This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Nº 101017331

D3.11 ODIN Platform v2

Deliverable No.	D3.11	Due Date	28/02/2023
Description	Platform architecture, components, integration, deployment manual, developer manual, user manual, verification and validation results.		
Type	Report	Dissemination Level	PU
Work Package No.	WP3	Work Package Title	Platform integration, Privacy Security and Trust + knowledge + cognition
Version	1.0	Status	Final

Authors

Name and surname	Partner name	e-mail
Ilias Kalamaras	CERTH	kalamar@iti.gr
Konstantinos Votis	CERTH	kvotis@iti.gr
Vasileios Lolis	CERTH	vaslwlis@iti.gr
Konstantinos Flevarakis	CERTH	kostisfl@iti.gr
Anastasia Blitsi	CERTH	akblitsi@iti.gr
Pilar Sala	MYS	psala@mysphera.com
Pablo Lombillo	MYS	plombillo@mysphera.com
Alejandro Medrano	UPM	amedrano@lst.tfo.upm.es
Daphne Plati	FORTH	daphni.plati@gmail.com
Luis Carrascal Crespo	INETUM	luis.carrascal@inetum.com
Sofia Granda-Sanz	INETUM	sofia.granda@inetum.com
Francesca Manni	PEN	Francesca.Manni@philips.com
Marcello Chiurazzi	SSSA	Marcello.Chiurazzi@santannapisa.it
Mike Karamousadakis	THL	mike.karamousadakis@twi.gr
Marta Millet	ROB	mmillet@robotnik.es
Saskia Haitjema	UMCU	s.haitjema@umcutrecht.nl
Anna Zondag	UMCU	A.G.M.Zondag@umcutrecht.nl

History

Date	Version	Change
10/11/2022	0.1	Initial draft containing table of contents and first skeleton content.
11/01/2023	0.2	Content Addition.
22/02/2023	0.3	Content Addition.
05/04/2023	0.4	Reworked to contain one-page tables for components.
16/05/2023	0.5	Version ready for peer-review

22/05/2023	0.6	Version ready for quality-check
23/05/2023	1.0	Final version

Key data

Keywords	System architecture, components, flow of operations
Lead Editor	Konstantinos Flevarakis (CERTH)
Internal Reviewer(s)	FORTH, UPM, CUB

Abstract

D3.11 “ODIN Platform v2” outlines the design of the second version of the ODIN platform. The first version of the document defined the overall architecture of the platform, while this version provides detailed information about each component and their connections. The document covers various aspects of the architecture, including Key Enabling Resources, Hospital Information System, resource gateway, resource management and federation, high-level platform services and management, external communication, security, and DevOps infrastructure. It also includes information on how the components of the platform are integrated using an API specification, how the platform and related infrastructure can be deployed, testing scenarios for system integration, documentation, and platform manuals.

Statement of originality

Table of contents

List of tables

List of figures

1 Introduction

This deliverable presents the ODIN platform architecture, its components and how they connect with each other to support the ODIN's use cases. This is the second version of the deliverable to be followed by another one final version. The scope of this deliverable is to go one step further than the previous deliverable and to provide an updated status of the ODIN components based on their implementation and integration status as well as to introduce a validation and a verification plan for the platform. The final version of the deliverable will report the complete and final status of every component in the platform with updated testing and user/developer manuals.

The ODIN platform aims to provide a system to support the interconnection of custom and diverse Key Enabling Resources, including Robotics, IoT and AI, that enable smart hospital functionalities. The platform provides the mechanisms for these resources to communicate with each other, in a way that is agnostic of the particular implementation of each resource, i.e. its communication protocol, data organization schema, etc. In a sense, the ODIN platform aims to act as an "operating system", which enables the communication between resources to achieve the goals setup by the ODIN use cases.

The deliverable aims to form a specification and a guide throughout the ODIN platform, acting as a map that can help readers locate the platform components, read details about their functionalities and specifications, and find examples of their usage. The platform components range from the Key Enabling Resources, to resource management and abstraction services, to high-level platform services and horizontal aspects

The rest of the deliverable is organized as follows:

- Section 2 presents an overview of the ODIN architecture, presenting the main architecture diagram showing the components involved in an ODIN instance and how they are connected.
- Section 3 presents the Key Enabling Resources of the architecture as well as the Hospital information system
- Section 4 presents each component of the ODIN architecture, grouping the components in functional categories.
- Section 5 presents the security-related components of the platform.
- Section 6 presents the DevOps infrastructure used to build the platform.
- Section 7 presents how each component is integrated to the platform by documenting the services that are exposed by each component to communicate and be used by other components.
- Section 8 describes issues related to the deployment of the ODIN platform at the envisioned sites as well as giving an updated deployment status.
- Section 9 introduces a verification and validation plan, providing test scenarios for the components and the reference use cases in ODIN.
- Section 10 contains manuals for the use of the platform, targeted to end-users, deployers and developers. More details will be provided in the next version of this deliverable.
- Section 11 concludes the deliverable.

