D3.5 Edge/Cloud orchestration tools II

Document Identification			
Status	final	**Due Date**	31/05/2022
Version	1.2	**Submission Date**	02/06/2022

Related WP	WP3	**Document Reference**	D3.5
Related Deliverable(s)	D2.3 Pledger Overall Architecture v1.0 D3.2 Edge/Cloud orchestration tools I	**Dissemination Level (*)**	PU
Lead Participant	ICCS	**Lead Author**	Alexandros Psychas (ICCS)
Contributors	ATOS i2CAT	**Reviewers**	Olga Segou, Ioannis Sarris (INTRA) Verena Stanzl (FILL)

Keywords:
Cloud, Edge, Orchestrator, Monitoring, Deployment, Scaling, Migration, Performance, Container, Kubernetes, Docker, Demo

Document Information

List of Contributors	
Name	Partner
Alexandros Psychas	ICCS
Antonio Castillo Nieto	ATOS
Orfefs Voutyras	ICCS
Estela Carmona Cejudo	i2CAT
August Betzler	i2CAT

Document History			
Version	Date	Change editors	Changes
0.1	15/04/2022	Alexandros Psychas (ICCS)	Document template and initial ToC provided
0.2	02/05/2022	Antonio Castillo Nieto (ATOS) Estela Carmona Cejudo(i2CAT) August Betzler (i2CAT) Adrian Pino (i2CAT) Alexandros Psychas(ICCS)	Initial versions with contribution to all Chapters
0.3	16/05/2022	Alexandros Psychas (ICCS)	Content homogenization and contribution alignment
0.4	20/05/2022	Verena Stanzl (FILL)	Internal review 1
0.5	23/05/2022	Olga Segou, Ioannis Sarris (INTRA)	Internal review 2
0.6	24/05/2022	Antonio Castillo Nieto (ATOS) Estela Carmona Cejudo(i2CAT) Alexandros Psychas(ICCS)	Review comments addressed by the contributors
0.7	25/05/2022	Alexandros Psychas (ICCS)	Version for quality review
0.8	25/05/2022	Carmen San Román (ATOS)	Quality assurance review
0.9	31/05/2022	Lara López (ATOS)	Project coordinator review
1.0	02/06/2022	Antonio Castillo Nieto (ATOS) Estela Carmona Cejudo(i2CAT) Alexandros Psychas(ICCS)	Review comments addressed by the contributors
1.1	02/06/2022	Carmen San Román (ATOS)	Quality assurance review
1.2	02/06/2022	Lara López (ATOS)	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Alexandros Psychas (ICCS)	02/06/2022
Quality manager	Carmen San Román (ATOS)	02/06/2022
Project Coordinator	Lara López (ATOS)	02/06/2022

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
CNCF	Cloud Native Computing Foundation
CPU	Central Processing Unit
CRD	Custom Resource Definition (Kubernetes)
DB	Database
DSS	Decisions Support System
Dx.y	Deliverable number y belonging to WP x
E2CO	Edge to Cloud Orchestrator application
ETL	Extract, Transform and Load process
FC	Functional Component
GPU	Graphics Processing Unit
HTML	HyperText Markup Language
IaaS	Infrastructure-as-a-Service
KVM	Kernel-based Virtual Machine
K8S	Kubernetes software. Container orchestrator software.
MAE	Mean Absolute Error
MANO	Management and Orchestration
ML	Machine Learning
OSM	Open-Source MANO
QoE	Quality of Experience
QoS	Quality of Service
RAE	Relative Absolute Error
RAN	Radio Access Network
REST	Representational State Transfer
SaaS	Software-as-a-Service
SLA	Service Level Agreement
SOE	Slicing and Orchestration Engine
SUC	System User Cases
Tx.y	Task y of WP x
UI	User Interface
VIM	Virtual Infrastructure Manager or hypervisor
VNF	Virtual Network Function
V2X	Vehicle-to-Everything
WP	Work Package

Executive Summary

This deliverable is the final updated version of the initial **deliverable** "D3.2 – Edge/Cloud orchestration tools"[1]. In this deliverable the updates and developments of the task T3.2 "Edge/Cloud orchestration tools" components are presented.

T3.2 focuses on edge/cloud orchestration tools and the necessary enhancements required to deliver the required Pledger functionality. With REST-based interfaces and graphical support, the emphasis is on usability. Increased configurability and integration capabilities are also planned, integrating to the database backend and enabling on-the-fly installation of user-defined strategies for edge/cloud provider selection based on "T3.1 – Performance Measurements and Classification" findings. Another goal is to optimize the application life cycle, which allows for the update of previously deployed software components. This enables ongoing functioning of edge/cloud applications and is required for project commercialization. This job also focuses on the difficulty of anticipating the dynamics of edge computing compute based on application Quality of Service (QoS) and Quality of Experience (QoE), hence allowing edge/cloud orchestration.

Following the Task's activities, this deliverable focuses on the developments of Pledger **Orchestration Subsystem**, one of Pledger's main Subsystems. This Subsystem is in charge of handling the orchestration of containerized applications (activities related to application deployment, infrastructure scaling up/down, and so on)., as well as the lifecycle management of network slices. This deliverable also includes components of the **Configurations Subsystem**, with the major emphasis on application profiling for IaaS resource management. More specifically this document presents the updates on the functionalities, baseline technologies, architecture, as well as updates on the interfaces and data models of the four tools reported in the context of this task. These components are:

- **Orchestration Subsystem**: Orchestrator, Slicing and Orchestration Engine, Radio Access Network Controller Framework and Monitoring Engine
- **Configuration Subsystem:** Application Profiler

Moreover, this deliverable contains all the necessary information needed for the installation and usage of these components and also demonstrators to showcase the usage and functionalities of these components.

1 Introduction

1.1 Purpose of the document

The scope of the current deliverable (D3.5) is to report and provide the updated functional and technical description of the components, tools and services of the Pledger Orchestration and parts of Configuration Subsystem. This deliverable is an update of D3.2 [1]which was the initial deliverable presenting the 1st iteration prototypes of the components. This deliverable also provides the installation and usage guidelines of the tools and components of the Subsystems as well as demos of the main functionalities.

1.2 Relation to other project work

T3.2 and by extent all the components that were developed or extended in the context of this task are tightly related and integrated with all the components of this WP (T3.1, T3.3). App Profiler communicates with Benchmarking Subsystem order to run all the necessary benchmarks for the training of the machine learning models. Also Orchestrator is tightly coupled with the SLA manager of T3.3 in order to send monitoring information and also to be notified about the SLA status of the services that are running and managed by the Orchestrator.

It is also important to mention that the components of T3.2 are also coupled with WP4 components mainly with the Decision Support System (DSS) of T4.3. As far as T3.2 is involved with this relation, the Orchestrator component is the actuator of DSS decisions and App Profiler aids DSS in the decision making process by characterising the applications or services based on their resource usage. This integration work is part of the WP5 and is documented and demonstrated in its associated deliverables.

1.3 Structure of the document

This document is structured in 6 major chapters:

- **Chapter 2** discusses the functional design choices of the components in alignment with the overall architecture of Pledger and describes the objective, functionality, and the interaction with the other modules in the platform.
- **Chapter 3** describes the technical aspects of the components (data model, interfaces, and component architecture) and the progress with respect to the initial baseline made in this period.
- **Chapter 4** documents the installation and usage guide of the components.
- **Chapter 5** shows demonstration of the components with the description of the validation scenarios.
- **Chapter 6** summarize the conclusions and next steps for this first iteration of the components.

2 Functional description

This chapter describes the main functional components and their updates in terms of functionality, interoperability and overall technical implementation within Task "T3.2 – Edge/Cloud Orchestration tools" and provides useful information related to the development of the Pledger **Orchestration & Configuration Subsystems.** The Orchestration and Configuration Subsystems are part of the Pledger Core system (see Figure 1 below), as identified in the deliverable "D2.3 Pledger Overall Architecture"[2], which acts as the roadmap for all development and integration tasks of the project. As far as the general architecture of Pledger is concerned all the components of T3.2 are positioned in the Management Subsystem. Management Subsystem interacts directly with **Evaluation Subsystem** and indirectly with the Support Subsystem. There are no major updates in the general position and interaction of the Subsystems of Pledger. The main updates are inside the components that constitute these general Subsystems (**Orchestration & Configuration Subsystems**).

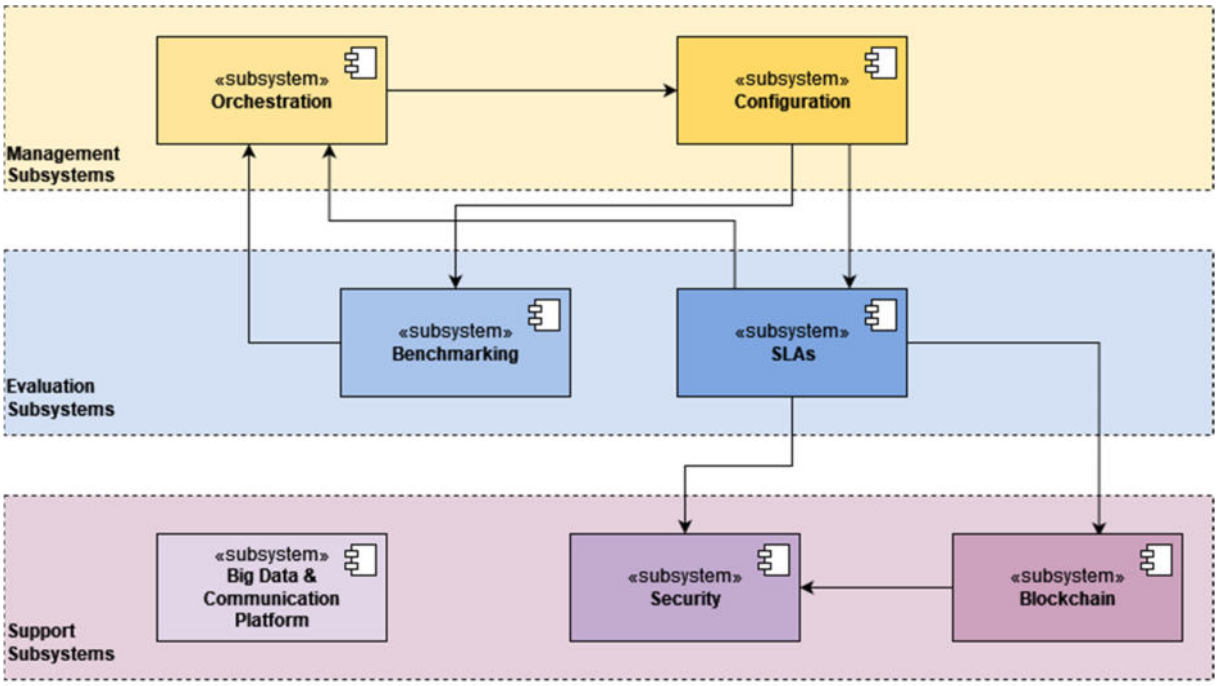

Figure 1: The Pledger Core Subsystems [2]

2.1 Orchestration Subsystem

The Orchestration subsystem is in charge of managing and operating the lifecycle of applications in the Edge-Cloud Continuum. This management covers not only the first deployment of applications but also the updated deployment of applications and runtime adaptations of the computational resources serving to these applications. This subsystem provides an abstraction layer to other components of the Pledger platform (e.g., DSS or Recommender) decoupling the technical details of the underlying infrastructure from the business logic of the Pledger platform.

Moreover, this subsystem acts as a link to other Virtual Infrastructure Managers (VIMs), container/process orchestrators or infrastructure resource controllers with proprietary APIs or interfaces, translating functional lifecycle management operations (like start, stop, scaling, among others) to the specific language of the underlying resource manager. With this architecture the subsystem and the Pledger platform might be extended to other types of resource managers implementing new adapters/plugins.

For this new release of this deliverable, we include the following improvements to the Orchestrator subsystem:

- Further integration with the SOE and RAN controller enabling the Pledger platform to orchestrate not only application workloads but also network slices and radio connectivity. Now we can provision on demand network slices, deploy VNFs compute components and setup radio access connectivity.
- The Orchestrator manager can now manage VM lifecycle operations on demand connecting with a hypervisor controller at the edge site. One of the challenges to this functionality is the lack of a standard API to interact with hypervisors or VIMs. This forced the implementation of a proprietary API of the chosen hypervisor. This integration is demonstrated within UC1 with a Linux KVM based hypervisor in the context of WP5.
- More visibility to the application lifecycle operations (deploy, start, stop, placement, scaling, etc.) is given using StreamHandler [28] as the channel to publish feedback about the last operation. In this way, any component of the Pledger platform could subscribe to this feedback topic to be aware of the operation result executed at one remote infrastructure. For example, app profiling of a new deployed application can be automatically launched with this event-driven approach.
- The Monitoring Engine component is not anymore part of the Orchestrator component and now it has its own development lifecycle. This is because basic functionalities of this component have been increased simplifying the queries for metrics values of different repositories with a centralized approach and decoupling producers and consumers of metrics using a middleware that allows indirect connections with the edge telemetries of the UCs. Now this component could be seen as a single point of service for store and retrieve metrics in the Pledger platform and an Extract, Transform and Load process (ETL) process to collect system and applications metrics from remote edge infrastructures and applications.

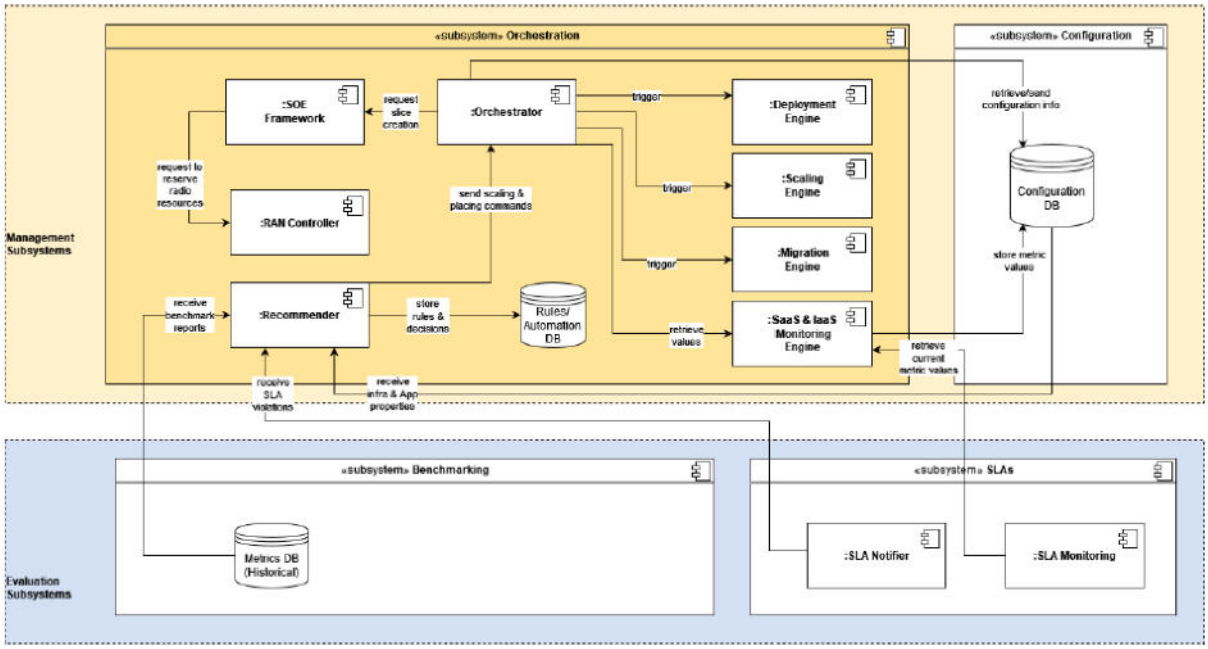

Figure 2: Orchestration subsystem Component Diagram and relation to other subsystems

During this iteration of T3.2, as part of the SOE's developments, all the extensions related to the use of Kubernetes as a virtual infrastructure manager have been completed, involving the development of specific API calls from the slice manager, as well as some developments related to the flows between OSM MANO and Kubernetes. OSM MANO is used by the SOE for the deployment of network functions.

Further integration between the SOE and RAN controller now allows the Pledger platform to orchestrate not only application workloads, but also network slices and radio connectivity. Now Pledger can

provision on demand network slices, deploy compute components based on virtualized network functions, and set-up radio access connectivity.

The set of new API calls are used for integration with the Orchestrator so that, when a request for an end-to-end slice deployment is made by the Recommender, this reaches the Orchestrator which, in turn, delegates the task to SOE. The same flow takes place when a request for a service deployment over a dedicated slice is made, and dedicated API calls have been developed for this task too.

In addition, dedicated API calls have been developed on the northbound interface of the RAN controller, such that it can be integrated with SOE for the instantiation of network slices including radio elements based on the protocol IEEE 802.11p.

