



D5.2 Pledger integrated demonstrator I

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

Related WP	WP5	Document Reference	D5.2
Related Deliverable(s)	D2.3 “Pledger Overall Architecture” D5.3 “Pilots operations and monitoring I” D5.6 “Pledger integrated demonstrator II”	Dissemination Level (*)	PU
Lead Participant	INTRA	Lead Author	Ioannis Sarris (INTRA) Olga Segou (INTRA) Ioannis Sarantidis (INTRA)
Contributors	ATOS, ENG, i2CAT, ICCS, INNOV, FILL	Reviewers	Orfefs Voutyras (ICCS)
			Estela Carmona Cejudo (i2CAT)

Keywords:
System integration, CI/CD, Pledger Core System, Microservices, StreamHandler

Disclaimer

This document is issued within the frame and for the purpose of the Pledger project. This project has received funding from the European Union’s Horizon2020 Framework Programme under Grant Agreement No. 871536. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

[The dissemination of this document reflects only the author’s view and the European Commission is not responsible for any use that may be made of the information it contains. This deliverable is subject to final acceptance by the European Commission.

This document and its content are the property of the Pledger Consortium. The content of all or parts of this document can be used and distributed provided that the Pledger project and the document are properly referenced.

Each Pledger Partner may use this document in conformity with the Pledger Consortium Grant Agreement provisions.

(*) Dissemination level: **PU**: Public

Document information

List of Contributors	
Name	Partner
Antonio Castillo	ATOS
Francesco Iadanza	ENG
Gabriele Giammatteo	ENG
Verena Stanzl	FILL
August Betzler	i2CAT
Nikolaos Kapsoulis	INNOV
Alexandros Psychas	ICCS
Ioannis Sarris	INTRA
Olga Segou	INTRA
Ioannis Sarantidis	INTRA

Document History			
Version	Date	Change editors	Changes
0.1	22/10/2021	Ioannis Sarris (INTRA)	Initial ToC
0.2	29/10/2021	Francesco Iadanza (ENG)	Updated section 4.2.2 Integration endpoints and provide integration sequence diagrams
0.3	05/11/2021	Antonio Castillo (ATOS)	Updated section 4.2.2 Integration endpoints and provide integration sequence diagrams
0.4	05/11/2021	Gabriele Giammatteo (ENG)	Updated section 4.2.2 Integration endpoints and provide integration sequence diagrams
0.5	05/11/2021	Alexandros Psychas (ICCS)	Updated section 4.2.2 Integration endpoints and provide integration sequence diagrams
0.6	05/11/2021	Nikolaos Kapsoulis (INNOV)	Updated section 4.2.2 Integration endpoints and provide integration sequence diagrams
0.7	05/11/2021	August Betzler (i2CAT)	Updated section 4.2.2 Integration endpoints and provide integration sequence diagrams
0.8	11/11/2021	Ioannis Sarris (INTRA)	Section 2, section 3 – Microservices approaches, and section 4 finalized
0.9	11/11/2021	Ioannis Sarantidis (INTRA)	Section 3 – StreamHandler finalized
0.10	12/11/2021	Verena Stanzl (FILL)	Updated Integration Plan
0.11	12/11/2021	Olga Segou (INTRA)	Executive Summary, Introduction, and Conclusions finalized
0.12	12/11/2021	Ioannis Sarris (INTRA)	Version ready for internal review
0.13	15/11/2021	Estela Carmona Cejudo (i2CAT)	Reviewed version from i2CAT
0.14	24/11/2021	Orfefs Voutyras (ICCS)	Reviewed version from ICCS
0.15	26/11/2021	Ioannis Sarris (INTRA)	Version after internal review
0.16	28/11/2021	Carmen San Román (ATOS)	Quality assurance review
1.0	30/11/2021	Lara López (ATOS)	Final version

Document name:	D5.2 Pledger integrated demonstrator I	Page:	2 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Ioannis Sarris (INTRA)	26/11/2021
Quality manager	Carmen San Roman Alonso (ATOS)	28/11/2021
Project Coordinator	Lara Lopez Muñiz (ATOS)	30/11/2021

Table of Contents

Document information	2
Table of Contents	4
List of Tables.....	6
List of Figures	7
List of Acronyms.....	8
Executive summary	10
1 Introduction	11
1.1 Purpose of the document.....	11
1.3 Relation to other project work.....	12
1.4 Structure of the document	12
2 Pledger CI/CD platform	13
2.1 Overview.....	13
2.2 Version control system.....	15
2.4 Automated build & testing.....	17
2.4.1 Create & configure Jenkins Pipeline	18
2.4.2 Jenkinsfile.....	21
2.5 Portainer.....	22
2.6 JFrog container registry	23
2.7 Security features of the CI/CD Platform.....	26
2.7.1 Encrypted communications over HTTPs.....	26
2.7.2 Hard disk encryption at rest.....	27
2.7.3 User authentication of the CI/CD services	27
2.7.4 Firewall protection of the VMs hosted in Hetzner Cloud.....	27
2.7.5 SSH key-based authentication	27
3 Integration patterns.....	28
3.1.1 Microservices approach	28
3.1.2 StreamHandler platform	29
4 Pledger core system integration.....	35
4.1 Pledger integration plan	35
4.1.1 Pledger integrated demonstrator I.....	38
4.1.2 Pledger integrated demonstrator II	38
4.2 Integration methodology	38
4.2.1 Integration matrix	38
4.2.2 Integration endpoints	40
5 Conclusions	54

5.1 Key take-aways	54
5.2 Planned work.....	54
6 References	55
Annex A – Pledger architecture	56

Document name:	D5.2 Pledger integrated demonstrator I				Page:	5 of 58	
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status:	Final

List of Tables

<i>Table 1: Pledger Integration Plan</i>	35
<i>Table 2: Pledger Core System Integration Matrix</i>	39
<i>Table 3: Pledger Core System Integration Endpoints</i>	40
<i>Table 4: Orchestrator - Recommender Integration endpoint description</i>	42
<i>Table 5: Orchestrator – SaaS/IaaS Monitoring Engine endpoint description</i>	42
<i>Table 6: Orchestrator – Configuration DB endpoint description</i>	43
<i>Table 7: Orchestrator – PLEDGER SOE Framework endpoint description</i>	44
<i>Table 8: Recommender – SaaS/IaaS Monitoring Engine endpoint description</i>	44
<i>Table 9: Recommender – Configuration DB endpoint description</i>	45
<i>Table 10: Recommender – Metrics DB endpoint description</i>	45
<i>Table 11: Recommender – SLA notifier endpoint description</i>	46
<i>Table 12: SaaS/IaaS Monitoring Engine – SLA Monitoring endpoint description</i>	46
<i>Table 13: Configuration DB – App Profiler endpoint description</i>	47
<i>Table 14: Configuration DB – UI Configuration Dashboard endpoint description</i>	47
<i>Table 15: Configuration DB – Benchmarking Scheduler endpoint description</i>	48
<i>Table 16: Configuration DB – SLA Manager endpoint description</i>	49
<i>Table 17: App Profiler – Benchmarking creator endpoint description</i>	49
<i>Table 18: SLA Notifier – SLA SC bridge endpoint description</i>	50
<i>Table 19: SLA Notifier – T&R Engine endpoint description</i>	51
<i>Table 20: Pledger SOE Framework – Pledger RAN Controller endpoint description</i>	52

List of Figures

Figure 1: Pledger Integration approach.	11
Figure 2: Pledger CI/CD Virtual Servers in Hetzner Cloud	13
Figure 3: Pledger CI/CD platform	14
Figure 4: Pledger private Gitlab Group	16
Figure 5: Pledger Jenkins jobs	17
Figure 6: Jenkins pipeline stages	18
Figure 7: New item creation in Jenkins	18
Figure 8: Connect the new pipeline to Gitlab	19
Figure 9: Enable project-based security	19
Figure 10: Add build triggers to the new pipeline	20
Figure 11: Build job after other projects finished	20
Figure 12: Configure the new pipeline	21
Figure 13: Jenkinsfile example	22
Figure 14: Portainer home screen	23
Figure 15: Portainer docker host management	23
Figure 16: Browser warning for untrusted certificates	24
Figure 17: Jfrog Private Docker Registry	25
Figure 18: JCR Storage limit settings	25
Figure 19: Pledger Registry max unique tags	26
Figure 20: Example of Microservices Architecture using StreamHandler	29
Figure 21: Overview of Integration process after M24	54
Figure 22: The Pledger Core Subsystems	57
Figure 23: The Pledger Core System	58

List of Acronyms

Abbreviation / acronym	Description
ACL	Access Control Lists
API	Application Programming Interface
App	Application
CA	Certificate Authority
CD	Continuous Deployment
CI	Continuous Integration
CPU	Central Processing Unit
DB	Database
DLT	Distributed Ledger Technology
DoS	Denial of Service
DSM	Design Structure Matrix
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
ID	Identifier
IoT	Internet of Things
JCR	JFrog Container Registry
K8s	Kubernetes
LDAP	Lightweight Directory Access Protocol
MitM	Man in the Middle
Mx	The #x Month of the project
QoS	Quality of Service
RAN	Radio Access Network
RBAC	Role Based Access Control
REST	Representational State Transfer
SaaS	Software as a Service
SASL	Simple Authentication and Security Layer
SC	Smart Contract
SLA	Service Level Agreement
SOE	Slicing and Orchestration Engine
SSH	Secure Shell
SSL	Secure Sockets Layer
SYN	Synchronize flag
T&R Engine	Trust and Reputation Engine
TLS	Transport Layer Security

Abbreviation / acronym	Description
UC	Use Case
UI	User Interface
URL	Uniform Resource Locator
VCS	Version Control System
VM	Virtual Machine
WP	Work Package
XMAS packet	Christmas tree packet
YAML	Yet Another Markup Language

Executive summary

Pledger is an innovative project that delivers a new architectural paradigm and a toolset that will pave the way for next generation edge computing infrastructures by enabling:

- ▶ edge computing providers to enhance the stability and performance effectiveness of their edge infrastructures,
- ▶ edge computing adopters to understand and manage the computational nature of their applications.

By providing this toolset, the project will also allow third parties to act as independent validators of QoS features in IoT applications, enabling new decentralised applications and business models, thus filling a large gap in the emerging edge/IoT computing market landscape.

This document describes the project's methodology and approach to the integration of the first end-to-end prototype and demonstrator. The produced work depicted in this deliverable is based on the best practices, methodologies and tools used and applied by INTRASOFT. There is a common set of DevOps, integration methodologies and relevant open source widely used tools by the software community supporting the entire software development, integration, testing, and deployment processes that INTRASOFT adopts and uses within EU funded projects indicatively STAMINA, LIFECHAMPS, CUREX, TRUSTONOMY, PLURAL and NUTRISHIELD. According to the Pledger project needs, the respective open-source tools are deployed and customized in the project's dedicated environment. INTRASOFT has proven know-how and experience in DevOps/DevSecOps methods and has applied them for software development, integration, testing and deployment in the context of many complex software systems and solutions in diverse application contexts, as part of both its commercial products and services as well as in research and development projects.

It provides a useful reference to the technical work underlying Pledger and integration good practices adopted by the consortium. The public nature of the document allows any interested reader to look into the integration of Pledger components that will be open-sourced. Pledger adopts a Continuous Integration/Continuous Deployment (CI/CD) approach to enable the quick development and automated building, testing, and deployment of source code. The CI/CD approach allows Pledger to reduce development time and costs, as well as enable security features and maintain a level of trust in the final software components.

Pledger also introduces integration patterns in the form of a microservice approach and the StreamHandler big data platform. These are important technologies that form the fabric for the deployment and interoperation of software components. The use of REST Application Programming Interfaces, and the introduction of publish/subscribe mechanisms facilitate the interconnection of the deployed software components that form the backbone of the Pledger infrastructure. Following the definition of the CI/CD pipeline and the integration patterns, an analysis is presented of the existing integration points among the varied components including the types of protocols utilised for inter-component communications.

This agile approach brings multiple benefits to Pledger and allows the monitoring of deployment, testing, and integration process towards the realisation of the first prototype and end-to-end demonstrator of the Pledger core subsystems.

Document name:	D5.2 Pledger integrated demonstrator I			Page:	10 of 58		
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status:	Final

1 Introduction

1.1 Purpose of the document

With the recent exponential proliferation of service-based infrastructures, it becomes clear that the complex and decentralised nature of the Edge-Cloud continuum introduces multiple technical challenges. Many services have critical QoS requirements and perform intensive operations from a computational, memory, network, or data transfer perspective. Hence, it is crucial to ensure that the infrastructure couples the benefits of low latencies on the edge, with the robustness and resilience of cloud infrastructures. Pledger is an innovative project that delivers a new architectural paradigm and a toolset that will pave the way for next generation edge computing infrastructures by enabling:

- ▶ **edge computing providers** to enhance the stability and performance effectiveness of their edge infrastructures, through modelling the overheads and optimal groupings of concurrently running services, runtime analysis and adaptation, thus gaining a competitive advantage,
- ▶ **edge computing adopters** to understand the computational nature of their applications, investigate abstracted and understandable QoS metrics, facilitate trust and smart contracting over their infrastructures, and identify how they can balance their cost and performance to optimise their competitiveness and monitor their SLAs.

The provided toolkit fills a significant need in the expanding Edge/IoT market environment. It allows for the integration and interoperation of decentralized applications, as well as independent third-party validation. This allows the creation of novel business models and improves the adoption of these technologies.

This document introduces the major components of the Pledger Core and focuses on the integration methodology and mechanisms applied within the platform, so as to realise the first end-to-end prototype of the Pledger Core subsystems. Pledger adopts a methodological approach, starting with the deployment of a CI/CD pipeline, along with a definition of the common integration patterns and a comprehensive listing of the integration points, culminating in the integration plan.

The overall approach is summarized in Figure 1.

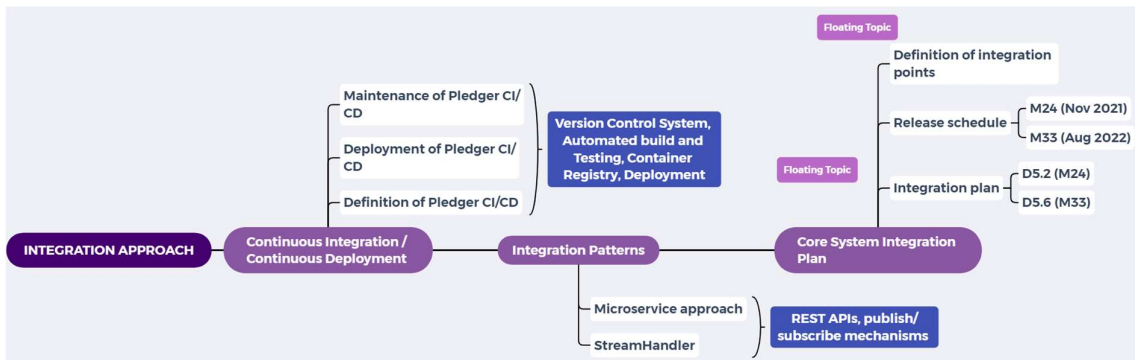


Figure 1: Pledger Integration approach.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	11 of 58
Reference:	D5.2	Dissemination:	PU
Version:	1.0	Status:	Final

1.3 Relation to other project work

This document provides reference to the deliverable D2.3 “Pledger Overall Architecture” [1], as the analysis herein is performed based on the Pledger conceptual architecture. For easy reference, an overview of the Pledger architecture is provided in Annex A. Following this document, D5.3 “Pilots operations and monitoring I” [2] will complement this work by providing additional information on the Use Case integration and UC-specific applications that will be deployed for the project’s integrated pilots. Furthermore, this document delivers the preliminary integration plan, while the final version of the integrated demonstration will be featured in the upcoming D5.6 “Pledger integrated demonstrator II” which will be delivered by M33 (Aug 2022).

1.4 Structure of the document

This document is structured as follows:

- ▶ **Introduction (present section)** presents the document scope and relation to the project work.
- ▶ **Section 2** describes the Continuous Integration/Continuous Deployment (CI/CD) platform set up for Pledger.
- ▶ **Section 3** presents the basic Integration patterns such as the microservice approach and the data management backbone.
- ▶ **Section 4** enumerates the Integration points identified within the Pledger Core and presents the integration plan.
- ▶ **Section 5** presents the key conclusions and the way forward in terms of the end-to-end integration of the project components.

Document name:	D5.2 Pledger integrated demonstrator I				Page:	12 of 58	
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status:	Final

2 Pledger CI/CD platform

The architecture of the Pledger CI/CD infrastructure is provided in this section (along with a detailed explanation of its software components), the purpose of which is to integrate, test, and release the Pledger software components.

2.1 Overview

The Pledger CI/CD platform is implemented as a collection of open-source software components with the goal of creating an automated build system capable of incorporating code changes performed by developers working on the different software components of the Pledger project.

The software components that comprise the Pledger CI/CD platform have been deployed on virtual hosts, which are essentially cloud servers that have been created on a public cloud provider (Hetzner Cloud [3]). These virtual servers are hosted in Germany and are following GDPR regulations. On each virtual server, effective firewall rules have been defined to securely manage inbound and outbound network traffic, ensuring that only authorized users and applications are permitted access. In addition, all the storage units (either ephemeral or persistent volumes) are encrypted in order to mitigate the risks of data exposure.

Figure 2 shows the project space created on Hetzner Cloud to meet the demands of Pledger, as well as some of the instantiated virtual servers.

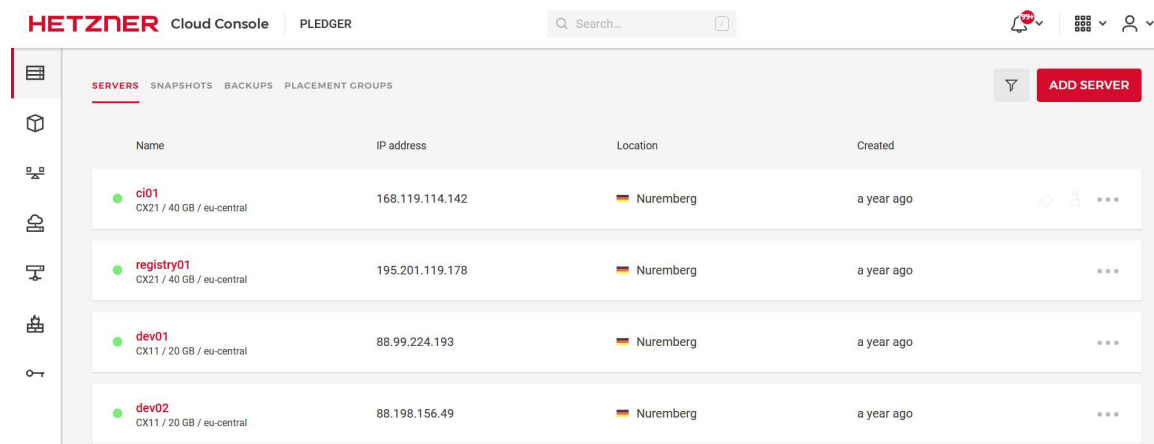


Figure 2: Pledger CI/CD Virtual Servers in Hetzner Cloud

Using Pledger CI/CD platform, developers will be able to push their source code changes, add new component features or integration endpoints, and push their code to Gitlab, the central source code repository in this environment. This action will trigger Jenkins to pull, compile, build, and test the most recent version of the source code. Jenkins will then create a dockerized image, which will be pushed to the private docker registry. Finally, once the components images have been pushed to the private docker registry, they may be pulled and deployed either in the isolated Docker VMs that compose the development environment or in the production K8s cluster that hosts the Pledger Core System software components.

By using this strategy, developers will be able to independently build, debug, and release their software components by defining their own automated pipelines in Jenkins. This way, whenever a software component needs to be updated and/or redeployed, the pipeline of this component will be automatically triggered by a Gitlab webhook, and new updates and modifications to the component will become available.

The following are the major features of the Pledger CI/CD platform:

- ▶ **Distributed CI/CD environment** including remote access and built-in security. Deploying the software components and services of the CI/CD platform on distinct Virtual Machines (VMs) simplifies the process of horizontally scaling the platform, if needed, by adding more VMs where necessary.
- ▶ **Encrypted communications** via TLS/HTTPS protocols ensure secure communication among the deployed software components of the CI/CD platform.
- ▶ **Encryption at rest** guarantees that data are unavailable to unauthorized parties in case of misplacement of the physical disk units that host the virtual servers.
- ▶ **Isolation and protection** from unauthorized access hosts due to defined firewall rules.
- ▶ **Triggerable events** (commit, push, merge, etc.) that will automatically start building, testing, and deploying software components with additional support for multiple environments (development, test, production) and multiple branches.