1.1 Deliverable context

Table 1 provides an overview of the context of the current deliverable, in relation to the project objectives and foreseen results.

Table 1: Deliverable context.

PROJECT ITEM	RELATIONSHIP		
Objectives	The deliverable is relevant to ODIN's Objective 1, as it describes and defines the architecture of the ODIN platform, which enables the secure federation of KERs empowered by all types of involved technologies (robotics, IoT, AI, databases, etc.).		
Exploitable results	All components described in the current deliverable are potential exploitable results, as well as the overall system architecture itself, as a reference architecture for similar endeavours.		
Workplan	D3.11 is attributed to the tasks of WP3, Platform integration, Privacy, Security and Trust + knowledge + cognition. Specifically, the task principally involved in the preparation of this deliverable is T3.1, DevOps and infrastructure. However, input from the other WP3 tasks is also used, regarding the ODIN ontology, the security infrastructure and the documentation and feedback provision.		
Milestones	D3.11 is a key deliverable of the PREPARATION (MS1) and IMPLEMENTATION (MS3) phases of the project.		
Deliverables	D2.1	ODIN co-creation workshop and end-user requirements	Regarding the requirements guiding the architecture design.
	D2.2	Hospital requirements report	Regarding the hospital requirements guiding the architecture design
	D2.3	ODIN Platform catalogue	Regarding the available resources that can be potentially shared and orchestrated
	D3.1	Operational framework	Regarding the DevOps architecture
	D3.2 – D3.3	Hospital Knowledge Base and ODIN semantic ontology	Regarding the design of the ODIN ontology
	D3.4 – D3.6	Privacy Security and Trust report	Regarding security mechanisms
	D3.7 – D3.9	Technical Support Plan and Operations	Regarding component documentation and feedback collection.
	D4.x	All WP4 deliverables	Regarding the implementation of WP4 components

	D5.x	All WP5 deliverables	Regarding the implementation of WP5 components
	D6.x	All WP6 deliverables	Regarding the implementation of WP6 components
	D7.x	All WP7 deliverables	Regarding system evaluation
Risks	The guidelines provided in this deliverable can help minimize the following risks identified in the Grant Agreement: • Technologies not available in time (by examining alternative solutions and supporting third-party solutions, if needed) • Technical problems during component/module development (by allowing to develop each component in isolation, reducing the dependencies on other system components) • Complexity of unification procedure (by defining a unified procedure for resource abstraction that can be applied to arbitrary resources) • Risk of time-consuming integration due to multiple technologies used (by providing means for resource abstraction, facilitating communication between diverse resources).		

2 ODIN platform architecture overview

Before going into details about the architecture of the ODIN platform, we provide definitions for some terms that will appear often in the rest of the text.

- A Key Enabling Resource (KER) is a piece of hardware or software (robots, IoT, AI) that provides essential functionality for the purposes of the ODIN project, i.e. the implementation of applications that assist in hospital operations.
- The ODIN platform refers to the set of components that handle KER registration, communication, orchestration and management towards the implementation of customizable applications.
- The core components of the ODIN platform are the components of the ODIN platform that are not KERs, i.e. the components that enable the communication and management of the KERs.
- An ODIN instance is an instantiation of the ODIN platform at a particular physical site, such as a hospital, a residence, or a cloud virtual machine. Different ODIN instances may employ different KERs, but the core components of the ODIN platform are the same in all ODIN instances.
- The ODIN DevOps infrastructure is the set of components that are responsible for the continuous development, integration and deployment of the ODIN platform at the ODIN instances. This infrastructure is architecturally outside the individual ODIN instances, but each ODIN instance is connected to allow its continuous support.

