D5.3 Pilots operations and monitoring I

Document Identification			
Status	Final	**Due Date**	30/11/2021
Version	1.0	**Submission Date**	30/11/2021

Related WP	WP5	**Document Reference**	D5.3
Related Deliverable(s)	D5.1 (Pledger Application for the use cases I), D5.2 (Pledger Integrated Demonstrator I), D2.2 (Pledger Requirements Analysis)	**Dissemination Level (*)**	PU
Lead Participant	FILL	**Lead Author**	Verena Stanzl (FILL)
Contributors	ATOS, ENG, HOLO, i2CAT, ICCS, INNOV, INTRA	**Reviewers**	Francesco Iadanza (ENG)
			Ioannis Sarris, Olga Segou (INTRA)

Keywords:
Pilots Operation, Pilots Monitoring, Metrics

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

Document Information

List of Contributors	
Name	Partner
Verena Stanzl	FILL
August Betzler	i2CAT
Estela Carmona Cejudo	i2CAT
Nour Fendri	HOLO
Carina Pamminger	HOLO
Francesco Iadanza	ENG
Gabriele Giammatteo	ENG
Alexandros Psychas	ICCS
Nikos Kapsoulis	INNOV
Antonio Castillo Nieto	ATOS

Document History			
Version	Date	Change editors	Changes
0.0	29/09/2021	FILL	Initial TOC
0.1	08/10/2021	FILL	First version with UC3 as example in Section 2 and 4
0.2	22/10/2021	FILL, ATOS, ENG	Updated version with inputs for Section 3
0.3	10/11/2021	FILL, i2CAT, HOLO	Integrated inputs from HOLO and i2CAT in Section 2, 3 and 4; Updated Section 1. Restructured Section 5 and 6.
0.4	12/11/2021	FILL, ICCS, ENG	Finalized Section 5 and 6. Included Executive Summary and Conclusion.
0.5	12/11/2021	FILL	Version ready for internal review
0.6	17/11/2021	ENG	Reviewed version from ENG
0.7	18/11/2021	INTRA	Reviewed version from INTRA
0.8	22/11/2021	FILL	General feedback integrated, Feedback for UC3 integrated, extended executive summary and conclusion
0.9	25/11/2021	FILL, i2CAT	Updates for UC2 integrated
0.10	29/11/2021	FILL, HOLO	Updates for UC1 integrated, updated Reference section, aligned list formatting, updated acronyms
0.11	30/11/2021	FILL	Version for Quality Assurance
0.12	30/11/2021	ATOS	Quality Assurance Review
1.0	30/11/2021	ATOS	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Verena Stanzl (FILL)	30/11/2021
Quality manager	Carmen San Roman Alonso (ATOS)	30/11/2021
Project Coordinator	Lara Lopez Muñiz (ATOS)	30/11/2021

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
API	Application Programming Interface
AR	Augmented Reality
CAD	Computer-Aided Design
CPU	Central Processing Unit
DCC	Decentralized Congestion Control
DLT	Distributed Ledger Technology
DSS	Decision Support System
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
ECODA	Edge to Cloud Offloading Decision Algorithm
FPS	Frames Per Second
GB	GigaByte
GPU	Graphics Processing Unit
HMD	Head Mounted Device
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
KPI	Key Performance Indicator
MQTT	Message Queuing Telemetry Transfer
MR	Mixed Reality
Mx	Month x of the project
NETCONF	Network Configuration Protocol
OS	Operating System
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RDNS	Risk Detection and Notification System
RSU	Road Side Unit
SLA	Service Level Agreement
SOE	Slicing and Orchestration Engine
SSD	Solid State Drive
UC	Use Case
UI	User Interface
V2X	Vehicle-to-everything
vCPU	Virtual CPU
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
VRU	Vulnerable Road Users

Abbreviation / acronym	Description
WP	Work Package
XR	Extended Reality
Tx.y	Task number y belonging to WP x

Executive Summary

This deliverable presents the initial work performed in task T5.3 “Pilots operation and monitoring” and documents the first of three iterations. In this iteration, the set-up of the pilots to perform the first execution of pilots has been performed in terms of infrastructure and integration with the Pledger system. It describes the infrastructure used in the pilots for executing the use cases and its set-up.

The second part deals with the integration of the pilots with the Pledger system. To enable the communication with the system as well as the launch of applications, network connectivity has been established. Furthermore, the integration with the different components (Orchestrator, Pledger Distributed Ledger Technology (DLT), Slice Orchestrator Engine (SOE), Radio Network (RAN) Controller, Software as a Service and Infrastructure as a Service (SaaS/IaaS) Monitoring Engine and StreamHandler Platform) is described and an overview and status of these integration end points is given.

Application metrics are crucial in this project to monitor the Quality of Service (QoS) and detect possible violations of the defind Service Level Agreements (SLA). Therefore, application-relevant metrics have been identified for each pilot. The focus was on defining metrics, which are dependent on the computational resource usage, those which represent general issues (e.g., a major network failure) where resource management would not improve the QoS and requires further analysis and intervention to fix, and SLA which are not considered for resource management and used for other purposes, like monitoring or smart contracts.

Furthermore, the first scenarios to be performed in this pilot operation phase are described. These include workflows for IaaS and SaaS Providers as well as first scenarios for the pilots to increase computational resources in case SLA violations are identified.

Lastly, the requirements defined in an earlier phase of the project have been reviewed and updated as well as extended with any additional requirements encountered during this iteration of the task. The overview of changed and new requirements is given in the last section. Finally, the link is established between the scenarios defined and the requirements validation for task T5.4 “Validation and Evaluation”.

1 Introduction

1.1 Purpose of the document

In this deliverable, the initial work performed in T5.3 "Pilots operation and monitoring" is described. This 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. The next iterations will be reported in D5.4 in M30 and D5.7 in M36.

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 first part of the pilot set-up involves the setting up of the infrastructure used for Pledger. This varies in difficulty and time depending on the environmental conditions and requirements and is the responsibility of the individual use case leaders.

The next step is to integrate the Pledger system into the individual use cases to perform the first phase of pilot runs. This step was done in close coordination with all technical partners of the project. There are parts of the integration that were performed the same way for all use cases, while others required more flexible solutions to meet the different requirements. In this document, these commonalities and differences are outlined.

Metrics were defined for the proper execution of the pilots to describe the Quality of Service (QoS) and to be used for optimization by Pledger. These are different for each use case and are used to prove the versatility of the platform to respond to different circumstances as the project progresses. As a final step, the first experimentation scenarios are defined. With their help, the requirements defined in the early phase of the project, were validated and new ones identified.

1.2 Relation to other project work

This task focuses on the pilots' operation and monitoring to validate the Pledger use cases, which have been developed and defined WP2 and the requirements to be validated were defined in T2.2. To ensure a smooth pilot operation, pilots' specific applications have been developed and described in D5.1 and will be utilized during the pilot runs to demonstrate the functionalities of the platform validate the effectiveness of the approach. The integration of these applications with the Pledger core components (developed in the dedicated tasks of WP3 and WP4) has been defined in the UC-specific architectures in D2.3. The evaluation procedure will be developed under the responsibility of T5.4.

1.3 Structure of the document

This document is structured in 6 chapters.

- **Chapter 2** gives an overview of the pilots and presents the set-up of the infrastructure.
- **Chapter 3** presents the integration of the Pledger system in the Use Cases.
- **Chapter 4** describes the metrics observed and used in the pilots to measure QoS.
- **Chapter 5** recaps the role of stakeholders in the project and presents different workflows for them. Furthermore, the first experimentation scenarios to be performed during the first pilot run are outlined.
- **Chapter 6** presents the methodology in terms of validation of requirements. For this purpose, the existing requirements have been reviewed and updated requirements are presented. Furthermore, the next steps for extending this methodology are presented.

2 Pilots Overview

This deliverable describes the set-up of the pilots in terms of preparation for the on-site preparation of the demonstrators' deployment. In this section, a summary describing the aim of the pilot and how it benefits from using the Pledger system is given. Furthermore, the infrastructure to be used to operate the pilots has been set-up, which is also described in the following.

2.1 UC1: Mixed Reality applications on the edge

2.1.1 Summary

Mixed reality applications are getting a significant momentum in a wide spectrum of business cases including on-spot training, building information modelling, inspection and quality control, Computer-Aded Design (CAD) visualisation, as well as other commercial applications [1]. One of the most common requests of industrial clients is the visualisation of 3D content. Whether it is for design review, prototyping, visual support during complicated work processes, or simply for optimised communication and collaboration purposes, visualisation is an essential part of Mixed and Augmented Reality (MR/AR) solutions.

The goal of this use case is to enhance the capabilities of Extended Reality (XR) solutions by coupling them with edge computing technologies and the corresponding load allocation and optimisation tools, to provide high level services in industrial environments. More specifically, the industrial environment which will run the pilots of this use case is the one provided by FILL in UC3.