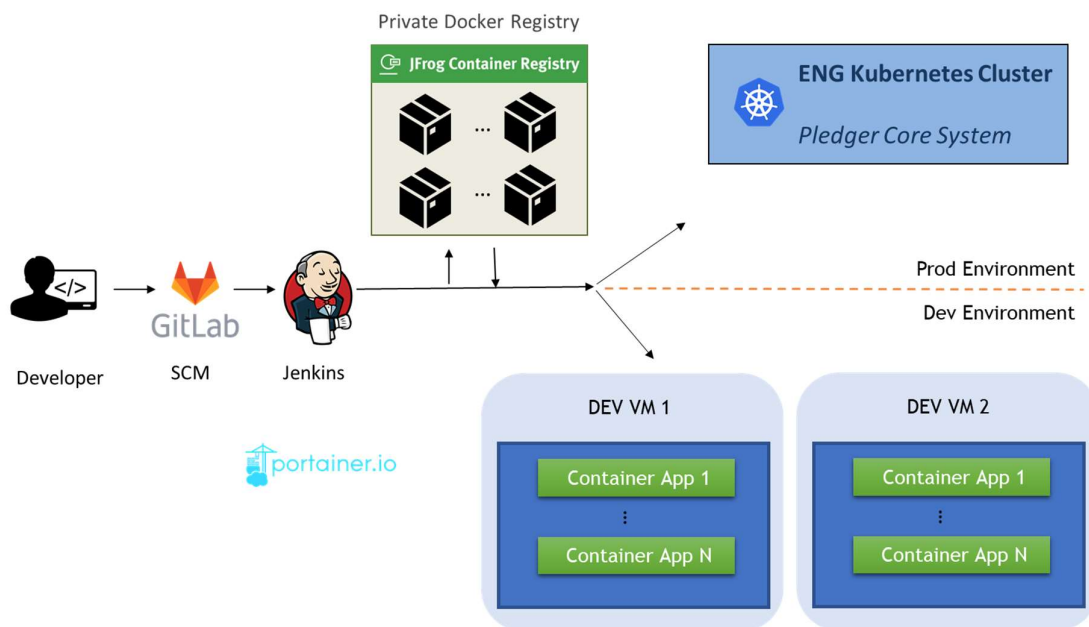


Figure 3: Pledger CI/CD platform

The following software components comprise the Pledger CI/CD platform:

- ▶ **Gitlab** [4]: A simple yet robust and user-friendly git Version Control System (VCS). It provides a web-based Graphical User Interface (GUI) with various built-in capabilities including version control, bug tracking, code review, wiki, etc. Multiple developers can concurrently create, merge, and edit their code independently, before pushing their changes to the common Gitlab repository. Rather than maintaining our own Gitlab server, we relied on the use of Gitlab projects hosted on gitlab.com for the purposes of the Pledger project.
- ▶ **Jenkins** [5]: An open-source automation server that serves as the Continuous Integration (CI) server, automating the software development lifecycle including building, testing, and deploying applications. Jenkins can automate operations such as software builds, unit testing, packaging, pushing container images to the private Docker registry, and deploying the images either in the production K8s cluster or on the separated development Docker VMs. A Jenkins Docker instance has been deployed in Hetzner cloud infrastructure to meet the demands of the Pledger project.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	14 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

- ▶ **Portainer** [6]: An open-source software application for administering container-based applications in a variety of virtualization settings. Portainer has been used to configure and administer the development environment of the Docker VMs, deploy applications, monitor application performance, troubleshoot issues, and implement role-based access control.
- ▶ **JFrog Container Registry** [7]: An open-source software application that implements a private Docker Registry for storing and distributing images of project assets. By integrating image storage and distribution into the Pledger development lifecycle, it is used to securely control where the container images are being stored. JFrog Container Registry Docker instance has been deployed in Hetzner cloud infrastructure.
- ▶ **Protected Docker Daemon [8] on development Docker VMs**: Containerized services can be deployed in the development environment before arriving on the production environment. Docker Daemon is a process that runs in the background of the development environment VMs and handles Docker images, containers, networks, and storage volumes. The Docker Daemon is always listening to and processing Docker API requests. Docker Daemon has been configured to listen to API requests over HTTPs only for clients that are authorized and have the appropriate client certificates.
- ▶ **K8s cluster** [9]: A Kubernetes cluster has been configured and used to host the production environment of the different services of the Pledger Core system.

2.2 Version control system

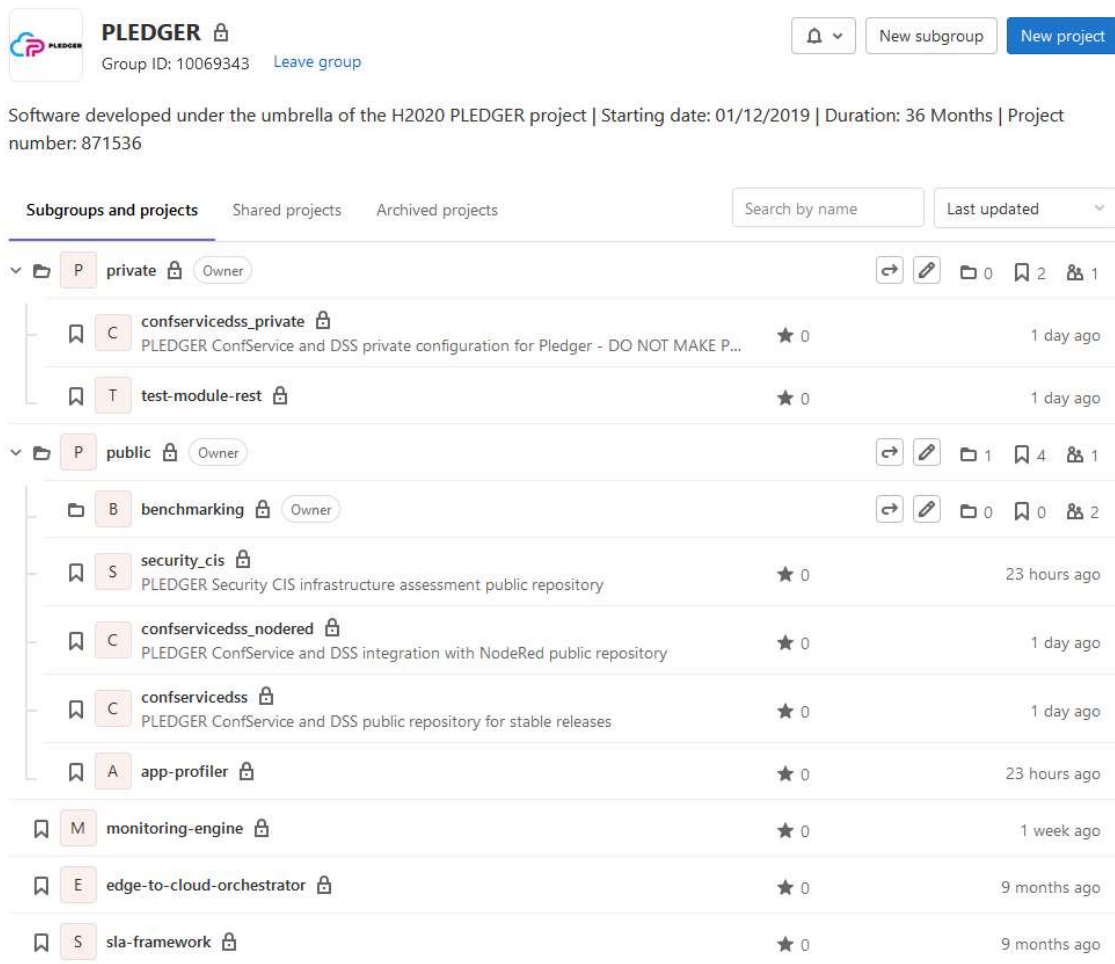
The process of recording and controlling software code changes is known as version control. Version control systems (VCS) are software tools that assist software development teams in managing source code changes over time. A VCS stores every change of the code in a special kind of database. It is a remote repository of files including software application's source code. If an error occurs, developers may compare previous versions of the code to fix the error while minimizing disruption to all team members.


Within the Pledger CI/CD platform, Gitlab is used as a VCS. It is an easy yet powerful and intuitive git VCS. Multiple developers can concurrently create, merge, and edit their code independently before pushing their changes to the common Gitlab repository.

A dedicated public Gitlab group named "PLEDGER" has been created for the purposes of the Pledger project, as illustrated in Figure 4. Each partner has access to their own group or individual projects under this group, which they may use to push their code. Under this group, two main subgroups were created:

- ▶ A **public subgroup** that contains the software components of Pledger that are publicly available. These public repositories can be easily forked into private Gitlab repositories by the open-source community, add new features, push back to the forked repository, and create a pull request in order to be included to the original repository. The link to the public subgroup is the following: <https://gitlab.com/pledger/public>.
- ▶ A **private subgroup** that contains the software components of the Pledger project that stays as proprietary software and intellectual property of the partner involved. The link to the private subgroup is the following: <https://gitlab.com/pledger/private>.

Document name:	D5.2 Pledger integrated demonstrator I				Page:	15 of 58
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status: Final



PLEDGER  Group ID: 10069343 [Leave group](#)

Software developed under the umbrella of the H2020 PLEDGER project | Starting date: 01/12/2019 | Duration: 36 Months | Project number: 871536

Subgroups and projects Shared projects Archived projects Search by name Last updated













- private**  Owner 0 2 1
 - confservicedss_private**  PLEDGER ConfService and DSS private configuration for Pledger - DO NOT MAKE P... ★ 0 1 day ago
 - test-module-rest**  ★ 0 1 day ago
- public**  Owner 1 4 1
 - benchmarking**  Owner 0 0 2
 - security_cis**  PLEDGER Security CIS infrastructure assessment public repository ★ 0 23 hours ago
 - confservicedss_nodered**  PLEDGER ConfService and DSS integration with NodeRed public repository ★ 0 1 day ago
 - confservicedss**  PLEDGER ConfService and DSS public repository for stable releases ★ 0 1 day ago
 - app-profiler**  ★ 0 23 hours ago
 - monitoring-engine**  ★ 0 1 week ago
 - edge-to-cloud-orchestrator**  ★ 0 9 months ago
 - sla-framework**  ★ 0 9 months ago

Figure 4: Pledger private Gitlab Group

At the time that this deliverable was released the repositories *monitoring-engine*, *edge-to-cloud-orchestrator* and *sla-framework* were directly under the subgroup “PLEDGER”. This is planned to be classified to */public* and */private* subgroups as a next step.

Finally, each Gitlab repository should include:

- ▶ **Jenkinsfile:** A definition file that contains the Jenkins pipeline including the steps that will be followed (build, test, push to registry, deploy, etc.) during the CI/CD process.
- ▶ **Dockerfile:** A text-based script providing instructions for creating a container image.
- ▶ **deployment.yaml:** A YAML definition file for specifying Kubernetes objects (pods, services, deployments, etc.). This needs to be included in case the component is deployed in K8s cluster.

Gitlab, like any other git VCS, has extensive branching features. In most circumstances, there is one main branch from which a developer working on a specific feature or bug fix creates an additional development branch. When the developers have finished their code modifications, they merge their branch back into the main branch.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	16 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

2.4 Automated build & testing

The Continuous Integration (CI) process automates the integration of code changes from multiple contributors into a single software project. Each integration cycle includes automated builds and unit tests on the most recent code changes to detect any errors as soon as they occur. CI's primary objectives are to detect and fix any bugs faster, to increase software quality, and to shorten the time it takes to verify and deploy new software updates.

The CI process includes the following steps:

- ▶ Developers implement a new feature and change the source code in their local repositories.
- ▶ Once completed, they commit their changes to the shared repository (which is in our case Gitlab).
- ▶ The CI server receives a notification that there are some changes to the shared repository.
- ▶ The CI server starts its pipeline described in Jenkinsfile by pulling first the latest version of the source code and executing any unit tests.
- ▶ The server creates deployable artifacts for testing.
- ▶ The CI server assigns a build tag to the code version that was just created.
- ▶ The CI server provides the development team with information on successful builds and tests, as well as logs when a build or test fails.
- ▶ The teams resolve any issues.
- ▶ It continues to integrate and run tests during the entire project.

For the Pledger project Jenkins was chosen as the CI server. Jenkins is operating in a Docker container on a Hetzner Cloud Virtual Machine (VM) as shown in Figure 5 and is available in the following link:

<https://static.200.2.203.116.clients.your-server.de/>.

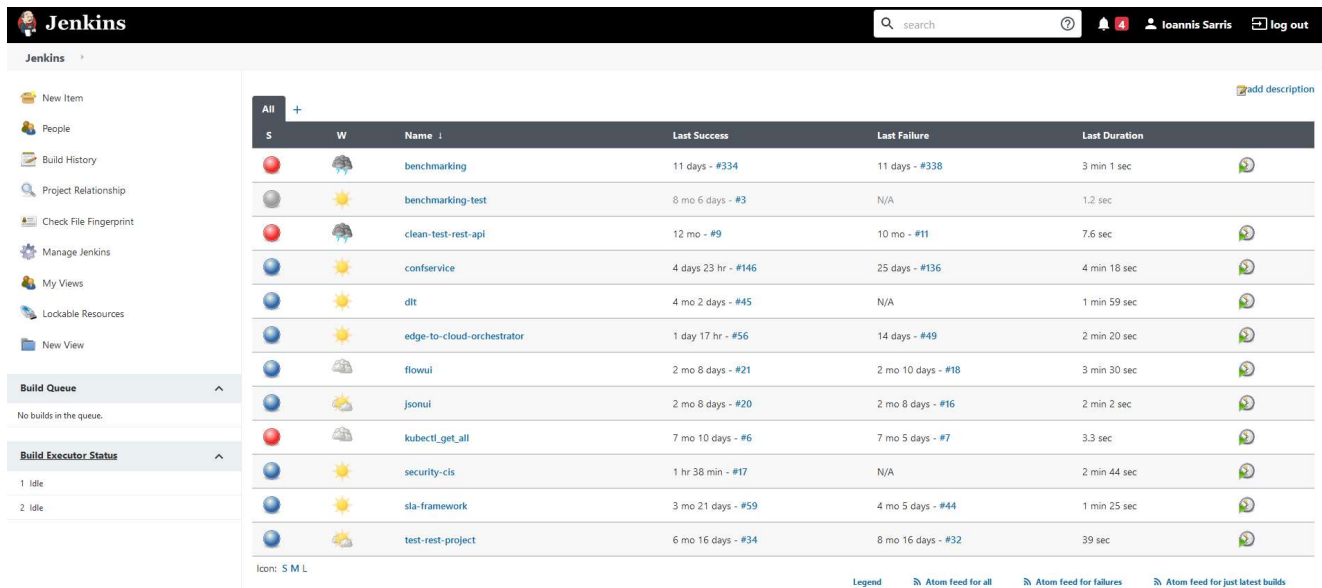


Figure 5: Pledger Jenkins jobs

Every repository of the Pledger public or private group on Gitlab is planned to be linked to a Jenkins pipeline, as shown in Figure 5. Each pipeline is described in a specific file known as Jenkinsfile. Every event that occurs as a result of source code changes on the Gitlab repositories (i.e., merge, push etc.) will trigger a new build on Jenkins and the steps described in Jenkinsfile will be executed. As indicated in the figure below, the pipeline contains compilation, build, and testing stages.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	17 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

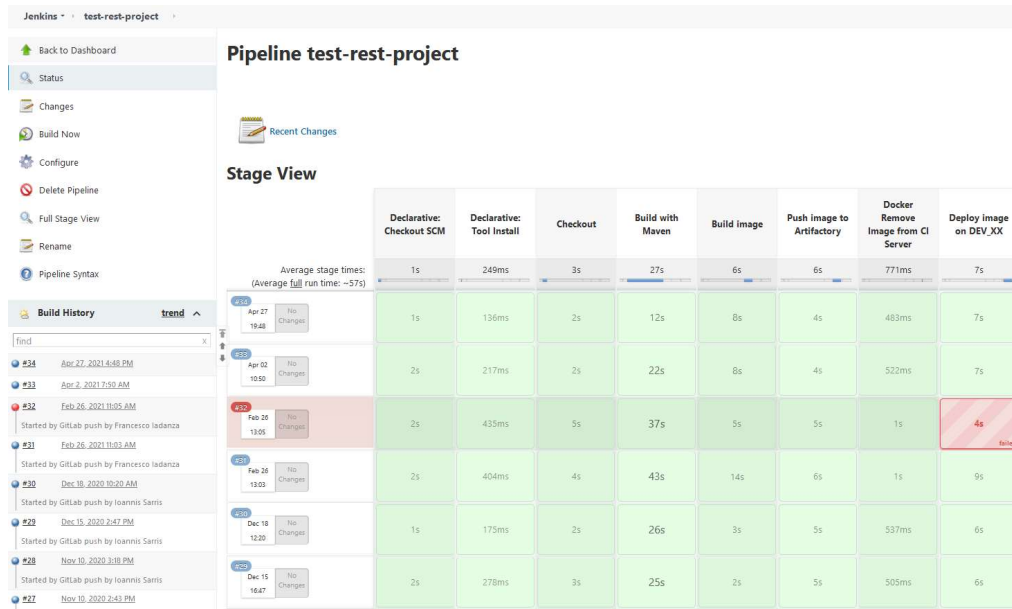


Figure 6: Jenkins pipeline stages

2.4.1 Create & configure Jenkins Pipeline

The following steps can be used in order to integrate Gitlab with Jenkins and enable automated pipeline triggers when source code is pushed to the repository:

- ▶ Create a New Item in Jenkins:

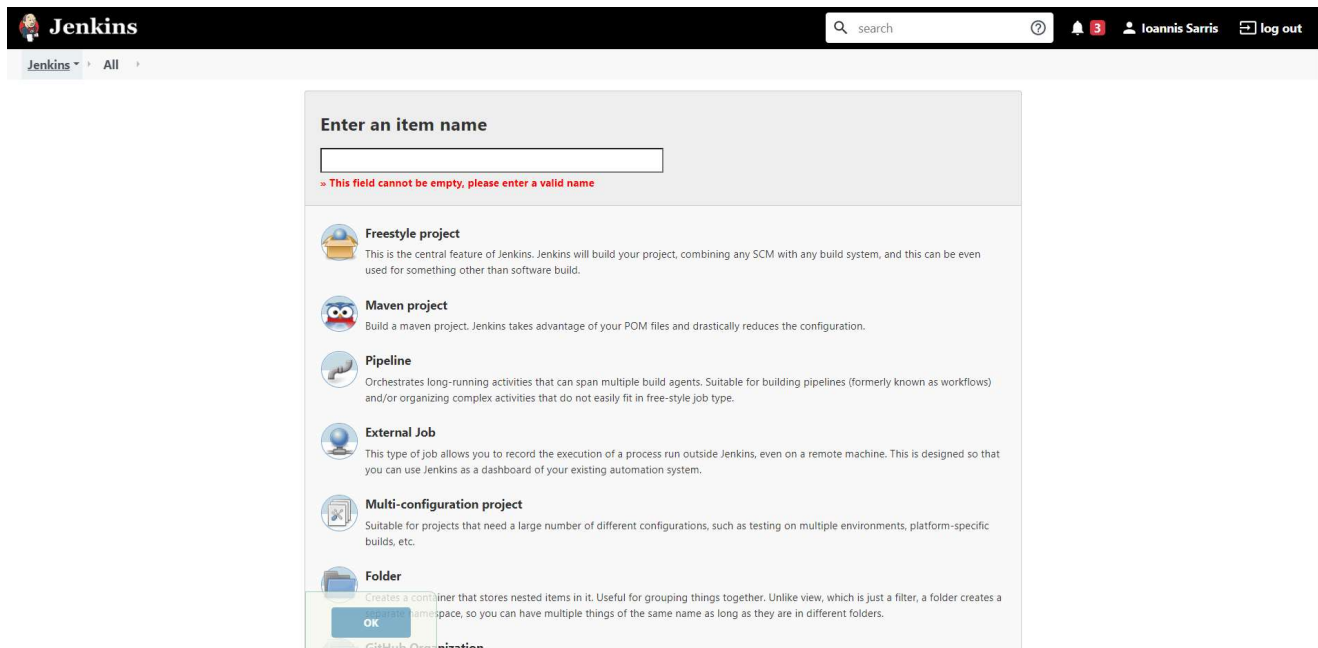


Figure 7: New item creation in Jenkins

- ▶ Enter an item name. It is recommended to use a name that corresponds to the Gitlab repository. Choose “Pipeline” or “Multibranch Pipeline” (if the repository has more than one branch i.e., main, development, etc.) and click OK.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	18 of 58	
Reference:	D5.2	Dissemination:	PU	
	Version:	1.0	Status:	Final

► In the **General** tab:

- Please select as Gitlab connection: “**Gitlab**”.