The architecture of the ODIN platform, as deployed in a typical ODIN instance can be seen in the diagram of

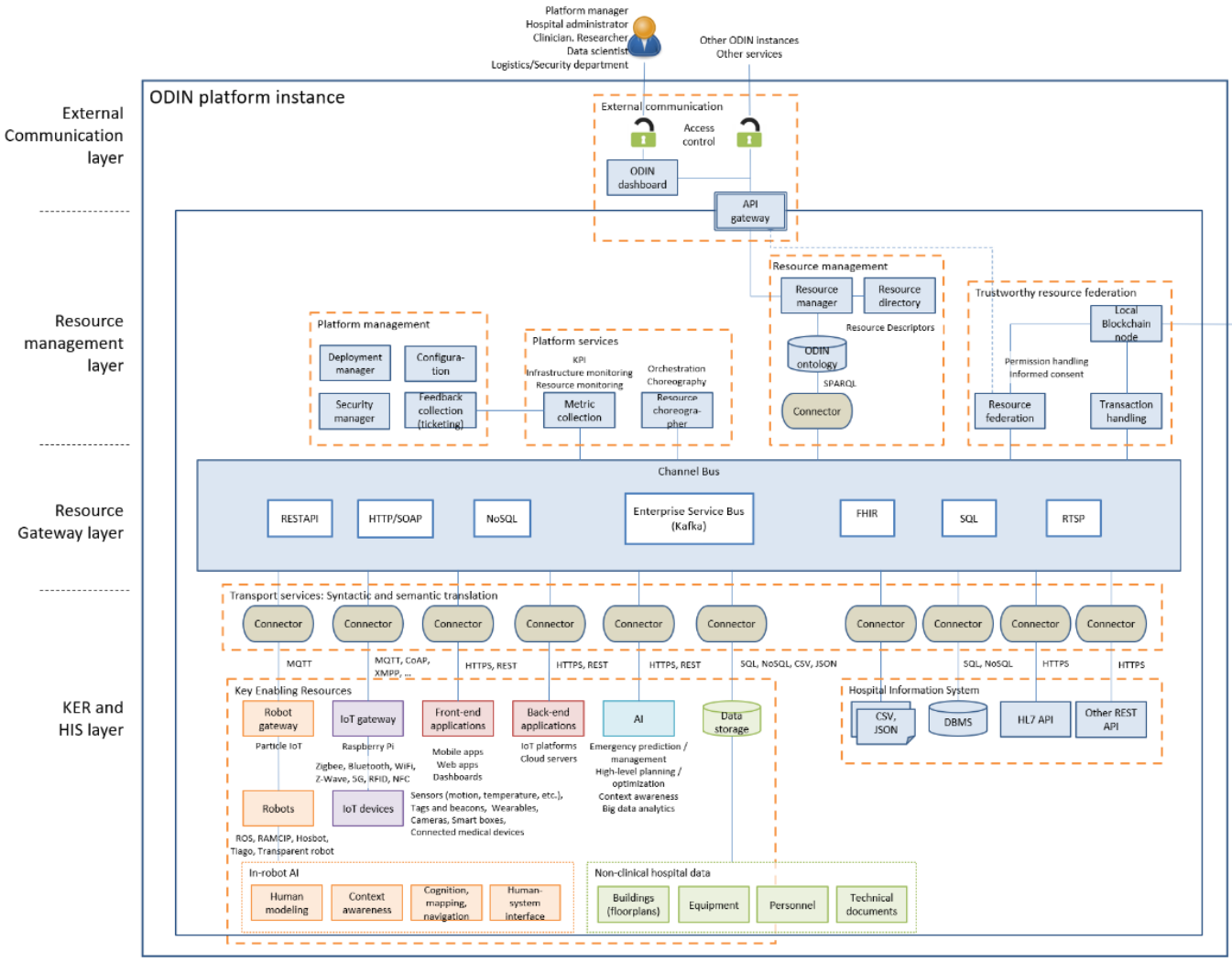

Figure 1. This diagram captures the main components involved in the ODIN platform and their connections and can be used as a map to position each platform component in relation to the other components.

ODIN follows a bus-oriented microservice architecture [1], where most components communicate with each other as microservices through a central message-passing bus. This reference architecture was chosen to facilitate the communication between diverse components and the extensibility of the platform as new KERs are provided, or as further components are developed, e.g. through open calls, or beyond the end of the project.