2.1.2 Infrastructure set-up

The UC1 PledgAR Workspace Application tackles different sub use cases:

- Prototyping: facilitates for the user to visualise and interact with CAD files.
- Training: enhances trainees learning experience through step-by-step AR tutorials.
- Collaboration: allows remote users to connect to the same session and collaborate on projects in an AR workspace.

As mentioned before, the Head-Mounted Displays (HMDs) are lacking processing power and rendering capabilities. Therefore, the user would have to sacrifice quality to keep a stable Frames Per Second (FPS) to run the application smoothly.

To tackle this issue PledgAR Workspace incorporates Holo-Light's ISAR technology which splits the application into a 2-component solution, oneone is running on an edge device and the other one on the smart glasses. This solution outsources the application rendering onto the powerful edge device and uses the smart glasses as viewing and input device.

Another limitation is the setup, maintenance, support, and cost control of applications running on the HoloLens 2. Using ISAR and the additional ability to run entire HoloLens applications on Windows machines allow the integration into the Pledger Framework. Using an Orchestrator for the VM image setup – which normally requires IT helpdesk assistance – reduces time and costs for users. Furthermore, maintenance can be streamlined through the VM image control, which also makes support easier. Another way to save costs is through the actual usage of VM resources and flexible resource management, whereas data security is increased through the link to the DLT.

Running on the VM, the PledgAR Workspace application will establish a connection between itself and the HMD. To ensure the secure data transmission, the DLT is protecting the relevant information needed for the peer-to-peer connections handshake.

This solution allows to mediate all previously mentioned pain points and enables the user to enjoy a great app experience while working with resource-heavy 3D Models.

The specifications of the edge device used for the remote rendering are:

- Graphical Processing Unit (GPU): GTX 10X0 min 5 GigaByte (GB) or higher.
- Operating System (OS): Windows 10 min. 10.0.17763 Build 17763.
- Central Processing Unit (CPU): Intel i7 or higher.
- Random Access Memory (RAM): 16 GB or higher.

2.2 UC2: Edge infrastructure for enhancing safety vulnerable road users

2.2.1 Summary

Deployed in Barcelona, UC2 explores mechanisms to enhance the safety of vulnerable road users (VRUs) in street layouts where risky situations can emerge due to the concurrency of train rails, pedestrian walks, and bicycle lanes. The VRUs on which UC2 focuses are pedestrians and users of bicycles (or electric scooters) that are transiting a tram station. Figure 1 shows the scenario on top of which the UC is built. The bicycles transiting this area use a bicycle lane that separates the tram lane from the pedestrian lane. This implies that bicycles are forced to cross an area that is potentially used by pedestrians that are entering or leaving a tram. Considering that the tram station can potentially block the sight, the risk gets even bigger.

To solve the issue, obtaining the locations of both VRUs and the tram, an intelligence that processes this information and detects possible risks, and a notification mechanism need to be implemented. This solution imposes certain requirements, like small end-to-end delays, assuring computing resources for the risk detection and notifications application and securing the generated data from unwanted or untrusted access. The Pledger Core system implements the necessary features to satisfy these requirements: Intelligent application placemenet, as well as its scaling and migration in case of computing resources shortage, as well as a DLT to assure that only trusted users can access the information generated in the application (events and traces).

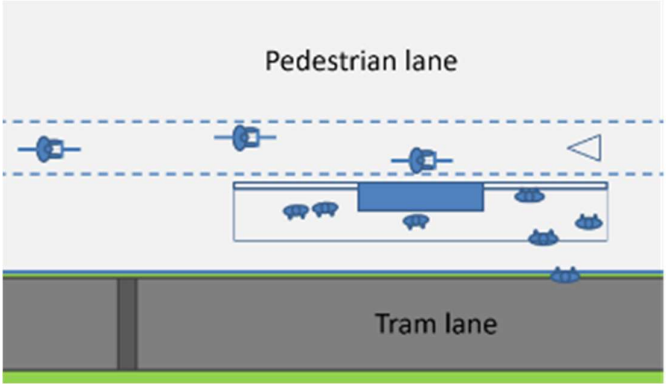

Figure 1: UC2 Scenario Layout

2.2.2 Infrastructure set-up

UC2 is being deployed in the city of Barcelona and consists of a three-tier compute infrastructure with two radio Road Side Units (RSUs) elements, to which several radio on-board units (OBUs) can connect. UC2's infrastructure comprises:

- A main data centre, formed by several computing nodes, and hosted at i2CAT, near "Zona Universitaria". Any non-latency-constrained services requiring heavy processing or large amounts of memory or hard disk space can be executed in the main data centre. In summary, the main data centre will host any services that do not need to run at the edge or far-edge, where less resources are available. The main data centre architecture is comprised of the following elements:

- A master Kubernetes node, hosted in an Openstack virtual machine with 4 virtual CPUs (vCPUs), 6 GB of RAM, and 70 GB of hard disk space.
- Two Kubernetes worker nodes, each hosted in an Openstack virtual machine with 2 vCPUs, 4 GB of RAM, and 50 GB of hard disk space.
- A VPN server, hosted in an Openstack virtual machine with 2 vCPUs, 4 GB of RAM, and 40 GB of hard disk space. This element can provide external access to the infrastructure through a public Internet Protocol (IP) adress.

- An edge compute tier, located in the so called 22@ area [2], also called the innovation district, in the East of the city. The edge compute node is more constrained than the main data centre in terms of CPU, RAM and storage resources. However, using edge resources brings large benefits in terms of (i) reduced transmission latency, essential for certain parts of the service, such as the processing of the data gathered from the VRUs on-street to detect situations of risk, and (ii) reduced energy usage, by reducing the energy consumption needed for transmitting the data to the main data centre. The edge compute tier is formed by:
  - A worker Kubernetes node, hosted in an Openstack virtual machine with 2 vCPUS, 4 GB of RAM, and 50 GB of hard disk space.
  - Far-edge compute nodes, located on lampposts by the tram stop that will be used for validating the UC. These nodes are formed by single-board computers with very high CPU, RAM and storage resource constraints. Therefore, these nodes will be used only for hosting radio technologies critical for the communication with the VRUs. The far-edge tier is formed by:
    - Two bare-metal Kubernetes 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.
    - Two RSUs, mounted on the same lampposts as the far-edge compute nodes. The RSUs provide IEEE 802.11p connectivity to UC2 users equipped with OBUs, such as bicycles, and they are connected to the far-edge compute nodes through dedicated Virtual Local Area Networks (VLANs).
    - Several OBUs mounted on bicycles and electric scooters that provide UC2 users with Institute of Electrical and Electronis Engineers (IEEE) 802.11p connectivity to the RSUs.

An overview of the provided infrastructure is given in Figure 2.

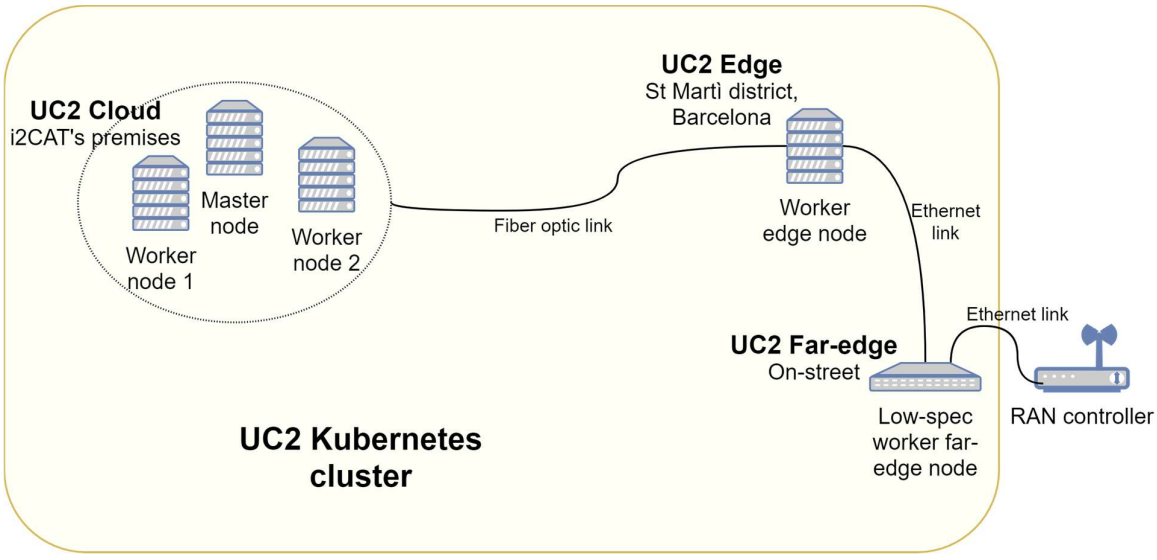

Figure 2: Infrastructure in UC2

2.3 UC3: Manufacturing the data mining on edge

2.3.1 Summary

FILL Cybernetics is the next evolutionary step of digital solutions, adding great value for FILL's customers to their machines. It is an Industrial Internet of Things platform focusing on the transformation of data to smart data and the integration of these smart data in the new smart digital engineering process. Its components are hosted on an edge device to digitize the entire production process. By collecting and recording the data together with analytics services the condition of a machine can be determined, and a complete overview of the production process is given. This enables the customers using Cybernetics to increase the profitability and production quality of a factory.
To conduct the UC, Cybernetics has been extended with analytical services to monitor the running process and to determine the process stability of the machine. These analytical services gather the data from the central message broker on the edge, produce the desired result and publish it on the message broker, from where it gets stored in a database. FILL Cybernetic's backend can then access these results and visualize them to the end user on the User Interface (UI).
These services have been described in detail in deliverable D5.1 [3].
With the help of Pledger system, Cybernetics can extend its limited computational ressources on the edge with additional ressources from the cloud to increase its services' QoS. These metrics will be defined in Section 4.