Moreover, additional standard API calls for the instantiation of network slices with 5G radio elements are being developed. The SOE and RAN controller will enable the deployment of end-to-end network slices based on 5G new radio (NR), in such a way that two (or potentially more) mobile operators can allocate a share of the same infrastructure and deploy their own core functions within their own slice. A proof of concept demonstrating the support for such 5G-enabled deployments using SOE's and RAN controller's 5G-related functionalities will be showcased as part of WP5 developments.

Table 1 introduces the general description and updates of the Orchestration subsystem components, while in Section 3 a more in-depth description of the technical and development updates of these components can be found.

Table 1: Orchestration Subsystem components

ID	Component	Functionality
FC.2.1	**Orchestrator**	As defined in D3.2 [1], "This component is in charge of managing the information related to runtime environment specifications (profiles), such resources preferred/ required (like RAM, vCPU, etc.) for the applications, and of executing basic operations like start, stop, update or get status of applications in a cluster or infrastructure". **Updates:** In this release the interface with the SOE and RAN controller is included extending the features of the Pledger platform not only to computing orchestration but also to network orchestration.
FC.2.2	**Deployment Engine**	As defined in D3.2 [1], "This component is in charge of deploying and removing the applications to the physical infrastructure, either in a swarm or different type of infrastructure. The "deployment" could be a first deployment, a redeployment due to runtime specification changes or an update of the version of the application". **Updates:** In this release the deployment of VMs at the edge is included.
FC.2.3	**Scaling Engine**	As defined in D3.2 [1], "This component implements a horizontal scaling (scale out/ in) of the application creating replicas of the same application (or subcomponents of the application in case of microservices, Y axis in scale cube [29]) or decreasing replicas. It also implements a vertical scaling (scale up) by calling the Migration Engine component to move the app to a node of the cluster with more resources (CPU, RAM, etc.) or with less resources (scale down)". **Updates:** In this release the vertical scaling of VMs is included.

ID	Component	Functionality
FC.2.4	**Migration Engine**	As defined in D3.2 [1], "This component implements the migration of applications already deployed, by moving the app to different infrastructure with more resources (e.g., edge to cloud, other node type with more resources like RAM, CPU) or with less resource (e.g., cloud to edge). The reasoning for this migration covers different needs like colocation for decreasing latency, lift & shift for more computer capacity, etc." **Updates:** In this release the migration from edge to cloud, cloud to edge, cloud to cloud and between Docker and Kubernetes workloads is included.
FC.2.5	**Monitoring Engine**	As defined in D3.2 [1], "This component collects metrics of cluster infrastructure and stores these values in a time series database. It is also the query interface for metric values from other subsystems or components". **Updates:** In this new release this component has its own development lifecycle and includes an ETL subcomponent to collect metrics from the edge via the StreamHandler component and store them in the central metric data store of Pledger.
FC.2.6	**Recommender**	Reported in Task "T4.3 – Decision Support mechanisms for Edge/Cloud computation moving" in the deliverable D4.3 Decision Support tools I[10]
FC.2.7	**Rules/ Automation DB**	Reported in Task "T4.3 – Decision Support mechanisms for Edge/Cloud computation moving" in the deliverable D4.3 Decision Support tools I[10]
FC.2.8	**SOE Framework**	The Pledger SOE Framework is responsible for processing slice creation. The SOE is now capable of deploying Kubernetes-based compute chunks and end-to-end network slices with radio elements based on the IEEE 802.11p protocol. Moreover, the SOE is capable of deploying OpenStack-based compute chunks and end-to-end network slices with radio elements based on 5G NR. This is in line with our vision of cloud-native network slicing, where a deployment-driven approach is followed [11], where resource sharing is decoupled from runtime constructs, while service creation and instantiation can be facilitated by a network function virtualization orchestrator (OSM MANO, in our case). **Updates:** Specifically, the developments performed during the last iteration of D3.2 are as follows: ▸ Design and development of a Python-based client for Kubernetes for certain SOE functionalities, e.g., the deployment of dedicated compute chunks. ▸ Development and extensions of a Python-based client for OSM MANO, which is used by SOE for functionalities related to the management of network-services-based applications. ▸ Integration of both the Kubernetes and the OSM MANO clients into SOE, including the development of required API endpoints, database management for the persistence of relevant parameters in a dedicated database, etc. ▸ Integration of all of the above developments within the API, business and models layers of SOE as necessary.

ID	Component	Functionality
		Implementation of additional API calls and integration with RAN controller for the deployment of end-to-end slices containing radio elements based on the 5G NR and on the IEEE 802.11p protocol.
FC.2.9	**RAN Controller**	The management and configuration of radio devices of an infrastructure is the responsibility of the Pledger RAN controller. Interfacing the Pledger SOE Framework, the RAN controller can be requested to reserve radio resources for a slice and to configure them to enable radio access connectivity as part of a service. **Updates:** During the last iteration of T3.2, the RAN controller has been extended to support IEEE 802.11p technology, involving developments related to the business layer logic, YANG model and NETCONF manager extension[33]. In addition, adaptations have been made in the SDN management and data planes for the integration of IEEE 802.11p radio devices. Moreover, relevant updates are being made in the scope of WP5 in order to support the deployment of network slices with 5G NR elements.

2.2 Configuration Subsystem

The **Configuration Subsystem,** as part of the **Management Subsystem,** is responsible to store the infrastructure and application configuration that is managed either manually by the IaaS/SaaS (Infrastructure/Software as a Service) providers or by tools that support automatic discovery features (e.g., the App Profiler). Part of the **Configuration Subsystem** is a T3.2 component called App Profiler (**FC.1.3**). This component is responsible for retrieving resource usage monitoring data and process them in order to create the application profiles which will be used by the recommender (DSS) in the deployment decision process.

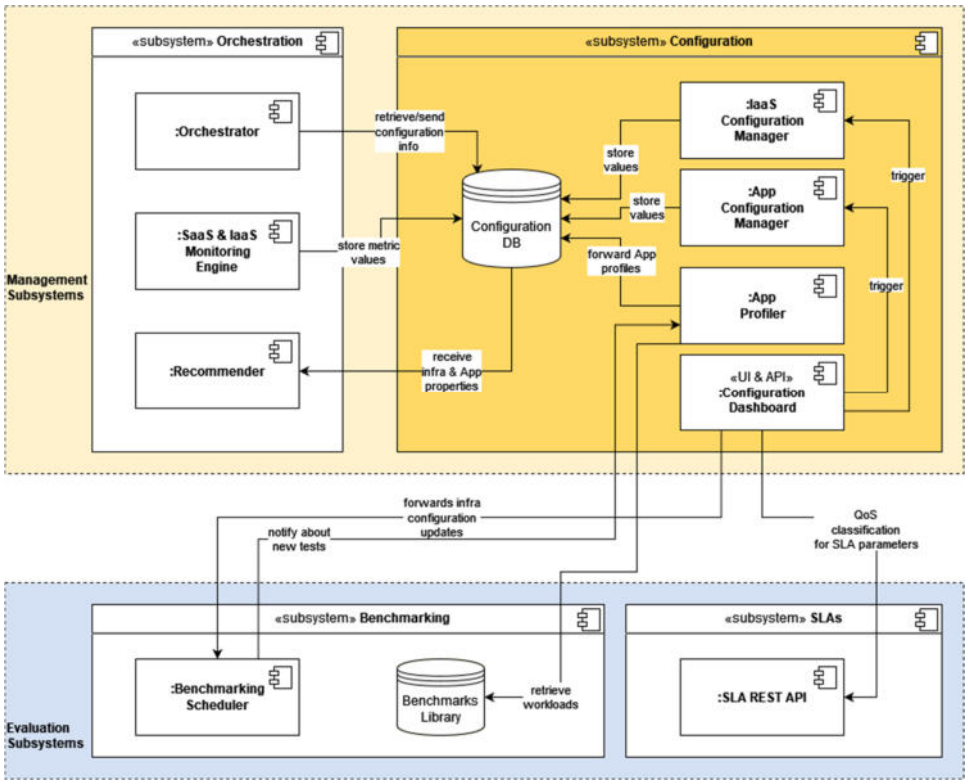

Figure 3: Configuration Subsystem component diagram and relation to other Subsystems

There is only one major update for the App Profiler component as far as the general Pledger architecture is concerned. In order to access the specific container info in a Kubernetes or Docker environment, App Profiler communicates with Orchestrator in order to retrieve the deployment information needed for the profile creation.

The table below (Table 2) describes the main functionalities of the Configuration Subsystem components; in this deliverable the main focus is on FC.1.3 App Profiler which is developed in the context of T3.2. All the other functional components of this Subsystem are reported in T4.3 related deliverables.

Table 2: Configuration Subsystem components

ID	Component	Functionality
FC.1.1	IaaS Configuration Manager	Reported in Task "T4.3 – Decision Support mechanisms for Edge/Cloud computation moving" in the deliverable D4.3 Decision Support tools I[10]
FC.1.2	App Configuration Manager	Reported in Task "T4.3 – Decision Support mechanisms for Edge/Cloud computation moving" in the deliverable D4.3 Decision Support tools I[10]
FC.1.3	**App Profiler**	The App Profiler component is responsible for the profiling of applications, based on their resource usage footprint on the underlying infrastructure. **Updates:** Functionality wise, App Profiler automatically creates profiles in a seamless way, without any additional user input after the initial deployment using the Orchestrator. Furthermore, App Profiler provides an API in order to parametrise the profile creation for a more detailed profile creation, in order to aid the adaptation of the tool.
FC.1.4	Configuration DB	Reported in Task "T4.3 – Decision Support mechanisms for Edge/Cloud computation moving" in the deliverable D4.3 Decision Support tools I[10]
FC.1.5	Configuration Dashboard	Reported in Task "T4.3 – Decision Support mechanisms for Edge/Cloud computation moving" in the deliverable D4.3 Decision Support tools I[10]

3 Technical description

This deliverable and this section are a continuation of the deliverable D3.2 [1] and it includes some descriptions from this deliverable to facilitate the comprehension of the changes of this release.

3.1 Baseline technologies and dependencies

3.1.1 Orchestrator (E2CO)

Table 3: Baseline technologies used by E2CO tool (Orchestrator subsystem)

Name	Description	Version
Lifecycle Manager [4]	As defined in D3.2 [1], "An Edge-To-Cloud Resource Orchestration tool that provides life-cycle management for composed services described as collections of containers and relying in container orchestration tools such as Docker Swarm and Kubernetes. It is being developed by ATOS under an open-source license (Apache 2.0), and it has been used in other H2020 projects".	Latest
Prometheus [5]	As defined in D3.2 [1], "Prometheus is an open-source monitoring system, written in Golang, and released with Apache 2.0. This tool can be integrated with container orchestrators like OpenShift and Kubernetes, and it can get metrics from the infrastructures and applications".	Prometheus Operator Stack latest version
Kubernetes [6]	As defined in D3.2 [1], "Kubernetes is an open-source container orchestration system for automating computer application deployment, scaling, and management, licensed under Apache 2.0. Originally designed by Google, Kubernetes is now maintained by the CNCF (Cloud Native Computing Foundation)".	Vanilla 1.19+
Grafana [7]	As defined in D3.2 [1], "Grafana is a data visualization and monitoring tool used to show data from external sources, like Prometheus".	latest
Docker [8]	As defined in D3.2 [1], "Container engine for containerized apps running in the edge".	1.18+
Kafka[9]	As defined in D3.2 [1], "Apache Kafka is an open-source distributed event streaming platform used for high-performance data pipelines and streaming analytics. It is part of the StreamHandler platform that handles data management in Pledger, and as such, interfaces with multiple components in the Pledger Core and Use Case deployments. Kafka is selected for its high throughput and scalability".	latest
Updates on baseline technologies in 2nd iteration		
SOE Framework	As defined in D3.2 [1], The Pledger SOE Framework is responsible for processing a slice creation and service deployment request, upon which an end-to-end slice including radio and compute elements of the infrastructure is generated.	latest

Table 4: Orchestrator dependencies with other components

Pledger Component	Interaction
Recommender or Decision Support System (DSS) component from Orchestration subsystem.	As defined in D3.2 [1], "Control commands of the app lifecycle to execute in the undelaying infrastructure because of the decision taken by the DSS".
Configuration subsystem	As defined in D3.2 [1], "Update internal metadata of available infrastructures and app/services in the system". In this release also the endpoint of the Prometheus associated with the application or infrastructure metrics for the first deployment.
SOE component from Orchestration subsystem.	As defined in D3.2 [1], "Send request to the **Slice Orchestration Engine** (SOE) to create and manage a slice in the network".

3.1.2 App Profiler Component

The following baseline technologies are used within the context of App Profiler in order to facilitate all the required functionalities and communication methods. One major update in these technologies is the change on the tools that are responsible for creating the Machine Learning (ML) models for the application classification. More specifically the Python libraries for ML model creation NumPy, Pandas and scikit-learn have been substituted by Weka [27] a ML Software in Java. Weka is able to process much bigger datasets especially through the command line interface and it can easily optimize the models and compere the models against a variety of other machine learning algorithms in order to produce an optimal result.

Table 5: Baseline technologies used by App Profiler

Name	Description	Version
Node-RED[32]	Node-RED is a flow-based development tool for visual programming developed originally by IBM (Apache License 2.0) for wiring together hardware devices, APIs and online services as part of the Internet of Things. Node-RED provides a web browser-based flow editor, which can be used to create JavaScript functions.	1.3.4
Docker[8]	Docker is a set of platforms as a service product that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.	20.10.6
NumPy, Pandas, scikit-learn	Usage of these python libraries has been discontinued from App Profiler.	N/A
Updates on baseline technologies in 2nd iteration		
Tiny-weka (2nd Iteration)	Weka is a collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization. Weka is open source software issued under the MIT Licence.	3.9

In the first iteration of App Profiler there were two main dependencies with other Pledger components, the Benchmarking Suite and the DSS. As far as these dependencies are concerned, the main interactions are fully developed and tested and no significant changes were made in the second iteration. A significant update on the interactions of App Profiler with other Pledger tools, is the connection of App Profiler with the **Orchestrator** in order to obtain the deployment information of a specific application.

Table 6: App Profiler dependencies with other components

Pledger Component	Interaction
Benchmarking Suite	As described in D3.2 [1]App Profiler uses benchmarks in order to classify applications, creating and training the ML models require a substantial dataset of benchmark profiles. Benchmarking Suite allows App Profiler to run a plethora of deferent benchmarks and workloads in deferent containerized environments (Docker, Kubernetes). Benchmarking Suite configures and runs the benchmarks and calls through a REST API App Profiler in order to extract the profile while the containerized benchmark is running.
Decision Support System (DSS)	As described in D3.2 [1], after App Profiler extracted the resource usage of an application, the DSS will be notified about the classification of this application. DSS obtains this information and makes an informed decision on where the application can be deployed based on the benchmark that the application resembles.
Orchestrator (E2CO)(2nd Iteration)	In order for the App Profiler to be able to access and extract infrastructure resource usage metrics for a specific application or service, it needs information on where the service is deployed, endpoints for communicating with Prometheus or Docker CLI, as well as unique identifiers for the service that it needs to be profiled. To obtain this information App Profiler interacts with Orchestrator and more specifically deployment engine of Orchestrator component, which can provide this information. In this way no additional user input is required for the profiler to operate seamlessly.

3.1.3 Slicing and Orchestration Engine (SOE) and RAN Controller Framework

SOE follows a modular architecture including a slice manager, multi-tier orchestrator, a network function virtualization orchestrator (OSM MANO), and a virtual infrastructure manager (Kubernetes and OpenStack), while also leveraging the use of MongoDB as a database. The slice manager and multi-tier orchestrator modules are explained in some more detail below, and all of the baseline technologies used by SOE and the RAN controller are summarized in Table 6.

- The slice manager is responsible for the logical partition and configuration of infrastructure resources, following a soft and deployment-driven approach. Specifically, the slice manager handles the registration and setup of computation, networking and radio resources. All infrastructure resources are registered in the slice manager's database. This relies on API-based communication with the virtual infrastructure manager (Kubernetes, OpenStack) and the RAN controller. Infrastructure partitions (i.e., chunks) are also registered in the slice manager's database. Albeit the creation of radio chunks is delegated to the RAN controller, these are also stored in the slice manager's database. In addition, network slice instances are also stored in the slice manager's database as collections of compute, network and radio chunks. Fig. 1 represents the slice manager's northbound and southbound flows, as well as the MongoDB integration.