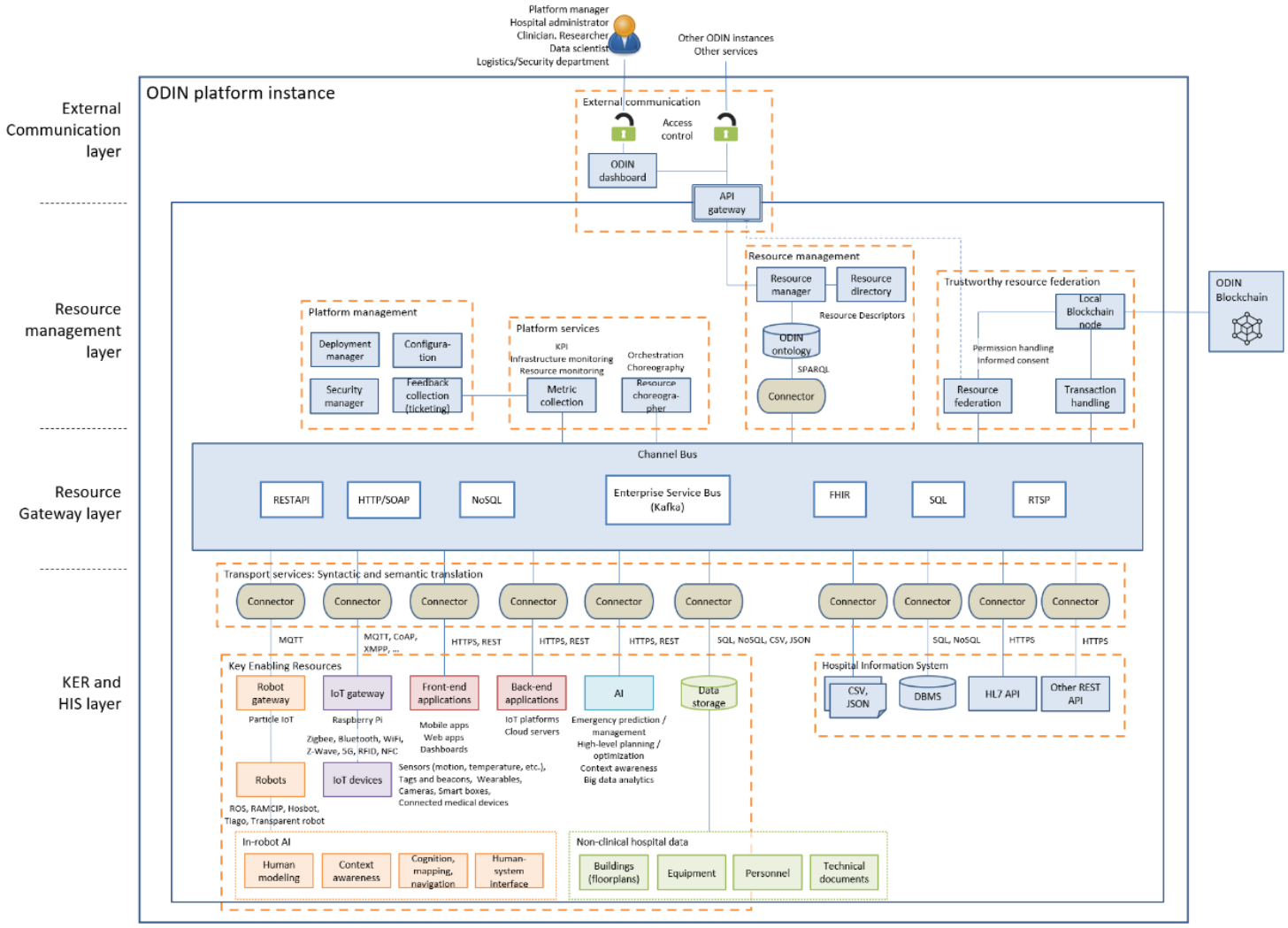

Figure 1: The architecture of the ODIN platform.