2.3.2 Infrastructure Set-Up

UC3 is hosted in FILL's own premises, the FILL Future Zone, where the standard machine tool SYNCROMILL is set-up for conducting the UC. An Industrial Personal Computer PC (IPC) is directly built into the machine providing one edge node with following ressources:

- CPU: Core i5-6442EQ
- 3x Gigabit Ethernet
- RAM: 16GB
- Solid State Drive (SSD): 480GB
- OS: Linux

Cybernetics and Docker Engine are installed and ready to host the analytical services developed in the UC. To extend the computational ressources with some cloud ressources, the Kubernetes cluster provided by ENG will be utitlized. This infrastructure is described in Section 3.1.1.

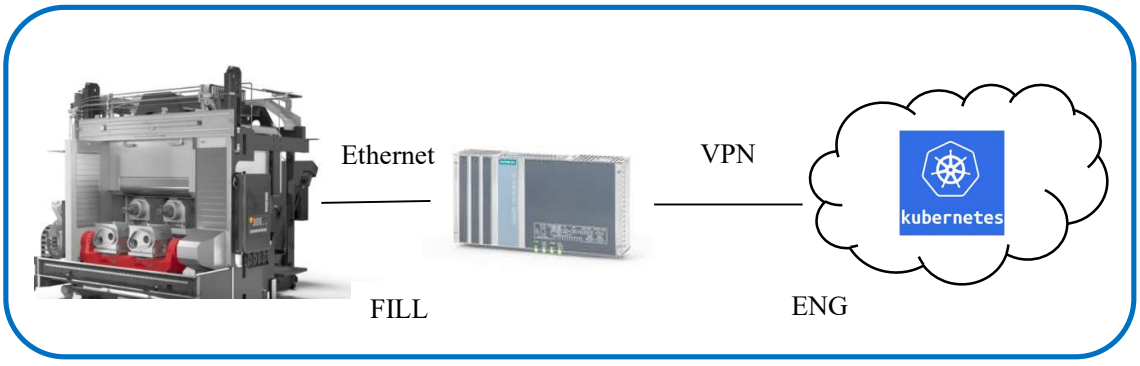

Figure 3: Illustration of the standard machine tool SYNCROMILL and the Industrial PC

3 Integration of the UCs with the Pledger system

To achieve the operational readiness of the UCs and to enable them to use the platform, some configurations must be done as well as some components need to be integrated into the UCs' environment.

First, the network connectivity must be established to enable the execution of applications and interactions with the Pledger core components. Furthermore, the service providers of application UCs need to provide metrics of their applications to enable the platform to monitor the QoS and take decisions accordingly. These aspects are general to all UCs are described along with the infrastructure used for hosting core components as well as extending computational resources for individual UCs. Furthermore, the integration of Pledger core components and UC components is presented. The following description is one way to achieve the connection to the platform and was chosen by the UC leaders in coordination with technical partners according to their requirements and expertise in different technologies.

3.1 General aspects

3.1.1 Connectivity

Besides the individual infrastructures of the UCs, a central infrastructure for Pledger is provided by ENG to host most of Pledger core components as well as to support all UCs in case they need to offload computation to the cloud using Linux containers. The proposed infrastructure comprises a dedicated Kubernetes [4] based cluster, currently made of four nodes, one master and three workers, which can be extended with additional worker nodes when needed. Such infrastructure is provided by ENG using its Fiware [5] node, an enhanced Openstack [6] based data centre located in Vicenza, Italy.

In the end, computation in Pledger can rely on all the UC and ENG cloud infrastructures, interconnected through Virtual Private Networks (VPNs) with OpenVPN [7] to support computation migration on different locations. Some of UC infrastructures already includes cloud features (e.g., UC2) and could be used to support offloading scenarios to different clouds based on different resources or hardware availability.

The high-level picture of Pledger infrastructure is shown in Figure 4.

For UC3, this process is not possible for machines on customer sites, as services must not be exposed on the public internet. This has been identified as a new requirement in Section 6.1 and taken into considerations for further development, e.g. to support distributed orchestration via StreamHandler. Nevertheless, for the first iteration of the pilot operation, an OpenVPN [7] connection will be established for the infrastructure set-up in FILL's own premises.

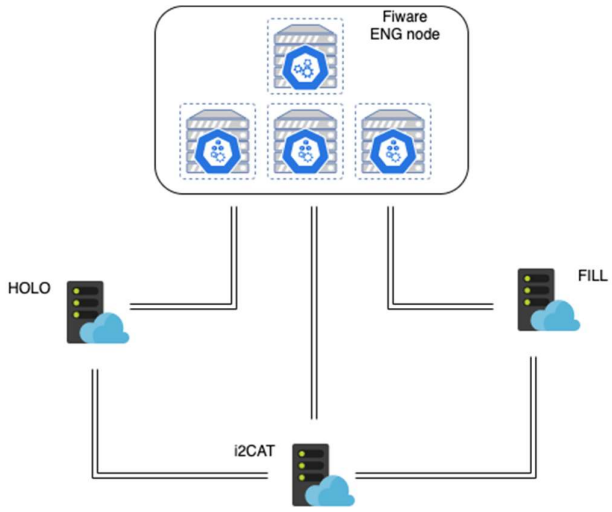

Figure 4: Pledger infrastructure

3.1.2 Providing application-relevant metrics to Pledger

Several applications have been developed in the individual UCs to be managed by Pledger. These applications have been described in detail in deliverble D5.1 [3] and an overview of them was given in Section 2. Based on the individual functionality and goals of the UCs, application-specific metrics can be defined and generated. These application-specific metrics are required to monitor the applications' health status as well as to monitor the QoS and detect possible SLA violations and to allow the Decision Support System (DSS) to take optimal decisions about App placement and resources management. Such application-specific metrics must be provided by the infrastructures integrated with Pledger, and two options are available:

- Usage of Prometheus service for the monitoring on the customer site.

  Prometheus [8] is an open-source monitoring framework that provides out-of-the-box monitoring capabilities for Kubernetes clusters. Prometheus uses Kubernetes [4] Application Programming Interfaces (APIs) to read all the available metrics from nodes, pods, deployments, etc. and is widely used in production environment.

  For example, a Prometheus engine is deployed in UC2's Kubernetes infrastructure. Metrics gathered by the Prometheus engine are exposed through an endpoint that is accessible to Pledger's core components. This enables the Monitoring subsystem to collect the metrics through this endpoint. Although this is a standard-de-facto monitoring solution, it could not be always available on the different infrastructures site, neighter Pledger aims to enforce such a requirement and instead promotes flexibility with the adoption of a lighter solution for monitoring that is described next.

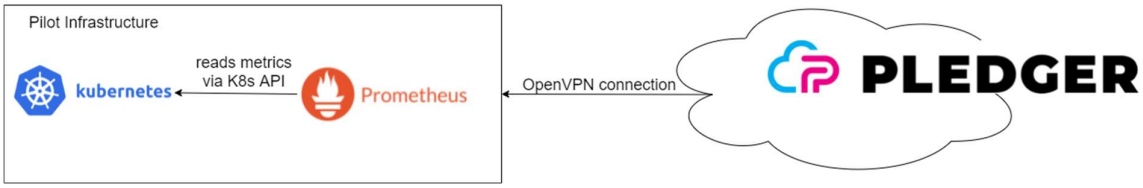

Figure 5: Providing metrics via a self-hosted Prometheus instance

- Usage of a central Prometheus instance on Pledger which receives application metrics by via StreamHandler Platform

The StreamHandler Platform (distributed event-driven message streaming platform based on Apache Kafka [9]) acts as the major backbone for Pledger integration. A detailed description of this component is given in deliverable D5.2 [10]. This component is based on the publish/subscribe mechanism and can be utilized for the transfer of application-relevant metrics for the Monitoring Engine to store them for the specific UC, thus requiring a small integration effort compared to hosting a Prometheus service. In the Monitoring Engine, a proxy module to transfer metrics values published periodically by the UC applications via StreamHandler to the centralized metric datastore (Prometheus time series database). This proxy module consumes the messages on the specific topic from StreamHandler for metric samples, processing them (e.g., format conversion, mapping) and inserting them in the metrics datastore. This enables any Pledger component to query these metrics anytime when necessary. The detailed description of the development of this proxy module belongs to the developments in WP3 and will be reported in the next version of WP3 deliverables.

