configuration	ATOS, HOLO	REST API, OpenVPN TCP	N/A	In progress	Done
1.1.2	Orchestrator – Client Unity Application	ATOS, HOLO	REST API, OpenVPN TCP	N/A	In progress	In progress
1.2.2	DLT – Client Unity Application	INNOV, HOLO	JSON, Whisper	N/A	In progress	In progress
1.3.1	SaaS/IaaS Monitoring Engine – VM Configuration	ATOS, HOLO	JSON, Kafka via StreamHandler	Topic: ▶ UC-metrics	In progress	Done
1.3.2	SaaS/IaaS Monitoring Engine – Client Unity Application	ATOS, HOLO	JSON, Kafka via StreamHandler	Topic: ▶ UC-metrics	In progress	Done

² Described in Deliverable D5.2 [3].

4.2 Metrics and Pilot KPIs

The developed solution for UC1 is an AR application reliant on interactions between the user and the augmented space generated around her/him. The application allows the user to load 3D models and freely manipulate them. However, this manipulation imposes certain requirements on the hardware side. Therefore, to ensure a smooth user experience, the metrics selected for UC1 (File loading time, Latency) allow for a direct insight on the application's performance and functionality as well as the user's QoE. These metrics have a direct impact on the application's performance and can be acted upon through the orchestrator and the decision-making components implemented within the Pledger Core system.

- ▶ **File loading time:** PledgAR Workspace allows the loading of 3D CAD models within the AR space. Nevertheless, these files have different sizes and depending on the edge device's capabilities, Random Access Memory (RAM) and Central Processing Unit (CPU) the loading time varies. This metric measures the loading time of the file currently being imported into the user's workspace. A limit of 17.6 seconds has been placed on the average loading time per 100 MegaByte (MB) and enables the calculation of the limit threshold for the file.
- ▶ **Latency:** PledgAR Workspace is making use of the hardware capabilities of external edge devices through the custom interactive remote rendering technology, ISAR. Latency is therefore defined as the time required for the movement made by the user to be replicated in the AR space (streamed from the application on the edge device to the HL2). For a consistent user experience, the latency should be near real-time, but not more than 50ms. ISAR provides the user with a controllable bandwidth that can be altered through API calls on demand while the application is on runtime. As the bandwidth has a direct influence on the application latency, this makes latency an ideal metric to monitor the application performance and functionality. These application-level metrics are periodically produced and published every 10 seconds. The monitoring Engine captures these metrics and forwards them to the Pledger decision-making components where SLA violations are triggered based on the defined thresholds.

Key metrics for integration and validation purposes: File loading time and latency

During the first pilot iteration for UC1 the experiments have been solely done on the file loading time metric. Results have shown that in a similar edge device hardware configuration, the average file loading time per 100MB increases the larger the file being loaded. Eventually, this triggers an SLA violation that leads to the increase in memory capacity (RAM) which lowers the average loading time. Initially, a configuration of 4CPUs and 4GB of RAM loads a 1.05 GigaByte (GB) file in 3 minutes and 40 seconds while the upgraded one (4CPUs and 8GB RAM) loads the same file in 2.5 minutes.

Although the file loading time metric helps in monitoring the QoE of the user, it does not account for the latency experienced in the case of a sub-optimal bandwidth allocation. Placing limits on the latency is extremely crucial because passing certain thresholds makes the application unusable. As in time of writing, the latency thresholds for SLA violations haven't been yet calculated as they require multiple bandwidth experiments.

In order to make the names of the metrics more speaking, self-explanatory and uniform in the future and also to simplify the distinction between UCs, the following names will be used in future documents:

- ▶ File loading time → ar_file_loading_time
- ▶ Latency → app_to_hl2_latency

Document name:	D5.4 Pilots operation and monitoring II			Page:	19 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

4.3 Scenarios performed in M24-M30

4.3.1 Automated upgrade of VM to enable faster loading times

4.3.1.1 Motivation

This scenario demonstrates the impacts and benefits on the QoE and Quality of Service (QoS) of the UC by using monitoring the SLAs and using the functionalities of decision making and service orchestration of Pledger.

A user using the PledgAR Workspace application can load multiple CAD files into the AR space. The size of these files puts the CPU and RAM under load. The higher the usage of the memory because of loading the files, the slower the subsequent CAD models are to load. Higher loading times result into decreasing QoE of the user, therefore an SLA threshold is set on the metric of loading time. If this threshold is exceeded, the VM needs to be upgraded with increased resources to enhance the QoE.

The goal of this scenario is for the user to load a file and have it pass the threshold of allowed loading time for a file which will trigger an SLA violation resulting in a scaling of the VM. Once the VM is upgraded and its resources increased, the user will have a smoother experience with the files he wishes to load.

4.3.1.2 Configuration

The configurations made for this scenario are related to setting up the SLAs and interfaces to ensure successful data transmission between the PledgAR Workspace application and the Pledger framework and consist of 4 steps:

1. Establishing the connectivity through OpenVPN:

The VPN connects the machine to the Pledger Core systems and allocates a static IP(Internet Protocol) to it, illustrated in Figure 4 . This allows the Orchestrator to access the VMs on the machine through the API interface and to access the ConfService.

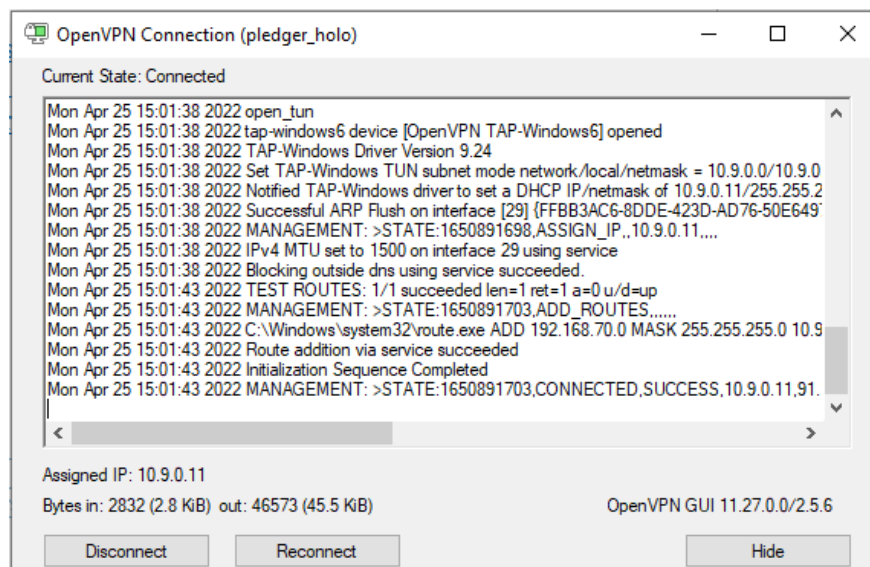


Figure 4: Connecting to the ENG cluster using VPN

2. Running the API interface

To perform this step, the connectivity must be established as described in Step 1. The interface is an API that can be used by the Orchestrator to control the VMs running on the machine as illustrated in Figure 5.

Document name:	D5.4 Pilots operation and monitoring II	Page:	20 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