Perhaps the most straightforward way to read the diagram of

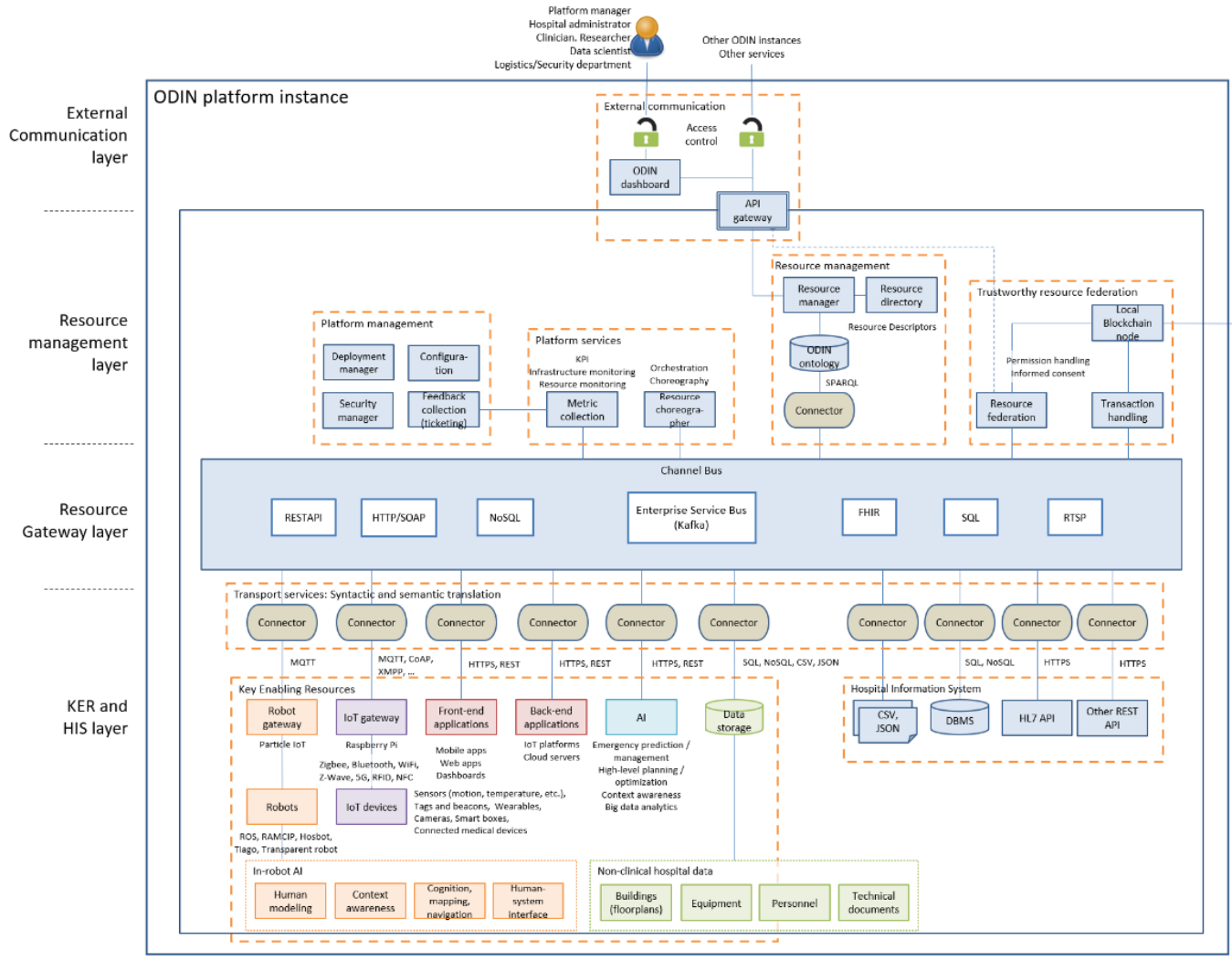

Figure 1 is in a bottom-up direction. Four layers can be distinguished, starting from "low-level" components at the bottom to user interaction at the top. However, these layers do not correspond one-by-one with the layers of other traditional architectures, e.g. with hardware at the bottom and user interfaces at the top. This is because the "low-level" components at the bottom, i.e. the KERs, are often complete systems by themselves, and range from hardware devices to high-level AI algorithms and front-end applications. They are termed as "low-level" in the current context because they form the pieces out of which higher-level applications can be built. The four conceptual layers of the ODIN platform are the following, starting at the bottom:

- The **KER and HIS layer**, consisting of the Key Enabling Resources and the Hospital Information System, which are the individual pieces that provide either existing or new functionalities in the hospitals, and need to be managed by the ODIN platform.
- The **Resource Gateway layer**, consisting of the central message bus, and the components responsible for enabling the communication between diverse resources.
- The **resource management layer**, consisting of core components for resource registration, semantic description, federation and monitoring.

- The **external communication layer**, consisting of components responsible for secure communication with the outside world.

At the bottom layer of the architecture lie the **Key Enabling Resources (KERs)**. These form the conceptual core of the ODIN platform and the ODIN project in general. They are the enabling technologies that power the implementation of smart functionalities in the hospital environments: robotic platforms, IoT devices, AI algorithms, databases, back-end and front-end applications. It is the purpose of the rest of the ODIN platform to support the communication between the diverse available KERs. The **Hospital Information System (HIS)** lies at the same layer, since it consists of already existing components for patient data and hospital management that need to communicate and harmonize with the KERs.