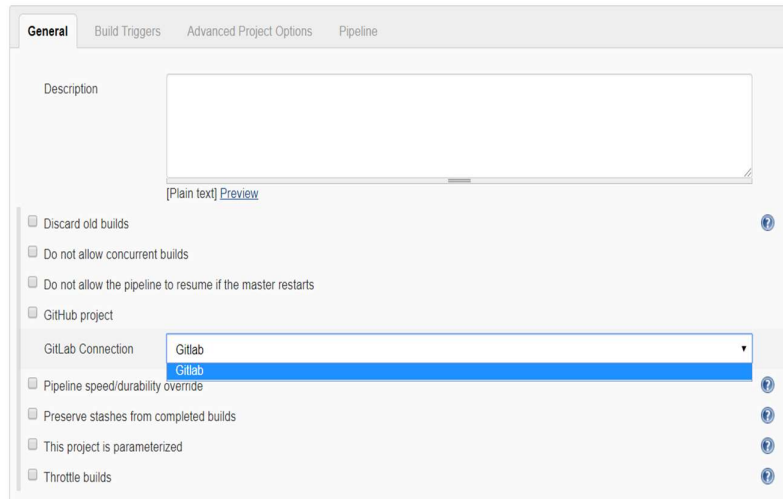


Figure 8: Connect the new pipeline to Gitlab

- Enable project-based security.
- Grant to admin user of Jenkins **plgadmin01** all the permissions of the job.
- Grant the dedicated users that need to have access on the jobs with specific permissions.

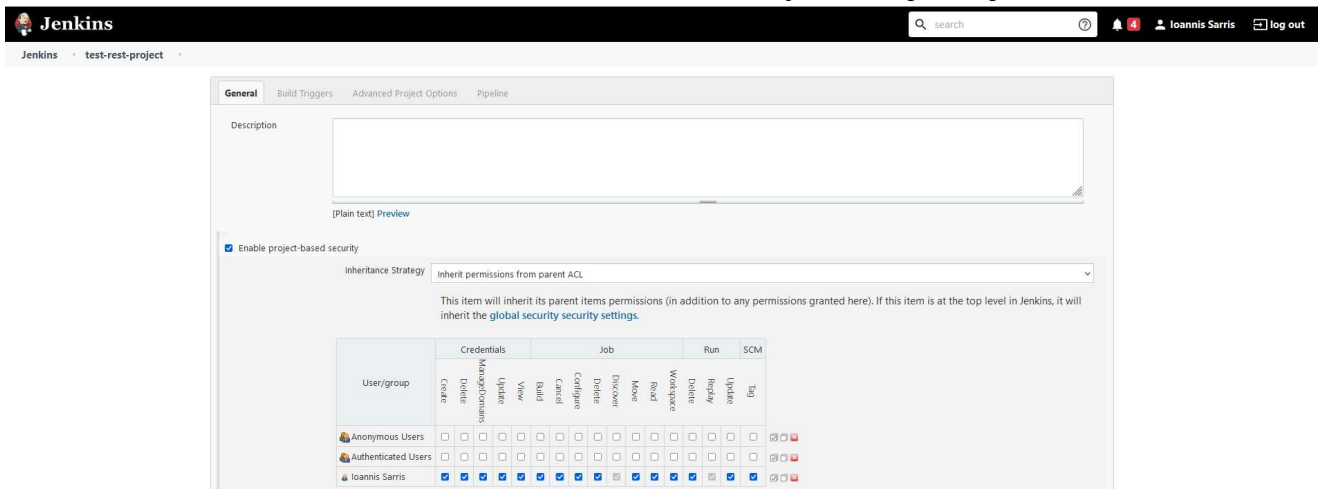


Figure 9: Enable project-based security

► In the **Build Triggers** tab:

- Select all boxes under Build when a change is pushed to GitLab.
- Rebuild open Merge Requests: Never.
- Comment (regex) for triggering a build: Jenkins please retry a build.

Build Triggers

Build after other projects are built

Build periodically

Build when a change is pushed to GitLab. GitLab webhook URL: <https://116.203.2.200/project/test-rest-project>

Enabled GitLab triggers

Push Events	<input checked="" type="checkbox"/>
Opened Merge Request Events	<input checked="" type="checkbox"/>
Accepted Merge Request Events	<input type="checkbox"/>
Closed Merge Request Events	<input type="checkbox"/>
Rebuild open Merge Requests	Never
Approved Merge Requests (EE-only)	<input checked="" type="checkbox"/>
Comments	<input checked="" type="checkbox"/>
Comment (regex) for triggering a build	Jenkins please retry a build

[Advanced...](#)

Figure 10: Add build triggers to the new pipeline

- The possibility to trigger other jobs after the job is finished is also available, if needed. This is very useful for creating dedicated jobs for testing purposes (integration test-jobs, acceptance test-jobs etc.).

Build Triggers

Build after other projects are built

Projects to watch

No project specified

Trigger only if build is stable

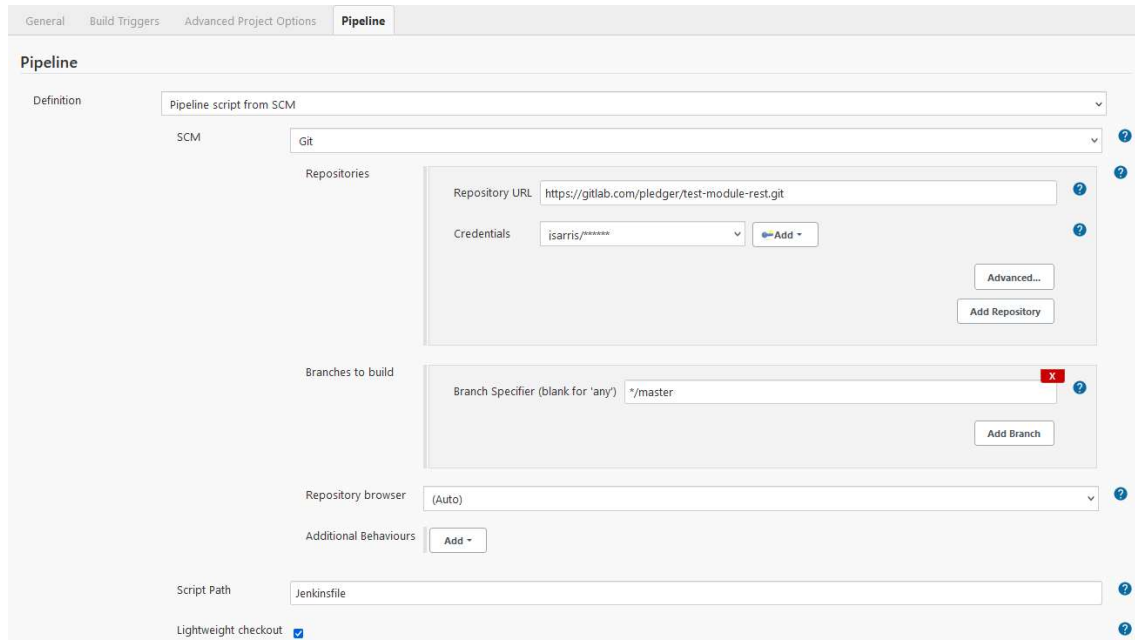
Trigger even if the build is unstable

Trigger even if the build fails

Figure 11: Build job after other projects finished

- In the Pipeline tab:
 - Definition: Pipeline script from SCM.
 - Add the Repository URL (e.g., <https://gitlab.com/pledger/private/test-module-rest.git>).
 - Add your Gitlab credentials (username/password).
 - In "Branches to build" optionally add the branch (e.g., */main, */development, etc.).
 - Add Script Path: <Path_to_Jenkinsfile>.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	20 of 58	
Reference:	D5.2	Dissemination:	PU	
	Version:	1.0	Status:	Final



The screenshot shows the Jenkins Pipeline configuration interface. At the top, there are tabs for 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline'. The 'Pipeline' tab is active. Below the tabs, the 'Definition' section is set to 'Pipeline script from SCM'. The 'SCM' dropdown is set to 'Git'. Under 'Repositories', the 'Repository URL' is 'https://gitlab.com/pledger/test-module-rest.git' and 'Credentials' is 'isarris/*****'. There are 'Advanced...' and 'Add Repository' buttons. The 'Branches to build' section has a 'Branch Specifier (blank for 'any')' set to '*/master' and an 'Add Branch' button. The 'Repository browser' is set to '(Auto)'. There is an 'Add -' button for 'Additional Behaviours'. The 'Script Path' is 'Jenkinsfile'. The 'Lightweight checkout' checkbox is checked.

Figure 12: Configure the new pipeline

► Press **save**, to save the configuration. The pipeline has been successfully created.

2.4.2 Jenkinsfile

Jenkins pipelines are specified in a test file called Jenkinsfile which allows engineers to implement pipelines in code format. Jenkinsfile allows one to write the steps required to create and execute a Jenkins pipeline. The pipeline is a collection of code-based instructions for continuous delivery that includes instructions for the complete build process. With pipeline one may build, test, and deploy the application. Jenkinsfiles are typically pushed into the Version Control System repository and are retrieved by Jenkins whenever a new build is started. Jenkinsfile has two syntaxes: declarative and scripting, both of which may be used to create CI/CD pipelines. Figure 13 illustrates an example of a Jenkinsfile that contains three stages: Build, Test, and Deploy, in both declarative and scripted pipeline.

```

// Declarative //

pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Building...'
      }
    }

    stage('Test') {
      steps {
        echo 'Testing...'
      }
    }

    stage('Deploy') {
      steps{
        echo 'Deploying...'
      }
    }
  }
}

// Script //

node {
  stage('Build') {
    echo 'Building...'
  }

  stage('Test') {
    echo 'Testing...'
  }

  stage('Deploy') {
    echo 'Deploying...'
  }
}

```

Figure 13: Jenkinsfile example

An example test project has been provided for this purpose with Jenkinsfile¹ (credentials needed) that is using the entire Pledger CI/CD pipeline.

2.5 Portainer

Portainer is a lightweight management UI that allows users to manage simply their docker hosts. It has been deployed as a single container on a VM in Hetzner Cloud. Portainer manages all the Docker resources including containers, images, volumes, networks, etc. Portainer is available via the following link: <https://static.200.2.203.116.clients.your-server.de:9000>

We have used self-signed certificates for the setup of SSL communication. For this reason, warnings that the site could not be verified as a trusted website will be displayed to the user's browser when accessing Portainer. In the context of Pledger, it has been utilized to manage the development environment of the project. The following figure presents two docker hosts that compose the development environment of the project. In addition, each docker host has a summary of its status.

¹ <https://gitlab.com/pledger/private/test-module-rest>

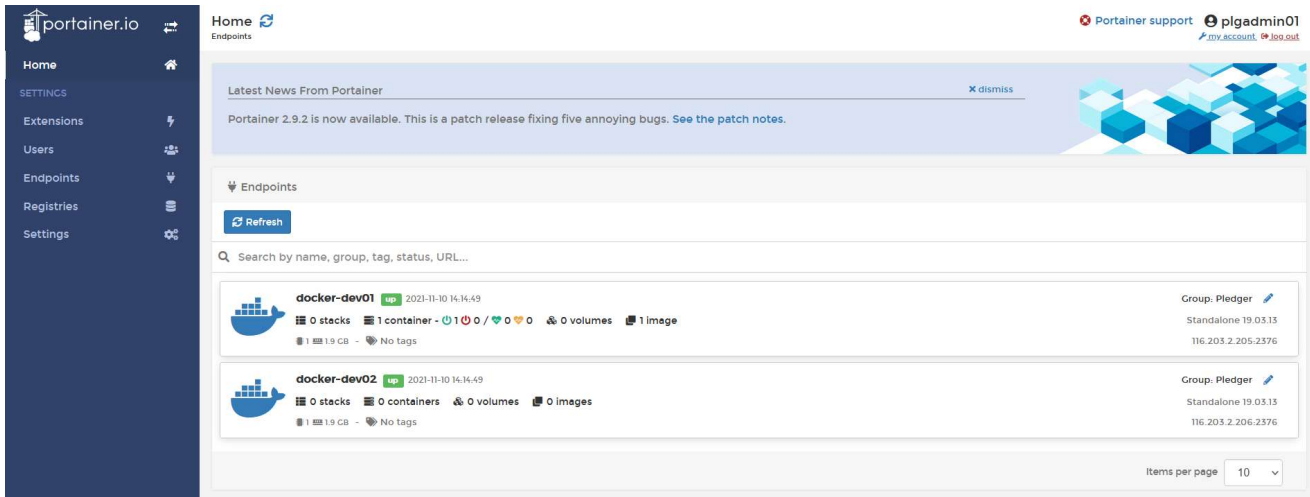


Figure 14: Portainer home screen

Finally, Portainer provides an extensive toolset that enables the monitoring and administration of each Docker instance individually, as illustrated in the following figure:

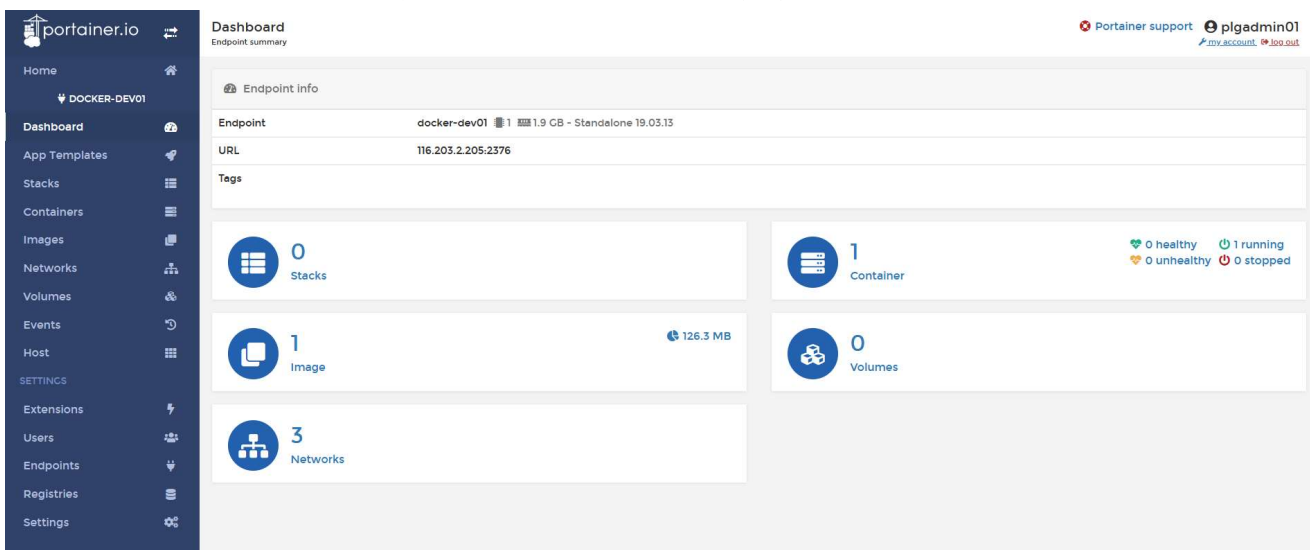


Figure 15: Portainer docker host management

2.6 JFrog container registry

One of the primary benefits of containers is the ease with which ready-to-use software images can be retrieved from a central repository. Users may push their image or their collection of tagged images to the same public registry, allowing everyone to benefit from the newly dockerized service.

However, users are sometimes reluctant to share their repository in public as it contains proprietary code or private information. Due to this limitation, setting up a private Docker Registry is a simple method to share images and manage their access across several Docker daemons and VMs.

In the context of the Pledger project, JFrog Container Registry (JCR) has been deployed and configured as a separate container on a dedicated VM to set up a secure private Docker Registry. The project’s developers may benefit using this registry to push and pull Docker images. JCR supports Docker and Helm registries, as well as generic repositories, allowing users to build, deploy and manage container images while offering sophisticated features with fine-grained access control and user-friendly UI. JCR

Document name:	D5.2 Pledger integrated demonstrator I	Page:	23 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

has no restrictions on the number of Docker Registries that can be utilized. JCR hosts two types of repositories:

- ▶ Local repositories
- ▶ Remote repositories

Virtual repositories may aggregate both local and remote repositories to establish controlled domains for artifact resolution and search. Local repositories are physical, locally maintained repositories where items may be deployed. Remote repositories act as a caching proxy for a repository managed already to a remote URL. With Virtual repositories, one may aggregate several local and remote repositories using a single URL. The repository is virtual in the sense that it may be resolved and obtain artifacts from it but it cannot be used to deploy anything on it.

The Pledger JCR server is accessible via the following URL:

<https://static.204.2.203.116.clients.your-server.de/ui/login/>

We have used self-signed certificates for the setup of SSL communication. For this reason, warnings that the site could not be verified as a trusted website will be displayed to the user's browser when accessing the Pledger JCR, similar to the one depicted in Figure 16. The user will have to proceed, click through the usual warnings for untrusted certificates, and accept the self-signed certificate.

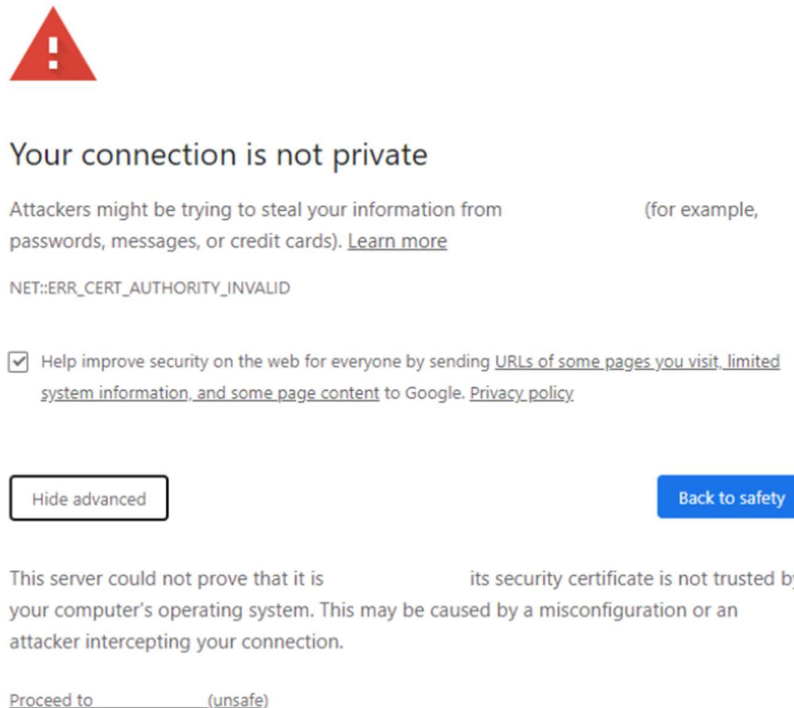


Figure 16: Browser warning for untrusted certificates

In the following figure a local repository has been setup with the following name: **plgregistry**

The URL of the private Docker registry is the following:

<https://static.204.2.203.116.clients.your-server.de:443/artifactory/plgregistry/>

Document name:	D5.2 Pledger integrated demonstrator I			Page:	24 of 58
Reference:	D5.2	Dissemination:	PU	Version:	1.0
				Status:	Final

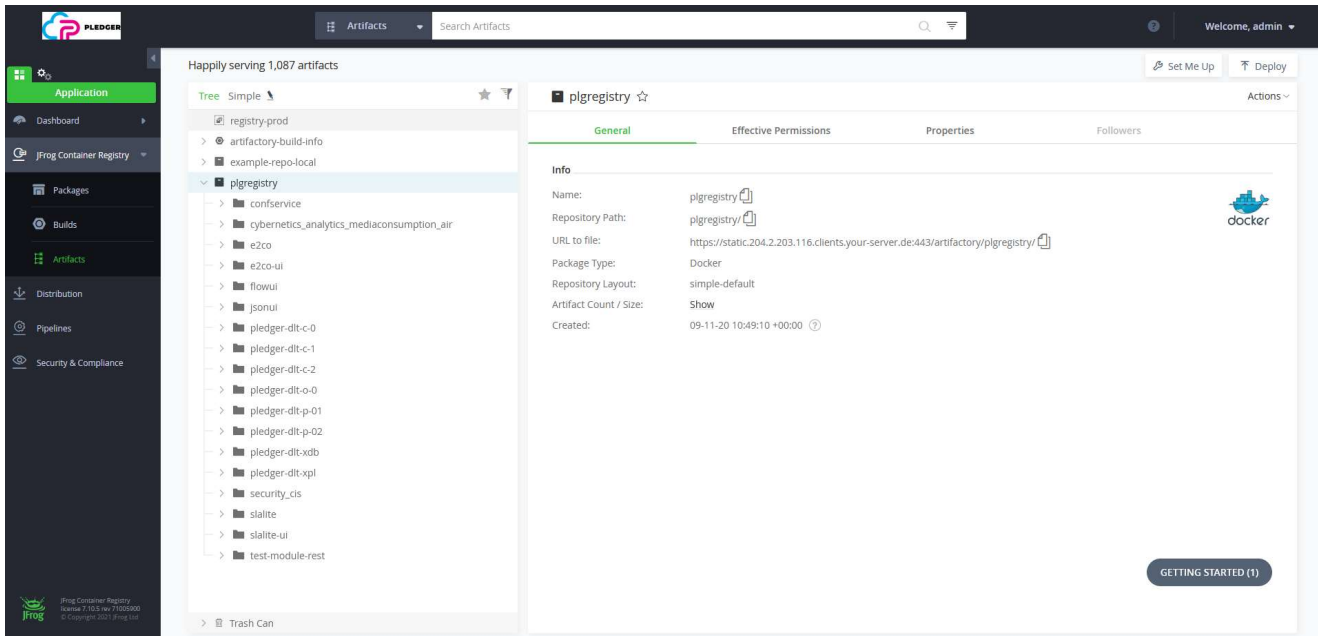


Figure 17: Jfrog Private Docker Registry

The following instructions can be used in order to be able to login the private docker registry once you have the appropriate user and password:

`docker login -u <user> -p <password>`

<https://static.204.2.203.116.clients.your-server.de:443/artifactory/plgregistry>

JCR allows you to restrict the storage capacity allowed for repositories in order to avoid running out of space. The storage capacity restriction has been setup to 90 percent of the disk space available of the VM. Any attempt to store binaries above the allocated sotrage will fail with an error. In addition, when 85 percent of disk space is achieved, a storage space warning is sent.