nou@localhost:~/Holo-Light/Interface$ node Interface
API up and running!
  
```

Figure 5: Running the API interface using the terminal

3. Configuring the SLA violation

The scenario is triggered once the metric “fileloading-time-per-mb” received by the SLA Lite exceeds the predefined threshold. Therefore, the adequate SLA configurations need to be set-up in the ConfService with the associated violations as well as the service involved, shown in Figure 6. The service optimization is shown in Figure 8 and the service creation and initial configuration is shown in Figure 7.

Create or edit a SLA

ID
30

Name
pledgar - fileloading-time-per-mb

DSS Resource management
active

Infrastructure Provider
filippo

Service Provider
filippo

Service
pledgar-workspace

Figure 6: Creating the SLA

Create or edit a Service

ID
1

Name
pledgar-workspace

Profile
cpu-intensive

Priority (1 is the lowest)
1

Initial Configuration
{"max_memory_mb":16000,"min_memory_mb":4000,"min_cpu_millicore":4000,"scaling":"vertical","initial_memory_mb":4000,"initial_cpu_millicore":4000,"max_cpu_millicore":4000}

Figure 7: Service creation and initial configuration

Document name:	D5.4 Pilots operation and monitoring II	Page:	21 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status: Final

Create or edit a Service Optimisation

ID
1

Name
opt_pledgar-workspace

Service
pledgar-workspace

Optimisation
scaling

Figure 8: Setup DSS for scaling

4.3.1.3 Set-Up and detailed description

Once the initial configurations, as described in the previous section, are finished, the VM can be launched by accessing the registered Apps within the, see Figure 9.

Manage an App

ID
2

Name
pledgar-workspace

Management Type
DELEGATED

Status
STOPPED

Action
START ...

Figure 9: Starting the application

This launches the Virtual Machine with the initial resource configuration as shown in Figure 10 (4GB RAM, 4 CPUs).

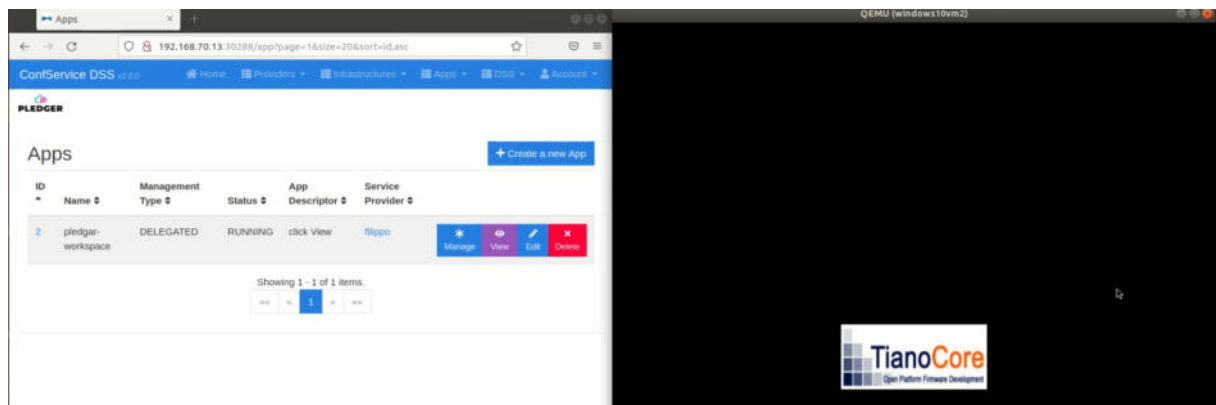


Figure 10: Orchestrator launching the initial VM

Document name:	D5.4 Pilots operation and monitoring II	Page:	22 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

As soon as the VM is started, the PledgAR Workspace is manually launched on the VM. Next, the ISAR Client application is started on the HL2 and the IP address displayed on the PledgAR Workspace application is typed in as illustrated in Figure 11.

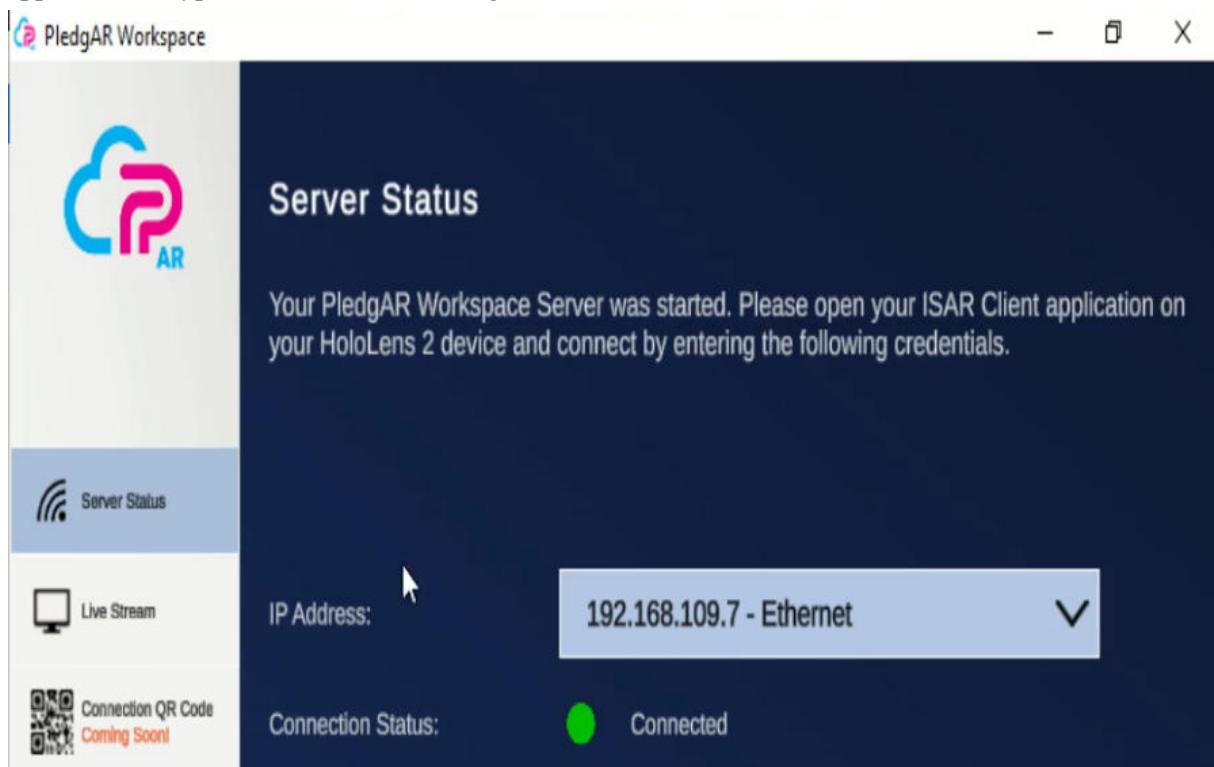


Figure 11: Connect Server to Client

Once connected, the user receives application's stream from the server (VM) where he/she selects to load a CAD model into the AR space (Figure 12) and picks a file with a size of 1.05GB (Figure 13). The file loading cube appears, and the application starts producing metrics (Figure 14).

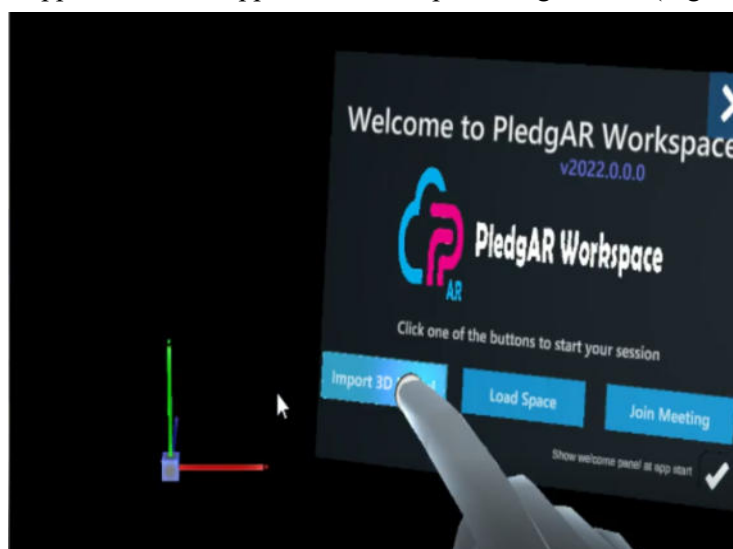


Figure 12: Importing a 3D model

Document name:	D5.4 Pilots operation and monitoring II	Page:	23 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final



Figure 13: Selecting the file to be imported

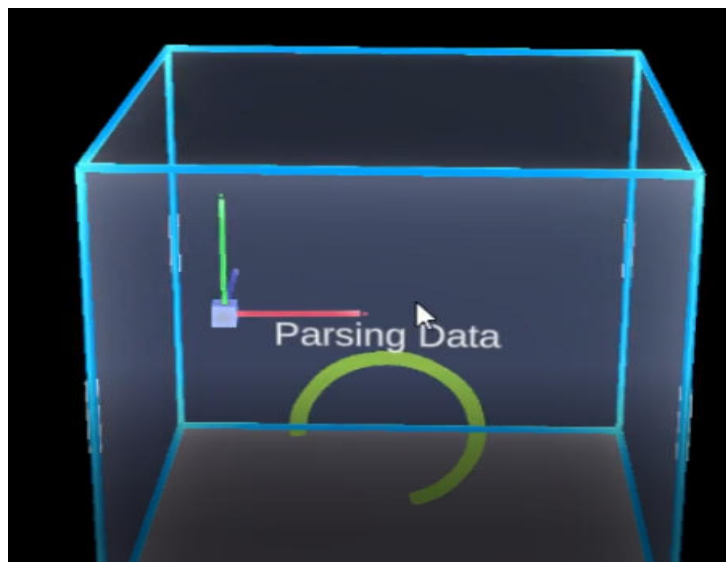


Figure 14: File loading cube

The file's loading time exceeds the defined threshold (180 seconds) and the SLA Lite triggers an SLA violation, shown in Figure 15.

Document name:	D5.4 Pilots operation and monitoring II	Page:	24 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

ConfService DSS v2.6.0 [Home](#) [Providers](#) [Infrastructures](#) [Apps](#) [DSS](#) [Account](#)

PLEDGER

SLA Violations

ID	Timestamp	Violation Name	Severity Type	Description	Status	SLA	
209	Mar 23, 2022, 3:29:51 PM	fileloading-time-per-mb	Serious	guarantee violation	open	pledgar - fileloading-time-per-mb	View

Showing 1 - 1 of 1 items.

«« « 1 » »»

Figure 15: SLA violation

The status of the violation gets updated until it reaches “closed_critical” (Figure 16). Following that, an event for scaling gets triggered by the DSS (Figure 17) whereupon the Orchestrator triggers the VM shutdown and the upgraded one (8GB, 4CPUs) is run instead, demonstrated in Figure 18.

PLEDGER

SLA Violations

ID	Timestamp	Violation Name	Severity Type	Description	Status	SLA	
209	Mar 23, 2022, 3:29:51 PM	fileloading-time-per-mb	Serious	guarantee violation	closed_critical	pledgar - fileloading-time-per-mb	View

Showing 1 - 1 of 1 items.

«« « 1 » »»

Figure 16: Violation status change

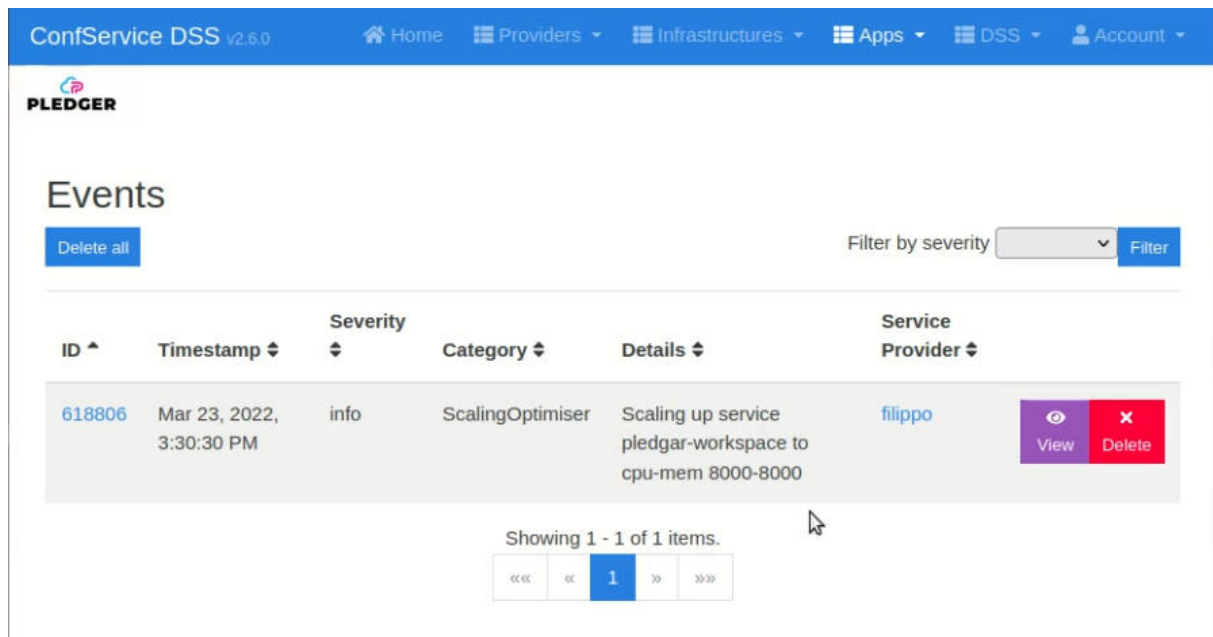


Figure 17: Scaling event triggered after receiving the SLA violation

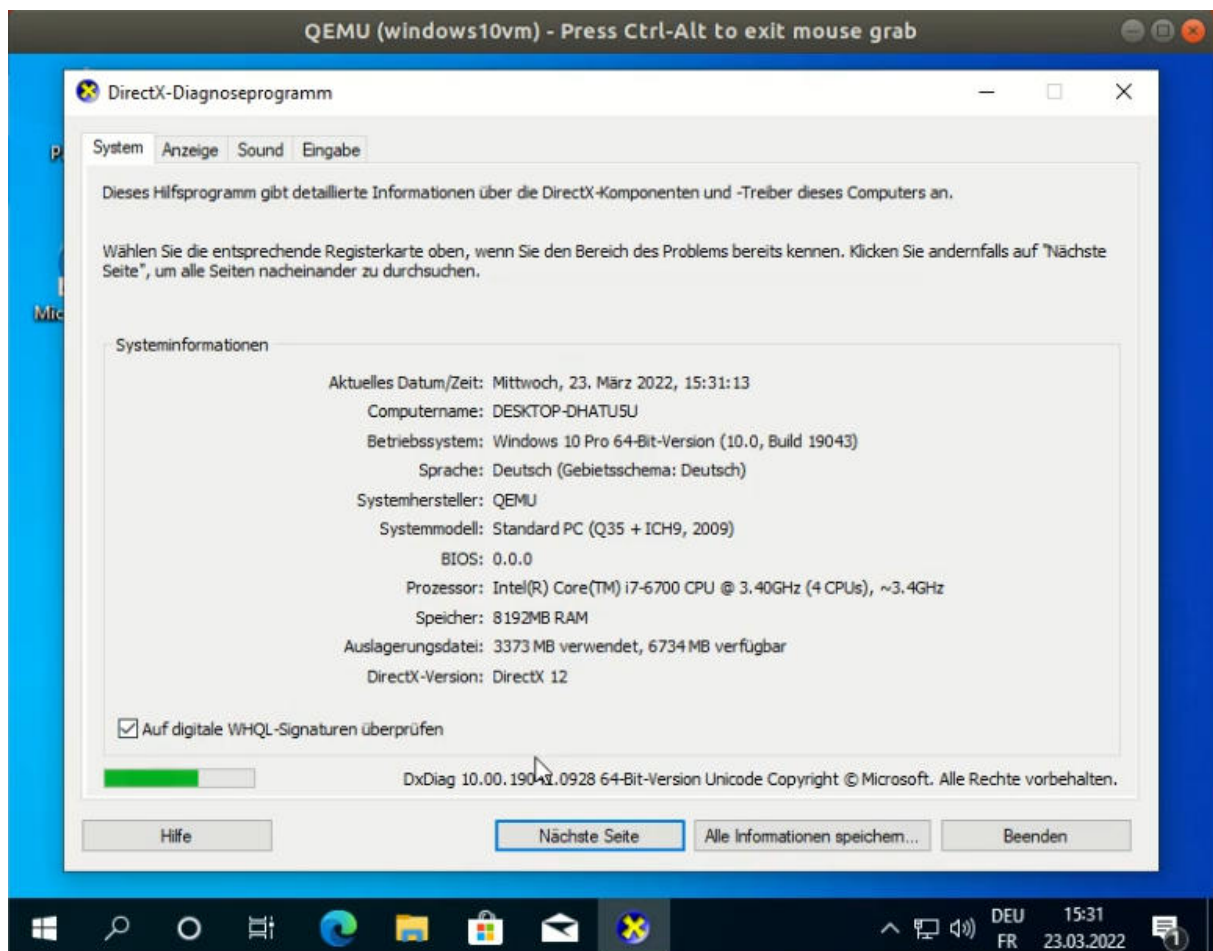


Figure 18: The upgraded VM is launched

Document name:	D5.4 Pilots operation and monitoring II	Page:	26 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

4.3.1.4 Benefits of using Pledger for the scenario

While using the application for any of the previously mentioned three case studies, the user is unable to judge the resources they are going to use. Therefore, they are unable to pinpoint the resources they require from the beginning. For that reason, having a VM that can scale automatically based on how well the application is performing, greatly enhances the user experience. Additionally, this helps in reducing resources and the associated costs (i.e., power consumption, VM resources).

Additional to the increased **flexibility**, a **reduction of cost** is anticipated. Cost can be reduced as the acquisition of new hardware(-parts) to support use cases. Another area of cost reduction is limited logistical requirements and labour costs that would be required to setup new hardware. Furthermore, this increased flexibility does not only reduce cost by freeing up the time of the IT, Development and Finance departments – which are usually involved in hardware purchase requests – but also the time of the user is made more **efficient**.

The user can concentrate on the task at hand, instead of having to test if files can run on the available hardware or not, or finding alternative solutions, such as using a different device, upgrading existing ones, etc. All these steps do not only distract from the actual work, but also consume a large amount of time from multiple employees. Using the Pledger Framework to scale VMs up if needed, therefore, solves multiple issues at once, whilst increasing efficiency of user time.

One of the most common challenges faced by Information Technology (IT) companies are rollouts to departments which presents multiple pain points such as bandwidth usage, IT effort especially if the devices are managed strictly (common in the industrial sector), increased error potential, and time effort. These difficulties are further exasperated in the case of AR and virtual reality applications in the industrial sector where the size hovers around 500MBs. An application like PledgAR Workspace – which is even larger in size – benefits greatly from the Orchestrator provided by Pledger. Instead of collecting devices and carrying out an update one device at a time, the user can automate the upgrade through scripts. The upgrade timeframe can be controlled reducing the risk of blocked worktime for the engineers. In addition, if the application is downloaded by an entire department, the bandwidth will be heavily impacted, resulting in a reduced download speed further slowing the rollout process. If an update was unsuccessfully the user might have to return the device to IT, blocking their worktime even further. As mentioned above, the Orchestrator can be used to plan and roll out updates during off hours, as well as managing VMs remotely through scripts to automate the entire process. Though the risk of a faulty update cannot be entirely removed, a user can, however, simply run a different VM with an already successfully upgraded application, or an older version of the application. What would not be necessary, is uninstalling the application, hoping to find the package and re-initiating the installation process, or looking for an alternative functional device.

The coupling with the framework also creates a more environmentally friendly workflow than the on-premises solution requiring the users to have hardware. In contrast, a centralized system that allows resource-sharing between people also allows the number of required resources to be decreased. Consequently, this solution lowers the costs since less devices need to be maintained and upgraded.

4.4 Scenarios planned until end of project

4.4.1 Integration of benchmarking results to ease the decision for the ideal VM

After finalizing the first scenario, further areas have been discovered. The current scenario – using the loading time based on the CPU load – will be enhanced through features of the benchmarking component. The starting point will be the creation of multiple scripts which can be used to recommend the ideal VM to the Orchestrator and further details will be clarified during the beginning of the next pilot iteration.

Document name:	D5.4 Pilots operation and monitoring II			Page:	27 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

4.4.2 Increasing the QoE considering AR resolution

In AR resolution has a massive impact on the user experience. If the performance is not ideal, this will not only impact the visual quality, but it also strains the user's eyes, which might even result in mild headaches. As the technology is relatively new and user acceptance is still a major point in any project, this point must be in the forefront.

The resolution is heavily impacted by the bandwidth. In short, the higher the bandwidth, the lower is the encoding time and resulting in lower latency. With ISAR it has been observed that the higher the latency, the lower the resolution is.

Current test results showed that latency can be reduced from 100 ms to 85 ms if the bandwidth is increased from 10 mbit to 50 mbit. The downside is, the higher the mbit rate is, the higher the network strain will be. This can have a very negative impact on other resource-intensive applications that might run in the same network. For that reason, this scenario must support also down-scaling. On a positive note, the increase of bandwidth will only be required for a short amount of time, when the decoder's performance is impacted. In the next iteration of the pilot operation, it will be determined, how this can be measured and monitored and how Pledger components can help improve the QoE in this aspect.