To provide the metrics to the StreamHandler platform, a producer needs to be developed to gather the required metrics and publish them to a topic of the Kafka.

For UC1, the application collects these metrics and publishes them directly to the topic, whereas, in UC3 these metrics are first extracted individually by the application itself or by another microservice and then transferred to the central message broker on the edge. Afterwards another service can publish these metrics on the topic. This process is described in more detail in Section 3.2.3 as part of the integration with the StreamHandler platform.

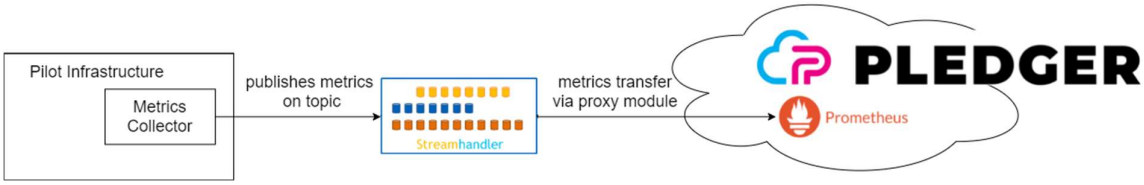

Figure 6: Export of metrics to Pledger Prometheus via StreamHandler Platform

The definition of these metrics as well as the identification of metrics worth monitoring is very individual for each UC and is described in detail in Section 4.

3.2 Integration with Pledger components

For the second part to achieve operational readiness, UCs need to integrate with some core components. These vary depending on which functions are needed and desired and the integration depends on various prerequisites. All UCs need to integrate with the Pledger Orchestrator to launch applications. Furthermore, they need to integrate with the SaaS/IaaS Monitoring Engine to enable the system to monitor the infrastructure and the running applications as well as the QoS to enable Pledger to optimize it. All UCs desire to use the Distributed Ledger Technology (DLT), as sensitive data is involved, which should be stored on the Pledger blockchain. The individual purpose is explained in Section 3.2.2.

Furthermore, UC2 integrates with the Slicing and Orchestration Engine (SOE) and Radio Access Network (RAN) Controller to support the provisioning of a dedicated end-to-end slice on top of the city-wide infrastructure to host the UC2 application. The slice includes both the (vehicular) radio and computing resources required to run the UC application.

Finally, UC3 is integrating with the StreamHandler platform to synchronize and transfer data between different infrastructures to support offloading computation in different scenarios.

These different integration activities are described in the following subsections in detail, where a summary of these integration endpoints and the status is given at the end.

3.2.1 Integration with the Pledger Orchestrator

To guarantee the connectivity and to ensure that the pilots' infrastructure can be reached by the Orchestrator, an OpenVPN connection is established for all UCs with an OpenVPN client is installed in the pilots' infrastructure to allow inbound and outbound traffic.

For UC1, the Orchestrator needs to launch and manage Virtual Machine (VM) instances to run the application. For this purpose, VmWare Player Hypervisor has been chosen as it provided the best performance for the application. These VMs need to allow the application to use its GPU and have Windows 10 OS. When a user decides to use the application, the Orchestrator will be informed and will launch a VM instance. Both the VM and client's Hololens will connect to the DLT and start the secure Web Real-Time Communication (WebRTC) signaling (Handshake), once successful the remote rendering (Stream back to the Client's HMD) starts.

This remote rendering utilizes the powerful GPU within the VM to overcome the shortcomings of the HMDs and thus provide the user with a smoother and more fulfilling XR experience.

For UC2, although the Pledger orchestrator gives the initial order of deploying and/or scaling applications, the deployment is managed by the Slicing and Orchestration Engine and RAN controller framework. In other words, when deploying an application, the Pledger Orchestrator communicates with the SOE and RAN controller framework, which executes the deployment in UC2's infrastructure. After the deployment, the UC application mostly interacts with the infrastructure elements (radio nodes, vehicular stack, etc.) and the DLT. Yet, the orchestrator can issue the UC2 application to be relocated. Integration with the orchestrator is achieved through an OpenVPN server that has been deployed in UC2's infrastructure.

For UC3, the applications are virtualized as Docker containers running on a Docker engine. To orchestrate these Docker containers, the Docker API needs to be reachable under a specified IP address, which has been tested.

3.2.2 Integration with the Pledger DLT

The Pledger DLT enables the UC to store sensitive information and provides secure authentication via mutual certificates handshakes.

In UC1, the DLT will be utilized to manage the secure handshake between the end user and the server. To integrate with the DLT, the blockchain nodes provided need to support the Whisper protocol and expose a secure endpoint to the nodes that allows the UC1 Application to connect to them.

The Whisper protocol is a double encryption protocol that allows for as much security as desired. The messages exchanged between the Whisper nodes are unknown and can only be decrypted for an authorized user with the correct topic and key. All nodes receive the message whether it can decrypt it or not and send it to their neighbours, therefore the true recipient of the message is never known to the network. In both scenarios, the DLT will handle the secure handshake between the Client (HMD) and the Server (VM) that will make the remote rendering work.

For remote rendering to work, some sensitive data about the Client and Server needs to be exchanged to establish a Peer-to-Peer connection between them. Therefore, Whisper (over the DLT) will be used to secure the signalling between the Client application and the Server application by encrypting the information being exchanged in a way that makes accessing this data almost impossible.

UC2's application generates logs containing sensitive information, e.g., related to the location of a VRU at a given time and it also contains information on critical events, such as possible accidents between VRUs and pedestrians. To assure that only trusted users have access to this information, any of the logs generated by the application are dumped to the secure DLT through the StreamHandler platform. Connection to the DLT node running in the core cluster will be achieved through Kubernetes NodePort [13]. Once on the DLT, the information is only accessible by trusted users or nodes.

In UC3 an application deals with the analysis of and details of parts produced by the machine. The information this result contains (e.g. the amount of parts produced, the quality of the parts, etc.) is considered sensitive and are considered to be stored on the Pledger DLT to grant access only to authorized parties. The details for this integration are still to be clarified.

3.2.3 Integration with StreamHandler Platform

The StreamHandler Platform does not only act as the backbone for integration of the Pledger core components, but it is also used also for other two purposes:

- Data transfer to the cloud to support the offloading of applications to the cloud in case limited exposure of services is required (UC3)
- Export of application metrics (UC1 and UC3)

About the data transfer, when offloading computation to the cloud, the data required for the analytical services to produce the desired result needs also to be offloaded to ensure the availability of the data and increase the robustness of the system and to avoid publishing services on the edge to be reached directly from the couterparts on the cloud. Having the StreamHandler as an additional hop is a compromise that increases security at the cost of adding some latency to the scenario which is tolerated in some scenarios. For UC3, two components have been developed to facilitate the integration with the message broker on the edge (RabbitMQ) and the StreamHandler platform in the cloud:

- Broker-To-StreamHandler component
- StreamHandler-To-Broker component

The Broker-To-StreamHandler component transfers (i.e., consumes and publishes) the raw data from the message broker on the edge (RabbitMQ [11]) to the StreamHandler. From there, the application can consume the data as soon as it is deployed in the cloud and perform its analytical task. As soon as the service generates a result, this result is published as a message on a separate topic. The message gets consumed by the StreamHandler-To-Broker component and published on the RabbitMQ from where it gets stored in the database. This process is illustrated in Figure 7.

The data is streamed continunously to the StreamHandler platform to support the offloading of the computation at any time. As data in UC3 gets analyzed batch-wise, a mechanism to synchronize the data and to provide the application the necessary information which data has already been processed on the edge and from where to start in the cloud was implemented. Besides the result, the application also publishes the according timestamp of the last processed data point. This timestamp is synchronized between edge and cloud using the components described.