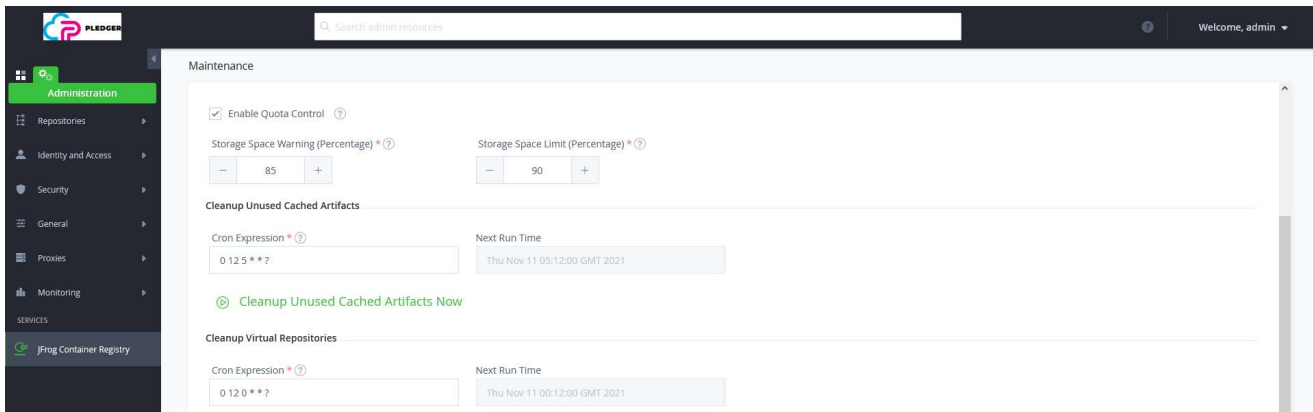


Figure 18: JCR Storage limit settings

Finally the feature "cleanup by max unique tags" has been activated and set to 5. This value is the number of the maximum number of unique tags that may be assigned to a single Docker image stored in **plgregistry** repository. When the number of tags exceeds this limit, older tags are deleted automatically.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	25 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

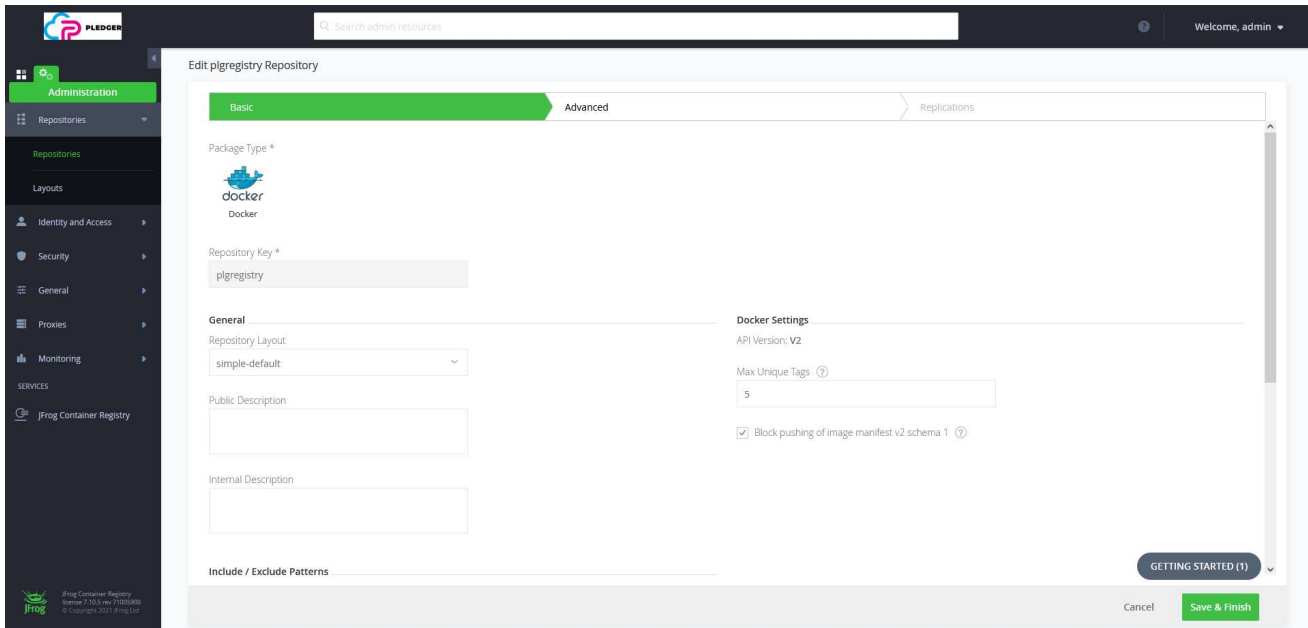


Figure 19: Pledger Registry max unique tags

2.7 Security features of the CI/CD Platform

A collection of security measures has been considered in the Pledger CI/CD solution to protect both the CI/CD software components and services and the deployed artifacts. In the following sections we describe in depth these security assets.

2.7.1 Encrypted communications over HTTPS

Access to the offered CI/CD services is secured with HTTPS to protect the connections of the users to the deployed applications. The continuous integration (Jenkins), and the management and the artifacts repository (JFrog Container Registry) have all been protected with HTTPS and provided an encrypted and safe client experience. Additionally, all data are delivered over HTTPS using the Transport Layer Security protocol (TLS), which provides three key layers of protection:

- ▶ Encryption - encrypting the exchanged data to keep it secure from eavesdroppers. That means that while users are using an HTTPS secured service, nobody can "listen" to their conversations, track their activities across multiple pages, or steal their information.
- ▶ Data integrity - data cannot be modified or corrupted during transfer, intentionally or otherwise, without being detected.
- ▶ Authentication - proves that users communicate and send data to the intended service. It protects against man-in-the-middle (MitM) attacks and builds user trust.

The OpenSSL library that provides an open-source implementation of the TLS protocol has been used to generate the private keys, certificate signing requests, SSL certificates (self-signed or Certificate Authority (CA)-signed) and for certificate format conversion.

Additionally, the connection to the servers that will be used for the deployment of the project's artifacts, including the testing, staging and production environments, has been also secured with HTTPS. Access to the Docker daemon socket is protected by enabling TLS, allowing Docker to be reachable through the network in a safe manner. From the server side, connections from clients authenticated by a certificate (self-signed or signed by a CA) are allowed to the servers in which the project's artifacts will be deployed. From the client side, clients can only connect to these servers with a certificate that can again be self-signed or signed by a CA.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	26 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

2.7.2 Hard disk encryption at rest

All hard disks mounted to the servers (virtual machines) that are used for the deployment of the Pledger CI/CD solution are encrypted. Disk encryption at rest ensures that files are always stored on disk in an encrypted form. The files only become available to the operating system and applications in readable form while the system is running and unlocked by a trusted user. An unauthorized person looking at the disk contents directly, will only find garbled random-looking data instead of the actual files, securing the actual data stored in cases that the hard disk or server is lost, stolen or discarded after its end-of-life.

2.7.3 User authentication of the CI/CD services

The services offered in the Pledger CI/CD solution, such as the source code repository (Gitlab), the continuous integration (Jenkins), and the artifacts repository (Artifactory) are secured using user authentication. The integration of an existing Lightweight Directory Access Protocol (LDAP) server or Active Directory implementation is possible.

2.7.4 Firewall protection of the VMs hosted in Hetzner Cloud

The infrastructure's security is guaranteed by firewall rules that control the authorized connections to the VMs used for the CI/CD solution. One of the most popular and widely used open-source firewall is iptables. A collection of iptables rules has been applied in the different firewall chains (input, output, forward, docker-user, etc.) of the VM to prevent several typical network attacks (SYN flood attacks, smurf attacks, prevent XMAS packets, land attacks, attacks by malfunctioning ICMP packets and other forms of Denial of Service (DoS attacks)). The default policy is to drop incoming, outgoing or forwarded packets from any source to any destination.

2.7.5 SSH key-based authentication

SSH access to administer and manage the CI/CD platform servers has been configured to use only key-based authentication. This is a more secure option than the most widely used password authentication techniques. Although passwords are securely sent to the server, it is not guaranteed that they are complex or lengthy enough to be guessed by persistent attackers. The combination of modern computing power and script automation makes brute forcing a password-protected account quite easy. Thus, SSH public key authentication is a more secure way in which we generate and store a pair of cryptographic keys and then configure the VMs to recognize and accept these generated keys.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	27 of 58				
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status:	Final

3 Integration patterns

This section highlights the main approaches, integration patterns, and methodologies adopted in order to facilitate the interactions and communications between software components of the Pledger Core System, resulting to its seamless integration. Additionally, it provides a detailed explanation of the individual components of the StreamHandler Platform which is used as a backbone communication middleware for the majority of the Pledger Core System software components. Finally, StreamHandler facilitates the integration of Pledger Core System with UC3 which is described in detail in D5.3 “Pilots operations and monitoring I”.

3.1.1 Microservices approach

The most significant challenge of microservices architecture is the software complexity of large applications. It is an alternative to the traditional monolithic approach used for application development which, under certain conditions, can remain a good choice. In the context of the Pledger project, we considered that the microservices approach represents a good fit, as the Pledger Core System consists of different software components, developed from different teams using different software stack and technologies.

The microservices approach achieves the above by adopting a strategy of putting together a large and complex platform from small individual building blocks. These distinct components of the Pledger Core System can be considered as separate software components which have their own code and resources. The entire functionality of the Pledger Core System is therefore realized and composed by the microservices available and deployed in the K8s cluster described in section 2.

Considering the microservice communication of the Pledger Core System, it has been considered from the beginning of the project that using a middleware as a backbone for microservice communication will provide a high-quality framework that creates decoupled, scalable and highly available system. For this purpose, the majority of the software components of the Pledger Core System use StreamHandler in order to be integrated. StreamHandler is a distributed event driven streaming platform based on Apache Kafka that offers the following powerful concepts:

- ▶ Publish/Subscribe to streams of events.
- ▶ Store these streams in a fault-tolerant way.
- ▶ Process these streams as they occur.
- ▶ Built-in capabilities for straightforward horizontal scalability.

Among other advantages and features, StreamHandler offers a distributed event driven streaming platform that can decouple various microservices (i.e., producers and consumers) in a reliable, scalable and fault-tolerant way. Finally, depending on the case, authenticated and encrypted communications that follow the RESTful paradigm were used.

In the following figure, an example of the Microservices Approach is presented using StreamHandler as streaming platform. Based on this approach, Pledger Core subsystems are implemented in ways that allow for them to be deployed independently from each other while always having in mind fully automated deployment approaches as they are described in the CI/CD related section (see section 2).

Document name:	D5.2 Pledger integrated demonstrator I			Page:	28 of 58
Reference:	D5.2	Dissemination:	PU	Version:	1.0
				Status:	Final

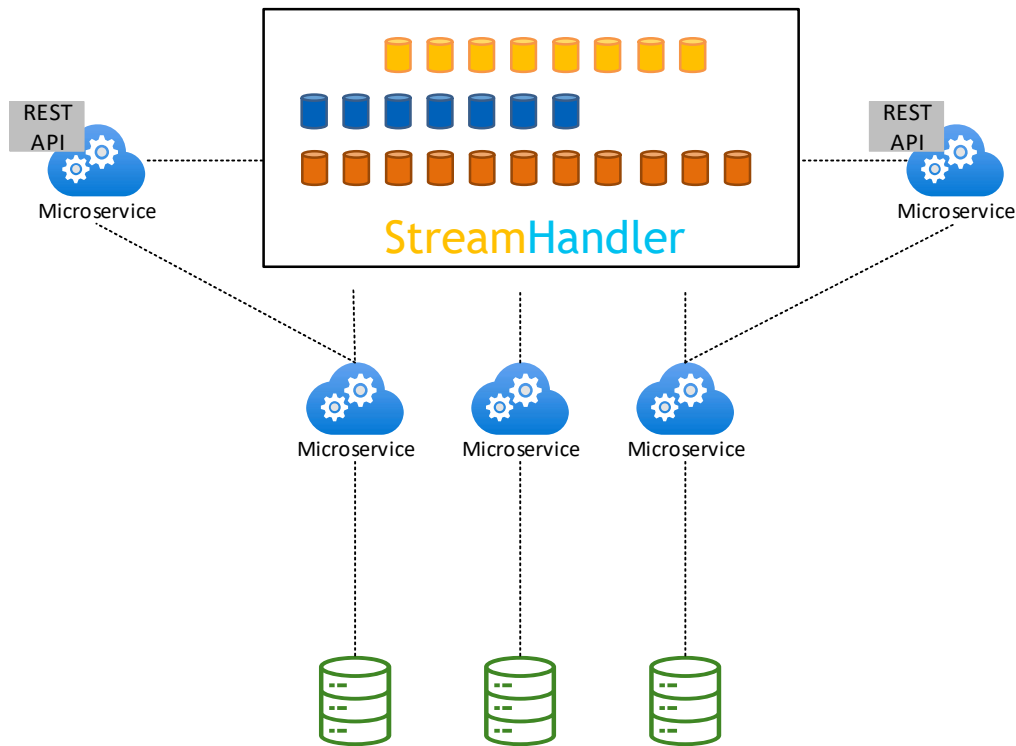


Figure 20: Example of Microservices Architecture using StreamHandler

3.1.2 StreamHandler platform

When it comes to microservices communication, the integration approach that is mostly used is REST communication via synchronous HTTP(S) requests. This is suitable to a wide range of applications and scenarios. The Request/Response pattern, on the other hand, creates point-to-point connections that tie both the sender and the receiver to sender, making it difficult to modify one component without affecting others.

As a result, in the context of Pledger, we used StreamHandler, as a backbone middleware for application communication. While other approaches also exist in order to successfully integrate different systems with the publish/subscribe mechanism like Message Queues ActiveMQ [10] (with the Enterprise integration patterns implemented in Camel) and RabbitMQ [11] with Exchanges and Routing algorithms respectively, these solutions can potentially accumulate business logic in the Message Queue and make solutions too complex to maintain and/or affect scalability.

With StreamHandler the business logic and solutions are implemented by incorporating distinct loosely coupled components (producers and consumers) which communicate with each other:

- ▶ Producers do not know who consumes the events. StreamHandler handles the scalability, fault-tolerance, and high availability of the data.
- ▶ Producers may produce a message while the consumers are down.
- ▶ New consumers may be added at any time with the ability of multiple consumers being able to read the incoming information from the same topic with different offsets per consumer.
- ▶ The consumers may process data at their own speed.

Considering the above benefits, StreamHandler plays an important role in the integration of the software components of the Pledger Core System. In the following sections, the detailed architecture and deployment specifications of the platform are provided, together with its cluster administration, monitoring and alerting dashboards and security features.

Document name:	D5.2 Pledger integrated demonstrator I	Page:	29 of 58	
Reference:	D5.2	Dissemination:	PU	
	Version:	1.0	Status:	Final

3.1.2.1 Architecture

The StreamHandler platform is used in Pledger to interconnect heterogeneous processing systems deployed at partner sites or in Pledger pilot sites for data exchange purposes. Figure 21 depicts the final architecture of the StreamHandler Platform which is comprised of the following components:

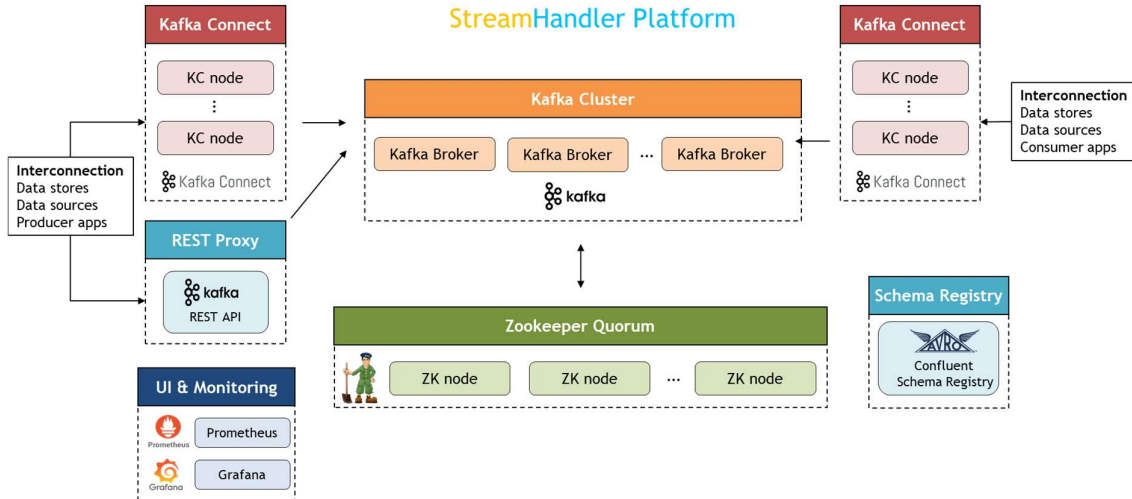


Figure 21: StreamHandler Platform Architecture

► Kafka Cluster

Multiple brokers typically collaborate to compose the Kafka cluster and achieve load balancing, redundancy, and fault-tolerance. For the purpose of the Pledger project, three brokers have been configured. The brokers themselves handle the actual storage of the incoming data and offer the functionality required for producers and consumers to do their tasks. In Kafka, data are stored in streams of messages or records that relate to a certain category known as a Topic. A Kafka record is made up of three parts: a key, the actual value, and a timestamp. While the data producer provides the actual value, the key may or may not be provided, and timestamp is inserted dynamically by Kafka.

For cluster administration and coordination, brokers use Apache Zookeeper. Each broker instance can handle hundreds of thousands read and write requests per second without affecting performance. Each broker has a unique ID and is responsible for partitions of one or more topic logs. Connecting to any broker, the client will implicitly gain access to the whole Kafka cluster. A minimum of three brokers should be used to ensure fault-tolerance: the bigger the number of brokers, the better the reliability.

► Zookeeper Quorum

Zookeeper Quorum of the deployed StreamHandler in Pledger consists of three Zookeeper nodes. Zookeeper is used by Kafka brokers to manage and coordinate the Kafka cluster. When the topology of the Kafka cluster changes, such as when brokers or topics are added or removed, Zookeeper notifies all nodes. For example, Zookeeper informs the cluster when a new broker enters the cluster or when a broker fails. Zookeeper also performs leadership elections among broker and topic partition pairs, assisting in determining which broker will be the leader for a certain partition as well as which brokers will be used as replicas. When Zookeeper alerts the cluster of broker changes, the brokers immediately start coordinating with one another and electing any new partition leaders that are needed. This safeguards against the unexpected unavailability of a broker.