The ISAR Bandwidth Settings API can be altered on runtime, allowing an up- and downscale ensuring a high-quality user experience. The following aspects will be explored: A user would use the application, once the latency exceeds the defined SLA threshold a violation will be triggered and communicated to the DSS. Afterwards the DSS will include the benchmarking results and instruct the Orchestrator with the required changes to the value. Finally, the Orchestrator applies the config change via an API call.

Document name:	D5.4 Pilots operation and monitoring II			Page:	28 of 53		
Reference:	D5.4	Dissemination:	PU	Version:	1.0	Status:	Final

5 Use Case 2: Edge infrastructure for enhancing safety vulnerable road users

UC2 explores mechanisms to enhance the safety of Vulnerable Road Users (VRUs) in city layouts where situations of risk may arise due to the proximity among tram tracks, footpaths and bicycle lanes and their concurrent use. In the specific layout under consideration, a bicycle lane is situated in between a footpath and a tram track. Upon the arrival of a tram to a stop, tram users are forced to transit through a bicycle lane to reach the pedestrian footwalk. User on bicycles or electric scooters circulating on the bicycle lane usually have a restricted view of the path ahead when approaching a tram station, which gets partially blocked by the tram station itself. Similarly, pedestrians exiting a tram may not have a clear view on the bicycle lane or might be distracted when exiting the tram. Pledger’s UC2 aims at enhancing the safety of VRUs upon risky situations arising from the scenario described above, through the implementation of a Risk Detection and Notification System (RDNS), which is deployed as a service in Pledger’s edge infrastructure, taking full benefits of the platform’s features.

The RDNS relies upon receiving information about the position of the VRUs and trams, as well as possible risks, in a timely manner. When a VRU (bicycle/scooter) approaches the tram stop while simultaneously a tram is arriving or already stationed, the RDNS generates a notification that is sent out to the VRU, notifying them about the imminent risky situation. For both sharing the location and transmission of messages, IEEE 802.11p [15] is used, featuring On Board Units (OBUs) on VRUs and Road Side Units (RSUs) connected to the Pledger infrastructure, that is composed of a three tier compute architecture (far edge nodes on the lamp posts, edge node in a small Data Center (DC) next to the street and a main DC in i2CAT, all connected over fibre), as represented in Figure 19.

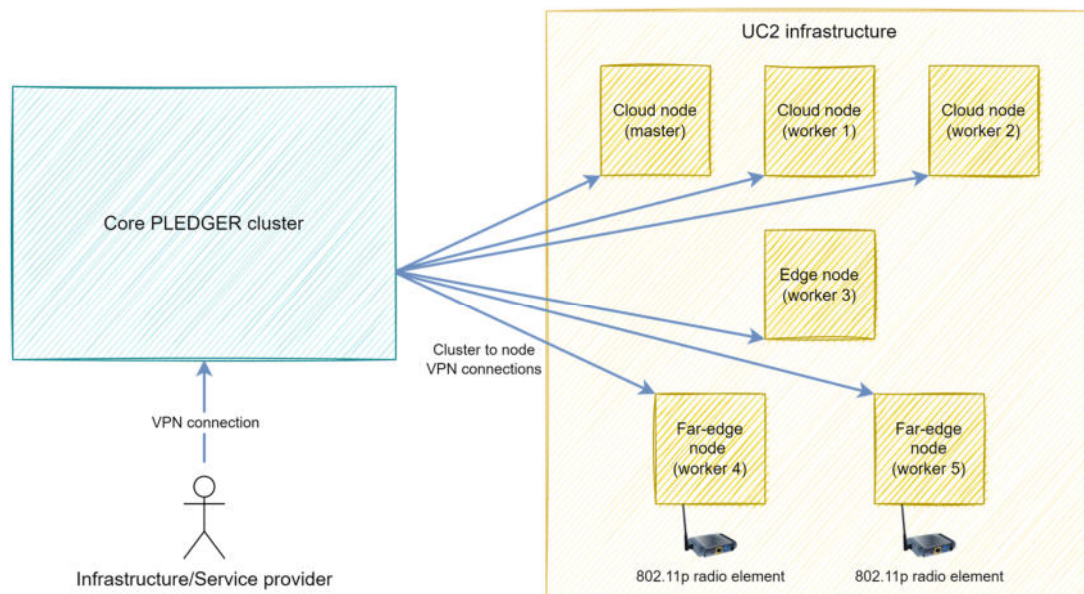


Figure 19: UC2 cluster setup and connectivity with Pledger's core infrastructure

The notifications sent by the RDNS can be either a warning, when the bicycle is still at a certain distance to the station, or it can be a “critical” notification once the bicycle enters the tram station area. When each type of notification is generated, depends in which area a VRU is currently moving. These areas, the warning and tram stop area, are definable and have been adjusted for the particular location in which the pilot will be executed. The layout of these areas and the different lanes is depicted in Figure 20.

Document name:	D5.4 Pilots operation and monitoring II	Page:	29 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

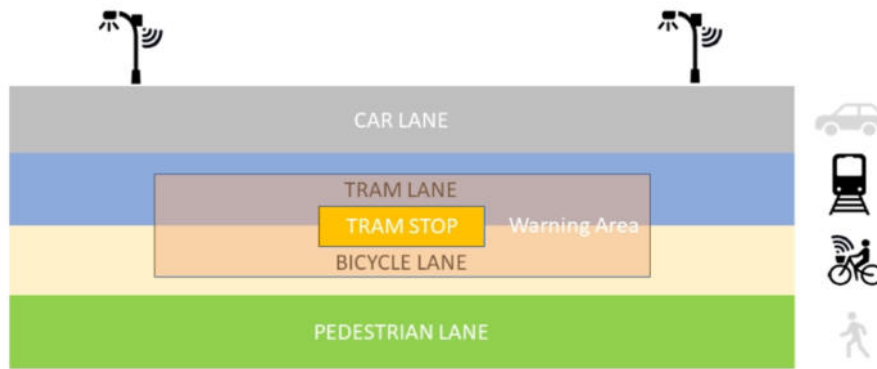


Figure 20: UC2 street layout visualization

5.1 Integration with Pledger

Pledger provides essential functionality to UC2 in the following aspects:

- ▶ Infrastructure management and orchestration
- ▶ SLA notification and recovery
- ▶ Protecting and securing sensitive data

The UC2 service relies on a subset of features provided by the Pledger platform that are key to satisfy the deployment and runtime requirements of the service, as described in the following.

UC2 is a service designed to be deployed within Barcelona's city-wide testbed that features a multi-tier compute architecture and radio connectivity that will be sitting on top of a dedicated network slice. The service requires a certain share of these resources: radio connectivity in the area where VRUs will be transiting and computing resources to host the applications that form the UC2 service. In order to assign the specific resources necessary to the UC as a subset of the overall available testbed resources, Pledger features the Slicing and Orchestration Engine (SOE) and radio access network (RAN) controller modules. With these two modules, a dedicated slice that includes both radio and compute resources can be created which is logically isolated from possible other slices in the same infrastructure. As such, UC2 has a set of well-defined resource requirements that can be translated into a descriptor file. As part of the integration work necessary for this UC, the Orchestrator (E2CO) needs to integrate with the SOE (that integrates with the RAN controller) to issue the reservation of radio and computing resources and to deploy the UC2 service on top of these resources.

UC2 relies on vehicular communications for exchanging information between VRUs and the RDNS service running in the infrastructure. This requires a Vehicle-to-Everything (V2X) stack³ instance to be deployed. Given the V2X Stack application is necessary to enable the radio, it needs to be deployed as part of the infrastructure

setup. This task is performed in the provisioning of the network slice, meaning an integration with the Pledger platform is necessary. Deploying the V2X stack (virtualized in a dedicated container) forms part of the day 0 configuration of the infrastructure, much like the configuration of the radio devices. As an outcome of this integration, the SOE will not only be capable to reserve the compute and radio resources and issue the configuration of the IEEE 802.11p radio to the RAN controller, but also deploy the virtualized V2X stack in the infrastructure to enable the V2X connectivity between the VRUs and the RDNS application. Figure 21 illustrates an exemplary deployment with an edge and two far-edge nodes, where the V2X Stack (Message Queuing Telemetry Transfer (MQTT) broker and the V2X COM

³ As described in D5.1: Pledger Applications for the Use Cases [4].

Document name:	D5.4 Pilots operation and monitoring II	Page:	30 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

modules associated to each IEEE 802.11p radio interface) have been deployed in the infrastructure to enable connectivity between the VRUs and the RDNS application.

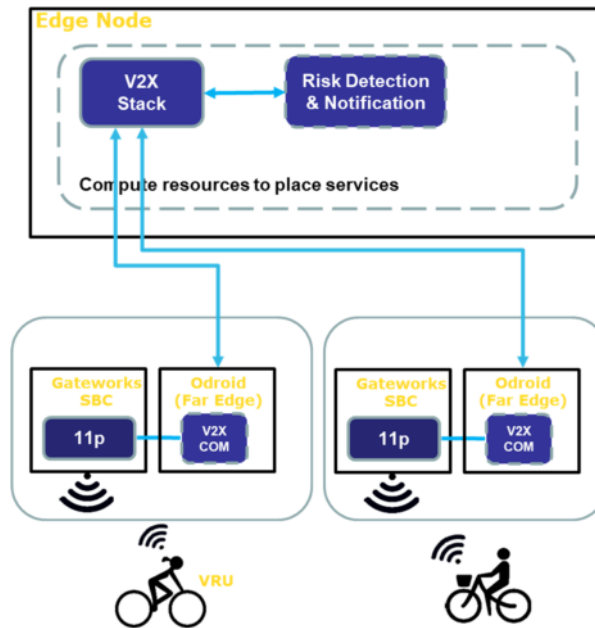


Figure 21: UC2's RDNS and V2X software stack deployment

The previously listed requirements (reservation of a slice to assure a set of resources from the infrastructure, V2X stack for vehicular communications) enable the basic operation of the UC service. However, to assure that the service operates as expected during runtime, application metrics are defined and certain thresholds for these values need to be respected. These requirements can be translated into a set of SLAs to be agreed between the service provider and the platform providers. Once settled and agreed, the service provider of the RDNS application expects that the Pledger platform performs the necessary operations, whenever possible, to assure the SLAs. Exposing, aggregating, and keeping track of the metrics requires an integration with the SLA subsystem and the DSS in particular.

Finally, UC2 during run time generates a set of data that includes application layer information about events that occurred on-street and the logs about the location of VRUs that have connected to the service. This (partially) sensitive information is stored locally in the application and is only to be exposed to trusted entities. To implement this trusted access, the data is written to the DLT, requiring an integration between the RDNS application and Pledger DLT component.

Summing up, UC2 requires a total of 4 specific integrations with the Pledger platform, the status is summarized in Table 2, showing the updates and progress since the last iteration presented in D5.3. The abbreviation follows the pattern explained in Section 4.1.3. The UC components are encoded in the scheme: Barcelona Infrastructure – 1 and Risk Detection and Notification System (RDNS) – 2.

The details on these integration points are presented in Sections 0 and 5.4.

Regarding the connectivity set-up, Figure 19 represents the three-layered structure of UC2's infrastructure and the required connectivity with the Pledger core platform. All core Pledger components run in the core Pledger cluster, except for the SOE and RAN Controller, which run in UC2's infrastructure. Therefore, connectivity from Pledger's cluster to UC2's infrastructure must be guaranteed. To avoid service exposure on the public Internet, the Pledger consortium opted for site-to-site OpenVPN configuration, which reduces such risk. Further, to enable pod-to-pod connectivity in future developments, individual OpenVPN connections have been set-up between the Pledger cluster and each one of UC2's infrastructure nodes.

Document name:	D5.4 Pilots operation and monitoring II	Page:	31 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

Table 2: Status integration endpoints UC2

Integration Endpoint	Components	Responsible	Data Type & Protocol	Publisher/Subscriber	Status M24	Status M30
2.1.1	Orchestrator – Barcelona Infrastructure	ATOS, i2CAT	JSON, OpenVPN TCP	N/A	Done	Done
2.2.2	DLT - RDNS	INNOV, i2CAT	JSON, Kafka via StreamHandler	Topic: ► vru_positions	In progress	In progress
2.3.1	SaaS/IaaS Monitoring Engine – Barcelona Infrastructure	ATOS, i2CAT	JSON, HTTP	N/A	Done	Done
2.3.2	SaaS /IaaS Monitoring Engine - RDNS	ATOS, i2CAT	JSON, HTTP	N/A	In progress	In progress