To implement high-level logic and support hospital use cases, the KERs and HIS need to communicate with each other. This is handled by the communication layer, which mainly consists of the **Enterprise Service Bus (ESB)**, which is the main message bus through which all components of the ODIN platform communicate. The latest version of the platform, V2, has made significant improvements to the **Channel Bus** by incorporating additional communication protocols such as HTTP, RestAPI, MySQL, and more. This upgrade aims to boost interoperability by allowing seamless interaction between different systems.. Communication between KERs is not trivial, since KERs are individual systems constructed by diverse manufacturers and developers, each using its own communication protocols and data representation schemas. To enable communication between KERs, each KER is connected to the ESB through a **connector**. A connector performs a two-stage transformation of the information produced or consumed by the KER, as shown in Figure 2: **semantic translation**, to transform the data in the common data representation used in ODIN, and **syntactic transport**, to transform the KER communication protocol to the ESB communication protocol.

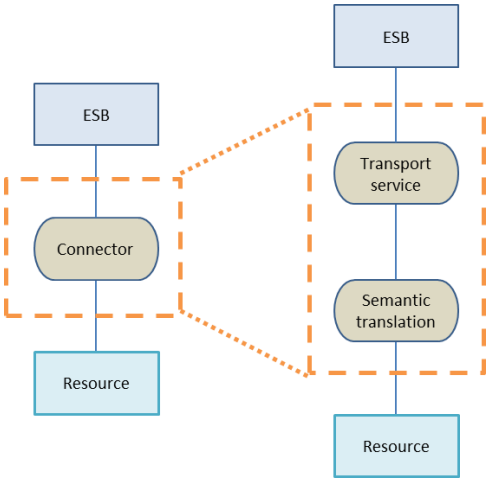

Figure 2: Each connector handles both semantic translation and transport to the ESB.

KERs can be registered to the ODIN platform using the **Resource Manager** component. Registration informs the platform that the KER is available for use, by specifying its details, such as identifiers, its corresponding semantic entity in the ODIN ontology, any exposed APIs, etc. The **ODIN ontology** is a central component holding information about all kinds of entities and information relevant to the ODIN platform, such as devices, algorithms, places, people, processes, etc., along with their semantic relationships.

One of the central ideas of the ODIN platform is to allow its implementation in a decentralized manner, by sharing resources between ODIN instances. An ODIN instance can use resources of another remote ODIN instance, as if they were local, forming networks of ODIN instances that collectively can achieve local or large-scale goals, as depicted in Figure 3. This networking is

achieved in a trustworthy manner through the use of the **Resource Federation** component, which handles permissions and consent between ODIN instances, and the **Transaction Handling** component, which logs all transactions performed between remote resources. Both permission requests and transactions are stored in the **ODIN blockchain**, to provide immutability and traceability of the stored information, and accountability and non-repudiation of the involved parties.

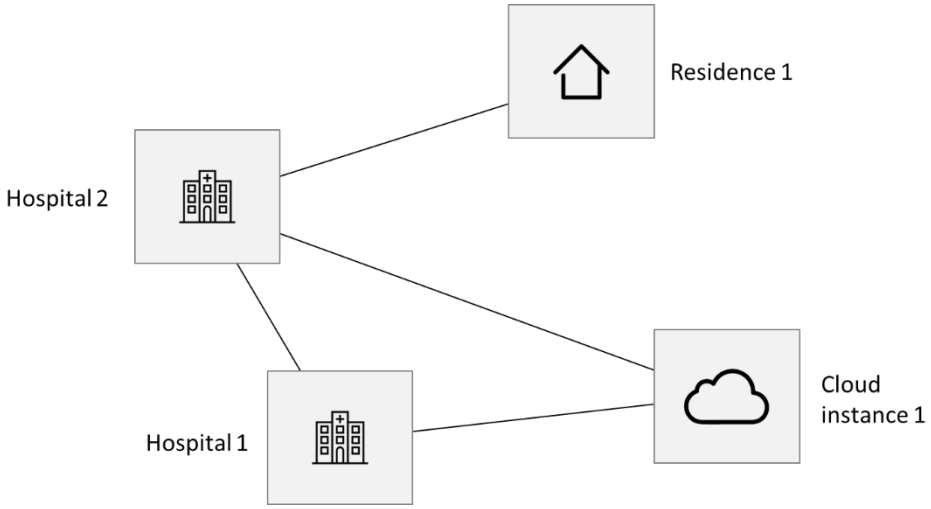

Figure 3: Connection of ODIN instances in a decentralized scheme.

Local or remote resources can be used by and ODIN instance to perform high-level logic that is relevant with the operation of the hospital, such as logistic services, or disaster handling. This logic is described using the **Resource Choreographer** component, by connecting KERs to each other in operational workflows that are then executed through the ESB. The **Metric Collection** component is connected to the ESB to collect performance metrics and KPIs regarding the use of the KERs, and make them available to high-level applications or the end users.