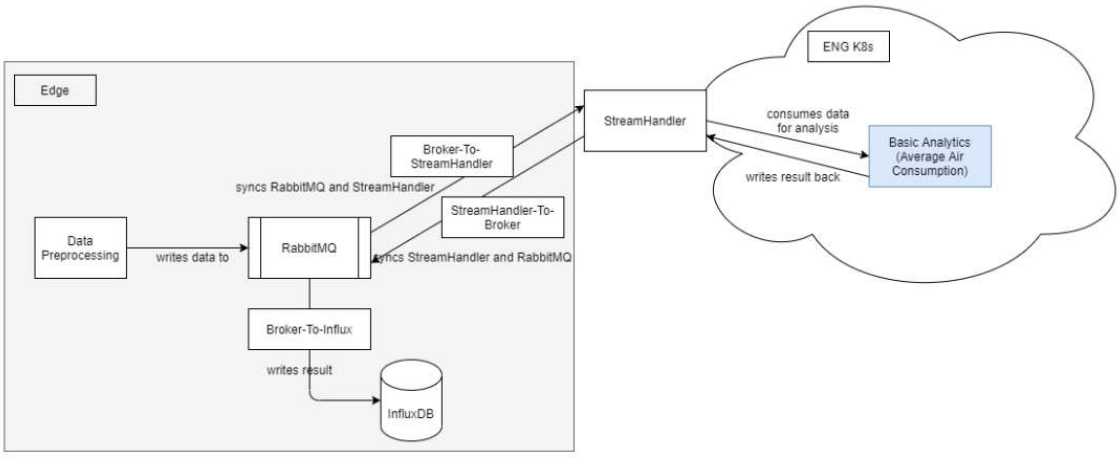

Figure 7: Integration with StreamHandler platform in UC3

About the export of metrics, UC1 and UC3 gather application metrics and send them using the StreamHandler through custom Kafka producers as it has been already described in section 3.1.2.

3.2.4 Integration with SOE and RAN controller

Infrastructure slicing and deployment of services are performed with support of the SOE and RAN Controller framework, which needs to be installed and running on top of the infrastructure. When a service provider requires the deployment of an application to the Pledger Orchestrator, the Orchestrator sends a request to the SOE, with details about the slice requirements and the service to be deployed.

In UC2, the SOE and RAN Controller framework is deployed via containers in a standalone Kubernetes cluster, on a single node installed on top of a virtual machine. Upon slice creation and service deployment, the framework communicates with other infrastructure elements, such as Openstack, through standard API interfaces.

Integration with the SOE and RAN Controller framework in UC2 involves:

- Integration between Pledger SOE Framework and Barcelona's infrastructure: The SOE Framework needs to register the computing infrastructure, which also needs to be configured to work with SOE. Since the Barcelona infrastructure is based on Openstack, this integration is achieved through a standard API interface.
- Integration between Pledger RAN Controller and Barcelona's infrastructure: the RAN Controller needs to register the radio infrastructure, which is then enabled to support slice configuration. This is achieved using Network Configuration Protocol (NETCONF [12]) and OpenFlow protocols.
- Integration between SOE and RAN Controller with the Vehicle-to-everyting (V2X) Stack: the V2X stack required by UC2's application needs to be supported by the framework, both for the physical configuration of the radio devices (RAN Controller) and for the instantiation of the containers that run the V2X stack as part of the infrastructure (SOE).

3.2.5 Integration with SaaS/IaaS Monitoring Engine

The SaaS/IaaS Monitoring Engine needs to be able to obtain metrics from the infrastructure to monitor it using either the central Prometheus instance on Pledger or using a self-hosted Prometheus instance.

In UC1, infrastructure related metrics are collected by the application and published to the StreamHandler directly, then consumed and transferred to the central Prometheus instance as described in Section 3.1.2.

In UC2, this is achieved using Prometheus, which is already deployed in UC2's Kubernetes infrastructure. Communication is achieved through a VPN connection to the cluster hosting the SaaS/IaaS Monitoring Engine, using a standard API interface.

In UC3, these metrics need to be obtained from the Docker engine. A separate microservice collects periodically these metrics via the Docker command line [13] to gather the container(s) resource usage statistics. These messages are published to the message broker on edge from where the get transferred to the central Prometheus instance of Pledger as described in Section 3.1.2.

3.2.6 Summary of UCs' integration end points and status

This section summarizes the different integration end points for the three UCs. 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 – 1, DLT – 2, SaaS/IaaS – 3, SOE – 4, RAN – 5 and StreamHandler – 6), whereas Z the UC component, described in the description of the table.

Table 1: UC 1 - Integration Endpoints.

The UC components for the identifier are: VM Configuration -1 and Client Unity Appliation -2

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

Table 2: UC2 - Integration Endpoints.

The UC components for the identifier are Barcelona Infrastructure – 1 and Risk Detection and Notification System (RDNS) – 2

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

Table 3: UC3 - Integration Endpoints.

The UC applications for the identifier are: Edge Server – 1 and Basic Analytics – 2

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

4 Application metrics to measure QoS

Measuring the performance and QoS of an application is very specific. As part of the set-up of the pilots, metrics which are worth monitoring as well as those describing the QoS have been identified to allow optimization and taking actions based on these metrics.

The main objective so far has been to identify the application metrics which are dependent on the computational resource usage, those which represent general issues (e.g., a major network failure) where resource management would not improve the QoS and requires further analysis and intervention to fix, so the DSS suspends any resource management on the related services, then SLA which are completely ignored by the DSS. In summary SLA are categorized as follows:

- SLA to be used by the DSS for resource management, as the related application metrics are known to be dependant on the actual resource usage.
- SLA to be used by the DSS to suspend any resource management on the related service as a major issue occurred and must be solved before any action from the DSS (e.g., a major network failure).
- SLA that are ignored by the DSS and are used for other reason (e.g. monitoring, smart contracting, etc.)

Along with application metrics, all UCs provide system metrics (like CPU and memory) to make the DSS aware of the current availability to better drive the offloading.

An initial set of application metrics has been identified, further testing and trial runs will be used to validate and extend them appropriately. This will be done during the next iteration and first actual pilot operation phase and reported in the next version of this deliverable in M30.

4.1 UC1: Mixed Reality applications on the edge

UC1 enhances the performance of HMDs using remote rendering with a VM deployed on a laptop. The application running on the VM will calculate the average time to load a file (CAD files are very and vary in size and therefore load time). Load time is a good indicator of CPU and Memory consumption. For this reason, load time is monitored further testing should allow us to find further ones like, for example the FPS of the application.

The metrics mentioned above permits us to get an idea about the QoS. For example, the load time of a file is heavily dependent on CPU and Memory, the higher these values the faster the loading of the file. These metrics will be exposed through a Kafka Broker to be consumed by the Monitoring Engine.

Table 4: Metrics in UC1

Metric	Source	Expected behavior	DSS action
File load time	Application	Increases in case ressources are too little	Execute scaling if the application is actually using the resources assigned and a SLA Violation is received

4.2 UC2: Edge infrastructure for enhancing safety vulnerable road users

Since UC2 application is deployed with the aim to increase safety for VRUs in face of possible risks they might encounter on-street, it is key for the application to run without interruptions. Further, if risk situations arise, warnings towards the VRUs should always be issued as fast as possible to give more time to react, e.g., allowing a cyclist to adjust their speed, or a pedestrian to look out for an approaching bicycle. Similarly, the vehicular software stack needs to be able to process all messages coming from vehicles towards the infrastructure containing positioning information and the other way around when warning messages are issued to the VRUs.

Based on these requirements, three parameters to determine the QoS for UC2 crystallize, namely the delay, which indicates how long it takes for vehicular information from on-street users to reach the application running in the Pledger infrastructure, packet losses, i.e., information that is sent by the VRUs or the application but gets lost along the way and finally, the load of the processing queues of the vehicular stack itself.

The delay is measured in the application as the difference between the timestamp of location information sent by VRUs equipped with OBUs and the current time on the application server. Given the fact that the OBUs are using satellites to obtain an accurate timestamp and that the infrastructure in Barcelona can be synchronized with the same satellite timestamp, the two ends of the packet transmissions can be synchronized.

The packet losses are measured by comparing the transmission ratio from the OBUs with the actual number of packets received by the infrastructure. At the Edge side, packet losses can be obtained from each V2X transmission flow by being filtered by the Station ID field or computed given the total number of current VRUs in the scenario. Decentralised Congestion Control (DCC) mechanisms effects on the VRU side will be considered to minimize its impact over the scenario.

The load of the queues of the vehicular software stack that forms part of the infrastructure is also used as an indicator of the performance of the applications. If it is observed that the Message Queuing Telemetry Transfer (MQTT) broker, a submodule of the vehicular stack responsible for communicating the V2X stack with other MEC modules, presents an abnormal load on the application queue, it should be considered as an indicator for the application performance health check.

Table 5: Metrics in UC2

Metric	Source	Expected behavior	DSS action
Delay	Application	End-to-end has to be below 1.5s, including message generation and transmission/reception delays on both sender and receiver.	Execute scaling/offloading if the App is actually using the resources assigned and a SLA violation is received.
Packet loss (radio)	V2XStack	No more than 5% consecutive packets lost shall be observed when operating under line-of-sight conditions.	Temporary suspend any DSS action on the related App.
Packet loss (application)	Application	No more than 0.1% of packet losses shall be observed on the node running the application. Otherwise, a network issue might be occurring.	Execute offloading to another node of the cluster.
Load of processing queues	V2X Stack	Vehicular messages are expected at a certain frequency for each connected user. If the load exceeds the value, an attack might occur	Ignore in terms of scaling/migration. This information is useful for monitoring purposes and can help to identify an attack.