2.4.1	SOE – Barcelona Infrastructure	i2CAT	JSON, REST API	N/A	In progress	Done
2.5.1	RAN Controller– Barcelona Infrastructure	i2CAT	XML, NETCONF	N/A	In progress	In progress

5.2 Metrics and Pilot KPIs

The metrics of interest in UC2 are those related to the capability of the RDNS to send relevant warnings to the system users via vehicular communications, such that risk situations are timely detected and can be acted upon. Connected and Automated Mobility (CAM) application metrics are standardized by ETSI [17], [17]. Considering the RDNS to be an application designed for vehicular communications, the relevant metrics identified in this UC are delay, packet loss and queue load, metrics that can significantly impact the service’s performance. By design, Pledger’s orchestration and decision-making modules act upon edge/cloud resources and the services deployed there. In this sense, the following UC2 metrics are explained and acted upon from the compute perspective⁴.

- Delay: measured as the time the position information generated by OBUs needs to reach the RDNS, delay is a key metric. It directly impacts on how fast a risk can be detected and a warning can be sent by the service to the VRU. The delay increases by over one order of magnitude if the services hosted in the infrastructure starve on resources (in particular virtual CPU (vCPU)). Upscaling or offloading the service to a less congested node can improve the delay. According to [17], [17], end-to-end delay is expected to be lower than 1.5 seconds, considering message generation and transmission/reception delays on both sender and receiver.
- Packet loss: messages between the VRUs and the RDNS can get lost. Few resources available to the RDNS increase the likelihood for packets to be dropped due to lack of processing capacities. Only by satisfying the services’ required resources and placing the service in a compute node that suffices

⁴ Note that certain phenomena external to Pledger’s orchestration and decision domain may impact UC2’s metrics; for example, congestion in a wireless link or the malfunction of a user device might incur in additional packet loss, delay or queue load.

these requirements, packet losses can be averted. According to [17], [17], no more than 5% consecutive packet loss shall be observed when operating under line-of-sight conditions, which is the case in UC2.

- ▶ Queue load: if many position messages coming from OBUs must be processed by the RDNS, the internal queues of the service might start filling up, and service congestion may arise. Only by assuring sufficient processing capacities (vCPUs), full queues and consequent packet losses can be averted. Measuring the queue load gives a useful indication on whether the service can service all the current users or whether an offloading / scaling-up might be necessary. According to [17],[17], assuming a minimum time interval between CAM messages generation of 100 ms, and considering two concurrent OBUs sending CAMs, a load of 20 CAMs/second is expected in optimal conditions.

All relevant metrics are measured at the application level and sent periodically to the Prometheus [18] server deployed in UC2's infrastructure. These metrics are gathered by Pledger's monitoring engine, such that SLA violation messages are triggered when the metric readings are over a certain threshold.

Key metric for integration and validation purposes: Delay

During the first iteration of the UC2 pilot, only the delay metric has been included in the experiments. In first instance, a series of experiments have been carried out to verify the delay performance under CPU stress. Specifically, the following tests have been performed: (i) the CPU of the node where the RDNS was running was stressed by deploying additional applications with large computational requirements, and (ii) the CPU requirements of the RDNS were increased by simulating a large number of users in the system. It was found that, in both scenarios, the delay increases by over one order of magnitude. In these two scenarios, the RDNS benefits from the functionalities offered by the Pledger platform. Specifically, under scenario (i), the RDNS needs to be offloaded to a node with more available CPU resources; and, under scenario (ii), a scale-up of CPU resources assigned to the RDNS is needed.

Since radio links haven't been included in the first pilot, and in order to cater for the additional transmission delay introduced by these, a conservative delay threshold of 800 ms has been set for the generation of the relevant SLA violations; i.e. when the measured delay is equal to or greater than 800 ms, an SLA violation message is generated by Pledger's SLA module. Note that this threshold value has been decided based on initial experiments and keeping in mind that radio elements will introduce further (small) delays. However, just like any other metrics, the threshold can be adjusted to a different value if needed, once more knowledge is acquired about the behaviour of the RDNS and the additional delays introduced by the radio elements.

As mentioned above, by the time of writing this deliverable, the delay metric has been used as key metric for integration and validation purposes, yet all three UC2 metrics will be available for defining SLAs in the pilot.

In the same way as in Section 4.2, the naming will also be adapted for future deliverables:

- ▶ Delay → obu_to_rdns_delay
- ▶ Packet Loss → rdns_packet_loss
- ▶ Queue load → rdns_buffer_overflow

Document name:	D5.4 Pilots operation and monitoring II			Page:	33 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

5.3 Scenarios performed in M24-M30

5.3.1 End-to-end integration with ConfService, E2CO and SOE for the deployment of a compute chunk and network service

5.3.1.1 Motivation

Network slicing allows infrastructure owners to run multiple logical networks on a common physical infrastructure, so that specific resource requirements can be assigned according to specific use case needs. In the context of UC2, Pledger’s orchestration and slicing framework -via SOE- allows for the provision of network slices with dedicated computing, network and radio resources, following a deployment-driven network slice model [19]. Slices are modelled as sets of infrastructure resources (compute, network and radio chunks), where Network Services (NSs) may be linked to the resource chunks.

UC2 particularly benefits from Pledger’s cloud-native network slicing capabilities. The required computational resources are assigned to UC2 through the isolation of groups of resources (vCPU & RAM) within the Kubernetes cluster. In this way, the RDNS and the V2X software stack can be deployed as NSs on top of a compute chunk, in one node or another of the cluster, opening up the possibility of performing offloading actions in a seamless manner, as required. In Barcelona, the Kubernetes cluster is composed of 4 nodes, including two far-edge nodes, one edge node and a main DC node, resulting in a three-tier architecture. Network isolation is natively provided by Kubernetes, through the creation of dedicated sub-nets along with their own Classless Inter-Domain Routing (CIDR), in a manner that results transparent to both the service provider and the infrastructure provider.

In addition, by incorporating the cloud-native network slicing paradigm, computational resources are natively managed by Kubernetes. Containerized workloads running on top of a slice (mainly Kubernetes pods governed by kubernetes deployments) will have a guaranteed amount of compute resources. Resources can be reserved at both the application level and the slice level. Under the Kubernetes resource management scheme, when an application or a namespace are not using all the resources reserved for them, these could be used by another application or namespace, respectively. Nevertheless, SOE introduces the condition that the sum of resources reserved by all namespaces (i.e. compute chunks) must not exceed the total amount of resources available in the cluster. In this way, each compute chunk is guaranteed to have all reserved resources available if needed. Once the slice for UC2 created, via the ConfService, EC2O & SOE, the Pledger Orchestrator deploys the application container on top of the slice, at which point the service is operational.

5.3.1.2 Configuration

The network slice creation on top of the services to be deployed later on is an action initiated by the ConfService core platform module. In the considered scenario, compute chunks are deployed⁵; specifically, this scenario demonstrates the configuration and deployment of a compute chunk. Specific compute chunk parameters can be adequately set-up through the ConfService UI, which communicates the configuration parameters to the E2CO through the Streamhandler Platform. In turn, the E2CO interfaces with the SOE’s NorthBound (NB) API, and the necessary calls with relevant configuration parameters are sent towards the SOE to request the creation of a compute chunk, thus triggering the reservation of the necessary resources. Figure 22 shows how the relevant compute chunk parameters are configured and reserved through Pledger’s ConfService. In the example, a total of 26 vCPUs and 66 GB of RAM are allocated to the compute chunk upon creation.

⁵ Network slices are composed by one or more compute, radio or network chunks; these are collections of compute, radio or network resources, respectively.

Document name:	D5.4 Pilots operation and monitoring II	Page:	34 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

Create or edit an Infrastructure

ID

8

Name

UC2-infra-30

Type

soe

Endpoint (eg. Kubernetes API server URL)

https://172.16.10.30:6443

Credentials (token) to allow monitoring [for K8S, if no kubeconfig in MonitoringPlugin, use the Base64.decode of the IP's service account secret]

.....

Monitoring Plugin (eg. {kubeconfig: '/var/myconfig', monitoring_type: 'metrics-server', goldpinger_endpoint: 'my_url'})