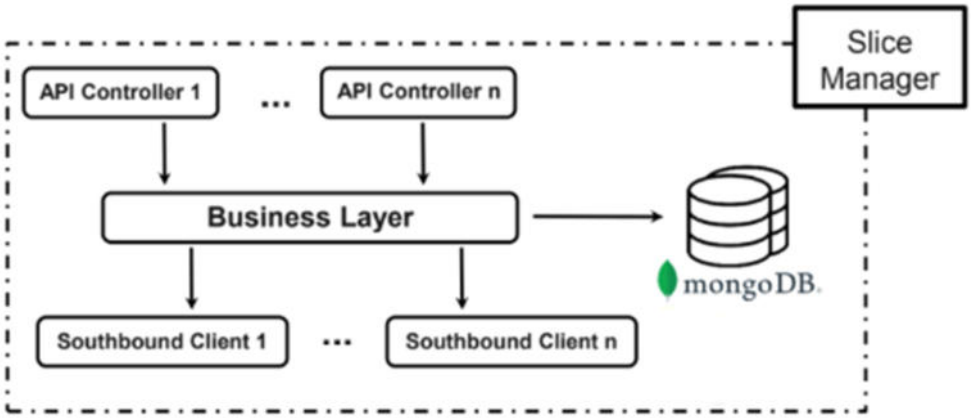

Figure 4: Slice manager flows

- The multi-tier orchestrator provides a single abstraction point between the slice manager and both the network function virtualization orchestrator (OSM MANO) and the virtual infrastructure manager (Kubernetes, OpenStack). The multi-tier orchestrator communicates with OSM MANO, Kubernetes and OpenStack through RESTful API calls, and it is responsible for the triggering of some high-level actions, namely the onboarding, instantiation and monitoring of network services and platform rules and configurations across the different orchestrators that lie underneath.

The SDN-enabled RAN controller leverages the use of NETCONF, Netopeer2, Open vSwitch, Prometheus, and Grafana, and supports the management and control a set of radio technologies to create radio chunks within a slice and connect them to the compute services.

Physical radio nodes that are to be managed, need to be registered inside the RAN controller. Each of these radio nodes can be equipped with a single or multiple radio interfaces. These radio interfaces are the manageable resources that can be included in a radio chunk to form part of a slice. The way these resources can be reserved and later on actively used, depends on the specific radio technology. Specifically, IEEE 802.11p interfaces use an "Outside the Context of a Business support system" (OCB) wireless operation mode to provide connectivity and cannot be virtualized. As such, the resource to be included in a chunk when using IEEE 802.11p would be the physical radio interface(s) and the radio service would be activated by configuring the physical radio interface(s) to operate in OCB mode. Other radio technologies are reserved and activated differently, e.g., dedicated Service Set Identifiers are radiated in the case of Wi-Fi for each service and they could be virtualized, offering a dedicated SSID for different services on top of the same physical interface. Another example is the Public Land Mobile Network Identifier (PLMNID) radiated in the case of 4G and 5G to provide data connectivity to users registered to the service. Different PLMNIDs could be radiated over the same physical radio, each assigned to a different slice and connected to different cores as per a *Muliple Operator Core Network* approach. The radio resources available to be reserved and configured to form part of a radio chunk are the ones that have been registered to the radio controller (this forms part of the infrastructure configuration). All the above baseline technologies and protocols used are summarized in Table 7:

Table 7: Baseline technologies used by SOE and RAN controller

Name	Description	Version
Slice Manager (SM)	As defined in D3.2 [1], it enables virtualized non-virtualized infrastructure resources to be logically partitioned, configured and reused (isolated from one another) for the creation of slices. Each slice has its own specific compute and network resource chunks that meet the SLA demands of each network service offered by it.	

Name	Description	Version
Multi-Tier Orchestrator (MTO)	As defined in D3.2 [1], the Slice Manager enables virtualized non-virtualized infrastructure resources to be logically partitioned, configured and reused (isolated from one another) for the creation of slices. Each slice has its own specific compute and network resource chunks that meet the SLA demands of each network service offered by it.	
MongoDB [34]	As defined in D3.2 [1], the Multi-Tier Orchestrator (MTO) lies between the slice manager and the network virtual function orchestrator (OSM MANO). It provides a single abstraction point between the slice manager and any other domain (e.g., MEC and cloud-native orchestrators) that needs to be integrated into the system. Moreover, it is responsible for the triggering of high-level actions such as onboarding, instantiation, and monitoring of (network) services and platform rules and configurations across the different orchestrators that lie underneath.	
Kubernetes	As defined in D3.2 [1], MongoDB is an open source NoSQL database used by SOE, that uses JSON-like documents with optional schemas.	Vanilla 1.19+
OSM [12]	As defined in D3.2 [1], in order to create container-based slices, SOE communicates with Kubernetes via its own Kubernetes client which, among other actions, handles the provision the network slice (isolation, dedicated resources, etc.) where OSM will instantiate the vertical services and apps.	R8
OpenStack[35]	As defined in D3.2 [1], OSM MANO is an ETSI-hosted open source community delivering a production-quality network function virtualization management and orchestration stack. SOE, via its southbound logic layer, communicates with OSM MANO to perform the instantiation of the vertical services and apps within these network slices.	
NETCONF [13]	As defined in D3.2 [1], OpenStack is an open source platform that uses pooled virtual resources to build and manage private and public clouds. Resources such as storage, CPU, and RAM are abstracted from a variety of vendor-specific programs and split by a hypervisor before being distributed as needed.	
Netopeer2[14]	As defined in D3.2 [1], NETCONF, designed by the IETF, is a well-known protocol used for the configuration and management of network devices. This protocol is used to register radio devices with the RAN controller, to configure them and to manage them. A NETCONF agent is required on the radio nodes for these interactions.	
OpenDaylight[15]	As defined in D3.2 [1], Netopeer2 is the NETCONF agent deployed on custom radio devices that are managed by the RAN controller.	
OpenFlow Protocol [16]	As defined in D3.2 [1], the OpenDaylight (ODL) Software-Defined Networking (SDN) controller is used by the RAN Controller for the configuration and management of radio nodes used in Wi-Fi-based wireless backhauling. Each node in such a wireless backhaul mesh is represented as a virtual switch, on top of which ODL operates for the configuration of tunnels that form part of an end-to-end network slice.	

Name	Description	Version
Open vSwitch [17]	As defined in D3.2 [1], the OpenFlow protocol is used by the RAN controller to manipulate the flows (forwarding rules) of the virtual switches on radio nodes that are managed by the ODL SDN controller, so that Layer-2 based tunnels can be generated.	
Radio technologies	As defined in D3.2 [1], Open vSwitch (OvS) instances run on the radio devices and are used to enable SDN-based data forwarding between wireless devices or wireless devices and compute nodes.	
Prometheus and Grafana	As defined in D3.2 [1], the RAN controller supports a variety of wireless technologies that can be included in end-to-end network slices created and managed by the SOE: Wi-Fi for RAN, Wi-Fi for backhauling, 4G (LTE), and 5G NR. For each technology, specific vendors or custom hardware designs that have gone through an integration process are supported by the RAN controller. Each radio technology might have further requirements, such as needing the deployment of an evolved packet core (4G) or a 5G core as part of a slice.	

Table 8 presents the main dependencies of SOE and RAN controller with other components.

Table 8: SOE and RAN controller dependencies with other components

Pledger Component	Interaction
Orchestrator (E2CO)	Integration of E2CO with the SOE's Framework northbound API interface. For this purpose, new functionalities related to the management of resources and services in Kubernetes will be added to the standard northbound interface of the SOE framework.
Recommender	The ConfService component of the Recommender passes over certain configuration parameters to the E2CO, which are in turn consumed by the SOE.
Radio Technology portfolio supported by the SOE and RAN controller framework	Support for vehicular radio technology (IEEE 802.11p), including support of VLANS in Kubernetes. Support for 5G NR technology.

3.1.4 Monitoring Engine

The following baseline technologies are used within this component:

Table 9: Baseline technologies used by Monitoring Engine tool (Orchestrator subsystem)

Name	Description	Version
Prometheus	As defined in D3.2 [1], "Prometheus is an open-source monitoring system, written in Golang, and released with Apache 2.0. This tool can be integrated with container orchestrators like OpenShift and Kubernetes, and it can get metrics from the infrastructures and applications".	Prometheus Operator Stack latest version
Kubernetes	As defined in D3.2 [1], "Kubernetes is an open-source container orchestration system for automating computer application deployment, scaling, and management, licensed under Apache 2.0. Originally designed by Google, Kubernetes is now maintained by the CNCF (Cloud Native Computing Foundation)".	Vanilla 1.19+
Kafka	As defined in D3.2 [1], "Apache Kafka is an open-source distributed event streaming platform used for high-performance data pipelines and streaming analytics. It is part of the StreamHandler platform that handles data management in Pledger, and as such, interfaces with multiple components in the Pledger Core and Use Case deployments. Kafka is selected for its high throughput and scalability".	latest

In the previous release of the Pledger platform this component was designed as a subcomponent of the Orchestrator component, but it was getting more relevance and functionalities as the project progressed. For this reason, we decided to create a separate component with its own identity.

Table 10: Monitoring Engine dependencies with other components

Pledger Component	Interaction
Recommender or Decision Support System (DSS) component from Orchestration subsystem.	Monitoring Engine acts as a single point of contact (SPC) to retrieve metrics values if DSS needs to query some metrics within a range of time.
Configuration subsystem	The endpoint of the Prometheus associated with the application or infrastructure metrics for the first deployment.
SLA subsystem	Retrieve metrics values in the evaluation phase of any SLA guarantee (periodic job) from the Monitoring Engine.

3.2 Components description and architecture specification

3.2.1 Orchestrator (E2CO)

Orchestrator has reached the expected level of maturity form the first iteration of the WP3 components, in this section all the important updates are highlighted. The main developments and architecture specification of Orchestrator can be found in D3.2[1].

This component provides an abstraction layer on top of infrastructure management tools, decoupling the rest of Pledger components from the technical implementation details and facilitates interoperability with diverse technical solutions or orchestrators/VIMs/engines.

The main functionality is the orchestration of containerized applications: deployment, scale up/down, placement, updates, etc. We also include the orchestration of VM using proprietary API of Hypervisors. Also, the management of clusters at different infrastructure levels (cloud, edge, on premise).

Progress in 2nd Iteration

As mentioned earlier in section 2, for this new release we include the following improvements to the Orchestrator component:

- Further integration with the SOE and RAN controller that allows the Pledger platform to orchestrate network slices and radio connectivity.
- The Orchestrator manager can now manage VM lifecycle operations on demand connecting with a hypervisor controller at the edge site.
- We provide more visibility to the application lifecycle operations (deploy, start, stop, placement, scaling, etc.) using StreamHandler as the channel to publish some feedback about the last operation.

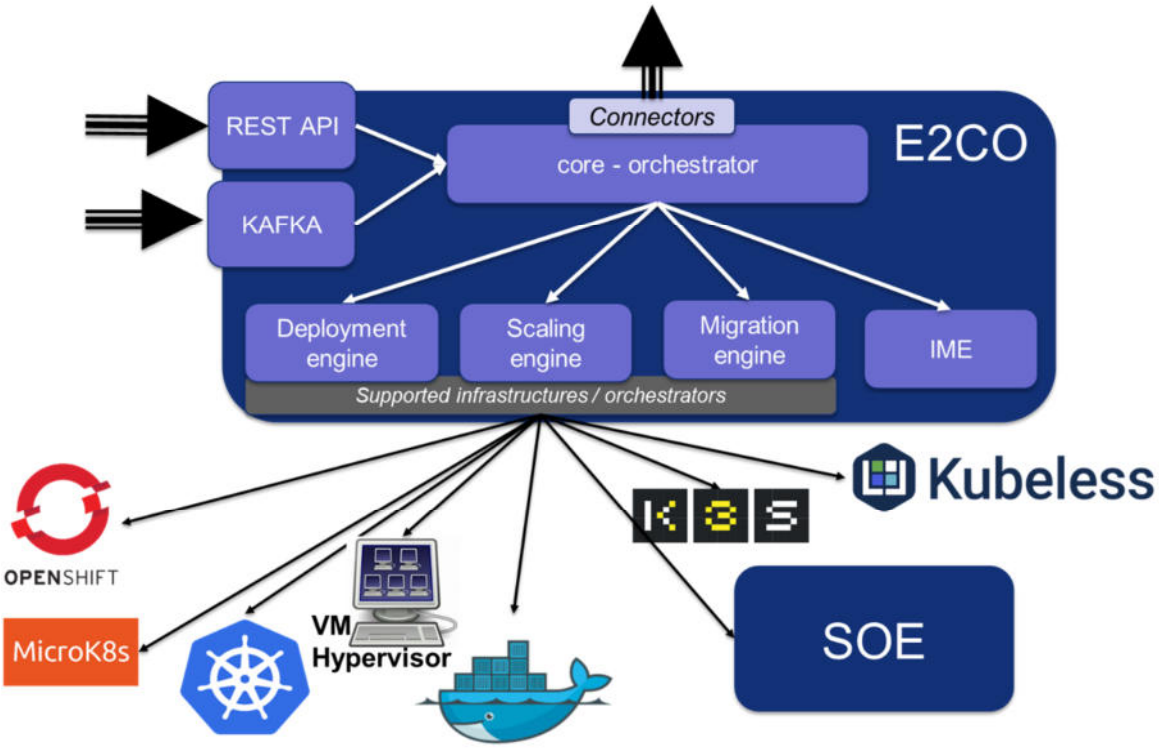

Figure 5: Orchestrator subsystem implemented app internal modules

3.2.2 App Profiler Component

The app profiler component is in charge of profiling application/services that are deployed and utilized as containers. App Profiler collects a containerized instance's resource consumption statistics and translates them to the resource usage of a known benchmark. Benchmarks are vastly used to examine the computing capabilities of specific hardware since they are designed to be simply installed and completed. This is not the case for generic applications; consequently, mapping applications to particular benchmarks using the App Profiler can aid in the formation of fundamental knowledge about how a certain program can perform on different platforms based on established benchmarks.

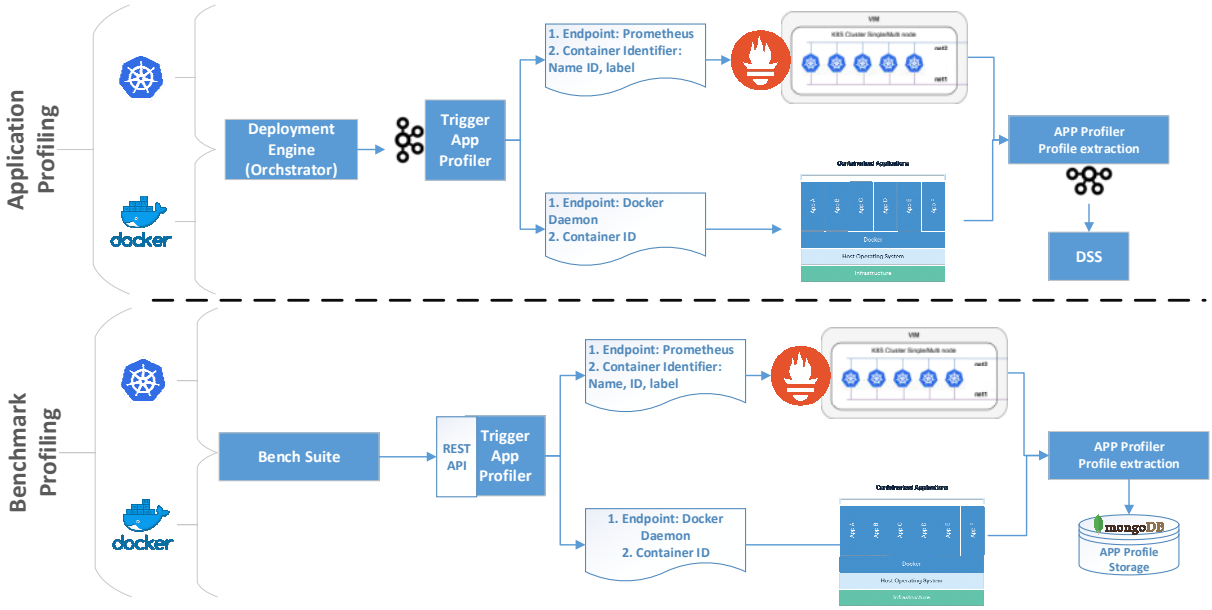

Figure 6: Application Profiler architecture and interactions

The application profiling system is built with the Node RED flow-based programming framework. This framework has a wide range of distinct nodes that aid in the communication of various components in order to achieve the desired functionality. Given the variety of cloud-edge hardware and APIs, it also supplies the essential communication nodes for data interchange and external resource management. As seen in the image above (Figure 6), the application profiling system works with various container environments (Docker, Kubernetes).

To function, the application profiling system requires two key components: the profiling component and the model trainer component. The multidimensional vector that reflects the profile of a benchmark or application's resource utilization is created by the profiling component. The model trainer component is in charge of developing the categorization model for the applications. All of the data from the profiles is put into the classifier trainer; a portion of this data is utilized for training, while the other subset is used to assess the validity of the created model.