► Schema Registry

The Schema Registry facilitates the creation and storage of data models that describe the data. It keeps a versioned history of all schemas, offers numerous compatibility options, and allows schemas to evolve based on the compatibility settings that have been configured. The Confluent Schema Registry, which is a Kafka add-on that offers a RESTful interface for storing and retrieving schemas, is used to implement the Schema Registry.

The Schema Registry's data models are expressed in Apache Avro [12], which may be seen as an enriched JSON format schema. Data written using Avro simply contain the values and an indicator of the schema that was used to produce the data. When reading the Avro data, the schema used when writing the values is used to understand the information. This permits data fields to be transmitted without overheads regarding the fields, etc., making serialization to be smaller in size and faster, resulting in data that is self-explanatory when paired with their schema. The Avro feature is used in Kafka to enable Schema Evolution. This is accomplished by using versioned Avro schemas for the messages that are exchanged, which allows the messages to be interpreted at the consuming end according to the schema that is pertinent to that specific piece of data. Furthermore, if a schema with extra fields is required to be backward compatible, setting a default value for the newly added field in messages that do not include it, ensures this compatibility.

► Kafka Connect

Kafka Connect is an open-source framework that is developed on top of the Producer and Consumer APIs in Apache Kafka. Its goal is to make data streaming between Apache Kafka and other data systems more performant, reusable, and reliable. When Apache Kafka is intended to be used as a centralized hub, Kafka Connect offers the tools required to optimize the individual connections between Kafka and systems providing data to or extracting data from it. The framework's simplifications and abstractions allow for the rapid definition and implementation of connectors capable of moving huge data collections into and out of Kafka. Among the many applications of Kafka Connect are the ingestion of databases or the automatic capturing of table updates, the reading of files from multiple locations, and so on into Kafka topics for further stream processing, while allowing fine tuning of the connectors to achieve desired performance characteristics such as low latency or high throughput, or even trading off delivery and replication guarantees to meet specific performance requirements.

Developers that use the Connect API are given all the required tools and infrastructure to create reusable, purpose-built connector plugins that can then be launched in a Kafka Connect cluster. The Connect cluster, which operates alongside the Kafka brokers, may be configured to distribute a connector's responsibilities across the cluster's available workers for parallel processing while also tracking the work accomplished and restoring the connectors in the event of failures with automatic offset management.

The Connect API is the most widely used by the industry and community since it was designed to standardize the integration of other data systems with Kafka. As a result, it simplifies connector development, deployment, and administration and enables the creation of reusable connectors that can be used to ingest data from or extract data from most of the common components and systems found in an organization. In the context of Pledger a Kafka Connect cluster has been configured with three nodes.

► Rest Proxy

A RESTful interface to a Kafka cluster is provided by the Kafka REST Proxy. It allows users to easily generate and consume messages, examine the cluster's status, and conduct administrative tasks without having to use the native Kafka protocol or clients. The interface allows you to list, create, and delete topics, retrieve information about topic partitioning and partition distribution among available brokers, produce or consume messages from a specific topic while also extracting details about

Document name:	D5.2 Pledger integrated demonstrator I				Page:	31 of 58	
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status:	Final

message offsets, and retrieve information about the number of Kafka brokers. In addition to the features indicated above, Producer and Consumer settings can be modified.

The value of the REST Proxy becomes clear when there is a need to interact with an application created using technologies that Kafka does not already support, or when an administrative interface is established to better monitor and analyze the state of the cluster. Furthermore, it enables and facilitates open-source tools that rely on the Rest Proxy to connect with the cluster, enhancing the functionality available to users. The Rest Proxy also supports the scripting of certain administrative activities, while allowing a stream processing framework that does not natively support Kafka to use it as a data source.

► **UI & Monitoring**

The UI & Monitoring node has a collection of tools for cluster configuration as well as an overview of cluster performance and health status. These are mostly open-source components that have been selected from the community and can be accessed when a user connects to a dedicated node with UI capabilities. The connection to that node's remote desktop is limited to a specified list of approved IP addresses, it is encrypted using HTTPS, and is password secured. Since noVNC is utilized, the user who wants to access the remote desktop does not need to install any special client.

The Prometheus Monitoring platform and the Grafana visualization application are the specific technologies used for cluster performance and health status. The platform components may export Java Management Extensions (JMX) metrics, which are then displayed in customizable Grafana dashboards. The information offered includes the Zookeeper quorum status, the state of the Kafka brokers, as well as informative graphs about incoming and outgoing data throughputs, message rates per broker and per topic, and other information on CPU and memory usage per broker. Likewise, healthy Schema Registry and Rest Proxy instances, as well as their associated connections, are monitored.

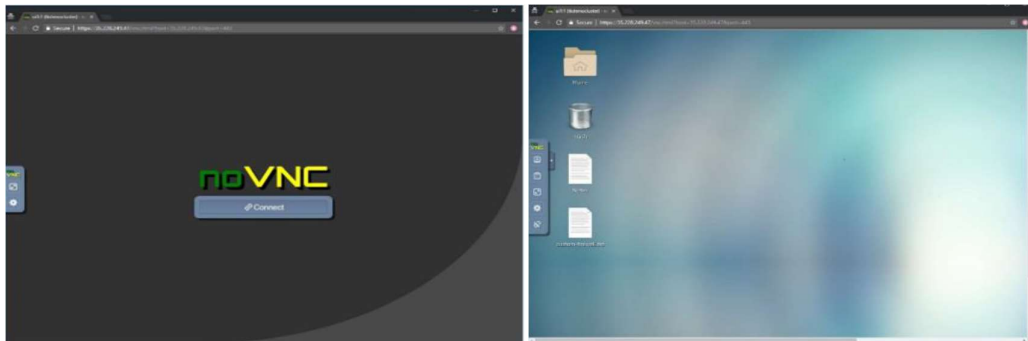


Figure 22: noVNC remote desktop

Aside from the Prometheus/Grafana setup, a Kafka GUI has been setup called AKHQ (previously known as KafkaHQ) for cluster administration purposes. More specifically, it provides users with the ability to search and explore data in a unified console across multiple clusters, manage topics, topic data, consumer groups, schema registry, connectors, and more. AKHQ offers tons of useful features, including multi-cluster management, message browsing, live tailing, authentication, authorization, read-only mode, schema registry, and Kafka Connect management. It supports Avro and is compatible with LDAP and RBAC (Role Based Access Control).

Document name:	D5.2 Pledger integrated demonstrator I	Page:	32 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

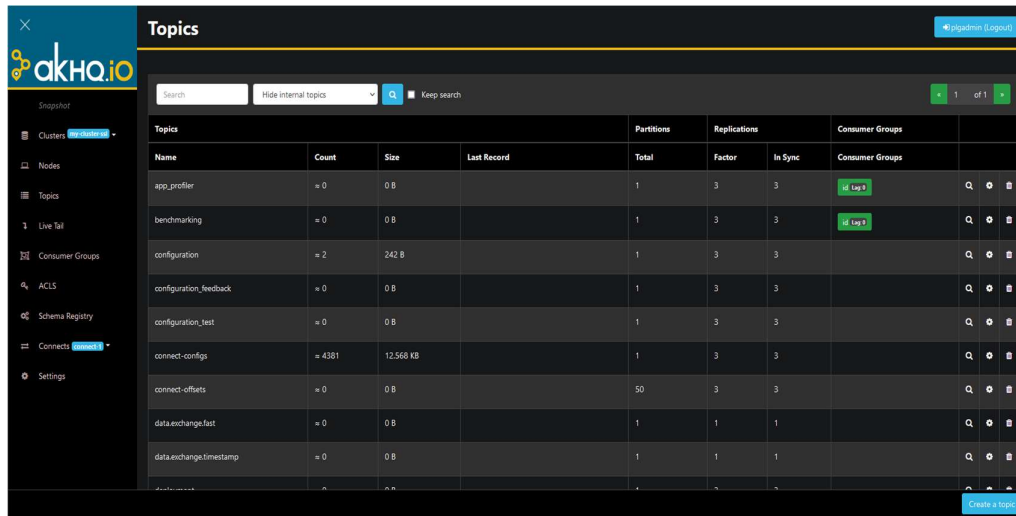


Figure 23: AKHQ UI with Topics details

3.1.2.2 Security

Apache Kafka enables back-end systems to share real-time data feeds with each other through Kafka topics. In general, any user or application can write messages pertaining to any topic, as well as read data from any topic with a standard Kafka setup. However, it is crucial to implement Kafka security when moving towards a shared tenancy model while multiple teams and applications use the same Kafka Cluster, or also when Kafka Cluster starts onboarding some critical and confidential information.

Kafka Security has three main components:

► Encryption of data In-Flight using SSL/TLS

Given that packets, while being routed to Kafka cluster, travel to networks and hop from machine to machine, there is the need to deal with man in the middle (MitM) attacks. Any of these routers could read the content of the data if it is PLAINTEXT. Therefore, data is encrypted and securely transmitted over the network with enabled encryption and careful setup of SSL certificates. Only the first and the final machine possess the ability to decrypt the packet being sent with SSL/TLS. However, this encryption comes at a cost. That means that in order to encrypt and decrypt packets, extra CPU cycles are needed for both the Kafka Clients and the Kafka Brokers.

Note: The encryption is only in-flight, and the data still sits un-encrypted on broker's disk.

► Kafka Authentication

This allows producers and consumers to authenticate to Kafka cluster, which verifies their identity. We can distinguish SSL and SASL authentication methodologies.

SSL Authentication

SSL Authentication is basically leveraging a capability from SSL called two ways authentication. Basically, a certificate is issued to the clients signed by a trusted certificate authority (CA), that allows Kafka brokers to verify the identity of the clients.

SASL Authentication

SASL refers to Simple Authorization Service Layer. The basic concept here is that the authentication mechanism and Kafka protocol are separate from each other. It is very popular with Big Data systems as well as the Hadoop setup.

Kafka supports the following shapes and forms of SASL:

- SASL PLAINTEXT
- SASL SCRAM
- SASL GSSAPI (Kerberos)

► **Kafka Authorization (ACL)**

Kafka needs to be able to decide what operations Kafka clients can perform or not as soon as they are authenticated. This is where Authorization comes in, which is determined by the Access Control Lists (ACL). ACLs have proven to be quite beneficial in preventing major incidents. Let's put this into perspective: suppose we have a topic that needs to be writeable from only a subset of clients or hosts, thus preventing any data corruption or deserialization errors. ACLs are also great if we have sensitive data and need to demonstrate to regulators that only particular application or users have access to it.

Document name:	D5.2 Pledger integrated demonstrator I				Page:	34 of 58	
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status:	Final

4 Pledger core system integration

Component integration is a process that involves following several multi-disciplinary approaches. The Pledger Core System consists of software components written in different programming languages, from different software teams. The use of micro-services, REST APIs, and publish/subscribe mechanisms address that kind of integration and provide end-to-end interoperability between the components.

In this section we firstly present the integration plan that has been established to drive the subsequent integration and deployment activities with the aim to release two major Pledger Core System releases, with the first one on M24 and the last one on M33. Additionally, the integration methodology is also provided together with the integration matrix of the software components of the Pledger Core System, offering a framework for identifying the integration endpoints of the platform. Finally, this section aims to identify the objectives and the priorities of the integration process, providing a timetable for their implementation, which will produce the two planned releases of the system.

4.1 Pledger integration plan

The software components of Pledger Integrated Demonstrators are implemented with different technologies and involve different technical requirements and configurations. The development of the integrated Pledger Core System and its Use Cases follows an iterative process that provides two major releases according to the integration plan which is elaborated further in this section. The integration process follows the CI/CD best practises aiming at reducing the errors during integration and deployment of the software components, while increasing project velocity. The development team of each component will use the Pledger CI/CD platform described in Section 2 to proceed with the development and integration tasks.

The integration plan includes two major releases.

- ▶ The first release on M24
- ▶ The final release on M33

Table 1: Pledger Integration Plan

Iteration	Integration activities	Components	Partners	Date
Init	Installation of CI/CD Platform	-	INTRA	M14
1st Pledger Release	Main development phase of the software components of the Pledger Core System with their unit tests and their bilateral integration tests prepared for the 1 st release	-	All technical partners involved in WP3, WP4 and WP5	M6-M24
	Installation of StreamHandler Platform	StreamHandler	INTRA	M15
	Integration of CI/CD with the K8s Cluster of the Pledger Core System	-	INTRA, ENG	M16

Iteration	Integration activities	Components	Partners	Date
	Integration of StreamHandler with K8s Cluster of Pledger Core System	Recommender, StreamHandler	INTRA, ENG	M16
	First version of the development of the consumers/producers and REST endpoints needed for integrating the Pledger Core System	All software components of PLEDGER Core System	ATOS, ENG, ICCS, INNOV, i2CAT, INTRA	M16-M24
Pledger Integrated Demonstrator I <i>Fist Version</i>	First Release of the Pledger Core System Demonstrator partially integrated with the UC1, UC2 and UC3. Integration with UCs is still in progress and will be released on M26. The majority of the Pledger Core System components is fully integrated.			M24
<i>1st Integration with UC1</i>	First integrated release of the Pledger Core System with UC1	PLEDGER Core System components <ul style="list-style-type: none"> i. SaaS/IaaS Monitoring Engine ii. Recommender (one-way, metrics consumption without additional action) iii. StreamHandler iv. Orchestrator (partially: orchestrator is integrated with local hypervisor of UC1 for the moment) UC1 components <ul style="list-style-type: none"> i. VM Configuration ii. Client Unity App iii. Server Unity App 	ATOS, HOLO, ENG, INTRA	M16-M26

Iteration	Integration activities	Components	Partners	Date
		iv. GPU		
<i>1st Integration with UC2</i>	First integrated release of the Pledger Core System with UC2	PLEDGER Core System components <ul style="list-style-type: none"> i. SaaS/IaaS Monitoring Engine ii. Recommender iii. Orchestrator (integration without SOE) UC2 components <ul style="list-style-type: none"> i. Barcelona Infrastructure Compute 	ATOS, ENG, i2CAT	M16-M26
<i>1st Integration with UC3</i>	First integrated release of the Pledger Core System with UC3	PLEDGER Core System components <ul style="list-style-type: none"> i. SaaS/IaaS Monitoring Engine ii. Orchestrator iii. Recommender iv. StreamHandler UC3 components <ul style="list-style-type: none"> i. Basic Analytics ii. Message Broker iii. Edge Server 	ATOS, FILL, INTRA	M16-M26
2nd Pledger Release	All the Pledger Core System software components are fully integrated, and the Pledger Core System is fully integrated with UC1, UC2 and UC3		All technical partners involved in WP3, WP4 and WP5	M24-M33
Pledger Integrated Demonstrator II <i>Final Version</i>	Final Pledger integrated release			M33

4.1.1 Pledger integrated demonstrator I

The first release of Pledger integrated demonstrator runs from M6 to M24 and comprises the following steps:

- ▶ Setup the infrastructure.
- ▶ Specify, implement, and test the majority of the integration endpoints of the Pledger Core System.
- ▶ First integration of the Pledger Core System with UC1, UC2, and UC3.

To begin with, the CI/CD infrastructure was set up together with the core infrastructure of the Pledger Core System such as:

- ▶ The StreamHandler Platform used as integration pattern to connect the different software components of the Pledger Core System.
- ▶ The K8s cluster that hosts the Pledger Core System.

Then the development, building, and testing of the software components of the Pledger Core Platform started. This includes the main development phase together with the development of the consumers, producers and REST endpoints needed for integrating the Pledger Core System. In addition, a first integrated prototype of the Pledger Core Platform with the different UCs will be delivered on M26 and is described in detail in D5.3 “Pilots operations and monitoring I”. Finally, the whole process will follow the CI/CD principles by using the services of the CI/CD infrastructure and will lead to the first Pledger integrated demonstrator I on M24.

4.1.2 Pledger integrated demonstrator II

The second release of Pledger integrated demonstrator will run from M24 to M33 and will include the following steps:

- ▶ Full integration the Pledger Core Platform software components.
- ▶ Full integration of the Pledger Core Platform with UC1, UC2 and UC3.

The outcome of this release will end up to the final version of Pledger Integrated Demonstrator II.

4.2 Integration methodology

For the identification and the documentation of the Pledger Core System integration, we created an integration matrix based on the format typically used in a Design Structure Matrix (DSM) [13]. DSM is a technique commonly used in system integration to represent the structure and interactions of complex systems. A design structure matrix is a square matrix that illustrates the connections between system elements. The system elements are labelled in rows to the left of the matrix and/or columns above the matrix. In our case, these elements represent software components. The DSM is the equivalent of an adjacency matrix in graph theory and is used in systems engineering and project management to model the structure of complex systems or processes, to perform system analysis, project planning and organization design.

4.2.1 Integration matrix

The integration matrix consists of all the software components of the Pledger Core System (described in D2.3 “Pledger Overall Architecture” [1]). Each row and column represent a software component of the Pledger Core System.

Table 2 provides the integration matrix of the Pledger Core System. It is an upper triangular square matrix where each element is an integration endpoint. The naming convention chosen for each integration endpoint is the following:

X.Y where X represents the row and Y the column of the integrated software components.

Using an integration matrix helped us to have a clear interface identification among the different software components of the Pledger Core System and reduced the risk of incompatibilities amongst the components.

Document name:	D5.2 Pledger integrated demonstrator I			Page:	38 of 58
Reference:	D5.2	Dissemination:	PU	Version:	1.0
				Status:	Final

Integration points	1. Orchestrator	2. Recommender	3. SaaS & IaaS Monitoring Engine	4. Configuration DB	5. App Profiler	6. UI Configuration Dashboard	7. Benchmarking Scheduler	8. Metrics DB (Historical)	9. Benchmarking Creator	10. SLA Manager	11. SLA Notifier	12. SLA Negotiator	13. SLA Monitoring	14. SLA-SC bridge	15. IDPS	16. Anomalies Detection Reasoner	17. Rules/Cases/Schemas	18. T&R Engine	19. Slicing & Orchestrator Engine	20. RAN Controller
1. Orchestrator		1.2	1.3	1.4															1.19	
2. Recommender			2.3	2.4				2.8			2.11									
3. SaaS & IaaS Monitoring Engine													3.13							
4. Configuration DB					4.5	4.6	4.7			4.10										
5. App Profiler									5.9											
6. UI Configuration Dashboard																				
7. Benchmarking Scheduler																				
8. Metrics DB (Historical)																				
9. Benchmarking Creator																				
10. SLA Manager																				
11. SLA Notifier														11.14				11.18		
12. SLA Negotiator																				
13. SLA Monitoring																				
14. SLA-SC bridge																				
15. IDPS																				
16. Anomalies Detection Reasoner																				
17. Rules Cases/Schemas																				
18. T&R Engine																				
19. Slicing & Orchestrator Engine																				19.20
20. RAN Controller																				

Table 2: Pledger Core System Integration Matrix

Document name:	D5.2 Pledger integrated demonstrator I				Page:	39 of 58	
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status:	Final

Please note that in Table 2 some lines are empty. This does not mean that we do not have an integration endpoint of each software component of the Pledger Core System. This is because the integration matrix is an upper triangular matrix and we decided to describe each integration endpoint and the bilateral interaction of the software components of the Pledger Core System with a single ID (X.Y) and not in the form of request-response (X.Y and Y.X).

However, there are components that are not integrated directly with other components such as:

- ▶ IDPS
- ▶ Anomalies detection reasoner
- ▶ Rules/cases/schemas

This is because these components are standalone components that exist in order to increase the security and the detection of anomalies of the system and do not interact directly with the other components for the moment.

4.2.2 Integration endpoints

After identifying the integration endpoints in Table 2, for each one of them we created a table to document relative information. Table 3 fields are filled in with the following information:

- ▶ **Identifier:** This is the unique identifier of the integration endpoint as presented in the Pledger Core System Integration matrix.
- ▶ **Components:** The name of the integrated components.
- ▶ **Responsible:** The responsible partner for the implementation and documentation of the relevant integration endpoint.
- ▶ **Data Type & Protocol:** The data type and the protocol used for the information exchange between the two components.
- ▶ **Publisher – Subscriber (if any):** This field is valid when the integration between two software components of the Pledger Core System is achieved through StreamHandler.
- ▶ **Status:** The current implementation status of the endpoint.