4.3 UC3: Manufacturing the data mining on edge

Within UC3, applications to determine the process stability of the machine have been developed. For this purpose, two services analyse the average air consumption as well as the average energy consumption of the machine and the end user can determine the process stability in terms of media consumption using the UI of Cybernetics. Furthermore, a service calculating the cycletime of the parts produced as well as some more information (type, target cycle time, error reasons, duration of subprocesses) has been developed. All services gather the required data from the central message broker (RabbitMQ on the edge, StreamHandler in the cloud), calculate the relevant information and push the results back to the message broker, from where the results get stored into a database. The services are virtualized as Docker containers.

To determine the health status of the containers and to get relevant metrics to infer the QoS, two sources have been identified. First, in the application itself, the duration for the process of calculating the relevant data can be determined. This is highly dependent on computational ressources, meaning with decreasing ressources the calculation process is expected to take longer leading to a decrease in QoS. These metrics can be extracted directly in the application taking the timestamp when calling the relevant function as well as the timestamp when receiving the result into consideration. Secondly, RabbitMQ metrics can be observed. RabbitMQ provides specific metrics to monitor the status of the cluster as well as nodes and queues [14]. In UC3, each application is connected via its own queue. Therefore, monitoring the queue metrics allows to monitor the applications' health status. If an application is up and running, it regularly consumes messages and acknowledges them accordingly. Therefore, the metric "Messages delivered recently" and "Message delivery rate" is expected to be constant. In case an application is down, these metrics decrease, and the metric "Number of unacknowledged messages" exceed application-specific limits. As soon as the metric "Number of messages ready for delivery" increases, the application is not consuming for longer time and will be monitored to create more insights. This metric has crucial impact, as it has a hard limit of ~7000 messages before messages get dropped leading into fatal loss of data. These metrics can be extracted directly using an Hypertext Transfer Protocol (HTTP) API [15].

Table 6: Metrics in UC3

Metric	Source	Expected behavior	DSS action
Calculation Time	Application	Increases in case ressources get limited	Execute offloading to the cloud if the App is actually using the resources assigned and a SLA Violation is received.
Messages delivered recently	RabbitMQ	Decreases in case application is not consuming/not performing well	Execute offloading to the cloud if the App is actually using the resources assigned and a SLA Violation is received.
Number of unacknowlegded messages	RabbitMQ	Increases in case application is down	Temporary suspend any DSS action on the related App until Calculation Time is fixed
Number of messages ready for delivery	RabbitMQ	Increases in case application is down – external action necessary (e.g., software engineer)	Ignore, used for monitoring.

5 Workflows and Scenarios for the Stakeholders

5.1 Overview Pledger Stakeholders

In deliverble D2.2 "Pledger Requirements Analysis" [17] the stakeholder's value chain has been identified, namely:

- IaaS Providers offering their infrastructure as a service and leverage resource monitoring
- System Integrators develop added-value services over those provided by the Pledger system
- SaaS Providers offering Software-as-a-Service and providing applications
- SaaS Consumers using the services provided by the SaaS Providers

These stakeholders come into play in several ways during the execution of the pilots. To be able to demonstrate the functionalities and benefits of the platform for the different roles and interests, different workflows and scenarios are defined.

IaaS providers use Pledger to configure and make their infrastructure available to SaaS providers. The SaaS providers use Pledger to find and deploy the right infrastructure for their applications and monitor and optimize the quality of service. System integrators are interested in defining scenarios that can bring to the design of added-value services leveraging on Pledger services.

In the following sections, the main IaaS and SaaS workflows are presented, along with the scenarios demonstrated by the pilots for the first phase of operation. The stakeholders' workflows and pilot demonstrators will be extended during the next iterations and reported in M30 and M36. All mentioned components are described in the associated deliverables of WP3 and WP4.

5.2 Workflows

5.2.1 IaaS Providers

For the IaaS Providers, two workflows have been defined. The first one describes, how their infrastructure can be added to the system and the possibility to customize the benchmarking process.

- Register infrastructure in Pledger

  To lease the infrastructure for SaaS Providers to be used in the Pledger system, the Infrastructure Provider need to register their infrastructure by following steps:

  1. Access Pledger Configuration Service
  2. Register infrastructure to enable Pledger SaaS Providers to utilize it
     - Configurations to be done: enpoint, monitoring endpoint, type, nodes, nodes properties (edge/cloud), enable benchmarking
  3. If Benchmarking is enabled: Benchmarking tests are performed on the infrastructure and their results stored to act as inputs for the DSS.

- Customization of benchmarking infrastructure in Pledger using Benchmarking UI

  When enabling the benchmarking option, generic benchmarking tests representing different kind of applications (e.g., database, analytics application) will be performed. The Infrastructure Provider has the possibility to customize them using the following steps:

  1. Schedule timing of specific tests: set time and intervals to run the tests
  2. Change sequence and ordering of tests

These workflows for the Infrastructure Provider are summarized in Figure 8.

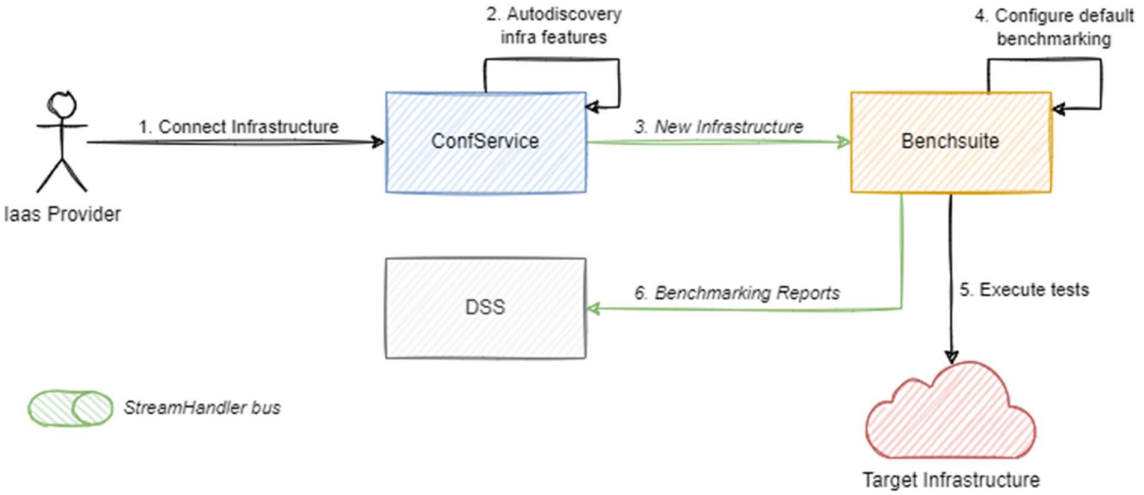

Figure 8: Workflow for the IaaS Provider

5.2.2 SaaS Providers

For SaaS Providers the following workflow has been defined to deploy an application:

- Deploy an application
  1. Provide application in Pledger registry
  2. Select App to launch and set configurations:
     - SLAs
     - Preferences for infrastructure (edge/cloud)
  3. DSS selects appropriate infrastructure (considering performances from previous benchmarking tests, results from App Profiler and SLAs)
  4. Orchestrator deploys application on selected infrastructure
  5. Monitoring Engine starts monitoring the application

This workflow is illustrated in Figure 9.

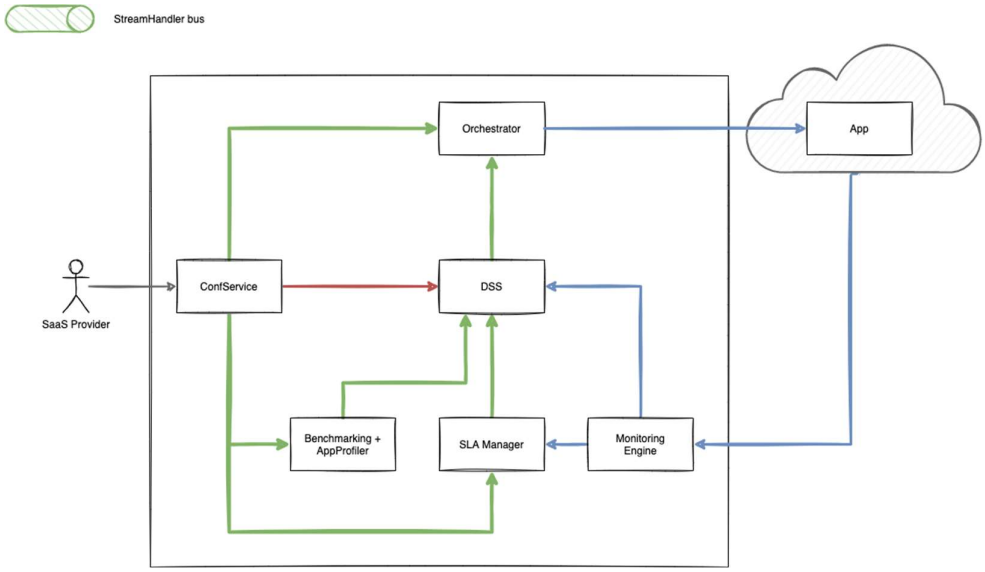

Figure 9: Workflow for Service Provider

5.3 Scenarios demonstrated in the Pilots using Pledger components

The preceding descriptions for workflows explain simple details of how stakeholders can interact with the platform and are common to all use cases. In the following, scenarios are now described that are used in one or more UCs and are customized to the different needs of the UCs. Afterwards, one scenario for each UC is outlined, which can be demonstrated in the individual environment using Pledger components.