Application Profile Creation

Application profiling is fundamentally the production of a multidimensional vector, with each dimension indicating a resource utilization measure; these dimensions are referred to as profile features. Features might be basic metrics like network packets received or bytes written, or they can be complicated metrics like Delta of CPU time, geometrical mean values, and other complex metrics that need sample collection and computations to be constructed. A profile must be able to be classified and categorized based on the values of its features in order to have any value. To be relevant in the categorization process, features must have three attributes:

- **Consistency in the Values:** Every feature computed by a benchmark's metric extraction should provide roughly the same result (minimum deviation). This indicates that the feature's value should be in the same order of magnitude for each iteration of profile extraction for a certain benchmark or application and workload.
- **Feature values differentiation:** Although the features should have similar values for each benchmark and workload, they should also have distinct values for various sets of benchmarks and workloads. The classification method requires features that can distinguish one benchmark from another in order to function effectively and accurately. Given that the random forest technique is utilized, this is

handled by the algorithm itself. If the algorithm determines that a feature is worthless because it does not clearly represent a distinction, the binary tree that accesses that feature is discarded.

- **Hardware agnostic:** Even if the features meet the two preceding characteristics, they must still be disassociated from the hardware configuration. Even if a feature has consistent values for a particular benchmark and workload, this consistency should be maintained with a minimum of variance even if the benchmark runs on several hardware configurations. If the characteristics lack this property, the classification process will be relevant and accurate only if the application or benchmark is profiled in the exact same configuration, limiting the scope of the research effort accomplished.

Classification Process

There have been advancements in the field of artificial intelligence (AI) and more complex training models, like artificial neural networks, work best when the data is high-dimensional, but the training takes a lot longer. Since the number of features that can describe an application's impact on the hardware is small and falls into specific categories (CPU, RAM, Storage IO, Network), ML-type classification algorithms were better [3] for this work especially if the profiles have the mentioned attributes. App Profiler used decision trees in the first iteration of the component, but after obtaining a substantial dataset for testing Random Forest Classifiers where chosen based on their accuracy and fast model creation.

More specifically the dataset created in order to train and test the App Profilers (Random Forest Classifier) accuracy and overall performance, consists of 2050 vectors (from 63 different benchmarks) obtained using three different hardware configurations. The exact specifications of the machines are described in the Table 11 below.

Table 11: Hardware specification for App Profiler accuracy testing

Machine #	Hardware specification
Configuration A	**CPU**: Intel(R) Core (TM) i7-3820 CPU @ 3.60GHz **RAM**: 32 GB (4 x Nanya NT8GC64B8HB0N) **Disk**: 256 GB (ADATA SU800)
Configuration B	**CPU**: AMD FX-8370 Eight-Core Processor **RAM**: 16 GB (2 x Mushkin 992125R) **Disk**: 256 GB (Samsung SSD 850)
Configuration C	**VM hosted on academic cloud service** **CPU**: 2 cores with 1 thread from an Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz **RAM**: 4 GB

The evaluation of the ML algorithms was performed using the 10-fold cross validation method [31] on the dataset. As far as the performance indicators for the ML models are concerned the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Relative Absolute Error (RAE) and simple accuracy of the predictions were evaluated (Table 12).

Table 12: App Profiler ML Model evaluation

Classification Algorithm	Mean Absolute Error (MAE)	Root Mean Squared Error (RMSE)	Relative Absolute Error (RAE)	Accuracy
App Profiler (Random Forest**)**	0.0056	0.0464	17.84%	91.88%
ForestPA	0.0071	0.0526	22.80%	88.88%
Bayes Network	0.004	0.0498	12.78%	88.78%
Random Tree	0.0037	0.0612	12.00%	88.20%
SPAARC	0.0042	0.0568	13.52%	87.04%
PART Decision List	0.0047	0.0619	15.10%	85.59%
REPTree)	0.0054	0.0586	17.29%	84.53%
MODLEM	0.0051	0.0716	16.42%	83.85%
HoeffdingTree	0.007	0.0765	22.35%	77.85%
Naive Bayes	0.0083	0.0719	26.62%	74.85%

SPAARC, Random Tree, Bayes Network, ForestPA, and Random Forest all perform fairly well, as shown in Table x PART Decision List. Random Forest, Bayes Network, and Random Tree produce the greatest MAE outcomes, as it is shown in the Table 12. This is likewise true for RMSE, with the exception of the final one. When RAE is included in, things become more complicated because Random Forest has the third best performance, and all of the aforementioned algorithms perform quite similarly. The actual accuracy clarifies the field and indicates that, while Random Forest is somewhat poorer in some Error Metrics (but still similar), it outperforms every other algorithm in terms of real predictions. This was especially true for the Random Tree, because, with a few exceptions, Random Forests often outperform Random Trees.

Progress in 2nd Iteration

As far as the updates of the 2nd iteration is concerned the table below (Table 13) demonstrates the most significant changes. It is important to mention that the significant developments on the App profiler regard mainly the machine learning models in order to implement a more robust mature and advanced machine learning software (Weka 3). It is important to mention the implementation of Kafka based message brokers in order to operate and communicate with the other Pledger components efficiently and easily. Finally, another significant development on the App Profiler is the development of the Node-RED flows responsible to handle the profiling of Kubernetes (Figure 6) based applications or services.

Table 13: 2nd iteration developments

General Functionality	Developments
Classification process	▸ Implementation of ML Algorithms ▸ Improve metric selection for application profiling ▸ ML Model accuracy assessment **Updates** ▸ Introduction of new ML model creator ▸ Development of model assessment
Flow based Distributed Architecture	▸ Implementation of Node-RED Framework ▸ Fragmentation subcomponent development and maintenance **Updates** ▸ Implementation of Kafka Brokers for communication with other Pledger components (DSS, E2CO) ▸ Development of Node-RED flows for Kubernetes container profiling
Ease of installation and use	▸ Implementation of REST-API for the configuration of a profiling process ▸ Implementation of connectors for different containerized environments (Docker, Kubernetes) ▸ Easy installation of App Profiler through Node-RED flows **Updates** ▸ Development of automatic profile creation on application deployment ▸ Advanced Application profiling based on the user input using the existing REST-API.

3.2.3 Slicing and Orchestration Engine (SOE) and RAN Controller Framework

The Slicing and Orchestration Engine (SOE) and Radio Access Network (RAN) controller are responsible for the lifecycle management of network slices, including the partitioning and management of shared virtual resources, as well as the orchestration of vertical services. Together, the SOE and RAN controller provide functionalities related to isolation, end-to-end connectivity, support for compute, network and radio chunks[1] (including support for radio technologies such as 5G NR and 802.11p), and support for verticals through the provisioning of dedicated slices.

Approach to network slicing

SOE follows a novel deployment-driven approach to network slicing [18], which leverages soft slicing principles, where vacant resources can be used for other services [26]. In this approach, slice resource sharing is decoupled from runtime constructs, so that it can be implemented at a pre-runtime phase, with the added flexibility that cloud resource orchestrators can be used for the management of slice resources. Through this approach, decreasing complexity and increasing re-use of slice model instances are achieved. In Pledger, a cloud-native approach is followed, and the use of Kubernetes is leveraged for the management of compute resources. Kubernetes supports multiple virtual clusters (i.e. namespaces or compute chunks) on top of a shared physical cluster. Kubernetes provides API isolation across

[1] I.e., collections of compute, radio and network resources that make up a network slice.

different namespaces, according to certain role-based access control policies. In addition, Kubernetes provides dedicated networking across the set of nodes comprising the infrastructure, such that applications running on the same namespace can communicate with each other, regardless of the physical node where they are actually deployed. Further, Kubernetes provides some intelligent resource management, such that a certain level of QoS can be guaranteed to applications, but unused resources are flexibly allocated as required to other loads running in the cluster. In summary, Kubernetes introduces capabilities related to the isolation [19] and the intelligent management of infrastructure resources [20],[21],[22]. In addition, SOE introduces a mechanism by which the sum of the maximum amount requested resources per slice cannot be larger than the total resources in the cluster.

On the other hand, service creation and instantiation through SOE is facilitated by a OSM MANO, which serves as a network function virtualization orchestrator. In this regard, the lifecycle management of network-service-based applications and virtual network functions is delegated by SOE to OSM MANO, as specified in [24]. Note that OSM MANO has not been extended as part of the work in PLEDGER; instead, it has been integrated with SOE, such that the lifecycle management of network services and functions is completely managed by OSM MANO – which is a functionality already built in within that tool.

Even though OSM MANO offers some network slicing capabilities, SOE's approach to network slicing is different to that followed by OSM MANO, in the sense that OSM MANO follows a service-driven approach, with a focus on the provisioning of network service chaining; this limits the ability to support the deployment of service models based on different standards (e.g., ETSI MEC), resulting in a limitation related to the slice resource management capabilities of OSM MANO. In contrast, SOE follows a deployment-driven approach, where slice resource management is also implemented as an added functionality, resulting in an increased flexibility for supporting diverse service models. Slices are modelled as sets of infrastructure resources with network services or other deployable linked to them as elements of the slice model (e.g., complete network service instances, MEC application instances, etc.)

It can be expected that in some use cases or scenarios featuring the Pledger platform, wireless, radio access network (RAN) technologies are used to provide connectivity to user equipment (UE). Where the SOE is used to enable slicing and managing computing resources, the RAN controller component enables the reservation slicing and radio resource management, as well as the deployment of a radio service. A radio service consists of the active connectivity provided by the physical radio (e.g., a Wi-Fi access point radiating a service set identifier, or a small cell radiating a public land mobile network identifier as in the case of 5G), and the necessary data plane connectivity towards the services hosted in the computing part of a slice. The data plane is managed via SDN.

The reservation of radio resources and the radio connectivity configuration is requested by the SOE over a REST API as part of the slice reservation workflow. In the Pledger project, UC2 features an infrastructure that is managed by the SOE and RAN controller components. The RAN technology used is IEEE 802.11p, providing connectivity to vulnerable road users. That means, that upon receiving the slice reservation request for UC2, the SOE enables the computing slice allocation, whereas the RAN controller does the allocation for the radio. Some technologies, like IEEE 802.11p, apart from requiring the configuration of the radio devices and setting up the data plane to connect with the services in the compute nodes, require additional software to be running in the infrastructure. For IEEE 802.11p, a V2X software (V2X stack) is necessary. The V2X stack handles the exchange of messages from the vehicular domain (radio side) with the service domain (compute side). As such, deploying the V2X stack forms part of the basic slice configuration and is part of the workflow, whenever IEEE 802.11p connectivity is requested. In contrast, in the 5G proof of concept currently being developed on the scope of WP5 activities, it is necessary to deploy a set of 5G core elements, which in our case will be running in an OpenStack-based virtual machine. In addition, a set of connectivity configurations (e.g., required VLANs) is required need to be performed during the deployment of end-to-end 5G NR-based network slices.

Architectural aspects

Figure 7 represents the SOE's modular architecture and interconnection with the RAN controller. SOE is primarily composed by a slice manager, a multi-tier orchestrator, a network function virtualisation orchestrator, and a virtual infrastructure manager; in addition, SOE communicates with the RAN controller for the deployment of end-to-end slices. SOE & RAN controller feature a set of standardized REST-based interfaces, which facilitate their integration with other Pledger components and infrastructure resources, as required. For example, dedicated API calls are provided for the deployment of compute, network and radio chunks, network services, etc., such that the E2CO can trigger those actions by sending the corresponding API call to the SOE. Moreover, the SOE also communicates with the RAN controller through a standardized REST-based interface for the deployment and configuration of radio elements (e.g., IEEE 802.11p, LTE, 5G, etc.)

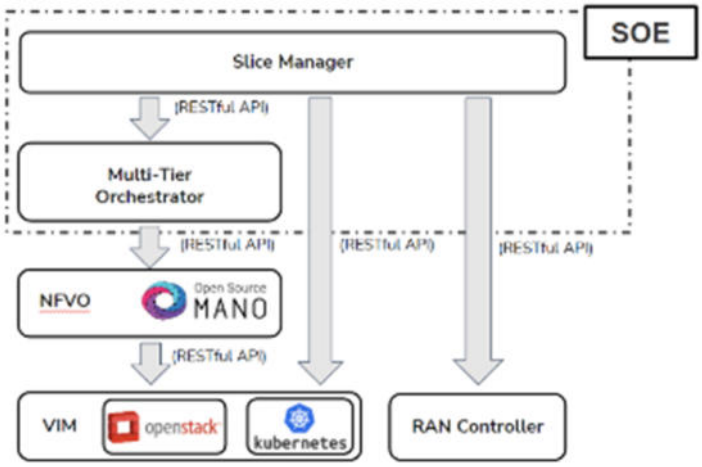

Figure 7: SOE modular architecture

The SOE Framework provides connectors that dynamically enable the allocation of resources for services (i.e., connectivity, application virtual machines or containers) in cloud and edge infrastructures based on OpenStack and Kubernetes. To that end, this engine leverages the current capabilities provided by OSM MANO also to allow the pre-provisioning of resources in different types of infrastructure (e.g., compute, networking, and radio chunks). As a result, services can be deployed on-demand and by introducing concepts like slicing, and the reutilization of the same infrastructure by multiple service providers becomes possible. These operations are exposed via a northbound RESTful API service that can be programmatically exploited by third parties to ensure the infrastructure couples with SLAs defined by the adopters (e.g., application developers, etc.) [23].

While the service-driven approach introduced by OSM MANO is based on a chain of network services only, the concept of resource chunk in SOE enables the logical partitioning of virtualized infrastructures, achieving multi-tenancy and isolation capabilities at both the infrastructure and service levels. SOE uses a set of resource chunks to compose a slice; these are associated to both an infrastructure layer and a service layer. At the same time, a specific chunk can be shared by several slices, increasing infrastructure re-use [18].

Progress in 2nd Iteration

The SOE and RAN controller are previously existing components that have been extended in the scope of Pledger to achieve some additional functionalities. All SOE and RAN controller developments that have been performed during this iteration of T3.2 were according to the initial plan and involved extensions of specific components rather than architectural changes. Therefore, the modular architecture is as previously presented in D3.2 [1] and given in Fig. 1.

On a top level, the main developments performed are as follows:

- The SOE and RAN controller have been extended to support devices based on the 802.11p radio protocol, in addition to previously supported technologies (including LTE, 5G, etc.) This extension enables the creation of end-to-end slices including radio elements based on 802.11p, therefore giving support to vehicular application such as that developed in UC2. This extension involved the following specific tasks:
  - RAN developments: The RAN Controller's catalogue of YANG models representing radio technologies has been expanded with the model for IEEE 802.11p radios in addition to 5G NR. On the radio node side, the necessary remote procedure calls have been implemented to be able to configure the radio interfaces remotely, following a well-defined business logic for reserving the radio resource and instantiating the radio service. Further, the SDN-based methodology to attach the radio interface to the data plane of the associated compute chunk was developed, taking into consideration the requirement of needing a dedicated virtualized V2X stack per each radio interface in the case of IEEE 802.11p-based network slices. This later point forms part of the integration work between the SOE and RAN Controller, given the virtualized V2X stack instances are instantiated as part of the day0 configuration of the slice. A similar approach is followed for the attachment of 5G radio interfaces to the data plane of OpenStack-based compute chunks.
  - An extension to the standard northbound API interface of the SOE, involving the creation of dedicated API calls for the deployment of end-to-end network slices including 802.11p-based radio elements, in addition to 5G NR elements.
  - Required integrations between SOE and RAN controller, including the definition of required flows and the implementation of necessary calls for the deployment of end-to-end slices.
- SOE has been extended to support the use of Kubernetes as the virtual infrastructure manager (VIM) and container orchestrator, supporting the possibility of deploying Container Network Functions (CNFs) on such VIM, while achieving a deployment-driven, cloud-native approach to network slicing. Through this approach, SOE delegates the management and isolation of virtual compute resources to Kubernetes & the instantiation of CNFs to OSM. This extension involved the following tasks:
  - Creation of the Kubernetes southbound client. SOE uses this client to perform any desired action in the Kubernetes cluster. Some examples are the Kubernetes cluster management or the compute chunk creation/deletion which in Kubernetes maps with the creation/deletion of namespaces and resource quotas. Other already supported southbound clients are OpenStack, OSM among others.
  - Extension of the OSM southbound client in order to support the Kubernetes-related actions required by OSM to be able to manage & deploy CNFs.
  - An extension to the standard northbound API interface of the SOE, involving the creation of dedicated API calls for the deployment of Kubernetes-based compute chunks, as well as instantiation of container-based network services.
  - Required extensions to SOE's business and service layers, as well as the implementation of necessary database-related flows.

3.2.4 Monitoring Engine

This component will manage the store and retrieve of metric values from different infrastructure resources and app/services in a central time series database to allow other component/subsystem (consumers) to query for specific metrics in a specific time interval. This metric database will be centralized in the Pledger hub cluster and with the help of some external tools/exporters/extractors/producers will gather metrics samples in a periodic interval and will send them to the central data-store.

The main functionality of the component remains as a central point of service to store and retrieve metrics requested or needed for the users of Pledger. We also extend the functionality to include a new ETL module or subcomponent to extract, transform and load the telemetry samples of the edge or remote infrastructures and applications in the central metric data store of Pledger. These metrics samples or scrapes will be sent by the infrastructure owners or applications owners via the StreamHandler component of Pledger to the Pledger hub (central) cluster. In this way, we avoid the usual security constraints at the edge that do not allow us to open inbound connections by default.