Table 3: Pledger Core System Integration Endpoints

Integration Endpoint	Components	Responsible	Data Type & Protocol	Publisher – Subscriber (if any)	Status
1.2	Orchestrator - Recommender	ATOS, ENG	Data Type: JSON Protocol: Kafka via StreamHandler	Topic(s): - deployment - deployment-feedback	Testing
1.3	Orchestrator – SaaS/IaaS Monitoring Engine	ATOS	Data Type: Text/plain Protocol: HTTPS	N/A	In Progress
1.4	Orchestrator – Configuration DB	ATOS, ENG	Data Type: JSON Protocol: Kafka via StreamHandler	Topic(s): - configuration	Done
1.19	Orchestrator – PLEDGER SOE Framework	ATOS, i2CAT	Data Type: JSON Protocol: REST API HTTPs	N/A	In progress
2.3	Recommender – SaaS/IaaS	ATOS, ENG	Data Type: JSON	Topic(s): - monitoring	In progress

Integration Endpoint	Components	Responsible	Data Type & Protocol	Publisher – Subscriber (if any)	Status
	Monitoring Engine		Protocol: HTTPs (either REST calls to Prometheus or Kafka via StreamHandler)		
2.4	Recommender - Configuration DB	ENG	Data Type: N/A Protocol: JDBC (SQL)	N/A	In progress
2.8	Recommender – Metrics DB	ENG	Data Type: JSON Protocol: Kafka via StreamHandler	Topic(s): - benchmarking	In progress
2.11	Recommender – SLA notifier	ATOS, ENG	Data Type: JSON Protocol: Kafka via StreamHandler	Topic(s): - sla_violation	Done
3.13	SaaS/IaaS Monitoring Engine – SLA Monitoring	ATOS	Data Type: Text/plain Protocol: HTTPS	N/A	Done
4.5	Configuration DB – App profiler	ENG, ICCS	Data Type: JSON Protocol: - HTTPs (Kafka via StreamHandler) - REST API	Topic(s): - app_profiler	In progress
4.6	Configuration DB – UI Configuration Dashboard	ENG	Data Type: N/A Protocol: JDBC (SQL)	N/A	Done
4.7	Configuration DB - Benchmarking Scheduler	ENG	Data Type: JSON Protocol: Kafka via StreamHandler	Topic(s): - configuration	Done
4.10	Configuration DB - SLA Manager	ATOS, ENG	Data Type: JSON Protocol: Kafka via StreamHandler	Topic(s): - configuration	Done
5.9	App Profiler - Benchmarking Creator	ICCS, ENG	Data Type: JSON Protocol: REST API HTTPs	N/A	In progress
11.14	SLA Notifier - SLA SC bridge	ATOS, INNOV	Data Type: JSON Protocol: Kafka via StreamHandler	Topic(s): - sla_contract - sla_violation	In progress
11.18	SLA Notifier - T&R Engine	ATOS, ICCS	Data Type: JSON Protocol: Kafka via StreamHandler	Topic(s): - sla_violation	Not started

Integration Endpoint	Components	Responsible	Data Type & Protocol	Publisher – Subscriber (if any)	Status
19.20	PLEDGER SOE Framework - PLEDGER RAN Controller	i2CAT	Data Type: JSON Protocol: HTTPs (REST API)	N/A	In progress

In the following tables, a detailed description of each integration endpoint is presented associated with its identifier, responsible, integration point purpose, and sequence diagram.

Table 4: Orchestrator - Recommender Integration endpoint description

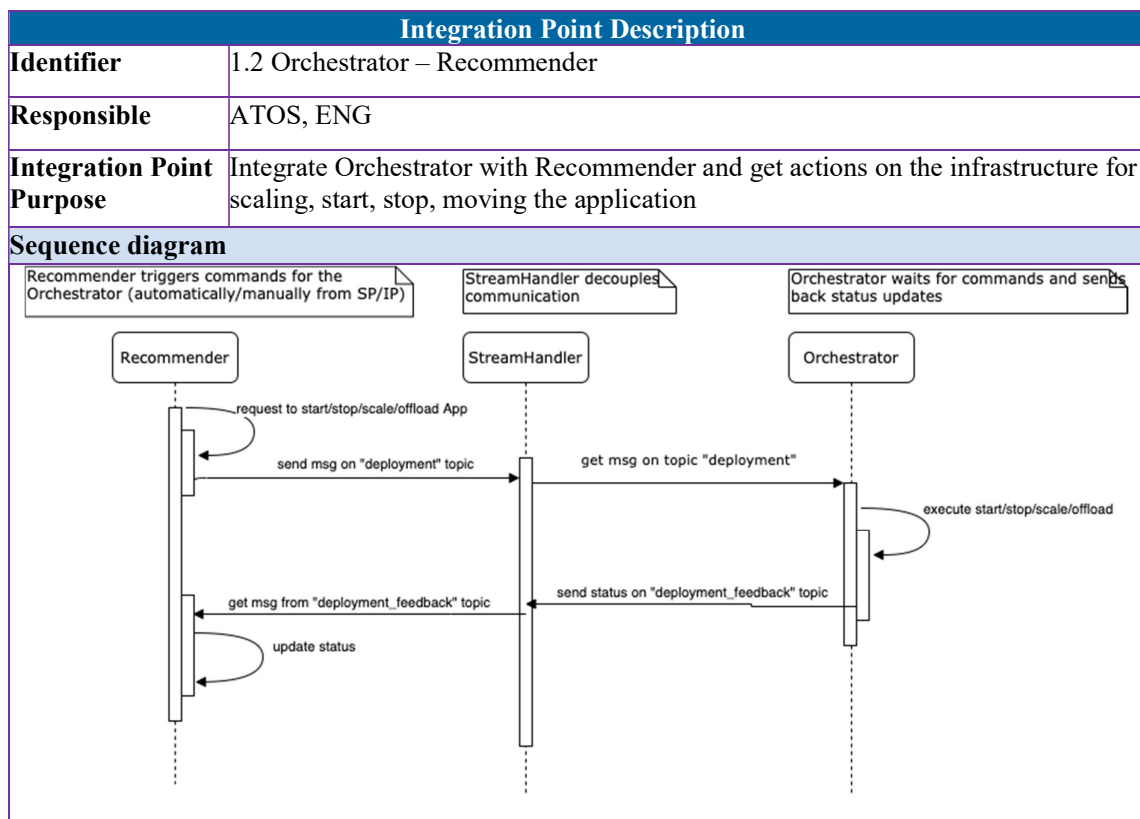
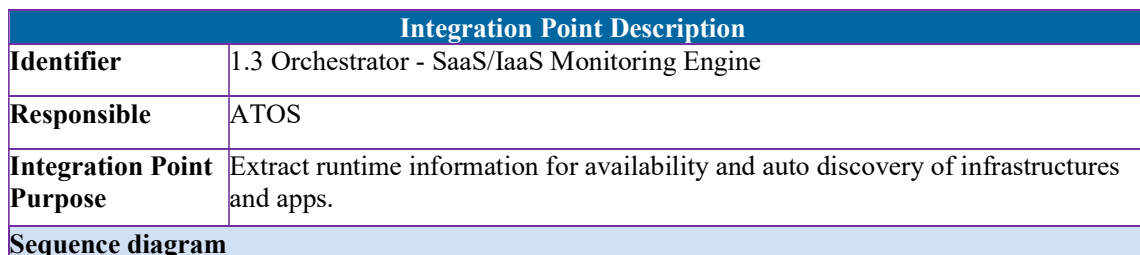


Table 5: Orchestrator – SaaS/IaaS Monitoring Engine endpoint description



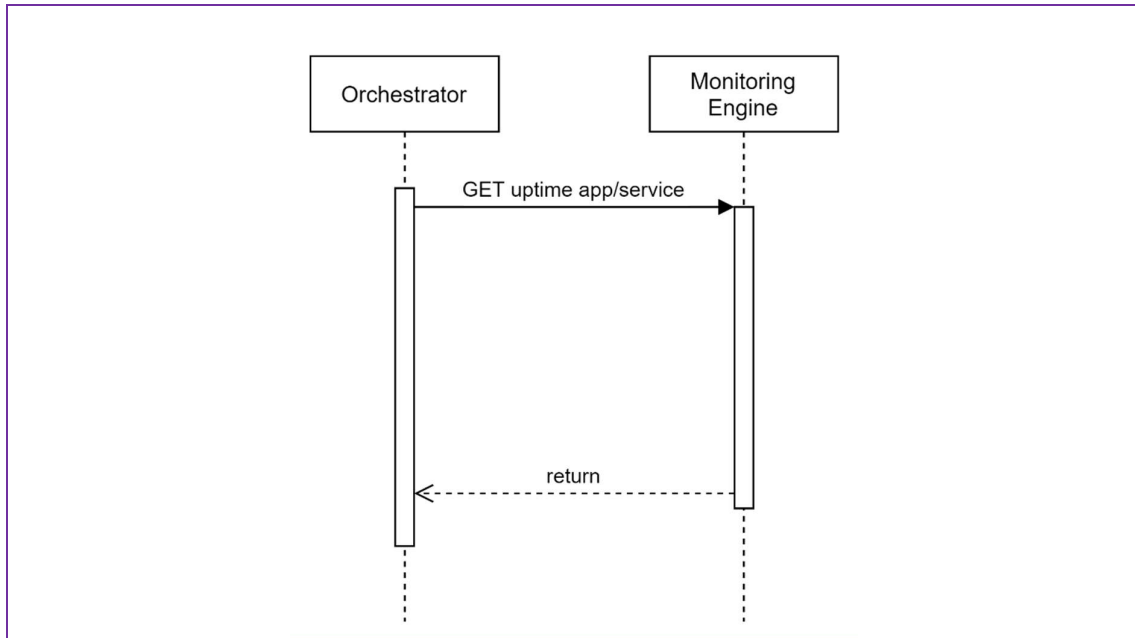


Table 6: Orchestrator – Configuration DB endpoint description

Integration Point Description	
Identifier	1.4 Orchestrator - Configuration DB
Responsible	ATOS, ENG
Integration Point Purpose	Get the updated configuration data for infrastructure and apps/services
Sequence diagram	
<pre> sequenceDiagram actor Provider as Service or Infrastructure Provider participant ConfService participant StreamHandler participant Orchestrator Note over ConfService: ConfService saves configuration and advertises updates Provider->>ConfService: update entity ConfService->>StreamHandler: advertise updated entity ID on topic "configuration" Note over StreamHandler: StreamHandler decouples communication StreamHandler->>Orchestrator: get msg on topic "configuration" Note over Orchestrator: Orchestrator waits for updates and gets them through REST calls Orchestrator->>StreamHandler: REST call StreamHandler-->>Orchestrator: return entity </pre>	

Table 7: Orchestrator – PLEDGER SOE Framework endpoint description

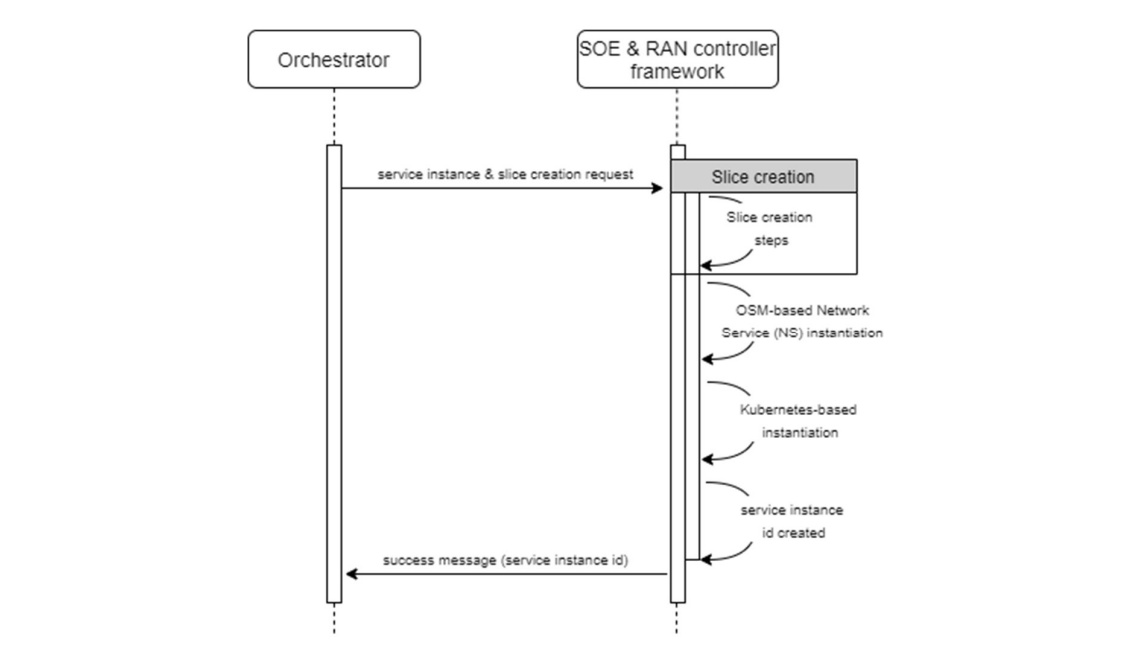
Integration Point Description	
Identifier	1.19 Orchestrator – PLEDGER SOE Framework
Responsible	ATOS, i2CAT
Integration Point Purpose	Request the instantiation of an application from the catalogue to be deployed on top of a previously created infrastructure slice.
Sequence diagram	
 <pre> sequenceDiagram participant Orchestrator participant SOE as SOE & RAN controller framework Orchestrator->>SOE: service instance & slice creation request activate SOE SOE->>SOE: Slice creation SOE->>SOE: Slice creation steps SOE->>SOE: OSM-based Network Service (NS) instantiation SOE->>SOE: Kubernetes-based instantiation SOE->>SOE: service instance id created SOE-->>Orchestrator: success message (service instance id) deactivate SOE </pre>	

Table 8: Recommender – SaaS/IaaS Monitoring Engine endpoint description

Integration Point Description	
Identifier	2.3 Recommender - SaaS/IaaS Monitoring Engine
Responsible	ATOS, ENG
Integration Point Purpose	Get the infra/app monitoring data. It supports either data extraction from Prometheus or msg consumption through Kafka depending on the availability from the different Use Cases.
Sequence diagram	

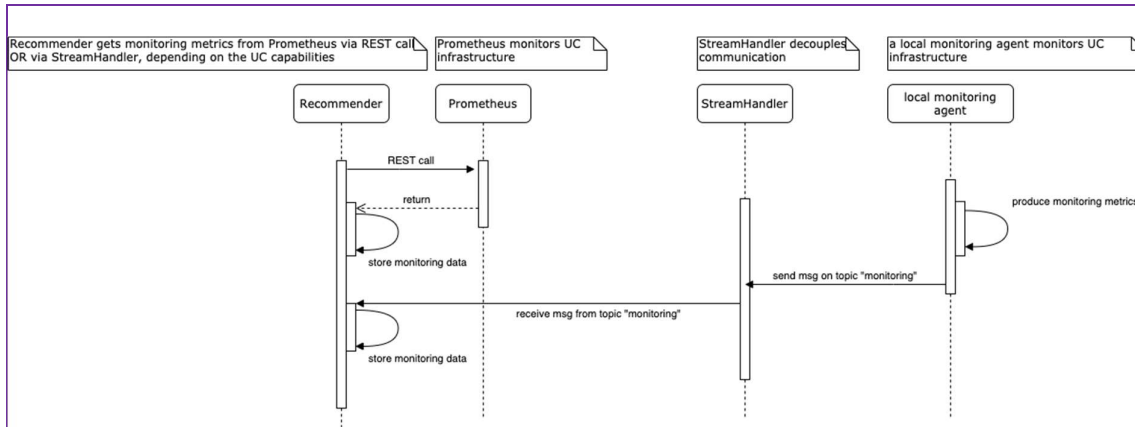


Table 9: Recommender – Configuration DB endpoint description

Integration Point Description	
Identifier	2.4 Recommender – Configuration DB
Responsible	ENG
Integration Point Purpose	Extract configuration data from the Configuration DB
Sequence diagram	
<pre> sequenceDiagram participant Recommender participant ConfService Recommender->>ConfService: JDBC call ConfService-->>Recommender: return </pre> <p>The Recommender communicates with ConfService to get configuration updates via JDBC calls</p>	

Table 10: Recommender – Metrics DB endpoint description

Integration Point Description	
Identifier	2.8 Recommender – Metrics DB
Responsible	ENG
Integration Point Purpose	Get the benchmarks reports with metrics
Sequence diagram	

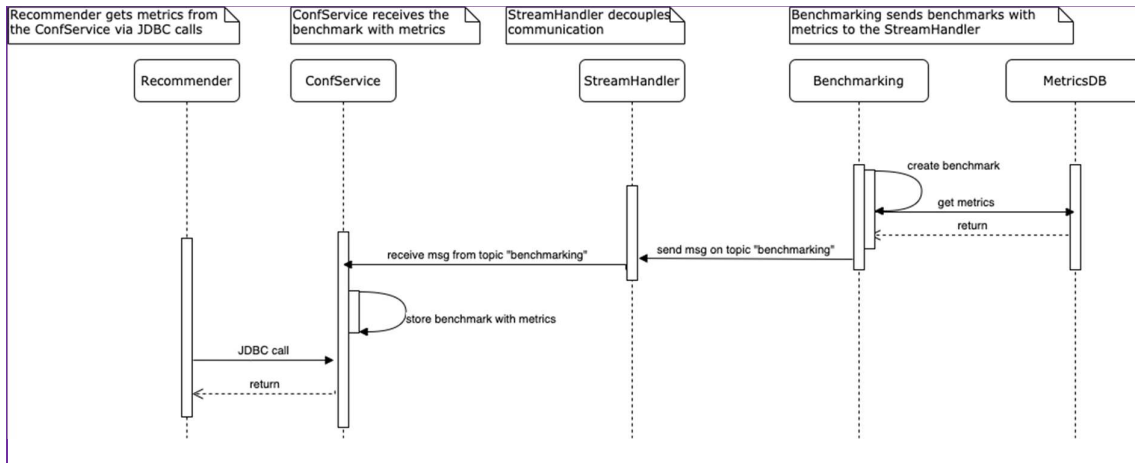


Table 11: Recommender – SLA notifier endpoint description

Integration Point Description	
Identifier	2.11 Recommender - SLA notifier
Responsible	ATOS, ENG
Integration Point Purpose	Get the SLA violation notifications
Sequence diagram	

Table 12: SaaS/IaaS Monitoring Engine – SLA Monitoring endpoint description

Integration Point Description	
Identifier	3.13 SaaS/IaaS Monitoring Engine - SLA Monitoring
Responsible	ATOS
Integration Point Purpose	Retrieve metrics samples periodically (polling) for the evaluation of SLA guarantees
Sequence diagram	