The ODIN platform also contains services for managing the platform itself, such as the **Deployment Manager**, which manages the deployed component microservices, the **Security Manager**, which manages access control across the ODIN platform and the **platform configuration manager**, for platform-level configuration. The **ODIN documentation** component provides access to the documentation of all ODIN components and KERs. The **Feedback Collection** component allows end-users to provide feedback regarding the use of the platform and report bugs and issues.

At the top layer of the architecture lie the components responsible for the communication with the outside world. External communication is handled by the ODIN **API Gateway**, which routes external calls to the relevant ODIN components. KERs that expose APIs to external services or users are made visible through the API gateway, to provide a common point of access and facilitate authorization. The **ODIN dashboard** provides a Graphical User Interface through which end users can have access to the functionalities provided by the ODIN platform or by the KERs. User and service authorization is handled by the ODIN **Access Control** mechanism.

The latest version of the Odin Platform introduces two main updates. The Channel bus, an upgraded version of the ESB bus, enables support for additional communication channel protocols such as HTTP and MySQL. Furthermore, components like platform documentation are now categorized as external components within the platform.

The above described components form the ODIN platform as it is installed in an ODIN instance. However, there are more components in the architecture of the ODIN ecosystem, which support the continuous development and deployment (DevOps) of the components of the ODIN platform.

This supporting infrastructure is depicted in Figure 4 and is described in detail in D3.1 “Operational framework”.

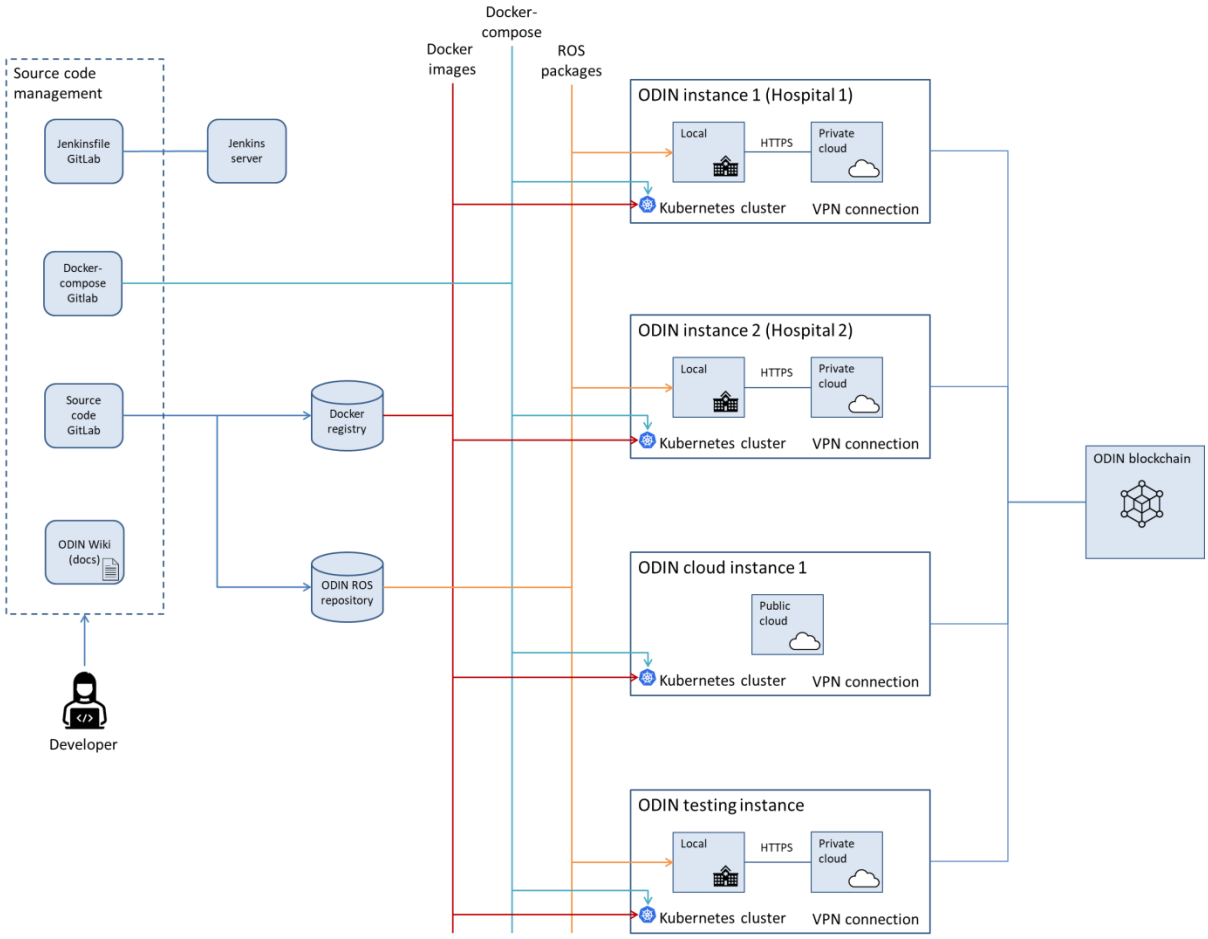

Figure 4: ODIN development and deployment infrastructure.