We also included a first proposal of a "***Continuum Monitoring***" system where the Monitoring Engine intermediate between the consumer and the producer of the metrics samples asking to the orchestrator component (E2CO) where is right now running an application to get the right/at to date metric data store to serving the metric request to the consumer. In this way we allow to Pledger to use a distributed datastore for metrics as an alternative to a single central repository or metric data store and we also decouple consumers and producers of metrics.

For this component we exploit Prometheus Operator as a baseline software playing with its different Kubernetes custom CRD (custom resource definitions) to allow a dynamic configuration and auto discovery of new endpoints or edge sites as sources/producers of metrics.

This component has to be understood as a manager of monitoring extractors (or exporters in Prometheus language) that feed Pledger metric datastores. There are an extend library of this extractors for Prometheus (third-party exporters [30]) when it is not possible to instruments systems and applications with Prometheus directly. It is also possible to create new exporters following some best practices. In the case of Pledger, we are not limited to only these exporters and in theory it is possible to integrate Monitoring Engine with external monitoring systems via our StreamHandler interface (Kafka) declaring new metric types in our ELT subcomponent (K2P or Kafka to Prometheus acronymous).

The metric data store supply by Prometheus is used to take advantage of the query language of Prometheus (PromQL) to retrieve or query for metrics values and the user interface of the Prometheus UI or Grafana dashboards.

This component or tool is a totally new development for this project. As mentioned before, for this component Prometheus Operator as a baseline software is used.

The Monitoring Engine component related to technical management and control of store and retrieve metric data will be implemented as two containerized applications with the following modules or microservices:

- **Monitoring Engine (manager)**: this module manages the query interface to serve request from consumers of metrics acting as a single point of service and will also implement the main business logic to find the current placement/infrastructure of an application and it associated metric data store. This component implements a REST API to allow other components of Pledger to query for some metrics values from metric data stores.
- **Kafka to Prometheus (ETL)**: this module implements the ETL process (extract, transform, load) for the feeding of remote or edge metrics (system or application metrics) to the Pledger central metric data store.

As we can see in the next figure, the Monitoring Engine component serves requests from consumers of metrics like SLA management subsystem or the DSS (aka Recommender) component on demand by

means of a REST API and also receives periodic metrics samples from remote or edge sites, like UC1/UC3 infrastructures/applications, to feed the central metric data store via StreamHandler interface.

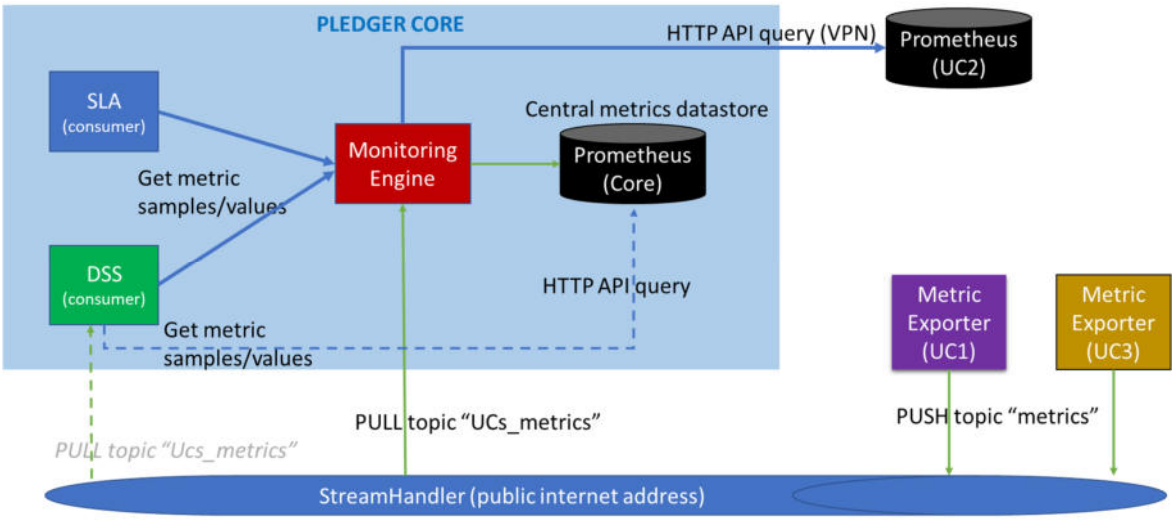

Figure 8: Monitoring Engine component context

Monitoring Engine Deployment diagram

All the containerized apps/services that composed the Monitoring Engine will be executed in a Kubernetes cluster with the following topology:

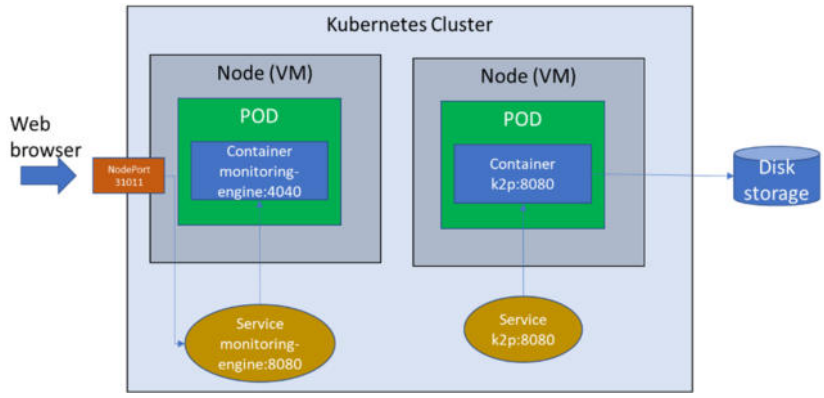

Figure 9: Monitoring Engine component deployment diagram in a Kubernetes cluster

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

3.3 Interfaces provided

3.3.1 Orchestrator (E2CO)

There are not changes in the OpenAPI REST API exposed by the Orchestrator (aka E2CO) component in this release. Please, check deliverable D3.2 [1] for more information.

New interfaces provided by SOE and RAN are described below. Interfaces with VM hypervisor will be described in the corresponding WP5 deliverable (no available at the edition of this doc).

3.3.2 App Profiler API

App profiler API for the basic processes such as profile extraction and classification can be found in the public GitLab repository of Pledger: https://gitlab.com/pledger/public/app-profiler#api-documentation

3.3.3 Slicing and Orchestration Engine (SOE) and RAN Controller Framework API

The SOE and RAN controller framework can be accessed through their northbound interface (NBI), where a set of RESTful API endpoints allow for the management of slices and services deployed on top of these slices. A detailed description of relevant endpoints is given in Annex I.

3.3.4 Monitoring Engine

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

The REST API services exposed by the Monitoring Engine component are the following:

- **/api/v1/query**: this endpoint requests an on-demand metric sample at one point in time.
- **/api/v1/query_range:** this endpoint requests metric samples in a closed time interval.

The same URL path pattern and payload results used by the Prometheus HTTP API have been followed to facilitate the transition from consumers of Prometheus metrics to this new single point of service.

Monitoring Engine query operations

Table 14: Monitoring Engine query operations with REST API

Operation	Method	URI	Description
READ	GET	/query?query=<string>&time=timestamp&appId=<id>	Read a metric value selected in the "query" parameter for the instant "time" (optional param) from the metric data store. The "appId" parameter filters metrics values by the Pledger application ID label.
LIST	GET	/query_range?query=<string>&start=timestamp&end=timestamp&appId=<id>	Retrieve a set of metric samples for a closed interval of time from the metric data store. The "appId" parameter filters metrics values by the Pledger application ID label.

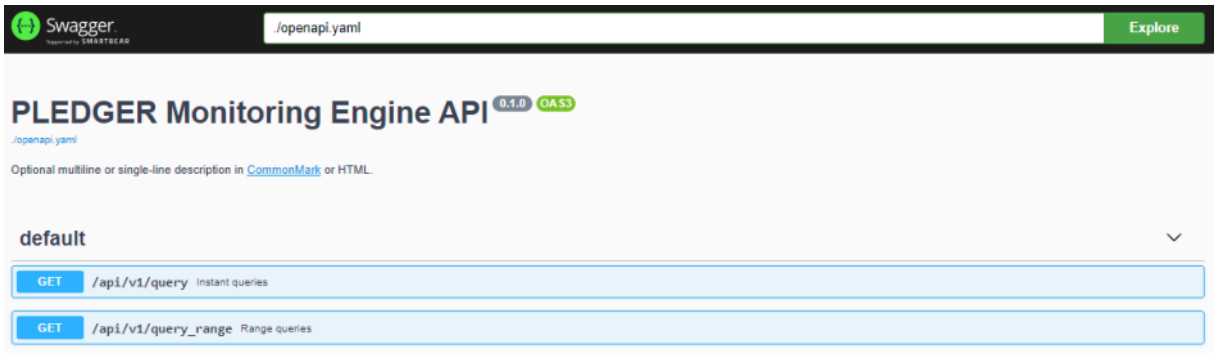

Figure 10: The Monitoring Engine component swagger interface for REST API

3.4 Data models

3.4.1 Orchestrator (E2CO)

No changes since last release. All details and information are given in the deliverable D3.2 [1].

3.4.2 App Profiler Data Models

App profiler uses two external data models in order to obtain the resource usage data from container environments. These two models are:

- Docker: For Docker based container App Profiler uses the Docker stats data model which can be found in the official website [25].
- Kubernetes: For the Kubernetes based environments an externa monitoring tools is used, which is Prometheus, Prometheus has specific resource usage exporter which follows a specific data model [26].

As far as the internal component data models, App Profiler uses a specific JSON based data model that is used to represent the profile vectors, this data model can be found in the official GitLab repository https://gitlab.com/pledger/public/app-profiler#schema-data-model

3.4.3 Slicing and Orchestration Engine (SOE) and RAN Controller Framework

The main data entities of the SOE and RAN controller are related to the management of infrastructure-related information. These entities describe compute, network and radio resources, as well as slices. Below, there are some example JSON files for the creation of compute chunk, radio chunks based on 802.11p, and end-to-end slices[2]. Additional JSON files used for the creation of 5G-based network slices in the 5G proof of concept being prepared within the scope of WP5 will be provided in due time in upcoming WP5 deliverables.

Table 15 shows an example of compute chunk creation request JSON file:

Table 15: Compute chunk creation request JSON

```
{
"name": "pledger-uc2-slice",
"user_id": "5b63089158f568073093f70d",
"description": "Compute on PLEDGER's K8s cluster node 1 on test-bed 2",
"compute_id": "5b63089158f568073093f70d",
```

[2] New features implemented during this iteration of T3.2.

```
"requirements": {
  "ram": {
    "required": 1024,
    "units": "MiB"
  },
  "cpus": {
    "required": 200,
     " limits": 400,
     " units": "m"
  },
  "storage": {
    "required": 100,
    "units": "Gi"
  }
}
}
```

Regarding the network side, network resource elements are created first through a POST/physical network operation, where a range of usable VLANs is provided for use by the V2X stack. An example network resource creation request JSON is provided in Table 16.

Table 16: Network resource creation request JSON

```
{
 "name": "public",
 "user_id": "5b63089158f568073093f70d",
 "physical_network_data": {
   "quota": {
     "tag_range": {
       "init": 50,
       "end": 90
     },
     "provisioned_tags": [
       20
     ]
   } }}
```

After successful network resource registration, a network chunk can be created through a POST/network_chunk operation. An example JSON for the creation of a network chunk is given in Table 17.

Table 17: Network chunk creation request JSON

```
{
 "name": "network_chunk1",
 "user_id": "5b63089158f568073093f703",
 "physical_network_id": "5b63089158f568073093f702",
```

```
 "compute_chunk_id": "5b63089158f568073093f701",
 "role": "data-network",
 "tag": 103
}
```


Similarly, radio resources are created through a POST/ran_infrastructure operation; a relevant example JSON for the creation of radio resources is provided in Table 18.

Table 18: Radio resource creation request JSON


```
{
 "name": "Barcelona",
 "user_id": "5b63089158f568073093f70d",
 "ran_infrastructure_data": {
  "password": "pass123",
  "username": "user1",
  "controller_url": "http://192.168.11.23:8008/",
 }
}
```


The creation of a radio chunk including two wireless IEEE 802.11p interfaces that have to have been previously registered in the RAN controller is showed in the JSON given in Table 19, where a radio interface is internally represented by a string right next after *selectedPhys*:

Table 19: Radio chunk creation request JSON


```
{
"name": "RadioChunk",
"user_id": "5b63089158f568073093f70d",
"chunk_topology": {
 "selectedPhys": [
    {
     "id": "f9af122a-c641-4084-ad61-2cdd9353fbc0",
    "type": "SUB6_ACCESS",
    "name": "phy0",
    "config": {
      "channelNumber": 36,
      "txPower": 200,
      "channelBandwidth": 20
     }
    }
  ],
```

```
  "selectedLinks": [
    {
     "id": "d66ed035-845e-4919-a746-85ed204cd53f",
     "key": "s0:phy0:3->s1:phy2:2",
     "srcPhyId": "11451e2b-475d-4f79-a75f-b5de6aa36bbe",
     "dstPhyId": "e8eff2b4-c317-47c6-b4d8-5d059336a7a0"
    }
   ]
  }
}
```


3.4.4 Monitoring Engine

To integrate this new component with the Pledger platform an internal data model was designed considering the global entities defined in the platform.

The next figure represents the relationship between these global entities and the internal entities of the Monitoring Engine for metrics management. In dark blue are represented the common entities of the platform and in green (attributes), light blue (entities/objects) and gold (types of labels) the internal data model of the component.

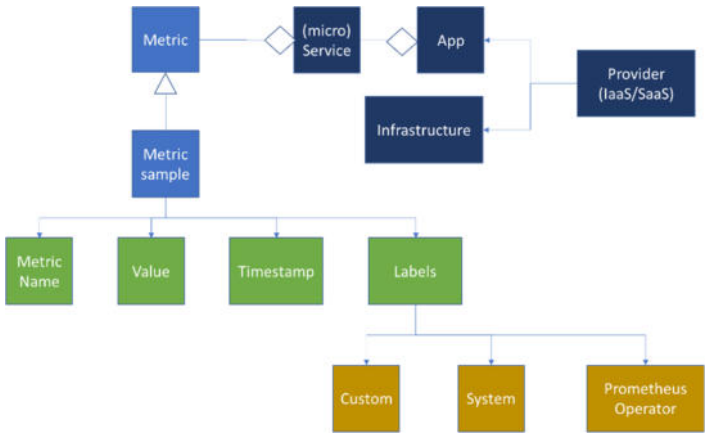

Figure 11: Monitoring Engine data model and the relation with Pledger platform

We also mapping the data model of the component with the entities (Kubernetes CRDs) of the Prometheus Operator package for integration purpose as shown in the next figure:

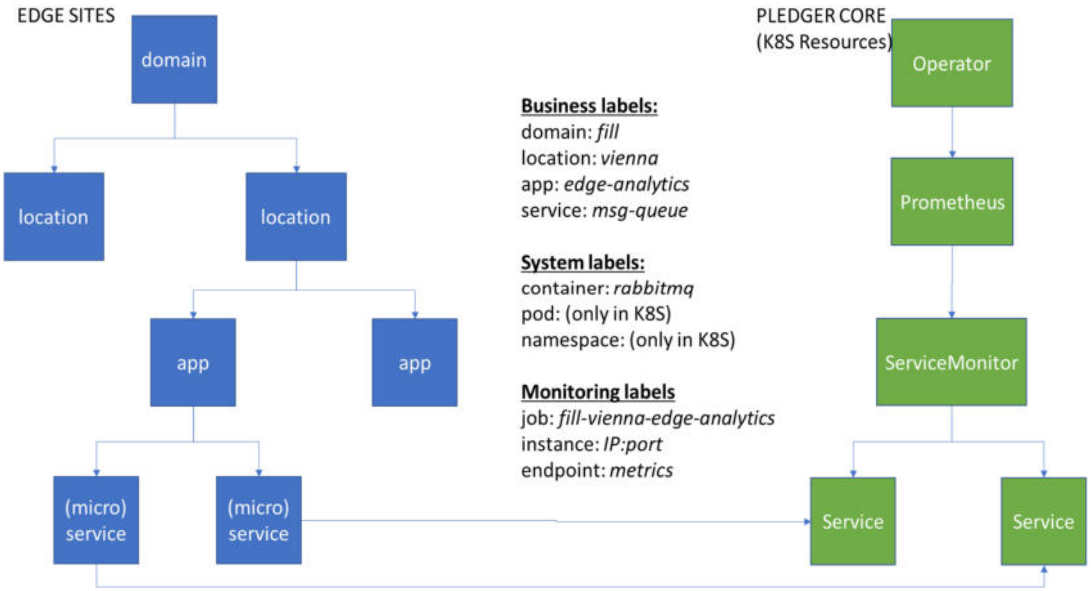

Figure 12: Monitoring Engine data model and the relation with Prometheus Operator package

Table 20: Monitoring Engine main entities description

Label	Description	Example
domain	Multitenancy ID	"fill"
location	Region, zone or DC (geospatial)	"vienna"
app	App name	"analytics"
service	Service name (part of an app)	"rabbitmq"
metric	Metric sample values with associated metadata (name, timestamp, labels, etc.)	"up {domain=fill,…} 1.0 1650534735"

Labels here are a key concept to facilitate the integration with the Prometheus metric data store and to help consumers to retrieve the correct metrics values using this key value pair fields as filtering.