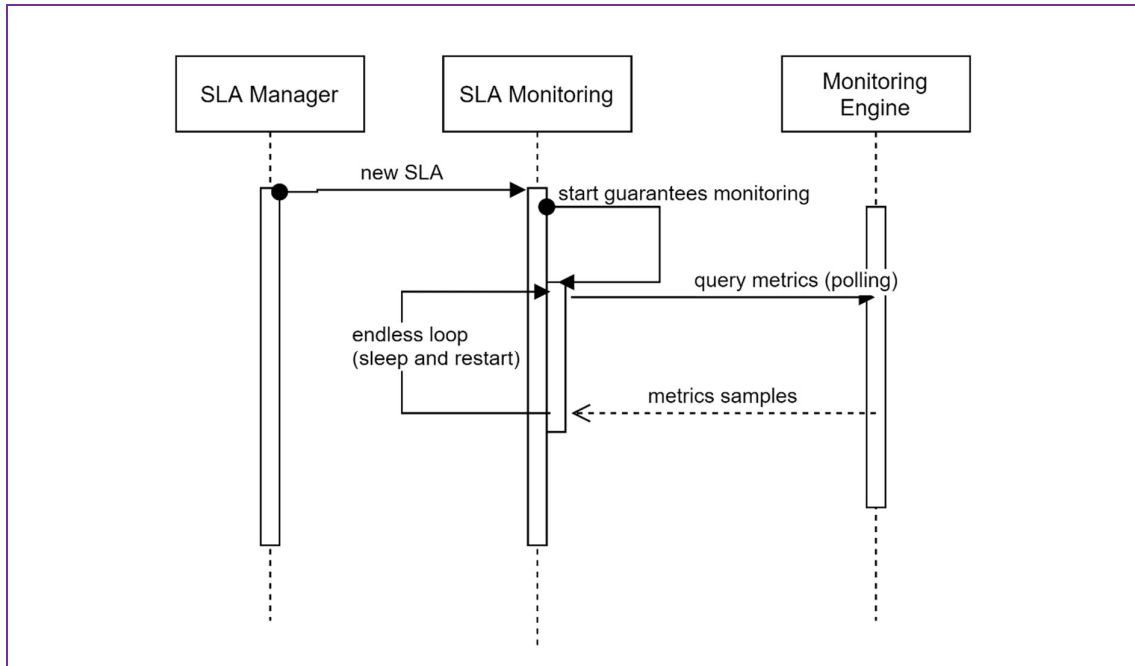


Table 13: Configuration DB – App Profiler endpoint description

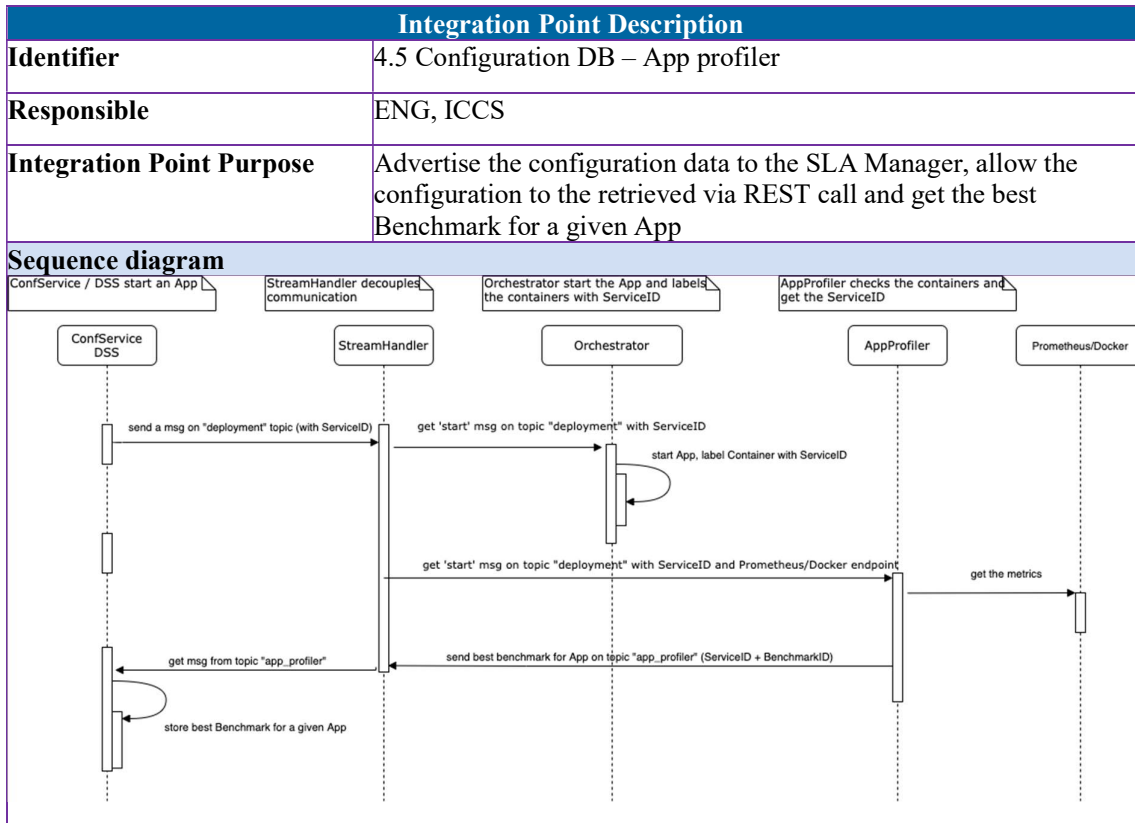


Table 14: Configuration DB – UI Configuration Dashboard endpoint description

Document name:	D5.2 Pledger integrated demonstrator I	Page:	47 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

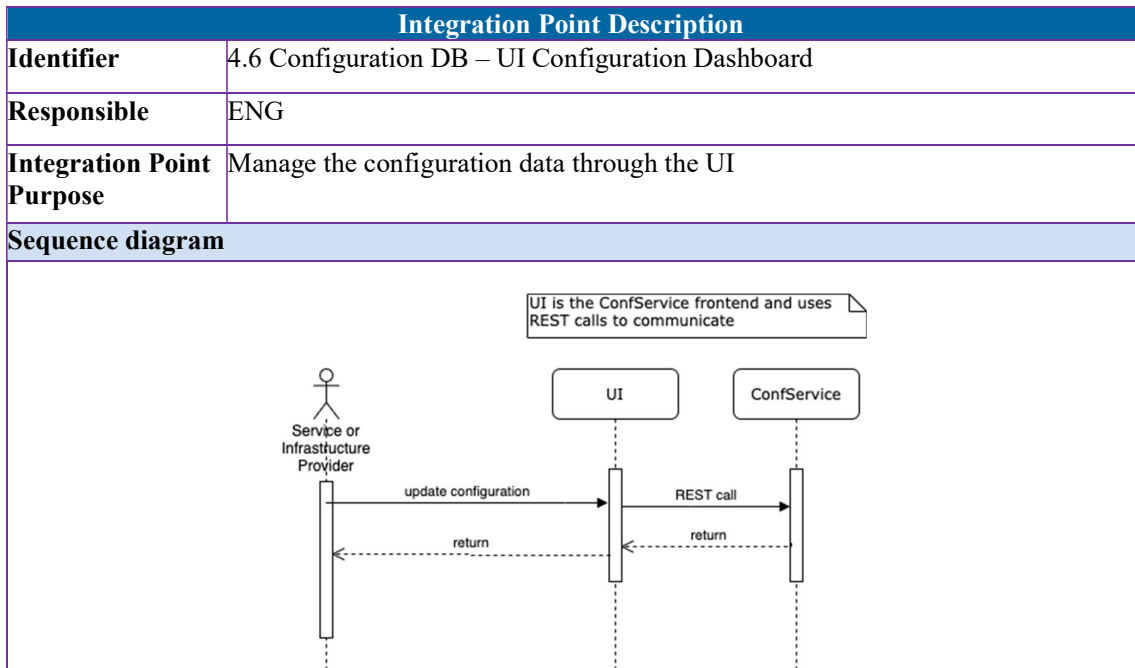


Table 15: Configuration DB – Benchmarking Scheduler endpoint description

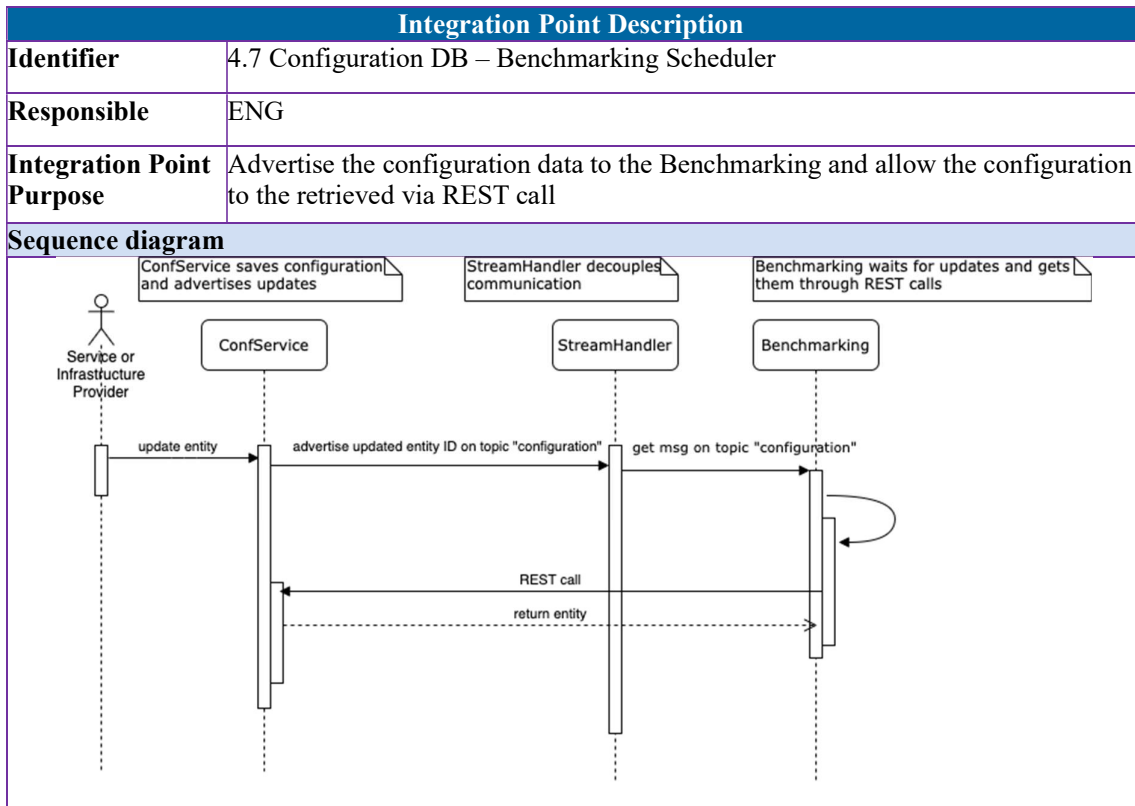


Table 16: Configuration DB – SLA Manager endpoint description

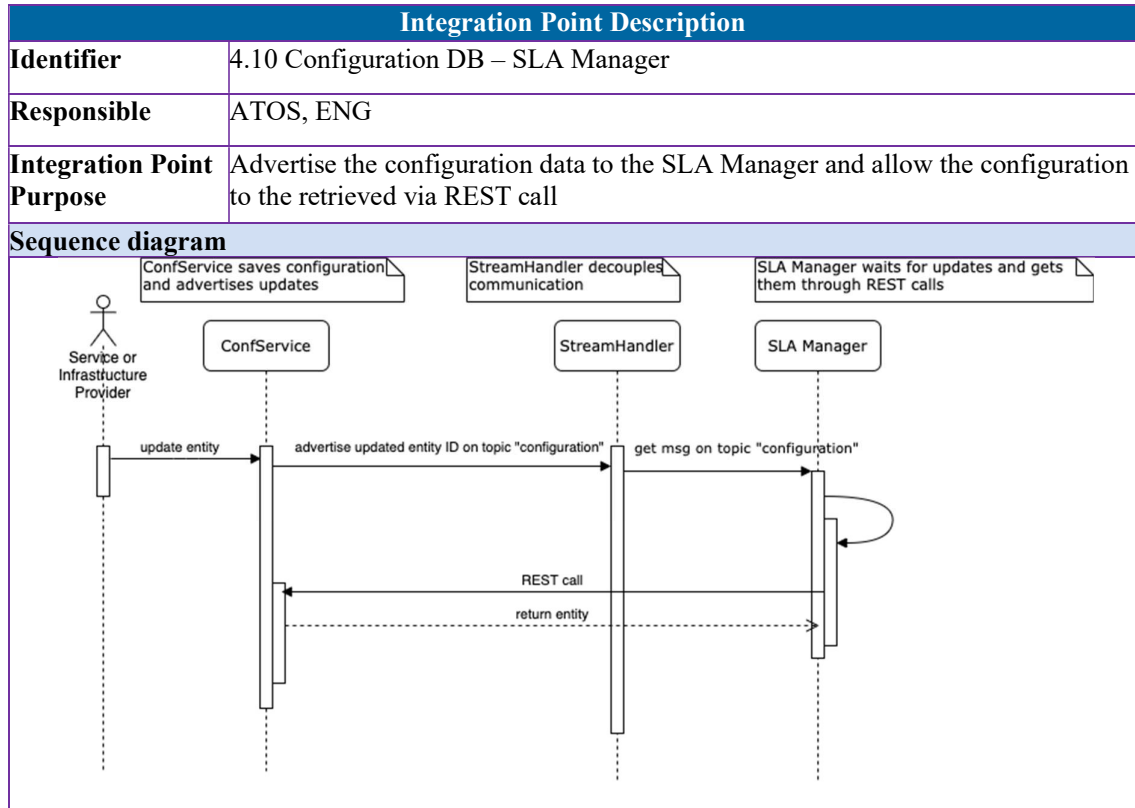
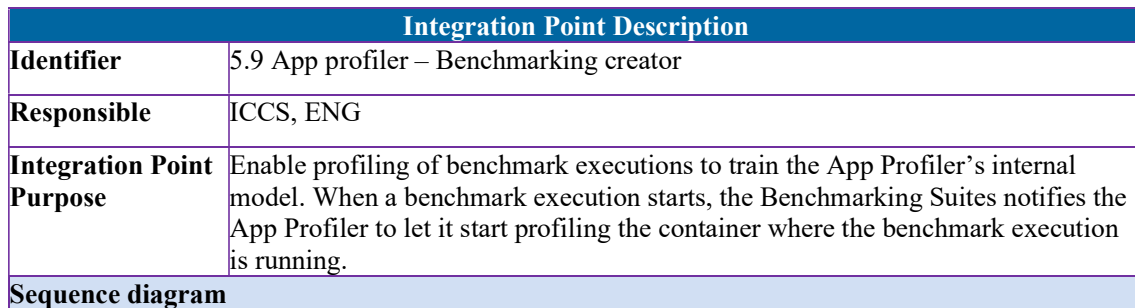


Table 17: App Profiler – Benchmarking creator endpoint description



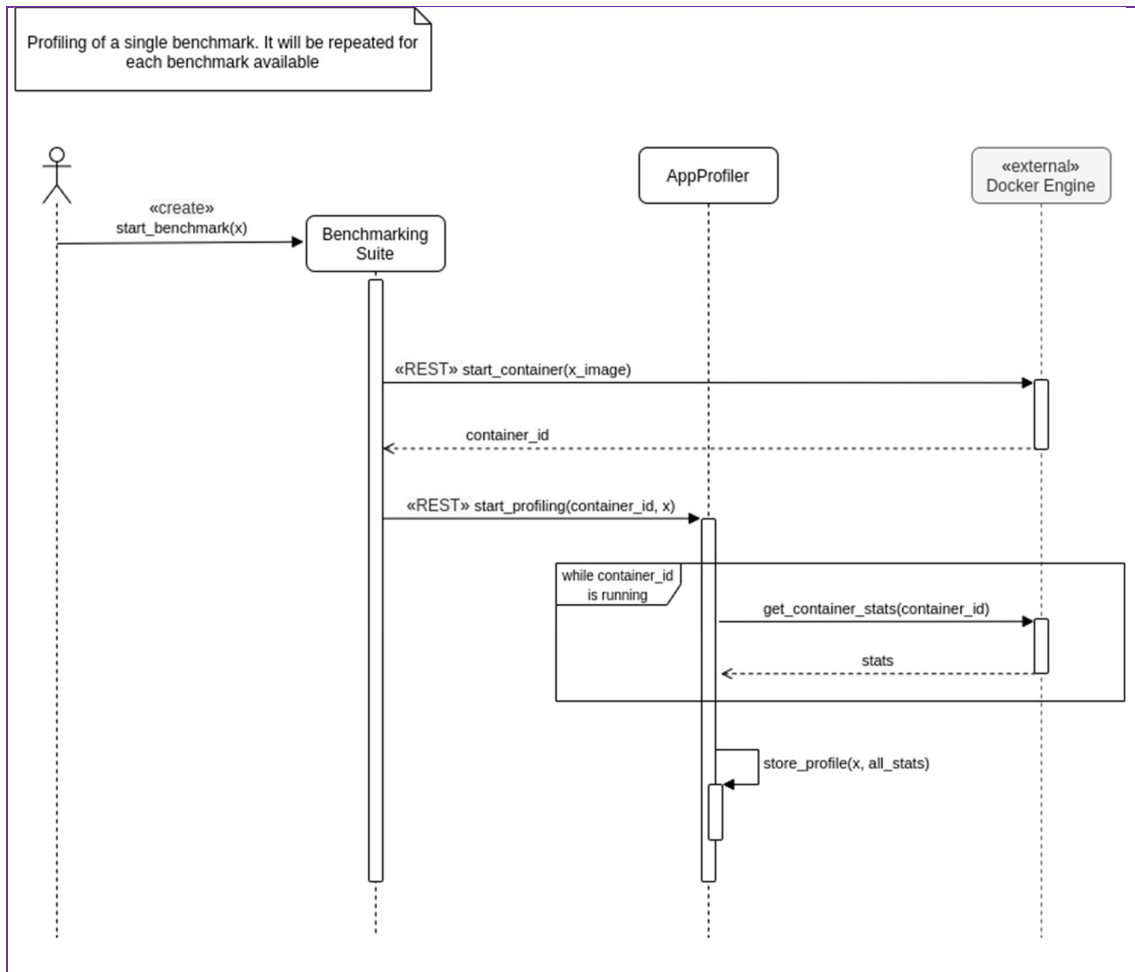


Table 18: SLA Notifier – SLA SC bridge endpoint description

Integration Point Description	
Identifier	11.14 SLA Notifier – SLA SC bridge
Responsible	ATOS, INNOV
Integration Point Purpose	Get the SLA related notifications for the blockchain SLA to Smart Contract Bridge and the SLA Violation workflows on the DLT.
Sequence diagram	

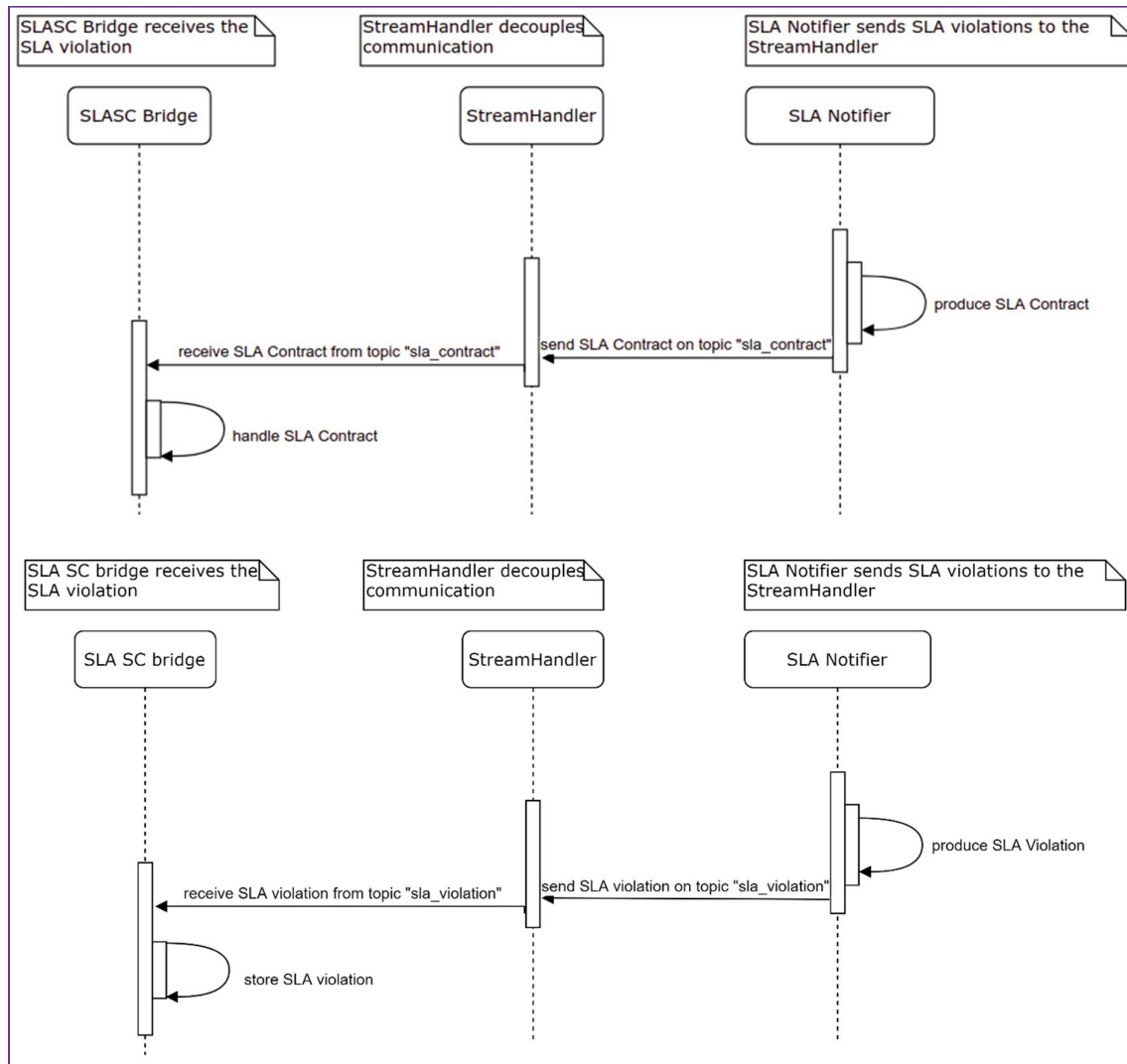


Table 19: SLA Notifier – T&R Engine endpoint description

Integration Point Description	
Identifier	11.18 SLA Notifier – T&R Engine
Responsible	ATOS, ICCS
Integration Point Purpose	Get the SLA violation notifications for providers audit records/trails
Sequence diagram	