5.3.1 Optimize QoS in case of lack of resources

This scenario is a first demonstration for Pledger to optimize the QoS in case ressources get little on the infrastructure the application has initially been deployed in. In this case the application is migrated to another infrastructure

Starting point: an application “App” is deployed, configured to use a max. amount of resources, is running and is providing metrics.

1. Monitoring Engine sends resources and application metrics.
2. The monitored application needs high amount of resources (CPU/RAM).
3. SLA Lite monitors sends SLA violations about application metrics over SLA thresholds.
4. DSS gets SLA violations and resource metrics and commands a migration to an alternative infrastructure (e.g., cloud).
5. Orchestrator migrates the application to the cloud.
6. Monitoring continues to provide resource and application metrics via Monitoring Engine.
7. Resources are available again - load on edge is gone, no SLA Violations are received for some time, so resource metrics decrease and fall under the defined threshold and DSS issues offloading back to edge.
8. Orchestrator relocates application to the Edge

5.3.2 Compute reservation and service deployment

Infrastructure slice reservation includes the creation of compute, radio and network chunks. This scenario demonstrates the configuration of a compute, as it would be configured as part of the process of infrastructure slice creation, reserving computation resources. Next, a service formed by cloud-native functions can be instantiated in the Kubernetes namespace within the compute resource quota assigned. Networking and radio resources are statically reserved and configured beforehand and are not demonstrated in this scenario.

1. Orchestrator launches service deployment request
2. Create namespace in Kubernetes
3. Assign resource quota to previously created namespace
4. Compute chunk is created
5. Instantiate service formed by cloud native functions in the compute chunk's namespace
6. Service up and running

5.3.3 Scenario for UC1

This scenario aims to help users to utilize the rendering power of edge devices that HMDs like the Hololens2 are unable to achieve [17].

Combining the components currently integrated and the flows described before, the following steps can be demonstrated:

1. Infrastructure registration of edge server
2. The SaaS provider launches of application on VM and the Orchestrator spins it.
3. Loading a file of bigger size, taking more time than defined threshold
4. SLA lite sends out a SLA Violation about App metrics over SLA thresholds

5. DSS commands a scaling up, then a scale down when no SLA violations are received for some time and resources are available again

This scenario will be further extended with the automatic deployment of the application via the Orchestrator which will allow the full scaling up and down scenarios triggered by the DSS depending on the resources availability and the SLA violations received.

5.3.4 Scenario for UC2

The Risk Detection and Notification System (RDNS) application is providing a critical service to VRUs: detecting risky situations and warning the VRUs (or other endpoints) about it, so that an accident can be avoided. To assure that this service never starves in terms of computation resources or suffers from connectivity issues, the SLA monitoring system implemented in Pledger closely observes how the compute node resource usage evolves or if there are any networking issues. Based on specific SLAs that define thresholds for these two aspects (CPU/Network errors), the platform can react once the thresholds are exceeded: an alternative location within the Kubernetes cluster for the RDNS application is determined and the application is migrated.

Combining with previous described steps, the following scenario can be demonstrated:

1. Compute resource reservation – Section 5.3.2
2. Deploy an application – Workflow #3 in Section 5.2.2
3. Optimize QoS in case of lacking resources – Section 5.3.1

5.3.5 Scenario for UC3

To monitor the process stability of the machine, several applications analyzing parameters for the process stability will be deployed. If one of the applications need more resources to accomplish its task for a certain time of period, SLA thresholds (based on the calculation time) exceed defined thresholds and one application will be moved to the cloud.

In combination with the other steps and workflows, this results in following scenario:

1. Infrastructure registration of the edge node and cloud – Workflow #1 in Section 5.2.1
2. Deploy an application on the edge – Workflow #3 in Section 5.2.2.
3. Optimize QoS in case of lack of resources by moving to the cloud and back – Section 5.3.1.

5.4 Additional Scenarios demonstrating functionalities of Pledger components

System Integrators might be interested to leverage on the Pledger components to develop added-value services, even though they are not yet support from the pilots.

5.4.1 DSS

For the DSS, a demo scenario will be prepared to demonstrate two algorithms: resource optimisation and Edge to Cloud Offloading Decision Algorithm (ECODA) [18] optimization, both using Kubernetes in Docker (KinD [19]) environment to allow an easy replication of the scenario and facilitate the adoption of this and the other Pledger components by system integrators.

The same demo will be accompanied by offline videos to be published on Pledger's YouTube channel and documented in the public repository to allow any interested developer to build the DSS and test the algorithms.

The scenario presented will comprise two Kubernetes clusters:

- the first with 1 cloud master, 1 cloud worker and 2 edge workers and with latency added on the cloud-edge connection.
- the second with 1 edge master and 1 edge worker node.

The demo scenarios below will show four dummy containers with different startup times and resource requests used by a service provider with limited resources on the edge.

The first demo will show the resource optimisation algorithm scaling/offloading the containers (edge to edge, edge to cloud on the same infrastructure, edge to cloud on a different infrastructure) according to the SLA violations received, which is aligned with the scenario used by the pilots.
The second demo scenario will show the ECODA optimisation algorithm scaling/offloading the containers based on the startup time, latency, resource requests and priority to minimise the ECODA utility function.
The third demo scenario will show a combination of the two optimisation algorithms above to have ECODA utility function on top of application resource limits that change dynamically, based on the SLA violations received.

5.4.2 SLA Lite

For the SLA component, two demonstrations have been drafted. First, for the SLASC Bridge functionality, instantiation of the component and deployment on the Pledger blockchain can be showcased. This component exploits the SLA definition of the SLA by interpreting it into blockchain contractual terms that trigger the dedicated transactional workflows on the ledger.
SLA Violations can be showcased by the continuous workflow functionality of an SLA Violation through the initiation of the corresponding blockchain transactional activities. The SLA Violation invokes the respective contractual workflow that includes the SLA customer refunding process on the blockchain ledger.

5.4.3 App Profiler

This demo aims to aid the process of resource selection for the Application owner, in container-based cloud and edge environments. One of the main needs of an application owner to migrate in a cloud edge environment is to understand the computational needs of their application. Understanding the computational needs of the application can help the application owner to select in a more informed way the appropriate resources to maintain the targeted QoS for their application. The Pledger Benchmarking component helps to estimate the resource usage of default applications, the App Profiler creates a profile of the specific application, the SaaS Provider wants to deploy and categorizes it into a benchmarking category to align it with the benchmarking results. This helps the DSS to decide on which infrastructure to deploy. This demo will demonstrate how a deployed application in a Kubernetes-Docker environment can be profiled and classified in a benchmark category. Following steps are required:

1. Deploy the application in a containerized environment
2. App Profiler is invoked and collects metrics
3. App Profiler creates the profile and computes mean values, deltas and variations
4. App Profiler feeds the profile to the classifier to produce the results

6 Link to validation and evaluation task

The task T5.3 “Pilots operation and monitoring” and the task T5.4 “Validation and Evaluation” are related in that this task executes the experimentation scenarios in the pilots to enable validation and evaluation of the results.

This paragraph deals with some aspects of the methodology of how the platform is evaluated and will be expanded in the next iterations of this task. It starts with the analysis of the requirements and their review for up-to-dateness. Then the process is presented how these requirements are validated in the project and an outlook to the next steps is given.

6.1 Update of requirements

In the deliverable D2.2 [16], the requirements for the development of the Pledger components have been defined. Since its release, the Pledger consortium produced a few more requirements changed or dropped some others. In this deliverable the result of this process is reported in terms of presenting the requirements which have been dropped or changed as well as a short note for explanation, as summarized in Table 7.

Before that, for convevience, the list below summarizes the requirements categories and the components involved:

- FU_CORE: functional requirement about Pledger core components;
- FU_APPL: functional requirement about Pledger Apps;
- NF_CTR: non-functional requirement about hardware and software constraints;
- NF_SEC: non-functional requirement about security;
- NF_PER: non-functional requirement about performance;
- NF_AVA: non-functional requirement about availability.