```
{kubeconfig: '/var/kubeconfig_i2cat_30/pledger-i2cat-30.kubeconfig', goldpinger_endpoint: 'http://172.16.10.30:30080', 'soe_endpoint': 'http://172.16.10.169:30989/api/v1.0', 'http://172.16.10.30:30090', prometheus_endpoint: 'http://172.16.10.30:30090'}
```

Properties (eg. {infrastructure_location: 'my_site'})

```
{infrastructure_location: 'Barcelona'}
```

Total Resources

```
{cpu_millicore: '26000', memory_mb: '66000'}
```

Infrastructure Provider

estela

Cancel Save

Figure 22: Configuration of compute chunk parameters through the ConfService

The reservation of slice compute resources is as follows. The SOE receives a request to create a Kubernetes resource quota (i.e. the amount of CPU and memory resources) that will be assigned to the slice. Subsequently, the SOE communicates with UC2 infrastructure's Kubernetes API, and a Kubernetes namespace is created with the required compute resources. Figure 23 represents SOE's endpoint along with the necessary payload that is required to deploy a Kubernetes-based compute chunk in UC2's infrastructure. In this example, 200 millicores (i.e. 0.2 vCPUs) are guaranteed to the compute chunk; if needed, additional vCPUs may be temporarily assigned to the compute chunk, up to 400 millicores (i.e. 0.4 vCPUs).

Isolation between the compute resources across slices (and the services running on them) is natively provided by Kubernetes. Note, that Kubernetes provides some elastic resource adjustment capabilities: reserved resources are guaranteed but, if not all resources are used at a given time, these are freed-up and can be used by other compute chunks. Therefore, Kubernetes provides a soft resource management approach, e.g., unused resources can be used by a different namespace (i.e., compute chunk) if needed.

Document name:	D5.4 Pilots operation and monitoring II	Page:	35 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

Edge/Cloud Compute Chunk

GET /compute_chunk Get Compute Chunks Information

POST /compute_chunk Create a new Compute Chunk

The body of the request

Example Value | Schema

```

{
  "name": "nova",
  "user_id": "5b63089158f568073093f70d",
  "username": "username",
  "password": "password",
  "description": "Compute node 1 on test-bed 2",
  "compute_id": "5b63089158f568073093f70d",
  "requirements": {
    "ram": {
      "required": 1024,
      "limits": 2048,
      "units": "Mi"
    },
    "cpus": {
      "required": 2,
      "limits": 4,
      "units": "M0"
    },
    "storage": {
      "required": 100,
      "limits": 200,
      "units": "GB"
    }
  }
}

```

→