4 Installation and usage guides

4.1 Requirements

4.1.1 Orchestrator (E2CO)

No changes since last release. Please, check deliverable D3.2 [1] for more information.

4.1.2 App Profiler Component

Application Profiler requirements fall into two categories:

General Software requirements: https://gitlab.com/pledger/public/app-profiler#software-requirements

Node-RED pallet requirements: https://gitlab.com/pledger/public/app-profiler#node-red-pallete

4.1.3 Slicing and Orchestration Engine (SOE) and RAN Controller Framework

The SOE Framework supports several installation methods, such as Python virtual environment, systemd, docker and Kubernetes. In the Pledger context, it is deployed via Kubernetes objects in a dedicated single-node Kubernetes cluster. Since the SOE framework includes a running instance of OSM and given that OSM's default installation method is based on a single-node Kubernetes cluster, the same Kubernetes cluster is used to host both SOE and OSM components, in order to achieve a better use of the available resources. The single-node Kubernetes cluster has been installed on top of an OpenStack-based virtual machine. The latest OSM release supported by SOE is OSM r11. SOE has the following minimum requirements: 8 GB of RAM, 4 vCPUs, 40 GB of SSD storage.

The RAN controller is virtualized and can be deployed in a VM instance. For use in an infrastructure, this dedicated VM has to be instantiated on any of the compute nodes forming part of the infrastructure during the day 0 configuration. Much like other Pledger core components, this instance is expected to run permanently, as it forms part of the infrastructure setup. The RAN controller has the following requirements: 16 GB of RAM, 8 vCPUs, 40 GB of SSD storage. Note that both the computing resource requirements for the SOE Framework and the RAN controller are mostly dictated by the integrated external software modules running as part of it (e.g., OSM).

4.1.4 Monitoring Engine

The following software components and credentials are pre-requirements to install the Monitoring Engine component:

- Kubernetes cluster ver. 1.20+ (this cluster will be the hub cluster)
- Credentials or service account in the hub cluster with permission to deploy applications.
- StreamHandler client certificate credentials as secrets object in K8S cluster.
- Endpoint configuration of the StreamHandler component.
- The two container images of the component in a container image repository.
- Prometheus Operator installed in the K8S hub cluster.

4.2 Installation

4.2.1 Orchestrator (E2CO)

A complete installation guide for the Orchestrator can be found in the official GitLab repository of Pledger: https://gitlab.com/pledger/public/edge-to-cloud-orchestrator#installation

4.2.2 App Profiler Component

A complete installation guide on how to install and operate App Profiler can be found in the official GitLab repository of Pledger: https://gitlab.com/pledger/public/app-profiler#pledger-application-profiler

4.2.3 Slicing and Orchestration Engine (SOE) and RAN Controller Framework

The SOE and RAN controller Framework application images are already configured and do not require an installation, just their deployment.

4.2.4 Monitoring Engine

A complete installation guide for the Monitoring Engine can be found in the official GitLab repository of Pledger: https://gitlab.com/pledger/public/monitoring-engine#installation

4.3 Usage

4.3.1 Orchestrator (E2CO)

A complete usage guide for Orchestrator can be found in the official GitLab repository of Pledger: https://gitlab.com/pledger/public/edge-to-cloud-orchestrator#usage-guide

4.3.2 App Profiler Component

A complete usage guide for App Profiler can be found in the official GitLab repository of Pledger: https://gitlab.com/pledger/public/app-profiler#pledger-application-profiler

4.3.3 Slicing and Orchestration Engine (SOE) and RAN Controller Framework

The northbound REST API of the SOE framework is by default accessed by a web browser through port number 8989 of the local host where it is installed, where the swagger UI with a full list of API calls can be found.

4.3.4 Monitoring Engine

A complete usage guide for Monitoring Engine can be found in the official GitLab repository of Pledger:

https://gitlab.com/pledger/public/monitoring-engine

4.4 Licenses

E2CO component is under the Apache license version 2

App Profiler component is under the Apache license version 2

SOE and RAN controller have a proprietary-type license.

Monitoring Engine component is licensed under Apache license version 2.

Some component dependencies and the correspondent licenses are indicated in section 3.1 Baseline technologies and dependencies of this document.

4.5 Source code repository

The source code repositories for **E2CO, App Profiler** and **Monitoring Engine** are available at the project GitLab.com account and can be publicly accessed.

The URL of the repositories where these components are hosted is as follows:

- **E2CO:** https://gitlab.com/pledger/public/edge-to-cloud-orchestrator
- **App Profiler:** https://gitlab.com/pledger/public/app-profiler
- **Monitoring Engine**: https://gitlab.com/pledger/public/monitoring-engine
- **SOE and RAN controller**: the code for these components is located in private repositories and will not be made open source.

5 Demonstration

5.1 Orchestrator(E2CO) and Monitoring Engine Demo

There are a few video recordings with full integrated demos of the tools described in this document plus other tools of the Pledger Core platform in the official YouTube channel of the Pledger project (https://www.youtube.com/channel/UCXV6V9rJ0ZvWhXeoWvDsArQ).

The video "Pledger integration demo#1" (https://www.youtube.com/watch?v=6aRlhWTi2k8) shows the core components of Pledger working together to demonstrate a runtime adaptation (placement) scenario in the Edge-Cloud continuum. For the point of view of this deliverable, the Orchestrator (E2CO) component part and its working functionalities of dynamically deployment and placement applications on different infrastructures in the Edge-Cloud Continuum, even with different VIM/controllers/orchestrator (from Docker container engines at the edge to Kubernetes cluster at on premise/cloud and back again) is highlighted.

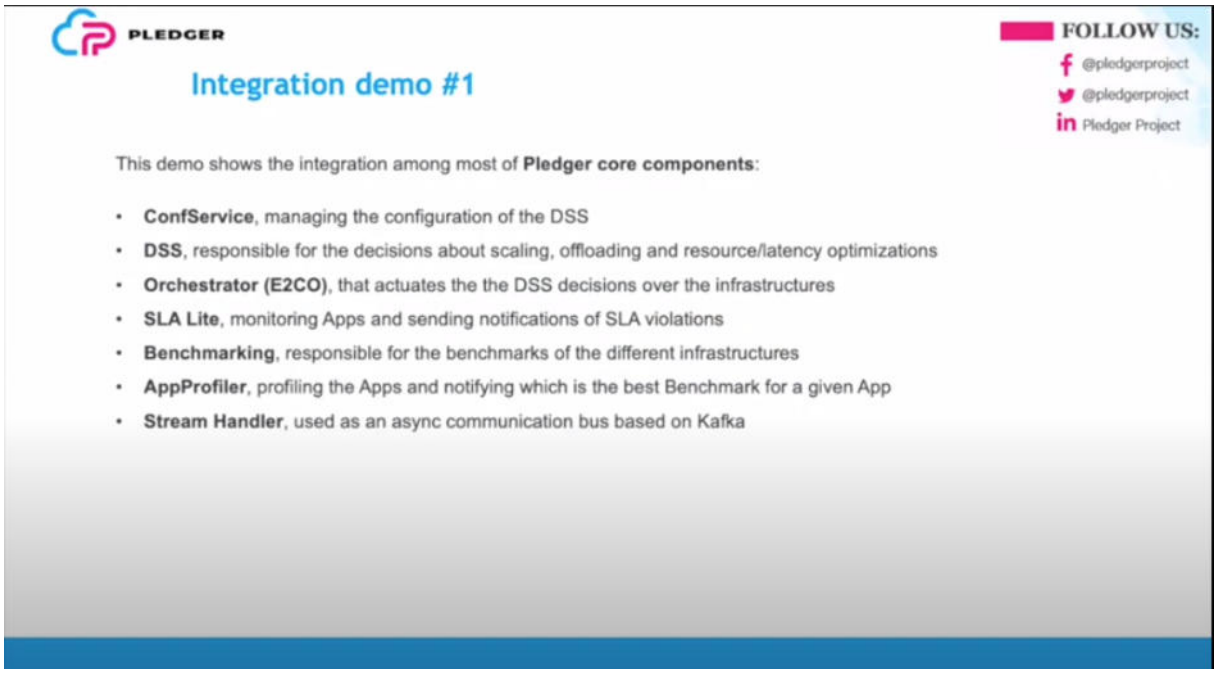

Figure 13: Pledger core components integrated demo snapshot

5.1.1 Scenario description

The service provider wants an app to run and fulfil some QoS (registered in an SLA object) and to keep the app on the edge as much as possible.

The main goal is to show a test app configured, started, monitored, offloaded from edge to cloud, and back to the edge when the system remediates the SLA violations with some runtime adaptations.

This demo will show the configuration and operation of a test app which is offloaded from the edge to the cloud and back reducing the SLA violations.

The sample app is configured on the ConfService to privilege edge over cloud nodes and has a guarantee used to monitor SLA violations. The app is offloaded by the DSS through the E2CO component, using the Stream Handler as a message broker. For the offloading, the DSS chooses the best node where to migrate the app using the benchmarks received.

The main steps for this demo are the following:

- **Step1**, configure de infrastructures needed for the edge (a VM with docker engine) and the cloud (K8S cluster) and the service provider preferences to keep an app on the edge as primary option as far as possible (ConfService component).
- **Step2**, configure an app with one service, SLA agreement with guarantees (expected QoS) and deployment options (ConfService component).
- **Step3**, Show Orchestrator (E2CO) and SLA manager updates (infrastructures, applications and SLA agreements created on the fly) via swagger API web UIs (Orchestrator, SLA Manager components).
- **Step4**, an app is deployed and started on the edge (as a docker container) as stated by the service provider (ConfService, Orchestrator components).
- **Step5**, the app is monitored, and an SLA violation is received when some load is artificially generated to trespassing the threshold defined by the SLA agreement (Monitoring Engine, SLA manager, DSS components).
- **Step6**, the DSS chooses the best benchmark tool for a service using label matching set manually or by the App Profiler (Benchmarking, App Profiler, DSS components)
- **Step7**, the DSS chooses the node (from a K8S cluster in the cloud) with highest score for the selected benchmark tool to offloading the app from the edge to the cloud (DSS, Orchestrator components).
- **Step8**, the app monitoring continues now in the cloud, with no more SLA violations for a while (Monitoring Engine, SLA manager components).
- **Step9**, the DSS triggers an offload back to the edge fulfilled the preferences of the service provider (DSS, Orchestrator components).

In the next picture we can see all the components involved in this demo, even the StreamHandler messaging middleware component used to facilitate the interoperability of the Pledger Core components.

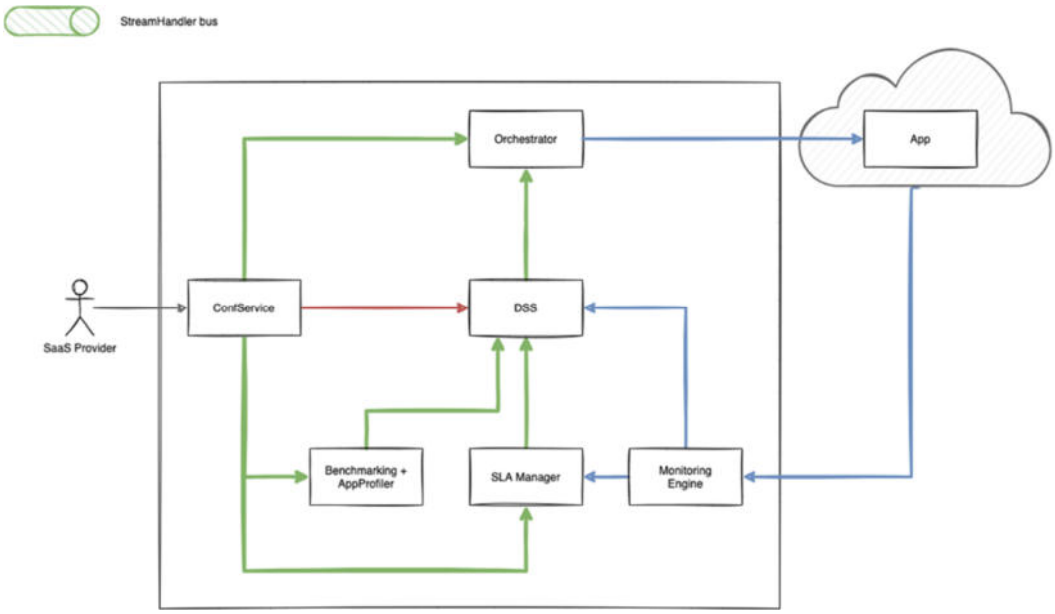

Figure 14: Pledger core components integration flows

5.1.2 Validation and Verification

In this scenario we can see how to create a rule of prioritization (user preferences) in the DSS component to select a non-critical component of an application to offload to the cloud in case some degradation of the edge infrastructure that can affect to other critical components or functionalities of the application. We can also see the comeback path from the offloaded component in the cloud to the edge infrastructure again when the constraints disappear.

We can check the SLA evaluation and violation notification (messages via StreamHandler) when we produce an artificial over utilization of the underlying infrastructure at the edge that trespassing the defined threshold of a metric value for the QoS requested by the Service Provider. We can also see other Kafka messages with different topics that show us what is going on in the background (behind the scenes) of the Pledger core components to take decision and execute runtime adaptations on live.

At the end the app returns to the edge as stated by the service provider keeping the QoS agreed between the infrastructure provider and the service provider as far as possible by means of automated runtime adaptations generated by the Pledger platform.

5.1.3 Demo

You may watch the video of demo shown the Orchestrator component in action together with the rest of Pledger Core components ("Pledger integration demo#1", https://www.youtube.com/watch?v=6aRlhWTi2k8) in the Pledger YouTube official channel.

5.2 App Profiler Demo

App profiler demonstrator is created in order to showcase the seamless operation of the component as well as the individual steps in order to profile an application and aid the DSS on the deployment decision. Application Profiler demo is part of the second integrated demo(demo2) of pledger core components. More specifically the integrated demo shows the interoperability and functionality of DSS, App Profiler and Benchmarking suite, but in the context of this deliverable we focus on the App profiler part of the demo.

5.2.1 Scenario description

When there is a violation on an SLA in the Pledger platform for a particular service or application, DSS enforces decisions on migrations in order to meet the targeted SLA guarantees. In order to achieve that DSS takes under consideration two important factors the capabilities of the resources and the needs of the application, App Profiler is responsible to provide to the DSS information about the latter. There are four steps in the process of obtaining the application profile and they happen in the following order:

- **Step 1** of the process to automatically profile the application is the actual deployment of the application or service in the infrastructure. As depicted in the image below application profiler is informed on the deployment of the application from deployment engine and starts the process of profiling.
- **Step 2**, after obtaining the deployment information about the application, App Profiler can trigger the operation of collection resource usage data.
- **Step 3**, App profiler collect the resource usage data and composes the profile vector that will be later used for the classification process.
- In the final **step 4** App Profiler takes the profile vector created in the previous process and classifies the application using the ML Models created in the training phase. Classification results are then sent to the DSS via the Stream Handler Component.

5.2.2 Validation and Verification

Based on the system use case requirements (SUC) App Profiler through this demo demonstrates how it is able to provide application profile details (SUC.01) to the DSS, creating by extend recommendations

for the IaaS resource selection(SUC.04). App Profilers classifications with the combination with the benchmark reports can aid the selection of the most fitting IaaS (SUC.05).

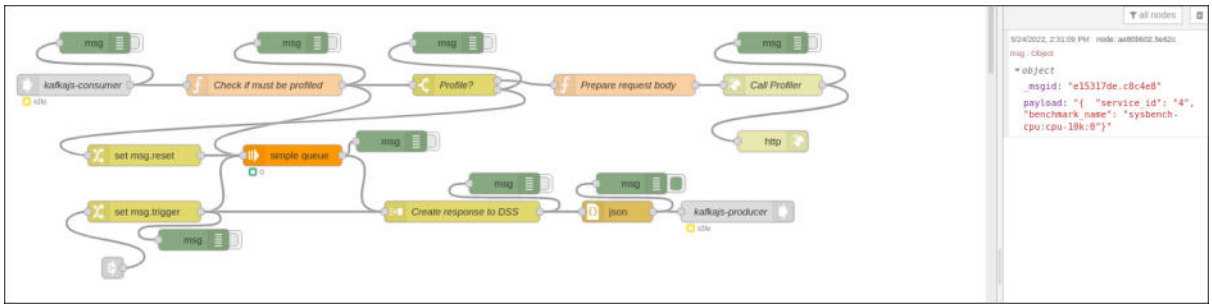

Figure 15: App Profilers Node-RED flow for resource usage metrics extraction

As it is depicted in the flow-based diagram of the Node-RED framework (Figure 15) App profiler has both Kafka Consumers and Producers (grey nodes) in order to seamlessly and asynchronously communicate with the deployment engine of Orchestrator and the DSS. After the finalization of the complex procedure of collecting and transforming the applications' resource usage profiles, as depicted in the box on the far right of Figure 15, App Profiler classifies the service to the most suitable benchmark, in this case sysbench-cpu-10K (where 10K corresponds to 10.000 iterations). This classification is sent to DSS in order to assess the performance indexes of this particular benchmark in order to find the most suitable infrastructure configuration for deployment or migration, for a given service or application.