Table 7: Summary of changed and dropped requirements

Code	Description	Relevant Stakeholders	Category	Status	Note
FR.08	The Pledger system could deploy event-driven apps with zero instances (FaaS).	SaaS Provider	FU_CORE	Dropped	Further analysis showed that no FaaS needed, therefore the effort has been moved to other activities.
FR.79	The Pledger system could support suggestions based on ML algorithms.	SaaS Provider	FU_CORE	Dropped	No data available for ML, effort moved to Lagrangian multiplier optimization used in ECODA algorithm
FR.16	The Pledger system must provide a dashboard to check which data has been released to whom.	SaaS Consumer	FU_CORE	Dropped	The release of data is handled via StreamHandler and a configuration is set about which data is pushed to the StreamHandler. Internal evaluation showed that no additional dashboard is necessary.

Code	Description	Relevant Stakeholders	Category	Status	Note
FR.18	The Pledger system must be able to offload computation to remote rendering services in case data is too complex, in order to overcome the device limitations and ensure functionality.	SaaS Provider	FU_APPL	Changed	Offloading scenarios for rendering changed to scaling scenario.
FR.38	The Pledger system must allow user end devices to be registered on the services that implement the VRU safety features.	SaaS Consumer	FU_APPL	Dropped	The application running within the Pledger ecosystem will expose an interface that allows to add/delete spcific users (VRUs). Only those users will be able to use the application.
FR.55	The Pledger system should allow the initiation of smart contract deployment from a user-friendly interface, hiding unnecessary details to the developers.	System Integrator	FU_CORE	Dropped	This requirement has been redirected towards exploring privacy features on the blockchain network.
NFR.19	The Pledger system could support connections among large numbers of edge locations having overlapping IP sub-networks.	IaaS Provider	NF_CTR	Changed	StreamHandler is used for the communication between different infrastructures; alternatives are under evaluation to allow (at least) direct POD to POD connectivity among different Kubernetes clusters.

Besides identifying the outdated requirements, also some new requirements have been identified during the set-up of the pilots as well as during the integration activities with the platform.

Table 8 gives an overview of these ones and their according categorization.

Table 8: Summary of new identified requirements

Code	Description	Relevant Stakeholders	Category
FR.ADD.1	The Pledger system should allow to configure domain features of the different nodes and infrastructures	SaaS Provider, IaaS Provider	FU_CORE
FR.ADD.2	The Pledger system should support low latency Apps on the edge and allocate them to optimise total latency with respect to resource consumptions	SaaS Provider, SaaS Consumer	FU_CORE
FR.ADD.3	The Pledger system should be able to automatically discover nodes HW features	IaaS Provider	FU_CORE
FR.ADD.4	The Pledger system should support vertical scaling of applications on VMWare	SaaS Consumer	FU_CORE
FR.ADD.5	The Pledger system should support edge and far-edge devices	IaaS Provider	FU_CORE
FR.ADD.6	The Pledger system should allow to prioritise node/infrastructures and offload accordingly when resources are not available	SaaS Consumer	FU_APPL
FR.ADD.7	The Pledger system should allow to distinguish between SLA which depends on allocated resources (e.g., number of frames processed per second) and those that are not (e.g., failure of radio access network)	SaaS Consumer	FU_APPL
NFR.ADD.1	The Pledger system should allow the integration with remote infrastructure without need to expose services on the public Internet	System Integrator	NF_SEC
NFR.ADD.2	The Pledger system should allow to orchestrate infrastructures using a distributed architecture	System Integrator	NF_AVA
NFR.ADD.3	The Pledger system should allow to integrate with monitoring easily and without need to setup dedicated gateway to collect data	System Integrator	NF_PER
NFR.ADD.4	The Pledger system should allow to assess the security capabilities of the infrastructures to help the SP to prioritise the Nodes where to deploy Apps	System Integrator	NF_SEC

6.2 Validation of requirements and next steps

The prioritization of the requirements was performed in deliverble D2.2 [16] according to the scheme of the MoSCoW method [20]. For validation, the requirements with the prioritization of the category "**M**ust have" and "**S**hould have" are used in the first step and further extended to those of the other categories. During this process, the steps of the workflows and scenarios performed during the execution of the pilots will be linked to individual requirements and thus checked for their fulfillment.

Further steps to be done to ease the validation and evaluation task include the definition and execution of further scenarios to validate the Key Performance Indicators (KPIs) derived in WP2.

7 Conclusions

In this deliverable, the initial work performed in the task T5.3 “Pilots operation and monitoring” was described. First steps included the set-up of the pilots in terms of infrastructure and connectivity with the Pledger core components involved to enable the execution of the Pledger pilots. These steps have been defined in close coordination with the technical partners, where common solutions for setting up the connectivity and first workflows for IaaS and SaaS Providers have been found. To measure the QoS of the applications running in the UCs, different solutions have been found to integrate with the Monitoring Engine. The definition of the metrics to measure the QoS has been defined by each UC individually, as this process is very individual to the applications provided by them.

During this first phase, also the requirements defined in an early phase of the project have been reviewed and updated, whilst during the set-up of the pilots, new ones have been identified.

A solid foundation was set for the execution of the pilots in terms of integration with the platform and setting up the infrastructure. The infrastructure for all pilots is set-up and metrics to monitor and to use for the system to improve the QoS have been defined.

The status of integration is different for each of the pilots. All integration end points in UC1 are still in progress, where the integration with SaaS/IaaS Monitoring Engine is in its final phase. For UC2, the integration with the SaaS/IaaS Monitoring Engine is already done, as well as the integration with the Orchestrator. The remaining integration end points are in progress. The integration with the Orchestrator is also done for UC3, as well as the integration with the StreamHandler to support offloading an application to the cloud. The other integration end points are in progress, except for the integration with the DLT, which is still under clarification.

The current status and closing the points in finalization will allow the outlined scenarios and demos to be carried out in the next phase. In further close coordination with the technical partners, these will be expanded in the future to include additional scenarios and more complex steps and reported in the next version of this deliverable in M30.

8 References

[1] *IDC Worldwide Augmented and Virtual Reality Spending Guide*, https://www.idc.com/tracker/showproductinfo.jsp?containerId=IDC_P34919, retrieved 2021/11/29

[2] *22@ area*, https://web.archive.org/web/20091220132715/http://www.22barcelona.com/index.php?lang=en, retrieved 2021/11/25.

[3] PLEDGER. *D5.1 Pledger Applications for the use cases*, A. Betzler, (2021) http://www.Pledger-project.eu/content/deliverables.

[4] Kubernetes home page https://kubernetes.io/ , retrieved 2021/11/16.

[5] Fiware home page https://www.fiware.org/ , retrieved 2021/11/16.

[6] Openstack home page https://www.openstack.org/ , retrieved 2021/11/16.

[7] OpenVPN home page https://www.openvpn.net/ , retrieved 2021/11/16.

[8] Prometheus, home page, https://www.prometheus.io/ , retrieved 2021/11/24.

[9] Apache Kafka home page, https://kafka.apache.org/ , retrieved 2021/11/24.

[10] PLEDGER. *D5.2 Pledger integrated demonstrator I*, I. Sarris (2021), http://www.Pledger-project.eu/content/deliverables , retrieved 2021/11/16.

[11] *RabbitMQ*, https://www.rabbitmq.com/ , retrieved 2021/11/24.

[12] R. Enns and M. Schönwälder and J. Schönwälder and A. Bierman (2011), *Network Configuration Protocol (NETCONF)*, RFC 6241, https://www.rfc-editor.org/info/rfc4741.

[13] *Using a Nodeport*, https://docs.openshift.com/container-platform/3.6/dev_guide/expose_service/expose_internal_ip_nodeport.html , retrieved 2021/11/24.

[14] *Docker CLI Documentation*, https://docs.docker.com/engine/reference/commandline/stats/, retrieved 2021/11/24.

[15] *Monitoring RabbitMQ*, https://www.rabbitmq.com/monitoring.html , retrieved 2021/11/24.

[16] *HTTP API RabbitMQ*, https://www.rabbitmq.com/management.html#http-api , retrieved 2021/11/24.

[17] PLEDGER. F. Iadanza and G. Giammatteo (2020), *D2.2 Pledger Requirements Analysis*, http://www.Pledger-project.eu/content/deliverables , retrieved 2021/11/16.

[18] *Understanding performance for mixed reality*, https://docs.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/understanding-performance-for-mixed-reality, retrieved 2021/11/29.

[19] E. Carmona and M. S. Siddiqui (2021) *An Optimization Framework for Edge-to-Cloud Offloading of Kubernetes Pods in V2X Scenarios. TechRxiv. Preprint.* https://doi.org/10.36227/techrxiv.16725643.v1.

[20] Kubernetes in Docker, https://kind.sigs.k8s.io/, retrieved 2021/11/17.

[21] MOSCOW Priorisation, https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation, retrieved 2021/11/24.