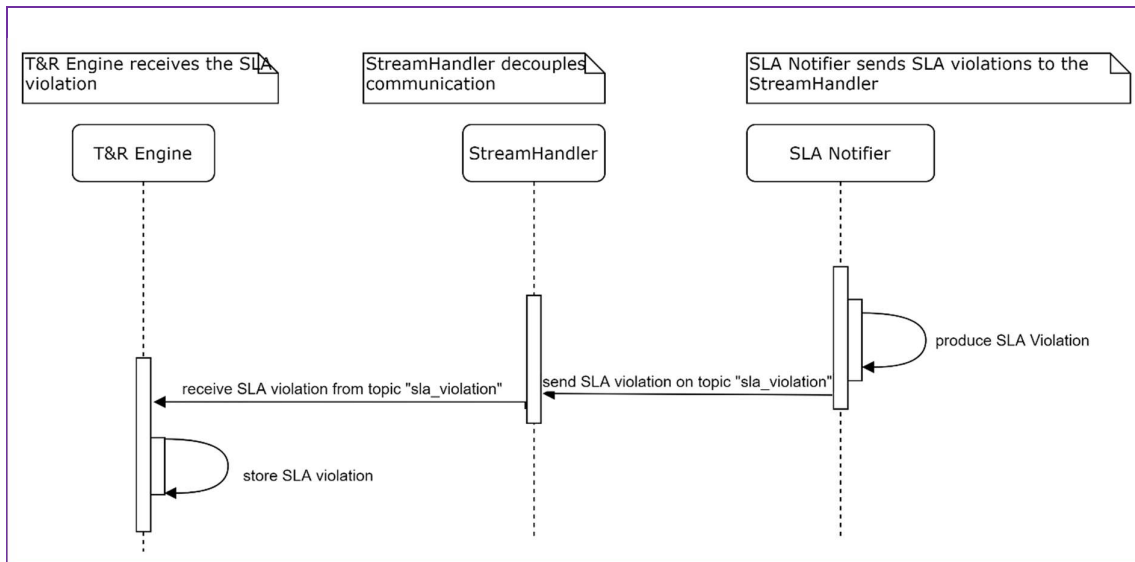
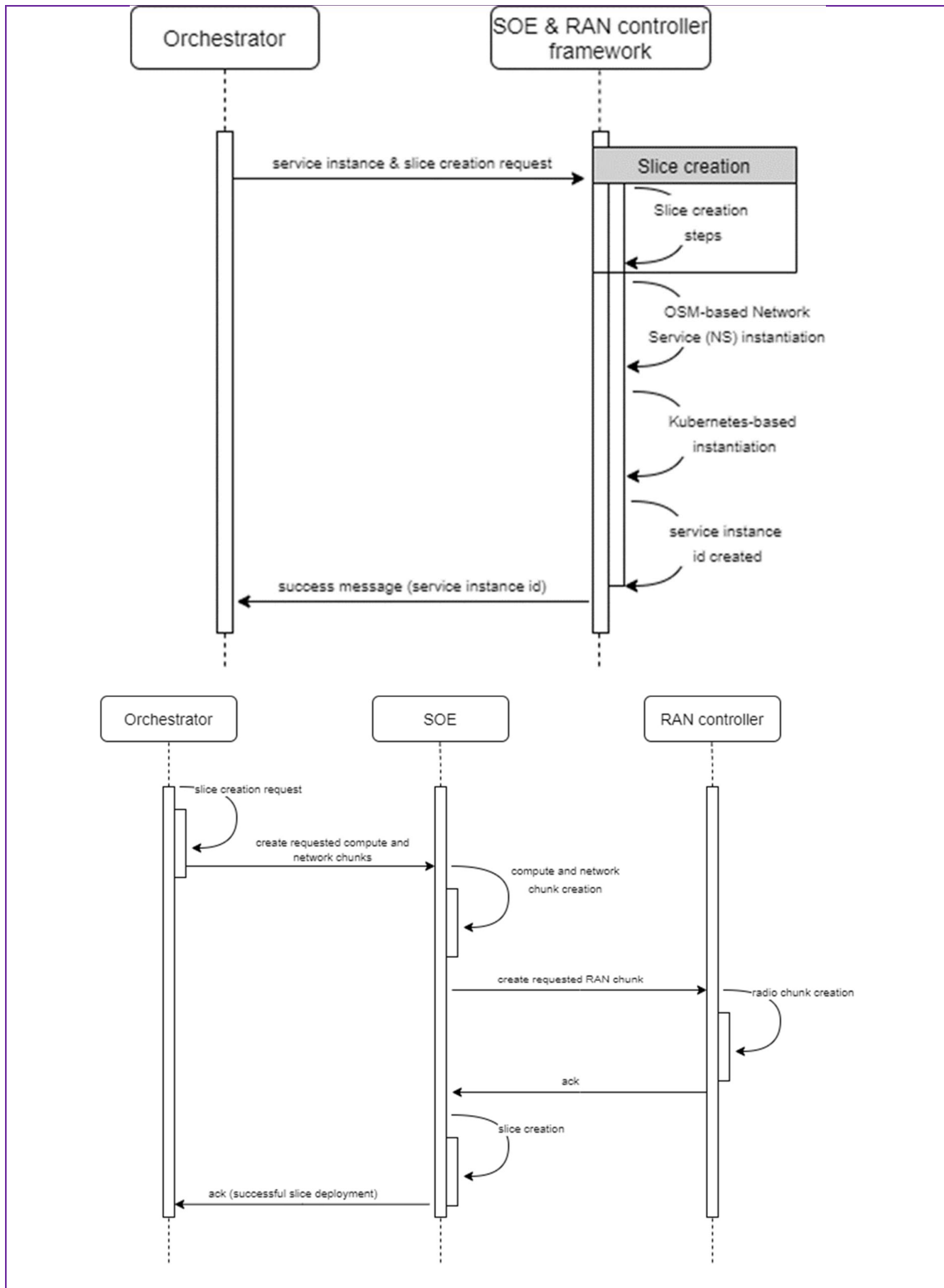


Table 20: Pledger SOE Framework – Pledger RAN Controller endpoint description

Integration Point Description	
Identifier	19.20 PLEDGER SOE Framework – PLEDGER RAN Controller
Responsible	i2CAT
Integration Point Purpose	Request the creation of an end-to-end slice comprising compute, network and radio resources.
Sequence diagram	



5 Conclusions

5.1 Key take-aways

This work approaches in detail the Pledger integration methodology. During M1-M24 and following the definition of the Pledger Core Architecture (found in D2.3 “Pledger Overall Architecture”), the partners exchanged information on the possible integration points among the deployed components, focusing on the available protocols and Application Programming Interfaces (APIs) as well as the exchange of functional tests.

Pledger adopts a Continuous Integration/Continuous Deployment infrastructure. By enforcing automation in the building, testing, and deployment of applications, CI/CD bridges the gap between development and daily operations. Although code can be deployed using multiple virtualisation paradigms, Pledger features a container registry and a K8S cluster, linking code delivery with deployment, following a microservice approach. This microservice approach has grown in popularity in recent years. Microservices provide the fabric for a modular architecture and bring agility in the deployment of the distinct functionalities of the Pledger Core. Any single application can be deployed, torn down and replaced in a matter of seconds without affecting other critical applications.

Furthermore, the introduction of the StreamHandler platform allows an additional layer of data integration, where applications can utilise publish/subscribe mechanisms to access the types of data they require for their operation. The use of the StreamHandler foundation allows the project to improve interoperability, data management as well as enforce security mechanisms in data access.

5.2 Planned work

This work features the first version of the Pledger integration plan, whereas the final version will be provided at M33 (Aug 2022) as part of the D5.6 deliverable “Pledger Integrated Demonstrator II” and will accompany Pledger’s Second Release. This upcoming document will report on all follow-up work after M24 (Nov 2021) and will provide the updated plan towards the second demonstrator. The technical work performed within the project will be monitored closely, and the timely exchange of information on protocols, interfaces and integration points will be provided by the Pledger partners. The integration plan will be refined to reflect the progress of the project activities and end-to-end testing will be performed. The expected outcome is a second, fully featured release of the Pledger Core systems. Figure 21 provides an overview of the integration process after M24 and towards the second release in M33.

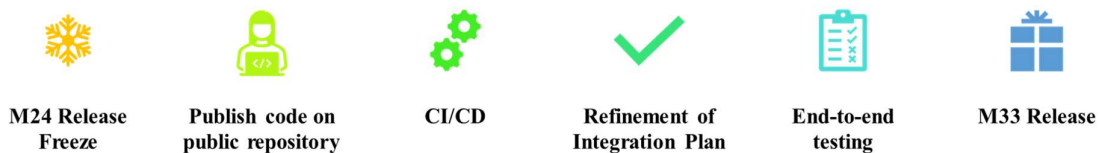


Figure 21: Overview of Integration process after M24

Document name:	D5.2 Pledger integrated demonstrator I	Page:	54 of 58
Reference:	D5.2	Dissemination:	PU
	Version:	1.0	Status: Final

6 References

- [1] PLEDGER. D2.3 – Pledger Overall Architecture, <http://www.pledger-project.eu/content/deliverables> . Voutyras, Orfevs. 2021
- [2] PLEDGER. D5.3 – Pilots operation and monitoring I, <http://www.pledger-project.eu/content/deliverables> . Stanzl, Verena. 2021
- [3] Hetzner Cloud home page, <https://www.hetzner.com/cloud>, Retrieved on 28/11/2021.
- [4] Gitlab home page, <https://about.gitlab.com/>, Retrieved on 28/11/2021.
- [5] Jenkins home page, <https://www.jenkins.io/>, Retrieved on 28/11/2021.
- [6] Portainer home page, <https://www.portainer.io/>, Retrieved on 28/11/2021.
- [7] JFrog Container Registry home page <https://jfrog.com/container-registry/>, Retrieved on 28/11/2021.
- [8] Docker home page, <https://www.docker.com/>, Retrieved on 28/11/2021.
- [9] Kubernetes home page, <https://kubernetes.io/>, Retrieved on 28/11/2021.
- [10] Active MQ home page, <http://activemq.apache.org/>, Retrieved on 28/11/2021.
- [11] RabittMQ home page, <https://www.rabbitmq.com/>, Retrieved on 28/11/2021.
- [12] Apache Avro home page, <https://avro.apache.org/>, Retrieved on 28/11/2021.
- [13] The Design Structure Matrix (DMS) home page, <https://dsmweb.org/introduction-to-dsm/>, Retrieved on 28/11/2021.

Document name:	D5.2 Pledger integrated demonstrator I				Page:	55 of 58	
Reference:	D5.2	Dissemination:	PU	Version:	1.0	Status:	Final

Annex A – Pledger architecture

For completeness purposes and in order to provide a better overview to the reader in this annex the latest Pledger Core System high level architecture is presented together with a brief description of the Pledger Core subsystems. A comprehensive breakdown of the main components and actor interactions are listed in deliverable D2.3 [1] which focuses on the Pledger architecture. Various constraints from the technological, scientific and market domain are taken under consideration, as captured in the requirements analysis (T2.2) and the various specifications for the Project Use Cases (T2.1).

Considering the different objectives of Pledger, two main types of subsystems are identified:

- ▶ **Core subsystems:** The Pledger core system is considered the system that aggregates all the core results and developed components of the project. Corresponding components result in the “minimum viable product” (MVP) of Pledger, the system that will be freely available to users and other projects and initiatives. Most of the core components represent the exploitable part of the project and contain all the main features that will deem Pledger successful as a project
- ▶ **Use Cases subsystems:** The Pledger Use Case subsystems are considered the subsystems that are developed for the pilots of the project. Use Case subsystems are considered the subsystems that fulfil the objectives of the pilots.

The core subsystems that have been identified are the following:

- 1 **Configuration subsystem:** Subsystem responsible for providing and storing the infrastructure and application configuration details.
- 2 **Orchestration subsystem:** Subsystem responsible for managing the orchestration of containerized applications (actions related to the deployment of applications, infrastructure scale up/down, etc.).
- 3 **Benchmarking subsystem:** Subsystem responsible for providing performance data of configured infrastructures to better characterise them and to optimise the orchestration with suggestions on application performance.
- 4 **SLAs subsystem:** Subsystem responsible for creating, managing and evaluating the SLAs associated to the applications running on the Pledger Cloud and Edge environment.
- 5 **Blockchain subsystem:** Subsystem responsible for providing the Pledger DLT and the corresponding middleware (set of tools offering features such as smart contracts and cryptocurrencies).
- 6 **Big Data & Communication Platform subsystem:** Subsystem responsible for exposing a high-performance distributed streaming platform for interconnecting, storing, transforming.
- 7 **Security subsystem:** Subsystem responsible for enhancing the security of the overall system as a whole.

For convenience, these subsystems are split in three different functional groups:

- 1 **Management subsystems:** Subsystems that are focused on the management of IaaS and SaaS.
- 2 **Evaluation subsystems:** Subsystems that are focused on the evaluation of IaaS.
- 3 **Support subsystems:** Supporting subsystems related to communications and security.

The following figure gives the subsystems view of the Pledger Core System. The view is not hierarchical and the interrelations between the subsystems that are provided are the main ones.

Document name:	D5.2 Pledger integrated demonstrator I			Page:	56 of 58
Reference:	D5.2	Dissemination:	PU	Version:	1.0
				Status:	Final

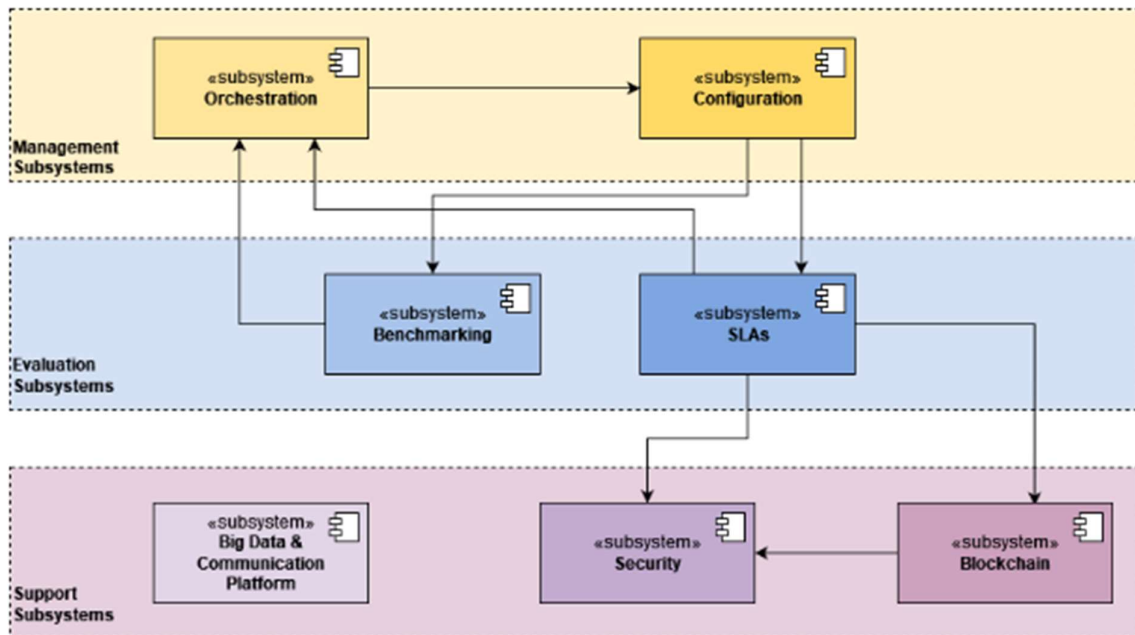


Figure 22: The Pledger Core Subsystems

The Big Data & Communication Platform subsystem is used as an integration pattern (through a publish/subscribe mechanism) to bridge the functional components of the different subsystems of the Pledger Core Platform and to facilitate the overall integration. Instead of placing the subsystem in the middle of the diagram and showing all of its bilateral arrows with other components, it has been decided to present the subsystem as isolated, so as to not lose the information of which components are communicating with each other.

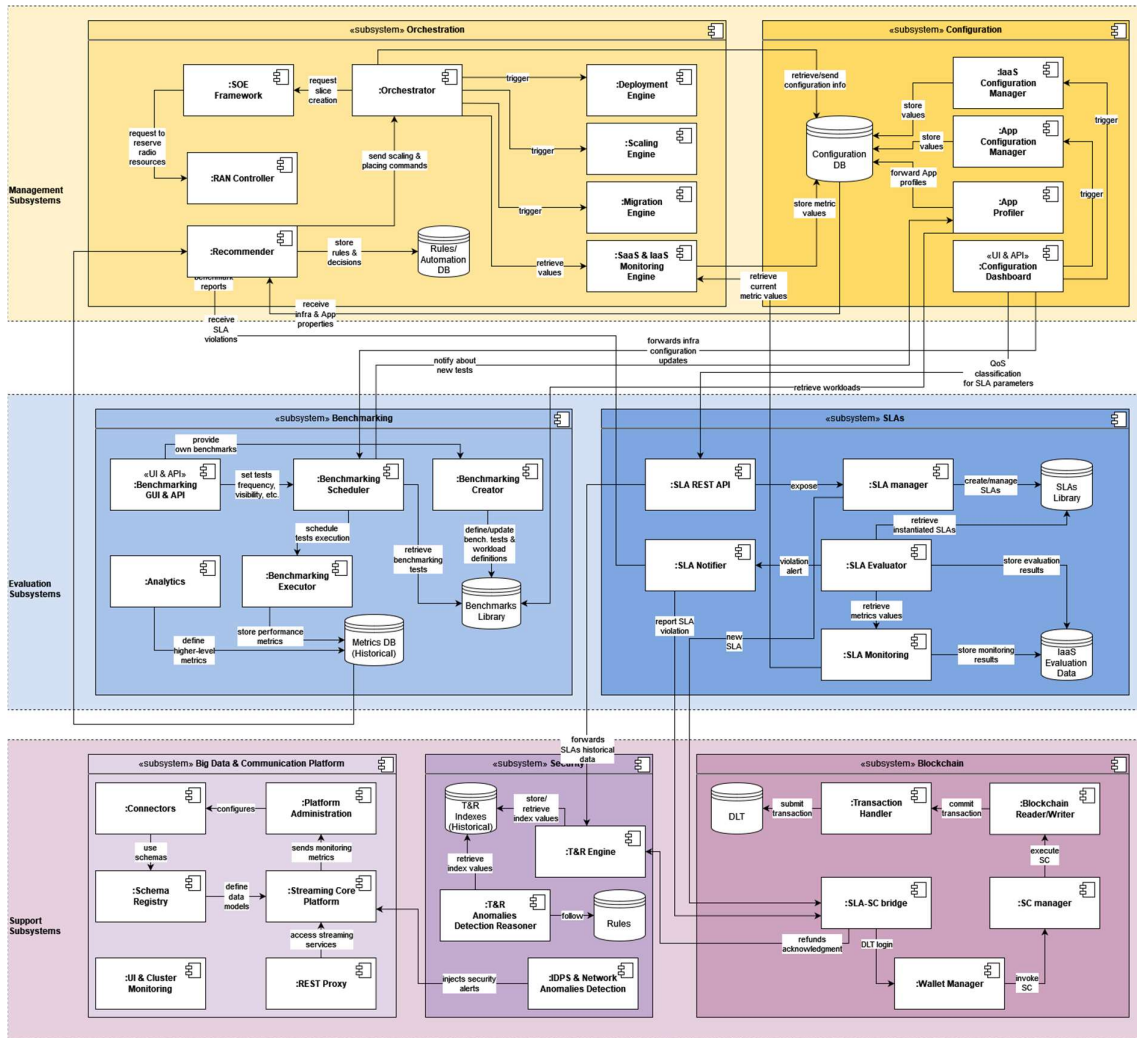


Figure 23: The Pledger Core System