```

{
  "name": "namespace-name",
  "user_id": "5b63089158f568073093f70d",
  "description": "Compute on PLEDGER's K8s cluster",
  "compute_id": "5b63089158f568073093f70d",
  "requirements": {
    "ram": {
      "required": 1024,
      "limits": 2048,
      "units": "Mi"
    },
    "cpus": {
      "required": 200,
      "limits": 400,
      "units": "m"
    },
    "storage": {
      "required": 8,
      "limits": 10,
      "units": "Gi"
    }
  }
}

```

Figure 23: SOE's compute chunk API endpoint along with the necessary payload

In addition, the deployment of UC2's RDNS in a dedicated network slice is performed in this scenario. Again, the application is launched from the ConfService UI. The application descriptor and the compute chunk of interest are sent to the E2CO, then to the SOE, and then to UC2's Kubernetes API interface; after this flow, the service is deployed in the requested compute chunk. Figure 24 shows how the service is configured through the ConfService UI, which allows the selection of the resources assigned to the service as well as the namespace (i.e. compute chunk) where the service will be instantiated. After the service is configured, the corresponding application can be launched from the ConfService UI, and finally it is deployed in UC2's Kubernetes infrastructure.

Manage a Service

ID

3

Name

risk-detector

Initial Configuration

{"max_memory_mb":500,"min_memory_mb":256,"min_cpu_millicore":120,"scaling":"vertical","initial_memory_mb":256,"initial_cpu_millicore":120,"max_cpu_millicore":500,"replicas":1}

Priority (1 is highest)

1

Runtime Configuration

{"replicas":1,"namespace":"pledger-uc2-slice","memory_mb":256,"infrastructure_id":8,"nodes_selected":"kubefar1","cpu_millicore":256}

Status

RUNNING

Action

Figure 24: Service management through ConfService

Document name:	D5.4 Pilots operation and monitoring II	Page:	36 of 53
Reference:	D5.4	Dissemination:	PU
Version:	1.0	Status:	Final

5.3.1.3 Set-Up and detailed description

This scenario involves the deployment of a compute chunk (i.e., the compute resources of a network slice) and the deployment of the RDNS over the previously created compute chunk. In addition, the V2X stack software components are manually deployed over the compute chunk, as these provide some functionalities needed for both the radio connectivity and the RDNS. In a later integration stage, though, the V2X stack will be deployed during the creation of the network slice.

The hardware platform used for this test scenario is UC2's Kubernetes cluster, formed by four Openstack-based [20] virtual machines (specifically, a master node with 4 virtual CPUs (vCPUs), 6 GB of RAM, and 70 GB of hard disk space; and three worker nodes with 2 vCPUs, 4 GB of RAM, and 50 GB of hard disk space) and two "far edge" bare-metal nodes (each hosted in an Odroid H2+ board with an Intel Quad-core processor, 32 GB of RAM and 500 GB of hard disk space).

Connectivity to and from the Pledger platform is achieved through dedicated VPN servers at i2CAT's and ENG's premises, where UC2's infrastructure and the core Pledger components are respectively located.

Figure 25 represents the integration between UC'2 application, running in i2CAT's infrastructure, and the Pledger components that are used in this scenario, along with the relevant flows among components. The basic flow of this scenario is explained below:

1. Prior to deploying a compute chunk in UC2's infrastructure, the infrastructure provider needs to use the ConfService UI to configure the amount of compute resources (vCPU, RAM) assigned to the compute chunk.
2. The infrastructure provider triggers the deployment of the compute chunk through the ConfService UI. After a few seconds, the compute chunk (i.e. Kubernetes namespace) is deployed in UC2 's infrastructure.
3. Prior to deploying the RDNS, the service provider needs to use the ConfService UI to configure the amount of compute resources (vCPU, RAM) assigned to the application, as well as the compute chunk (i.e. Kubernetes namespace) where it will be deployed.
4. The service provider triggers the deployment of the RDNS through the ConfService UI. After a few seconds, the RDNS is deployed in the selected compute chunk.

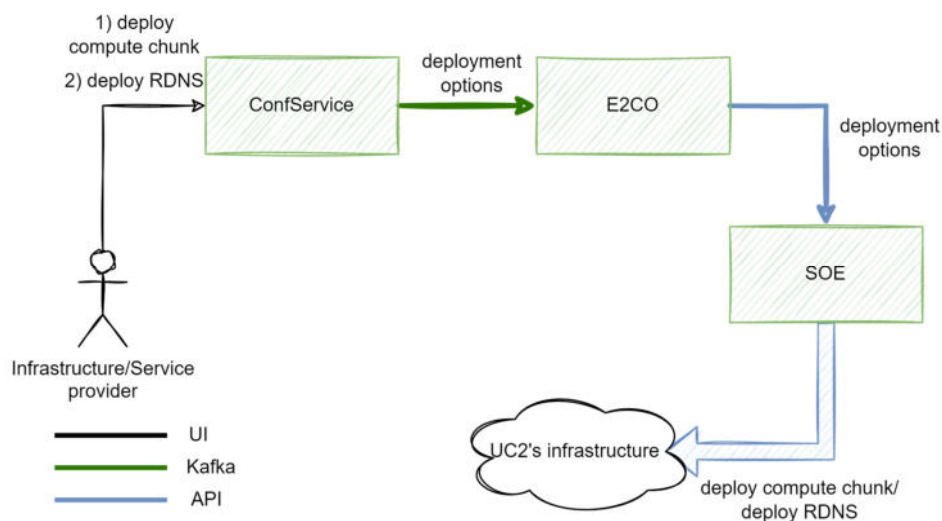


Figure 25: UC2's setup with ConfService, E2CO and SOE

Document name:	D5.4 Pilots operation and monitoring II	Page:	37 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

5.3.1.4 Benefit of using Pledger for the UC

The infrastructure owner benefits in several aspects from being able to deploy end-to-end slices to host specific services. An important aspect is that with a single, general-purpose infrastructure deployment composed of radio and compute resources, a whole series of verticals and their use cases can be deployed, as long as the resources required by those services can be served. Once a service provider decides to use the infrastructure, Pledger reserves the desired end-to-end slice from the available infrastructure resources. The alternative of deploying dedicated hardware as new use cases plan on deploying their services is hardly scalable and more expensive. Operation and maintenance are also less trivial, as more equipment needs to be maintained. Another important aspect is the ease and speed with which new slices can be deployed: within few minutes the general-purpose infrastructure is configured in such a way that the necessary radio and compute services are secured, and the service becomes operational.

5.3.2 Integration with the SLA and DSS for application scale-up upon delay violations

5.3.2.1 Motivation

In UC2, SLAs are defined in terms of a series of relevant application metrics. To demonstrate the validation of the SLA and DSS integration with the UC, the scenario was developed around the end-to-end delay metric, which measures the time that position packets generated by the VRU's on-board units need to reach the RDNS application, plus the additional time required by the RDNS for processing the packets. The processing speed of vehicular messages slows down when the RDNS becomes resource constrained, for example because the number of VRUs in the system increases and the originally allocated resources for the application no longer suffice. A slow down of the processing speed of vehicular packets results in an increase of the average delay and once the delay becomes too long, alert notifications might not reach the VRUs in time for preventing a risky situation. By defining an SLA according to a given delay threshold, computational resources assigned to the RDNS can be adjusted by the DSS in a timely manner, potentially preventing a lack of resources at the application level, which would contribute to the malfunctioning of the RDNS.

5.3.2.2 Configuration

Initially, the RDNS application is deployed by the service provider through the ConfService portal. Upon deployment, the service provider is able to select the SLA of interest, the action to be performed by the DSS in the event that an SLA violation occurs, and the initial number of resources assigned to the application.

In this test scenario, as detailed in the motivation, a high delay is related to a lack of computational resources at the application level. Therefore, the SLA of interest is the delay, which is selected as shown in Figure 26. Further, the DSS should perform a scale up of resources when a delay violation occurs, as depicted in Figure 27. Furthermore, Figure 28 shows the amount of resources initially assigned to the application, which in this case are 256 MB of RAM and 120 millicores of CPU, respectively.

Document name:	D5.4 Pilots operation and monitoring II			Page:	38 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

Create or edit a SLA

ID

31

Name

risk-detector - delay

DSS Resource management

active

Infrastructure Provider

estela

Service Provider

august

Service

risk-detector

Cancel Save

Figure 26: SLA selection

Create or edit a Service Optimisation

ID

2

Name

opt_risk-detector

Service

risk-detector

Optimisation

scaling

Cancel Save

Figure 27: DSS action selection

Document name:	D5.4 Pilots operation and monitoring II	Page:	39 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

Create or edit a Service

ID
3

Name
risk-detector

Profile
cpu-intensive

Priority (1 is the lowest)
1

Initial Configuration
{"500","min_memory_mb":"256","min_cpu_millicore":"120","scaling":"vertical","initial_memory_mb":"256","initial_cpu_millicore":"120","max_cpu_millicore":"500","replicas":"1"}

Runtime Configuration
{"replicas":"1","namespace":"pledger-uc2-slice","memory_mb":"500","infrastructure_id":"8","nodes_selected":"kubefar1","cpu_millicore":"500"}

Figure 28: Initial service resource assignment

5.3.2.3 Set-Up and detailed description

This scenario involves the deployment of a single instance of UC2’s application, the RDNS, along with a chain of some additional containerized applications (the V2X stack), and some virtualized VRUs, all running in UC2’s Kubernetes cluster. The virtualized VRUs can be used to increase the number of (virtual) VRUs in the UC, without the need of physical OBUs. Just like the actual physical devices, they generate vehicular messages, reporting virtual positions and increasing the number of vehicular messages forwarded to and from the vehicular application (the RDNS, in this case). This increases the computational load on the RDNS application, which in return requires more computing resources to process the messages.

The hardware platform used for this test scenario is UC2’s Kubernetes cluster, formed by four Openstack-based virtual machines (specifically, a master node with 4 virtual CPUs (vCPUs), 6 GB of RAM, and 70 GB of hard disk space; and three worker nodes with 2 vCPUs, 4 GB of RAM, and 50 GB of hard disk space) and two “far edge” bare-metal nodes (each hosted in an Odroid H2+ board with an Intel Quad-core processor, 32 GB of RAM and 500 GB of hard disk space).

Connectivity to and from the Pledger platform is achieved through dedicated VPN servers at i2CAT’s and ENG’s premises, where UC2’s infrastructure and the core Pledger components are respectively located.

Figure 29 represents the integration between the UC2 RDNS application, running in UC2’s infrastructure, and the Pledger components that are used in this scenario, along with the relevant flows among components. The basic flow of this scenario is explained below.

1. The service provider deploys the RDNS, after configuring the desired SLA, action to be performed by the DSS upon delay violation, and initial resources assigned to the RDNS.
2. The monitoring engine continuously monitors application metrics generated by the RDNS, which are output by UC2’s dedicated Prometheus server.
3. Upon a delay violation, the SLA sends out a violation notification, which is received at the DSS. The DSS then performs a scale up of the application.
4. If additional delay violations occur, the DSS will perform additional scale-ups of resources assigned to the RDNS.
5. If no delay violations are received for a period of time, the DSS gracefully reduces the resources assigned to the RDNS, such that non-required resources are not reserved anymore and could be used by other applications in the same compute chunk, if required.

Document name:	D5.4 Pilots operation and monitoring II	Page:	40 of 53	
Reference:	D5.4	Dissemination:	PU	
	Version:	1.0	Status:	Final

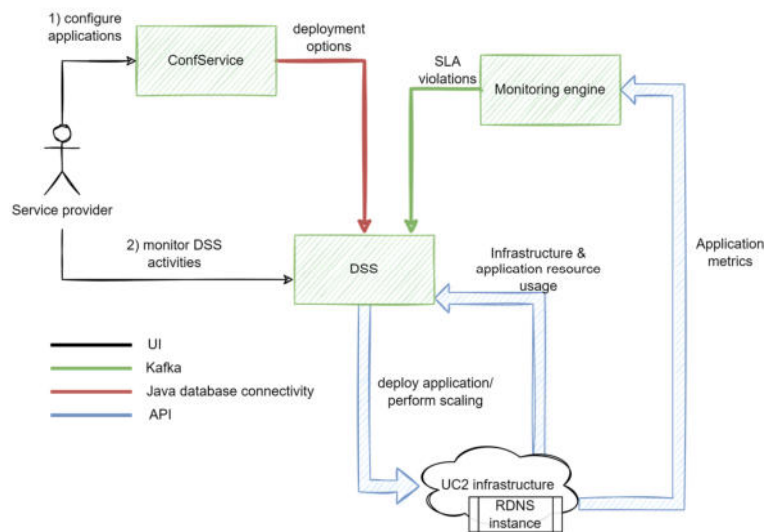


Figure 29: UC2's set-up with SLA and DSS

5.3.2.4 Benefit of using Pledger for the UC

From the service provider's perspective, the RDNS should always run smoothly, without having to monitor the application metrics manually and continuously, and without having to manually perform configuration changes upon varying environmental conditions. In addition, the utilization of available resources should be optimized; therefore, resources allocated to the RDNS should be closely adjusted to those needed at any given time, without incurring in over provisioning. In this sense, the benefits introduced by using the Pledger platform in this specific scenario are three-fold:

- ▶ Continuous, automated monitoring of relevant application metrics, as well as automated notifications related to SLA violations.
- ▶ Automatic scale up of application resources upon delay violations, helping to meet the application delay requirements.
- ▶ Automatic scale down of application resources, allowing for a more efficient resource management.

5.4 Scenarios planned until end of project

5.4.1 End-to-end integration for the deployment of a compute + radio chunk and network service

UC2 is deployed in a city-wide testbed that includes radio resources providing connectivity to vehicular use cases (two IEEE 802.11p radios nodes). Given the infrastructure is supporting multi-tenancy, we expect Pledger to be able to request the creation of a slice for UC2 that includes both radio and computing resources. Going beyond the initial integration done in 5.3.1, where it is shown how Pledger reserves dedicated compute resources and deploys the UC2 service, this scenario includes the reservation of radio resources into a slice deployment. By including these radio resources into a slice, radio access to user equipment (the VRUs in this particular case) is provided. Apart from reserving the radio resources (IEEE 802.11p radio nodes), also the required V2X stack application to enable vehicular communications needs to be deployed, as detailed in D5.1 [4]. In the following a detailed description of the setup and the different steps to be demonstrated are presented.

At radio level, UC2's infrastructure features two radio nodes (Single Board Computers) equipped with IEEE 802.11p transceivers. The radio nodes have direct layer-2 connectivity to the compute nodes of the infrastructure, where computing resources for applications are available as part of the Kubernetes cluster distributed among the far-edge, the edge and the i2CAT cloud. The resources available in this city-wide UC2 infrastructure can be split among different service providers by creating slices that

Document name:	D5.4 Pilots operation and monitoring II	Page:	41 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

contain the required resources. For UC2, radio connectivity forms part of the requirements of the service, so VRUs can connect and receive/transmit data from/to the risk detection and notification system. To reserve the necessary radio, but also the computing resources, both the RAN Controller and the SOE Framework are needed as part of the Pledger platform to reserve a radio chunk and a compute chunk, respectively. The end-to-end integration with the SOE framework, as detailed in Section 5.3.1, has already been demonstrated and showcased how a compute chunk is reserved and a service is deployed in it.

This planned scenario will additionally feature the reservation of radio resources and the activation of the radio service, i.e., the steps necessary to provide radio access connectivity. Similarly, like in the scenario explained in Section 5.3.1, the ConfService will be able to request the reservation of the resources and then the deployment of the application. Part of this request that is handed over to E2CO will include the desired radio resources to be used in the deployment. Again, E2CO will hand this request to the SOE Framework, where the request will be translated into the necessary operations (resource reservation, slice creation), and this time radio resources will also form part of the request. To reserve the radio resources, SOE will communicate with the RAN controller over a RESTful API to request the reservation of the IEEE 802.11p interfaces for the slice with a legal set of radio configuration parameters. The RAN controller will receive this request and will issue the configuration of the physical interface via the NETCONF protocol [21] and will then proceed to configuring the data plane in the radio devices so that traffic from and to the radio is redirected to the network to connect with the compute nodes.

In addition to this base configuration of the radio devices, for vehicular communications, the deployment of the V2X stack is required. The V2X stack has been completely virtualized so that it can be deployed in the compute nodes, just like an application service. The difference to the RDNS application service is that the V2X stack deployment is treated as a part of the slice configuration and is completely handled by the SOE Framework and the RAN Controller. Once deployed, the V2X stack forms part of the infrastructure and is left unchanged until the slice is deleted.

Integration work will focus on being able to issue the deployment of the necessary modules of the V2X stack as part of the end-to-end service creation flow, so that the entire radio configuration happens upon request, just like it is already done for the compute nodes.

5.4.2 Integration with the DLT for the secure management of application data

The RDNS processes sensitive data related to the position of application users at given times. This data, due to its nature, should be securely stored, such that it cannot be accessed by non-authorised parties. This can be achieved by using Pledger’s DLT module: application logs are sent to the DLT when generated, where they are securely stored and inaccessible to malicious users. This functionality requires of the following integration steps:

- ▶ Creation of a Kafka producer which will send application logs to Pledger’s Kafka server in a secure manner, under the “vru_positions” topic.
- ▶ Creation of a Kafka consumer on the DLT side, which will subscribe to all messages under the topic “vru_positions”.
- ▶ Implementation of an additional logic to store in the DLT the application logs received through Kafka.

Document name:	D5.4 Pilots operation and monitoring II			Page:	42 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

6 Use Case 3: Manufacturing the data mining on the edge

In UC3 “Manufacturing the data mining on the edge”, analytical services are developed and deployed to determine the process stability of the machine SYNCROMILL. These services analyze the media consumption (air and energy) of the machine as well as the parts produced by the machine in terms of cycle time and quality. Furthermore, another service to determine the thermal stability and to automatize the warm-up process of the machine is under development at the time of writing. These modules are integrated into Cybernetics Analyze, which is FILL’s Industrial Internet-of-Things solution for collection and processing of data digitizing the entire production process.

The services are deployed on an edge device (nerve-host), which is installed directly in the control cabinet of the machine with Docker Engine installed. To extend the computational resources with some cloud resources, the Kubernetes cluster provided by ENG will be utilized. This set-up and the connectivity to the Pledger core system is illustrated in Figure 30.

In this iteration of the pilot execution, the focus was set on performing an offloading scenario using the different functionalities of Pledger. In Section 6.2.3, the scenario for offloading the media-consumption service, more precisely, the air-consumption app will be offloaded to the cloud in case resources on the edge infrastructure get limited.

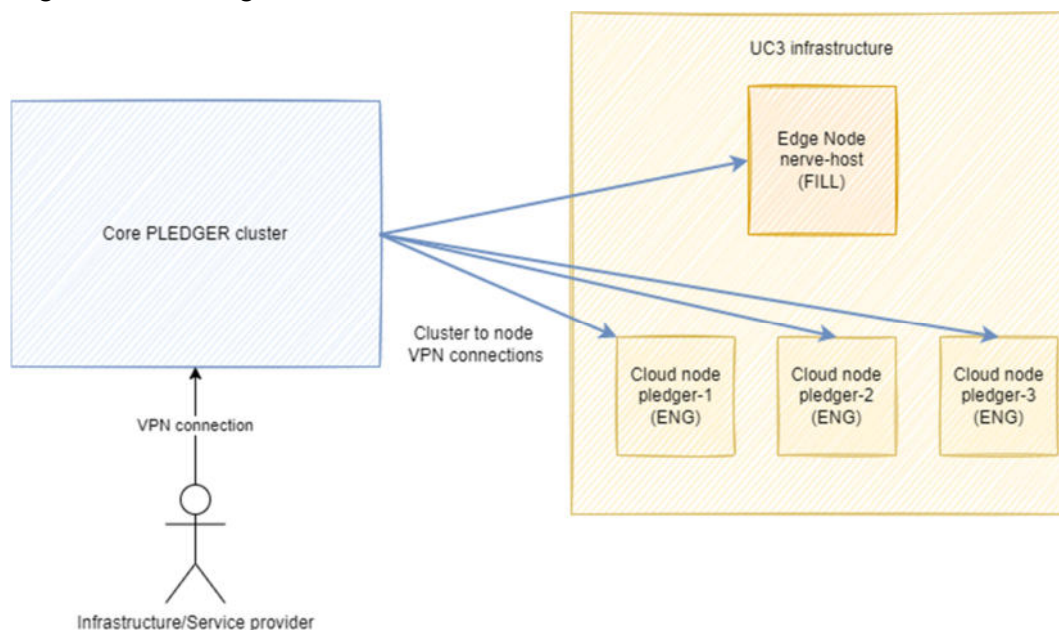


Figure 30: UC3 set-up and connectivity with Pledger's core infrastructure

6.1 Integration with Pledger

The UC benefits from the following Pledger functionalities:

- ▶ Infrastructure management and service orchestration
- ▶ Monitoring performance, QoS and SLAs
- ▶ Infrastructure benchmarking and application profiling
- ▶ Secure management of sensitive data

Document name:	D5.4 Pilots operation and monitoring II	Page:	43 of 53
Reference:	D5.4	Dissemination:	PU
	Version:	1.0	Status:
			Final

To make use of the components and tools provided by Pledger, configurations of the infrastructure, the application and the SLAs need to be entered in the ConfService. The details for the UC3 are given in Section 6.3.1.2 describing the configurations for the performed scenario using Pledger for offloading a service to the cloud.

To monitor infrastructure and service-related metrics, integration with the Monitoring Engine has been performed. This integration end point has already been established via extracting the metrics using a dedicated microservice for this purpose and pushing them to a dedicated topic on the StreamHandler component, as described in the deliverable D5.3 [1].

Integration with the Orchestrator was performed by establishing the connectivity using an OpenVPN connection and opening the docker port of the edge node. However, during the testing the service orchestration of the media-consumption app between edge and cloud, some more requirements arose to enable specifically the offload of the application to the Kubernetes cluster. The application requires data from a message broker, where on the edge an instance of RabbitMQ and in the cloud the StreamHandler based on Kafka is utilized for the data transfer. Furthermore, on the edge Docker is used, whereas in the cloud the container runs on Kubernetes. For smooth execution of the application, configuration files were established to configure the data source (RabbitMQ [22], StreamHandler) for the different environments. These files were packed together with the other parts of the application in one Docker container. In this context, the Orchestrator was extended to also consider environment variables for the deployment of the service. These environment variables point to the location of the configuration file in the container and are configured for Kubernetes (env) and Docker (env_docker) each. This configuration is shown in Figure 31.

To benefit from the secure management of sensitive data, integration with the DLT is required. The analysis results of parts produced by the processing machine are considered sensitive and will be stored on the DLT to provide access only to authorized parties. This will be performed in the next iteration of the pilot, described in Section 6.4.1.

Table 3 gives an update of the table presented in the deliverable D5.3, showing the status of the different integration endpoints, following the pattern as described in Section 4.1.3. The UC applications for the identifier are Edge Server – 1 and Basic Analytics – 2.

Table 3: Status integration endpoints UC3

Integration Endpoint	Components	Responsible	Data Type, Protocol	Publisher/Subscriber	Status M24	Status M30
3.1.1	Orchestrator – Edge Server	ATOS, FILL	REST API, OpenVPN TCP	N/A	Done	Done
3.2.2	DLT – Basic Analytics	INNOV, FILL	JSON, Kafka via StreamHandler	N/A	Not started	In progress
3.3.1	Saas/Iaas Monitoring Engine – Edge Server	ATOS, FILL	JSON, Kafka via StreamHandler	Topic: ▶ UC-metrics	In progress	Done
3.3.2	Saas/Iaas Monitoring Engine – Basic Analytics	ATOS, FILL	JSON, Kafka via StreamHandler	Topic: ▶ UC-metrics	In progress	Done
3.6.2	StreamHandler – Basic Analytics	INTRA, FILL	JSON, Kafka via StreamHandler	Topics: ▶ fill-data-fast ▶ fill-data-result ▶ fill-data-timestamp	Done	Done

ConfService DSS 12.8.0 Home Providers Infrastructures Apps DSS Account

PLEDGER

Create or edit a Service

ID: 4

Name: air-consumption

Profile: sysbench-cpu-cpu-10k:0

Priority (1 is the lowest): 1

Initial Configuration: {"max_memory_mb":500,"min_memory_mb":100,"min_cpu_millicore":100,"scaling":vertical

Runtime Configuration: 0

Deploy Type: DOCKER

Deploy Descriptor