5.2.3 Demo

The demo for the App Profiler (Demo 2) can be found in the official YouTube Channel of Pledger: https://www.youtube.com/watch?v=oJGZCa8SUUI

5.3 SOE and RAN Controller demo

The SOE and RAN Controller are demonstrated to showcase the deployment of an end-to-end slice based on IEEE 802.11p, and a network service over such slice.

Note that, as part of the activities being carried out in WP5, a proof of concept for the deployment of end-to-end network slices based on 5G NR is currently being developed. A relevant demo will be documented in upcoming WP5 deliverables.

5.3.1 Scenario description

Pledger's end-to-end network slice deployment video demonstrates the use of SOE and RAN controller for the deployment of an end-to-end network slice over UC2's Kubernetes-based infrastructure. The main goal is to show the configuration of the resources that compose the slice (compute, network and radio), and the provisioning of the end-to-end slice on top of which the UC2 application then can be deployed. The end-to-end network slice creation workflow is depicted in Figure 16.

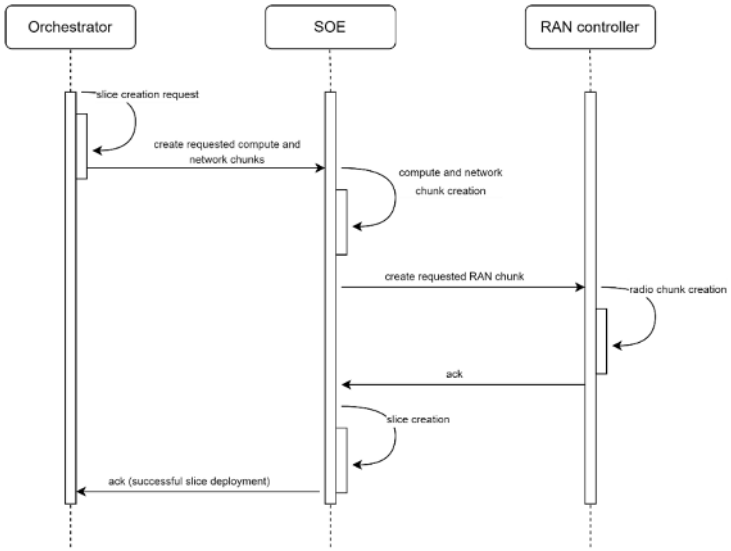

Figure 16: End-to-end network slice creation workflow

To provision the slice, several SOE endpoints (one per type of chunk) need to be consumed specifying the characteristics/requirements of the different chunks that will form the slice. The slice compute capabilities are covered in the compute chunk API endpoint, as reflected in Table 9. Radio characteristics of the slice are specified in the radio chunk, as depicted in Fig. 5, and the interconnection of compute and radio devices is performed with the help of the network chunk. The compute and radio chunk are dimensioned according to the requirements specified by the UC2 application.

Next, after all resource chunks have been set up, relevant parameters such as chunk ids -needed to identify the newly created chunks- are parsed to SOE, and an end-to-end slice is finally deployed.

The steps required to deploy an end-to-end network slice and a network service can be summarized as follows:

- **Step 1,** Compute chunk creation
- **Step 2,** Network chunk creation
- **Step 3,** Radio chunk creation
- **Step 4,** Creation of a network slice as a collection of chunks
- **Step 5,** Deployment of a radio service, in order to provide end-to-end connectivity
- **Step 6,** Deployment of a network service instance

5.3.2 Validation and Verification

In the end-to-end slice deployment scenario, a network slice with compute, network and radio elements is deployed as per the user's request.

On the compute side, this scenario can be validated and verified by checking whether a namespace has been created on Kubernetes with the specified resource quotas passed to SOE.

On the network side, the validation can be achieved by executing a Kubernetes describe command on some of the components of the V2X stack, specifically on the V2X-COM modules that run in Kubernetes. After the network chunk configuration is successfully applied and the slice is activated, a V2X-COM pod is created for each radio element. SOE will then execute a NetworkAttachmentDefinition Kubernetes object, which will create a dedicated interface on each V2X-COM pod; this interface will be automatically connected to a dedicated VLAN for communication with the radio elements.

The correct creation of the radio chunk can be validated by checking whether the radio interfaces have been correctly configured with the desired physical configuration. Once the configuration request has been received by the radio node and executed, an ocb0 interface will be present in the list of interfaces (ifconfig command). The specific configuration can be requested with the $iw ocb0 info command.

On the radio side, the integration of the radio nodes on the network slice can be validated and verified by revising whether the ocb0 radio interface for vehicular communications in each radio node has been correctly attached to a virtual switch together with a VLAN interface; the VLAN of the interface has to match the VLAN of the corresponding V2X-COM pod that is deployed in the compute chunk.

Moreover, the success of the slice activation can be validated and verified by sending network traffic through the dedicated VLAN between each radio element and the Kubernetes cluster and verifying that no traffic is leaked outside of that VLAN.

5.3.3 Demo

An end-to-end slice deployment demonstration video is currently being prepared and will be made available on Pledger's official YouTube channel. However, at the time of writing this deliverable, an already available video demonstrates the creation and validation of a compute chunk creation, as well as the deployment and validation of a network service on a dedicated compute chunk. Below, Figure 17 shows how a compute chunk is created from the ConfService user interface; subsequently, a Kubernetes namespace is created with the required compute resources. Figure 18 shows UC2's master node's terminal screen, where the required namespace has been created; in addition, Figure 19 shows that the newly created compute chunk is also registered in OSM MANO. In addition, Figure 20 and Figure 21 respectively show how a service is launched through the ConfService, and how the corresponding network service is instantiated in OSM MANO.

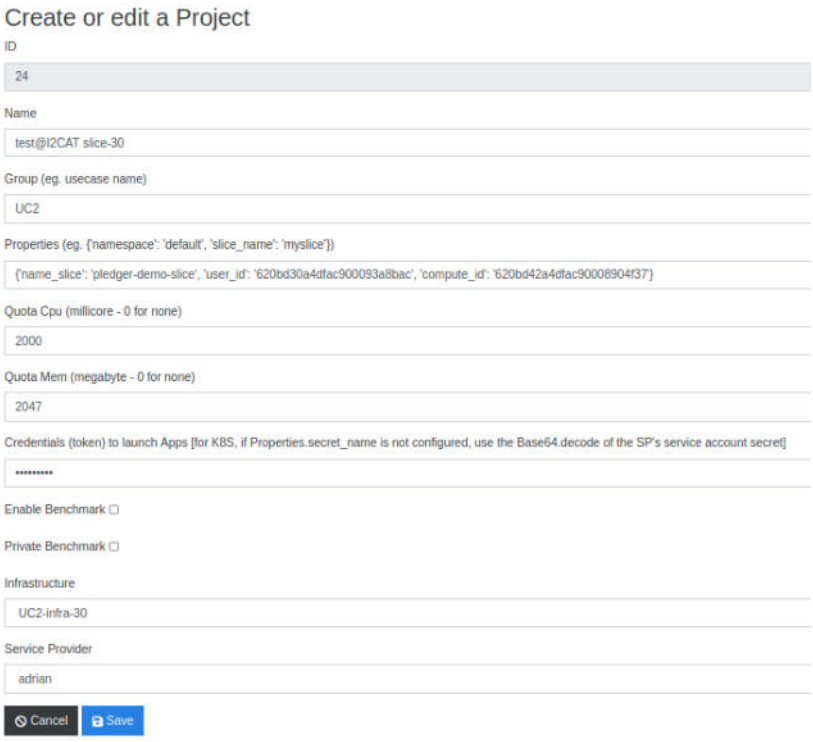

Figure 17: Compute chunk creation through the ConfService UI

```
adripino@adripino:~$ kubectl get ns
NAME                   STATUS   AGE
argocd                 Active   107d
default                Active   107d
kube-node-lease        Active   107d
kube-public            Active   107d
kube-system            Active   107d
metallb-system         Active   17d
monitoring             Active   106d
node-feature-discovery Active   17d
openebs                Active   17d
pledger-demo-slice     Active   23s
pledger-uc2-slice      Active   106d
adripino@adripino:~$ kubectl describe resourcequota -n pledger-demo-slice
Name:           pledger-demo-slice
Namespace:      pledger-demo-slice
Resource        Used  Hard
--------        ----  ----
limits.cpu      0     2
limits.memory   0     2047Mi
requests.cpu    0     2
requests.memory 0     2047Mi
adripino@adripino:~$
```

Figure 18: Terminal view after the compute chunk creation

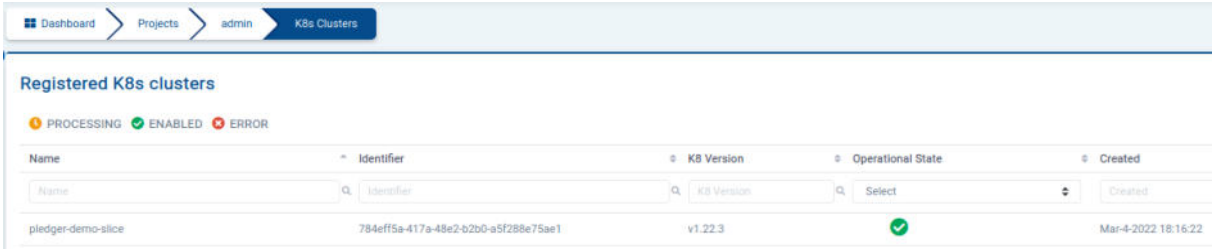

Figure 19: The compute chunk is registered in OSM MANO as a Kubernetes namespace

Figure 20: Instantiation of a service through the ConfService UI

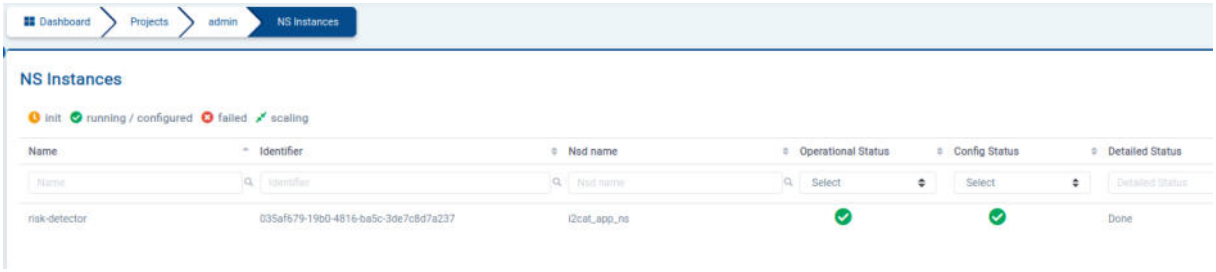

Figure 21: A network service instance is instantiated in OSM

6 Conclusions and next steps

In this deliverable D3.5, the work performed in WP3 Task T3.2 during the M18-M30 was documented in order to reach the goal "MS7 - Second iteration of WP3 prototypes", which was achieved. As far as the development of second iteration is concerned the components of T3.2 are fully integrated and reached the desired level of functionality.

This deliverable presented all the important updates of Orchestrator, App Profiler, SOE and RAN Controller Framework (as summarized in the Table 21) in the second iteration of these components and also introduced the functionalities and architecture of the Monitoring Engine. Furthermore, the functionalities of the components were demonstrated through specific scenarios, these demos constitute the integrated demos of Pledger that showcase how Pledger is able to deliver the envisioned system.

Table 21: Components update summary

Component	Technical updates summary
Orchestrator	Orchestrator major updates are listed below: ▸ **Integration**: Further integration with the SOE and RAN controllers for network slices and radio connectivity. Enhanced feedback status of lifecycle operations. ▸ **Developments**: Management of VMs using Linux KVM based hypervisor.
App Profiler	Application Profiler's major updates are listed below: ▸ **Integration:** App profiler is fully integrated with Orchestrator and DSS system in order to deliver seamless operation and automatic profiling. ▸ **Developments:** As mentioned in the section 3.2.2 App Profiler has adapted a different ML tool for the data processing and model creation Weka. Also there where heavy developments in order to facilitate the needs of Kubernetes based container profiling.
SOE and RAN Controller Framework	SOE's and RAN controller's main updates are as listed below: ▸ **Integration:** the SOE is integrated with the RAN controller for the deployment of network slices with radio elements based on the IEEE 802.11p protocol. In addition, the SOE is fully integrated with the ConfService and Orchestrator for the deployment of compute chunks and network services, while integration efforts for the deployment of radio chunks, network chunks and end-to-end slices are ongoing. ▸ **Developments:** all developments related to the cloud-native slicing functionalities of the SOE have been completed during this iteration, as per the initial development plan (i.e., support of Kubernetes-based slices, support of container network functions, and alignment with ETSI NFV.) The RAN controller has been extended to support 802.11p technology (including business model, Yang model, NETCONF manager extension, SDN management and data plane adaptations.)
Monitoring Engine	Monitoring Engine major updates are listed below: ▸ **Integration**: new query API to centralize retrieving of metrics values (single point of service). ▸ **Developments**: new ETL service to collect metrics from the edge decoupling producers and consumers.

Since this deliverable marks the end of WP3 further major developments are not foreseen for the components. The effort for the components will be focused on the UC integration and general usability,

in order to ensure the adaptation and correct evaluation of the KPIs of the project. More specifically, effort will be focused on the seamless operation and integration of the components of the Pledger platform as well as on the adaptation and usage of the components for the pilots in order to produce the overall evaluation of these components.

7 References

[1] PLEDGER. D3.2 –Edge/Cloud orchestration tools I v1.0. Psychas, Alexandros. 2021. http://pledger-project.eu/D3.2.pdf, retrieved 2022-05-31

[2] PLEDGER. D2.3 – Pledger Overall Architecture v1.0. Voutyras, Orfefs. 2020. http://pledger-project.eu/D2.3.pdf, retrieved 2022-05-31

[3] Frosst, Nicholas, and Geoffrey Hinton. "Distilling a neural network into a soft decision tree." arXiv preprint arXiv:1711.09784 (2017).

[4] ATOS Research & Innovation Sodalite project page. https://booklet.atosresearch.eu/sodalite retrieved 2022-05-04

[5] Prometheus home page. https://prometheus.io , retrieved 2022-05-25

[6] Kubernetes home page. https://kubernetes.io , retrieved 2022-05-25

[7] Grafana home page. https://grafana.com , retrieved 2022-05-25

[8] Docker home page. https://www.docker.com , retrieved 2022-05-25

[9] Kafka home page. https://apache.kafka.org , retrieved 2022-05-25

[10] PLEDGER D4.3 Decision Support tools v.1.0. Iadanza Francesco. 2021 http://www.pledger-project.eu/D4.3.pdf

[11] ETSI Network Functions Virtualisation, https://www.etsi.org/technologies/nfv, retrieved 2022-04-28

[12] Open Source MANO (OSM) home page. https://osm.etsi.org, retrieved 2021-10-01

[13] Enns, R., Ed., et al., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011

[14] Netopeer2 – NETCONF Server, https://github.com/CESNET/Netopeer2, retrieved 2022-05-31.

[15] OpenDaylight home page. https://www.opendaylight.org, retrieved 2021-10-01

[16] SDN Resources: OpenFlow. https://opennetworking.org/sdn-resources/customer-case-studies/openflow/, retrieved 2021-10-01

[17] Open vSwitch home page. https://openvswitch.org, retrieved 2021-10-01

[18] A. Papageorgiou et. al., On 5G network slice modelling: Service-, resource-, or deployment-driven?, Computer Communications, 149, 2020, 232-240

[19] Kubernetes Network Policies, https://kubernetes.io/docs/concepts/services-networking/network-policies/, retrieved 29/04/2022

[20] Kubernetes Resource Quotas, https://kubernetes.io/docs/concepts/policy/resource-quotas/, retrieved 29/04/2022

[21] Configure Memory and CPU Quotas for a Namespace, https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/, retrieved 29/04/2022

[22] Configure Quality of Service for Pods, https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/, retrieved 29/04/2022

[23] G. Baldoni et. al., (2020). “Enhancing the Performance of 5G Slicing Operations via Multi-tier Orchestration,” 23rd Conference on Innovations in Clouds, Internet and Networks (ICIN 2020).

[24] H. Khalili, et. al. (2019). “Network Slicing-aware NFV Orchestration for 5G Service Platforms," 2019 European Conference on Networks and Communications (EuCNC). Valencia, Spain.

[25] https://docs.docker.com/engine/api/v1.41/#operation/ContainerStats

[26] https://prometheus.io/docs/prometheus/latest/querying/basics/

[27] https://www.cs.waikato.ac.nz/ml/weka/

[28] PLEDGER D5.2 Pledger integrated demonstrator I v1.0 2021

[29] Scale cube home page. https://uniknow.github.io/AgileDev/site/0.1.10-SNAPSHOT/scale-cube.html, retrieved 2022-05-31

[30] https://prometheus.io/docs/instrumenting/exporters/

[31] Fushiki, Tadayoshi. "Estimation of prediction error by using K-fold cross-validation." Statistics and Computing 21.2 (2011): 137-146.
[32] https://nodered.org/
[33] An introduction to NETCONF/YANG, https://www.fir3net.com/Networking/Protocols/an-introduction-to-netconf-yang.html, retrieved 2022-05-31
[34] MongoDB homepage, https://www.mongodb.com/, retrieved 2022-05-31
[35] OpenStack homepage, https://www.openstack.org/, retrieved 2022-05-31

Annex I (SOE and RAN Controller Framework API)

As shown in Figure 22, the northbound interface of the SOE and RAN controller framework offers a set of API calls for the management of individual sets of compute, network and radio resources, chunks of these resources, network slicing and deployment of applications, among others. However, not all these API calls are relevant to the integration of the SOE and RAN controller framework in Pledger. Below, more details are given for the specific calls that are foreseen as relevant in the integration with Pledger. Note that a few additional calls will be documented in upcoming WP5 deliverables in the scope of the 5G proof of concept currently being developed.

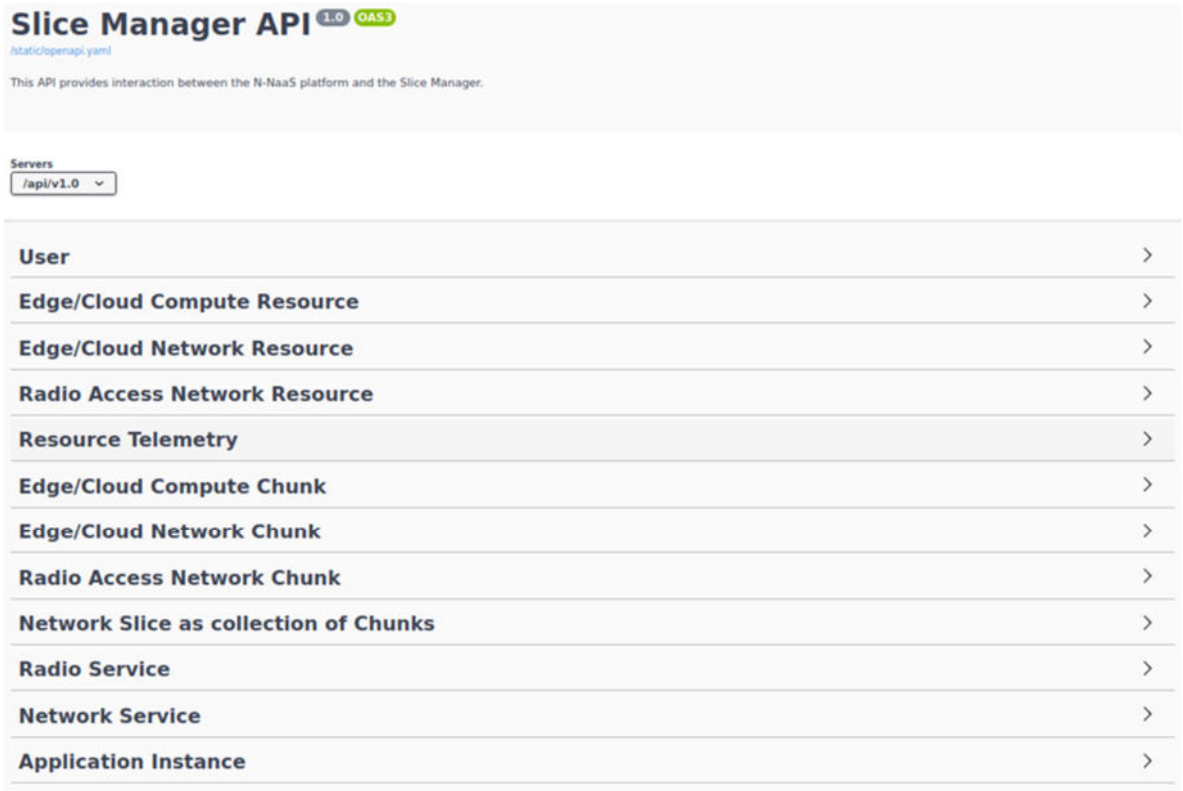

Figure 22: SOE and RAN Controller Framework swagger interface for REST API

Compute resource operations

Operations related to the management of individual (virtualized and non-virtualized) resources are represented in Figure 23 and Table 22. These operations are related to the retrieval and registration of available compute resources, as well as their registration and deletion.

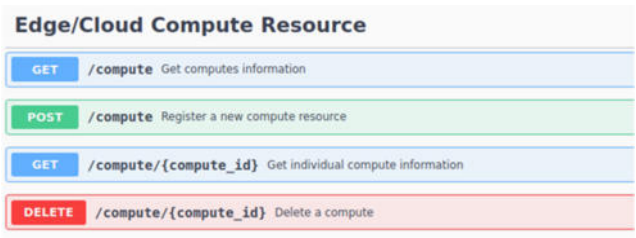

Figure 23: SOE and RAN Controller Framework REST API swagger interface for resource management

Table 22: SOE and RAN controller framework exposed REST API methods for resource management

Operation	Method	URI	Description
READ	GET	/compute	Get compute information.
UPDATE	POST	/compute	Register a new compute resource
READ	GET	/compute/{compute_id}	Get individual compute information
DELETE	DELETE	/compute/{compute_id}	Delete a compute resource

Compute chunk operations

Operations related to the management of chunks of compute resources are represented in Figure 24 and Table 23. These operations are related to the retrieval and registration of chunks of compute resources.

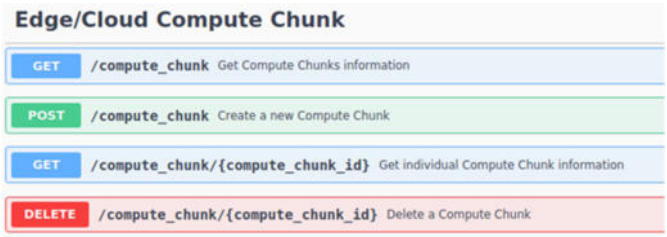

Figure 24: SOE and RAN Controller Framework REST API swagger interface for resource chunk management

Table 23: SOE and RAN controller framework exposed REST API methods for resource chunk management

Operation	Method	URI	Description
READ	GET	/compute_chunk	Get compute chunks information.
UPDATE	POST	/compute_chunk	Create a new compute chunk.
READ	GET	/compute_chunk/{compute_chunk_id}	Get individual compute chunk information
DELETE	DELETE	/compute_chunk/{compute_chunk_id}	Delete a compute chunk.

Network resource operations

Operations related to the management of individual network resources are represented in Figure 25 and Table 24. These operations are related to the retrieval and registration of available compute resources, as well as their registration and deletion.

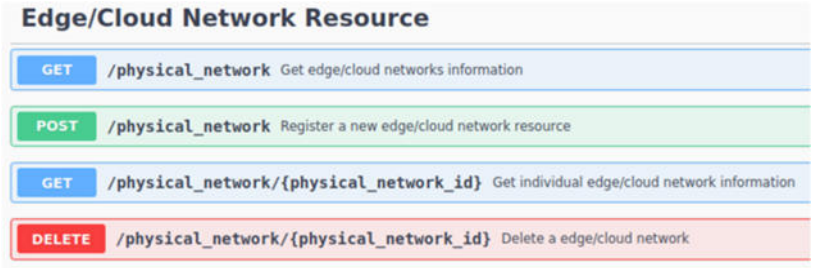

Figure 25: SOE and RAN Controller Framework REST API swagger interface for network resource management

Table 24: SOE and RAN controller framework exposed REST API methods for network resource management

Operation	Method	URI	Description
READ	GET	/physical_network	Get network information.
UPDATE	POST	/physical_network	Create a new network resource.
READ	GET	/physical_network/{physical_network_id}	Get individual network resource information.
DELETE	DELETE	/physical_network/{physical_network_id}	Delete a network resource.

Network chunk operations

Operations related to the management of chunks of network resources are represented in Figure 26 and Table 25. These operations are related to the retrieval and registration of chunks of compute resources.

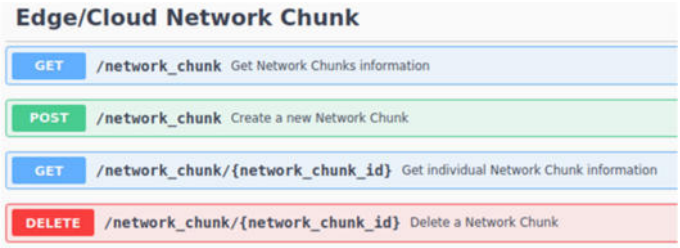

Figure 26: SOE and RAN Controller Framework REST API swagger interface for network chunk management

Table 25: SOE and RAN controller framework exposed REST API methods for network chunk management

Operation	Method	URI	Description
READ	GET	/network_chunk	Get network chunks information.
UPDATE	POST	/network_chunk	Create a new network chunk.
READ	GET	/network_chunk/{network_chunk_id}	Get individual network chunk information.
DELETE	DELETE	/network_chunk/{network_chunk_id}	Delete a network chunk.

Radio resource operations

These operations are related to the radio configuration of IEEE 802.11p and sub-6 Wi-Fi (2.4 GHz and 5 GHz) devices; its API endpoints can be found in Figure 27 and Table 26.

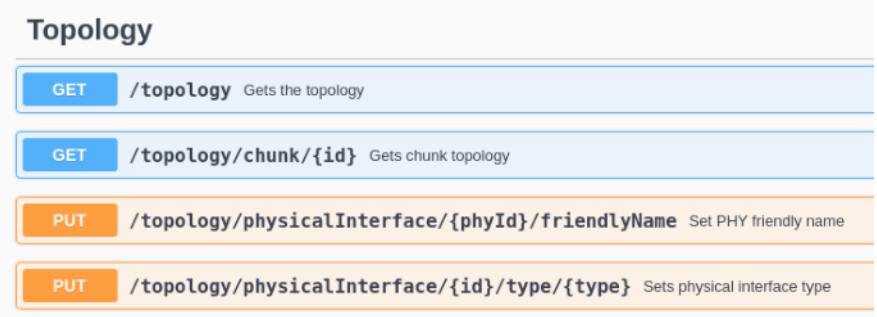

Figure 27: RAN Controller's REST API swagger for Radio resource operations

Table 26: RAN controller exposed REST API methods for radio resource operations

Operation	Method	URI	Description
READ	GET	/topology	Retrieves the RAN controller's topology with all the radio resources classified in boxes (devices) and physical interfaces (representing specific ethernet ports, PCIe interfaces, etc., present on the device)
UPDATE	PUT	/topology/physicalInterface/{id}/type/{type}	Changes the working mode of a selected physical interface (11p interfaces can act as normal Wi-Fi access point or as an actual 11p node)
UPDATE	PUT	/toplogy/physicalInterface/{id}/sub6/config	Modifies the physical interfaces radio parameters (channel, bandwidth and txPower)
READ	GET	/topology	Retrieves the RAN controller's topology with all the radio resources classified in boxes (devices) and physical interfaces (representing specific ethernet ports, PCIe interfaces, etc., present on the device)

Radio chunk operations

The RAN controller defines a Radio chunk as a subset of devices from the whole RAN controller's topology which is owned or reserved by a user (chunk owner). The chunk operations are detailed in Figure 28 and Table 27**.** These set of calls can be used for various radio technologies, including 5G NR and 802.11p.

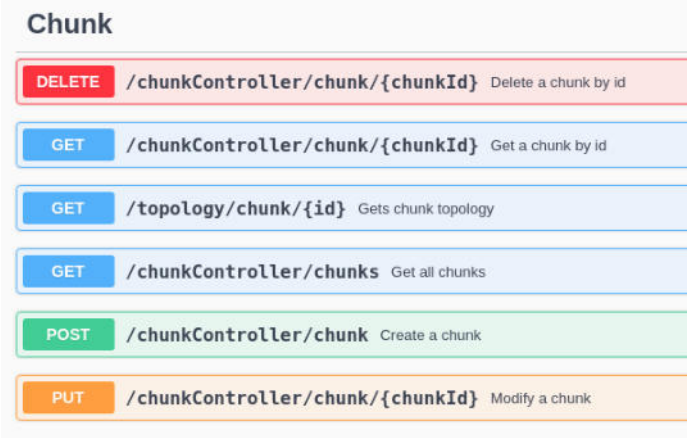

Figure 28: RAN Controller's REST API swagger for Radio chunk operations

Table 27: RAN controller exposed REST API methods for Radio chunk operations

Operation	Method	URI	Description
READ	GET	/chunkController/chunks	Retrieves all the chunks present on the RAN controller
READ	GET	/chunkController/chunk/{chunkId}	Retrieves the selected chunk
READ	GET	/topology/chunk/{id}	Displays the topology (similar to the endpoint described on table X) but limits the output to the resources available on the selected chunk

Operation	Method	URI	Description
CREATE	POST	/chunkController/chunk	Creates a new Radio chunk
UPDATE	PUT	/chunkController/chunk /{chunkId}	Modifies the selected chunk, makes possible to add or delete interfaces from the chunk
DELETE	DELETE	/chunkController/chunk /{chunkId}	Deletes the selected chunk

Network slice operations

Operations related to the management of network slices are represented in Figure 29 and Table 28. These operations are related to the retrieval, registration and deletion of network slices as collections of chunks and are applicable to network slices based on different radio technologies, such as 5G and IEEE 802.11p.

Figure 29: SOE and RAN Controller Framework REST API swagger interface for network slice operations

Table 28: SOE and RAN controller framework exposed REST API methods for network slice operations

Operation	Method	URI	Description
READ	GET	/slic3	Get slice(s) information.
UPDATE	POST	/slic3	Create a new slice.
READ	GET	/slic3/{slic3_id}	Get individual slice information.
DELETE	DELETE	/slic3/{slic3/id}	Delete a slice.

Application instance operations

Operations related to the deployment and management of application instances are summarized in Figure 30 and Table 29; these operations are mainly related to the retrieval of information, and the deployment and deletion of service instances.

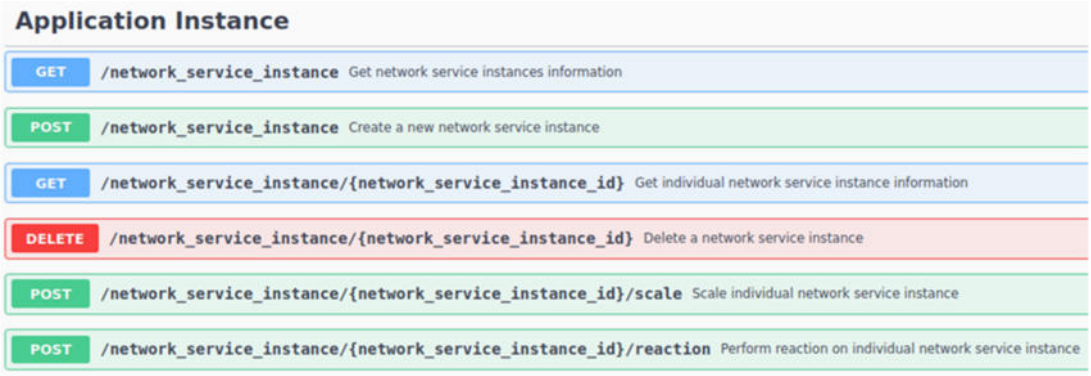

Figure 30: SOE and RAN Controller Framework REST API swagger interface for application instance operations

Table 29: SOE and RAN controller framework exposed REST API methods for application instance operations

Operation	Method	URI	Description
READ	GET	/network_service_instance	Get network service instance information.
UPDATE	POST	/network_service_instance	Create a new network service instance.
READ	GET	/network_service_instance_{network_service_instance_id}	Get individual network service instance information.
DELETE	DELETE	/network_service_instance_{network_service_instance_id}	Delete a network service instance.
UPDATE	POST	/network_service_instance_{network_service_instance_id}/scale	Scale individual network service instance.
UPDATE	POST	/network_service_instance_{network_service_instance_id}/reaction	Perform reaction on individual network service instance.
READ	GET	/compute_chunk/{compute_chunk_id}	Get individual compute chunk information.