```

- name: FAST_VARIABLE_CONFIG
  value:
- name: SLOW_VARIABLE_CONFIG
  value:
- name: MACHINE_SIGNAL_CONFIG
  value:
- name: CYBERNETICS_CONFIG
  value:
- name: period_sec
  value:
- name: n_packages
  value:
resources:
  limits:
    cpu: PLACEHOLDER_CPU_MILLCOREm
    memory: PLACEHOLDER_MEMORY_MBMI
  requests:
    cpu: PLACEHOLDER_CPU_MILLCOREm
    memory: PLACEHOLDER_MEMORY_MBMI
env_docker:
- name: FAST_CONFIG
  value:
- name: SLOW_CONFIG
  value:
- name: FAST_VARIABLE_CONFIG
  value:
- name: SLOW_VARIABLE_CONFIG
  value:
- name: MACHINE_SIGNAL_CONFIG
  value:

```

Status: STOPPED

App: media-consumption

Figure 31: Configuration for service deployment and environment variables for UC3

Document name:	D5.4 Pilots operation and monitoring II	Page:	45 of 53
Reference:	D5.4 Dissemination: PU	Version:	1.0 Status: Final

6.2 Metrics and Pilot KPIs

During the set-up and the first pilot iteration, metrics serving different the following purposes have been discussed and implemented:

- ▶ Metrics to monitor the functionality of the UC applications
- ▶ Metrics to monitor the performance of the UC applications
- ▶ Metrics/KPIs to determine the performance of the UC

The different metrics in these categories, their intention and purpose as well as their implementation is described in the following subsections, as well as how Pledger can improve the performance of the UC.

6.2.1 Metrics to monitor the functionality of the UC applications

All UC applications communicate with an instance of RabbitMQ to gather the data needed to perform the specific analytical task using a dedicated queue for every application. Monitoring the queue metrics [23] provided by RabbitMQ gives the service providers the possibility to monitor the functionality of the UC and enables them to narrow the problem in case issues arise.

During the pilot execution, the metrics “Message publishing rate” and “Message delivery rate” have been identified to estimate the correct retrieval and delivery of the data.

Furthermore, the metric “Number of messages ready for delivery” gives the service provide the possibility to determine, whether the application is running and actively consuming messages. This behavior is also reflected in the metric “Message delivery rate”. Using this metric, an SLA can be specified to define the service availability on the edge.

These metrics are applicable to all applications in UC3.

6.2.2 Metrics to monitor the performance of the UC applications

Besides monitoring the functionality of the UC applications, further metrics have been defined to determine the performance of the applications. The metric “calculation_time” is retrieved on application-level and determines the time required for the actual processing of the data and calculation of the desired analytical result including data retrieval and result provisioning. Especially in latency-critical application this is a key metric to determine the performance of the applications and will be considered in the remaining part of the project for the application to determine the thermal stability of the machine.

6.2.3 SLAs/KPIs to determine the performance of the UC

Using the metrics described before, SLAs and KPIs for the UC can be defined, which can be further developed into smart contracts to give guarantee of specified thresholds and automatize the refunding process in case of violations.

These SLAs/KPIs are in the final definition phase, but focus on two metrics: First, the service availability on the edge for the media-consumption app using the “Message delivery rate” and estimating the time, where this metrics remains 0. A value of 0 indicates, the service is not available on the edge and either down due to some technical issues or running in another infrastructure, e.g., in the cloud.

Second, the “calculation_time” will be considered for the thermal-stability app. For this application, “good” performance is defined, if the variance of the metric is low, meaning the results are delivered continuously without any outliers or delay guaranteeing the smooth warm-up process in the shopfloor. The exact thresholds for these metrics will be determined via experiments during the next iteration. This metric also affects the QoE, as a stable calculation time with low variance results in consistent delivery of results leading to satisfying QoE. Therefore, the “calculation_time” will be further developed to better estimate the QoE.

Summarizing, the following metrics will be considered further for evaluation and validation purposes:

Document name:	D5.4 Pilots operation and monitoring II			Page:	46 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

- ▶ “mediaconsumption_availability_on_the_edge”: Determining the availability on the edge for the media-consumption application.
- ▶ “thermalstability_result_availability_time”: Determining the amount of time required to process the data, determining and provisioning the analytical result to determine the thermal stability of the machine.

The naming of these metrics follows the same conventions as for the other UCs.

6.3 Scenarios performed in M24-M30

6.3.1 End-to-End integration with service deployment, monitoring and offloading to the cloud in case of an SLA violation

6.3.1.1 Motivation

The UC3 has strong dependence on the edge. Applications analyzing parts produced by the machine contain sensitive data, which should not leave the edge. Furthermore, determining the thermal stability has latency requirements in terms of few ms, as the machine needs to react on the result, requiring the application to be running on the edge. Furthermore, this application has varying needs in resources, depending on the actual process in the shopfloor resulting in time phases with higher loads to be prioritized. The media-consumption application is prioritized on the edge to make results available in shorter time. However, in case of limited resources, it can be offloaded to the cloud to guarantee the QoS of all applications, including the thermal stability application and parts-produced app.

This scenario demonstrates the end-to-end integration of the UC3 with the media-consumption app and the offload to the cloud in case resources get limited. In this iteration, the stressing of the infrastructure is done manually due to the current implementation status of the thermal stability application. In the final pilot, all applications will be available and will replace the manual stressing with actual resource consumption.

6.3.1.2 Configuration

The configuration of this scenario consists of three steps, which are directly implemented in the ConfService:

1. Configuration of the Service to be orchestrated
2. Configuration of the App deployment options
3. Configuration of the associated SLA

The configuration of the service to be orchestrated includes besides some basic information about the naming, the link to the docker image, initial memory and computation limits and the environment variables as described in Section 6.1.

The App deployment options are configured using ranked options, with two rankings. First ranked, is the edge node of the UC3 infrastructure. Second ranked are the available cloud resources, where three nodes are provided by partner ENG and two by partner i2CAT. Therefore, the orchestrator deploys the media-consumption first on the edge node and in case of an SLA violation, a node from the second ranking will be used to deploy the application based on the recommendations of the DSS. These deployment options are shown in Figure 32

The SLAs are configured using the associated metric in the Prometheus instance of the Monitoring Engine and defining thresholds with different severities, including warning and catastrophic. The values of these thresholds are shown in Figure 33.

Document name:	D5.4 Pilots operation and monitoring II			Page:	47 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

App Deployment Options 4

Ranked options (lower values are better)

```

app: media-consumption

service: air-consumption
- ranking 1 : service deployed on
  -- node#9 'nerve-host' [edge] on infrastructure 'UC3-edge' with id:4
- ranking 2 : service deployed on
  -- node#11 'pledger-1' [cloud] on infrastructure 'ENG-infra' with id:5
  -- node#12 'pledger-2' [cloud] on infrastructure 'ENG-infra' with id:5
  -- node#13 'pledger-3' [cloud] on infrastructure 'ENG-infra' with id:5
  -- node#35 'kubenode1' [cloud] on infrastructure 'UC2-infra-30' with id:8
  -- node#36 'kubenode2' [cloud] on infrastructure 'UC2-infra-30' with id:8
  
```

App
media-consumption

[← Back](#)

Figure 32: App deployment option for the media-consumption service in UC3

ID	Name	Threshold Warning	Threshold Mild	Threshold Serious	Threshold Severe	Threshold Catastrophic	SLA	Constraint	
35	calculation-time	> 30			> 70		air-consumption - app-responsiveness	click "View" or "Edit" for details	View Edit Delete
37	UC3 e2e guarantee	> 80	> 100	> 120	> 140	> 160	UC3 e2e	click "View" or "Edit" for details	View Edit Delete

Showing 1 - 2 of 2 items.

Figure 33: SLA guarantees of UC3

6.3.1.3 Set-Up and detailed description

The basic set-up of this scenario consists of the running software components of Cybernetics, including the Message Broker, the database for storing the application results and the microservice extracting the infrastructure and application metrics and sending them to the StreamHandler. Furthermore, a Grafana dashboard built on top of the Monitoring Engine's database is implemented to observe the performance of the application as well as for monitoring the infrastructure. Furthermore, the OpenVPN connection between ENG and FILL is established to access the ConfService as well as to enable the Orchestrator to reach the infrastructure.

Document name:	D5.4 Pilots operation and monitoring II	Page:	48 of 53	
Reference:	D5.4	Dissemination:	PU	
	Version:	1.0	Status:	Final

After the login in ConfService as a Service Provider, the service and SLAs are configured as described in Section 6.3.1.2. Afterwards, the application media-consumption is started, and the Orchestrator deploys it on the edge node of UC3 infrastructure.

The application is running, and the resulting analytical results are stored in the database and can be observed using the Cybernetics UI. Furthermore, the infrastructure and the application's performance can be observed using the Grafana dashboard built on top of the Monitoring Engine's database.

In this scenario, the infrastructure is stressed using a small script sending continuous requests to a webserver running on this infrastructure, resulting in CPU load on the edge node, which can also be observed on the Grafana dashboard as shown in Figure 34. The SLA Lite component detects this SLA violation and sends out a notification. The DSS consumes this notification and recommends the offload to the cloud. The decision about which node to use is based on the results of App Profiler and Benchmarking. In this case, the DSS recommends the offload to pledger-node 3, as illustrated in Figure 35. The orchestrator undeploys the application on the edge and deploys it in the cloud.

The stressing of the infrastructure is stopped, resulting in decreasing load on the edge device. After two minutes without any SLA violation, the DSS triggers the offload back to the edge node. The orchestrator undeploys the container in the cloud and deploys it again on the edge.



Figure 34: Grafana dashboard of UC3 for monitoring the infrastructure and performance of the service

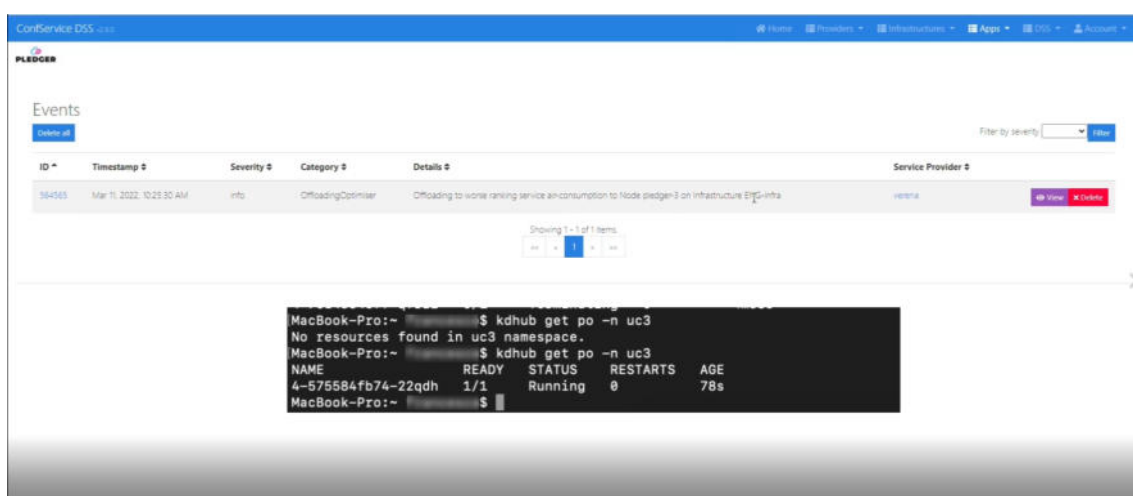


Figure 35: The DSS recommends the offload to the cloud. The terminal shows the application running in the cloud

6.3.1.4 Benefit of using Pledger for the scenario

The computational resources needed for the different applications depend highly on the running process in the shopfloor. Depending on the production mode, more parts are produced to be analyzed and the thermal stability needs to be determined requiring dedicated amounts of resources. In case these two services need almost all available resources on the edge, the media-consumption app gets limited resources to perform its task leading to delayed analytical results – or in worst case even no result at all - and therefore decreasing the QoE of the end user. By offloading to the cloud, the app gets sufficient resources to perform its task and providing the periodical analytical results in stable frequency. Using Pledger, the SLA violations can be detected and the process of service orchestration to the suitable infrastructure can be automatized leading to stable performance of the different applications increasing the QoE of the end user.

6.4 Scenarios planned until end of project

6.4.1 Integration with the DLT for the secure management of sensitive information about parts produced

The parts-produced application derives sensitive data related to the number of parts produced by the specific machine as well as their status about the quality. This data allows to draw conclusions about the workload of the factory as well as the scrap rate, which is considered as a sensitive information about the business. This data should be securely stored, giving access to only authorized parties, e.g. the service provider and the end user (i.e. the customer). Using the Pledger DLT module, the results of the application are sent as JavaScript Object Notation (JSON) to the DLT when generated and securely stored. This functionality requires of the following integration steps:

- ▶ Extending the already existing Broker-to-StreamHandler module (described in Deliverable D5.3) sending the analytical results to the StreamHandler to a dedicated “uc3-dlt” topic in a secure manner.
- ▶ Creation of a consumer on the DLT side, which will subscribe to all messages under this topic.
- ▶ Implementation of an additional logic to store in the DLT the messages received through StreamHandler.
- ▶ Implementation of an additional logic to retrieve the information about the total amount of parts produced as well as the number of good parts and scrap parts.

6.4.2 Smart contracts

In the remaining phase of the project, the use of smart contracts in the UC will be explored. For this purpose, the SLAs defined in Section 6.2.3 will give the basis to implement these contracts using Pledger components. At the time of writing, the involved parties (e.g. service provider, infrastructure provider) are under definition as well as the frequency of evaluation.

The SLAs will be defined in the SLA Lite module of Pledger adding the necessary information in the ConfService. This process will be very similar to the ones described in Section 6.3.1.2 when configuring the SLAs for the offload scenario.

Document name:	D5.4 Pilots operation and monitoring II			Page:	50 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

7 Conclusions

In this deliverable, the work performed in the second iteration of the task T5.3 “Pilots operation and monitoring” was performed. All UCs demonstrated experimentation scenarios related to service orchestration and performance optimization by scaling resources or offloading applications to the cloud.

UC1 implemented a scenario launching a VM to start the PledgAR Workspace application and loading CAD files to work with. The file size is directly correlated with the loading time and the required resources needed. Using the loading time as a measure for QoS and QoE, an SLA can be defined to increase the VM’s resources in case loading times exceed a certain threshold.

Due to the current integration status of the E2CO and the UC application, a manual launch of the PledgAR Workspace application was necessary. In the next iteration, this gap will be closed to provide a full end-to-end demonstration.

UC2 implemented two scenarios, one for demonstrating the functionality of the SOE framework by reserving dedicated computing and network resources before deploying the actual application.

The end-to-end delay metric increases with limited resources and when exceeding a certain threshold, alert notifications might not reach the VRUs in time for preventing a risky situation. Setting an SLA on this metric and increasing computational resources in case of its violation represented the second scenario.

Both scenarios have been performed separately, because UC2's application requires a radio chunk reservation and configuration, which is not yet fully automatized in the current implementation stage of Pledger’s SOE and RAN controller; therefore, the scaling capabilities were demonstrated over a statically configured network slice featuring the required compute and radio resources. However, during the final pilot, it will be possible to deploy the entire network slice with an instance of UC2's application running on it, and then apply scaling and/or application over the application instance running on top of that slice.

UC3 presented an end-to-end scenario for deploying a UC application, monitoring its performance, and offloading the application to the cloud, to give priority to other applications on the edge in case they need more resources for a certain amount of time. In this iteration, the stressing of the infrastructure was performed manually. With the finalization of the UC application development in Task T5.1, also these other, prioritized applications can be included in the scenario completing the overall picture of the UC and the manual stressing will be replaced with running the further applications.

The further developments for the UCs to extend the scenarios are clarified – implementing the remaining integration end points and completing the end-to-end scenarios. Furthermore, emphases must be set on measuring QoE and sharpening the pilot KPIs to advance the evaluation and validation work during Task T5.4 “Validation and overall evaluation”.

Document name:	D5.4 Pilots operation and monitoring II			Page:	51 of 53
Reference:	D5.4	Dissemination:	PU	Version:	1.0
				Status:	Final

8 References

- [1] PLEDGER. D5.3 – Pilots operation and monitoring I. Stanzl, Verena. 2021. <http://pledger-project.eu/content/deliverables>. Accessed 9 May 2022.
- [2] Kubernetes home page. <https://kubernetes.io/>. Accessed 9 May 2022.
- [3] PLEDGER. D5.2 – Pledger integrated demonstrator I. Sarris, Ioannis. 2021. <http://www.Pledger-project.eu/content/deliverables>. Accessed 9 May 2022.
- [4] PLEDGER. D5.1 – Pledger Applications for the use cases. Betzler, August. 2021 <http://www.Pledger-project.eu/content/deliverables>. Accessed 9 May 2022.
- [5] OpenVPN home page. <https://www.openvpn.net/>. Accessed 9 May 2022.
- [6] PLEDGER. D2.3 – Pledger Overall Architecture v1.0. Voutyras, Orfefs. 2020. <http://www.Pledger-project.eu/content/deliverables>. [Accessed 9 May 2022]
- [7] ISAR SDK home page. <https://holo-light.com/products/isar-sdk/>. Accessed 9 May 2022.
- [8] HoloLens 2 home page. <https://www.microsoft.com/en-US/hololens/hardware>. Accessed 18 May 2022.
- [9] PLEDGER. D4.1 Trust, Security and Privacy related tools I. Segou, Olga. 2021. <http://www.Pledger-project.eu/content/deliverables>. Accessed 9 May 2022.
- [10] VMWare home page. <https://www.vmware.com/>. Accessed 9 May 2022.
- [11] VirtualBox home page. <https://www.virtualbox.org/>. Accessed 9 May 2022.
- [12] QemuKVM home page. <https://wiki.qemu.org/Features/KVM>. Accessed 9 May 2022.
- [13] Apache Kafka home page. <https://kafka.apache.org/>. Accessed 9 May 2022.
- [14] Confluent Kafka home page. https://docs.confluent.io/platform/current/clients/confluent-kafka-dotnet/_site/api/Confluent.Kafka.html. Accessed 9 May 2022.
- [15] IEEE 802.11p home page. <https://standards.ieee.org/ieee/802.11p/3953/>. Accessed 9 May 2022.
- [16] ETSI TS 102 637-2 V1.2.1. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative - Awareness Basic Service*.
- [17] ETSI TS 101 539-1 V1.1.1. *Intelligent Transport Systems (ITS); V2X Applications; Part 1: Road Hazard Signalling (RHS) Application requirements specification*.
- [18] Prometheus home page. <https://www.prometheus.io/>. Accessed 9 May 2022.
- [19] Papageorgiou, A. et. al. *On 5G network slice modelling: Service-, resource-, or deployment-driven?*, Computer Communications, 149, 232-240. (2020).
- [20] Openstack home page. <https://www.openstack.org/>. Accessed 9 May 2022.

Document name:	D5.4 Pilots operation and monitoring II	Page:	52 of 53	
Reference:	D5.4	Dissemination:	PU	
	Version:	1.0	Status:	Final

- [21] R. Enns and M. Schönwälder and J. Schönwälder and A. Bierman, „Network Configuration Protocol (NETCONF)”, RFC 6241, <https://www.rfc-editor.org/info/rfc4741>. (2011).
- [22] RabbitMQ home page. <https://www.rabbitmq.com/>. Accessed 9 May 2022.
- [23] RabbitMQ queue metrics home page. <https://www.rabbitmq.com/monitoring.html#queue-metrics>. Accessed 9 May 2022.

Document name:	D5.4 Pilots operation and monitoring II			Page:	53 of 53		
Reference:	D5.4	Dissemination:	PU	Version:	1.0	Status:	Final