The DevOps infrastructure provides a continuous path from the developer of a component to the deployment of the component at the ODIN instances. The source code produced by the developer and the associated documentation is submitted to the ODIN **source code repository** and the ODIN Wiki (implemented within the source code repository). The source code is built into Docker images which are uploaded to the ODIN **Docker registry**, while the whole process is managed by an **orchestration server** (Jenkins). Packages developed for the robotics applications can be uploaded to the **ODIN ROS repository**, to be installed in the robots operating in the hospitals.

The Docker images available in the Docker registry and the Docker-compose files describing how to deploy each component are used by the **Kubernetes clusters** acting as a substrate in the ODIN instances, to deploy the KERs and the components of the ODIN platform. The ODIN instances themselves are deployed either at the local infrastructure of the hospitals, or at **private cloud virtual machines** allocated to the hospitals, in case the hospitals do not possess the necessary computational resources.

The above overview can be used as a quick reference of the components involved in the ODIN platform. Sections 3, 4, 5, 6 are providing more detailed descriptions of all the above outlined

components, and more details can be found in relevant deliverables. Table 2 provides a list of all components, with links to the corresponding sections in this deliverable, and references to the relevant tasks and deliverables where one can find more information about their functionalities and implementation.

Table 2: ODIN platform components and relevant references for more information.

Category	Component	Section	Relevant task(s)	Deliverable(s)
KERs	Robotic platforms	3.1.1	T5.1-T5.5	D5.1-D5.9
	Internet of Things	3.1.2	T4.1, T4.2	D4.1, D4.2-D4.4
	Artificial Intelligence	3.1.3	T6.1-T6.3	D6.1-D6.7
	Datasets	3.1.4	T3.2, T4.2	D3.2-D3.3, D4.2-D4.4
HIS	Hospital Information System	3.2	T3.2, T4.2	D3.2-D3.3, D4.2-D4.4
Resource gateway	Enterprise Service Bus	4.1.1	T4.3	D4.2, D4.2-D4.4
	Transport services	4.1.2	T4.3	D4.2, D4.2-D4.4
Resource manage-ment	ODIN ontology	4.2.1	T3.2	D3.2-D3.3
	Resource manager	0	T4.1, T4.2	D4.1, D4.2-D4.4
	Resource descriptor	0	T4.2	D4.2-D4.4
	Resource directory	4.2.4	T4.2	D4.2-D4.4
Resource federation	Resource federation	4.3.1	T4.4	D4.5-D4.7
Platform services	Metric collection	4.4.1	T4.6	D4.2-D4.4
	Resource choreographer	4.4.2	T4.5	D4.5-D4.7
Platform manage-ment	Deployment manager	4.5.1	T3.1	D3.10-D3.12
	Access control manager	4.5.2	T3.1, T3.3	D3.4-D3.6, D3.10-D3.12
	Platform configuration	4.5.3	T3.1	D3.10-D3.12
External communi-cation	API gateway	4.6.1	T4.3	D4.2-D4.4
	Administration dashboard	4.6.2	T3.1	D3.10-D3.12
Horizontal aspects	Platform security	5	T3.3	D3.4-D3.6
	DevOps infrastructure	6	T3.1	D3.1, D3.10-D3.12

The next two sections of the document will be used to provide all the additional information to the reader about the ODIN platforms KER's and components. Every KER and component will be described in a form of table with a brief information about it. Moreover, it will point to the relevant deliverables for detailed information. Section 3 will describe the KERs and Section 4 the platform components. This deliverable offers also a more detailed description for the components that are not described in different deliverables.

3 ODIN platform Key Enabling Resources and Hospital information System

This section describes the Key Enabling Resources (KERs) available in ODIN. These are the enabling components that offer all needed functionality to implement specific use cases. The section also describes the Hospital information system.

3.1 Key Enabling Resources

This section describes the Key Enabling Resources (KERs) available in ODIN. These are the enabling components that offer all needed functionality to implement specific use cases. They are at the bottom of the ODIN architecture diagram of

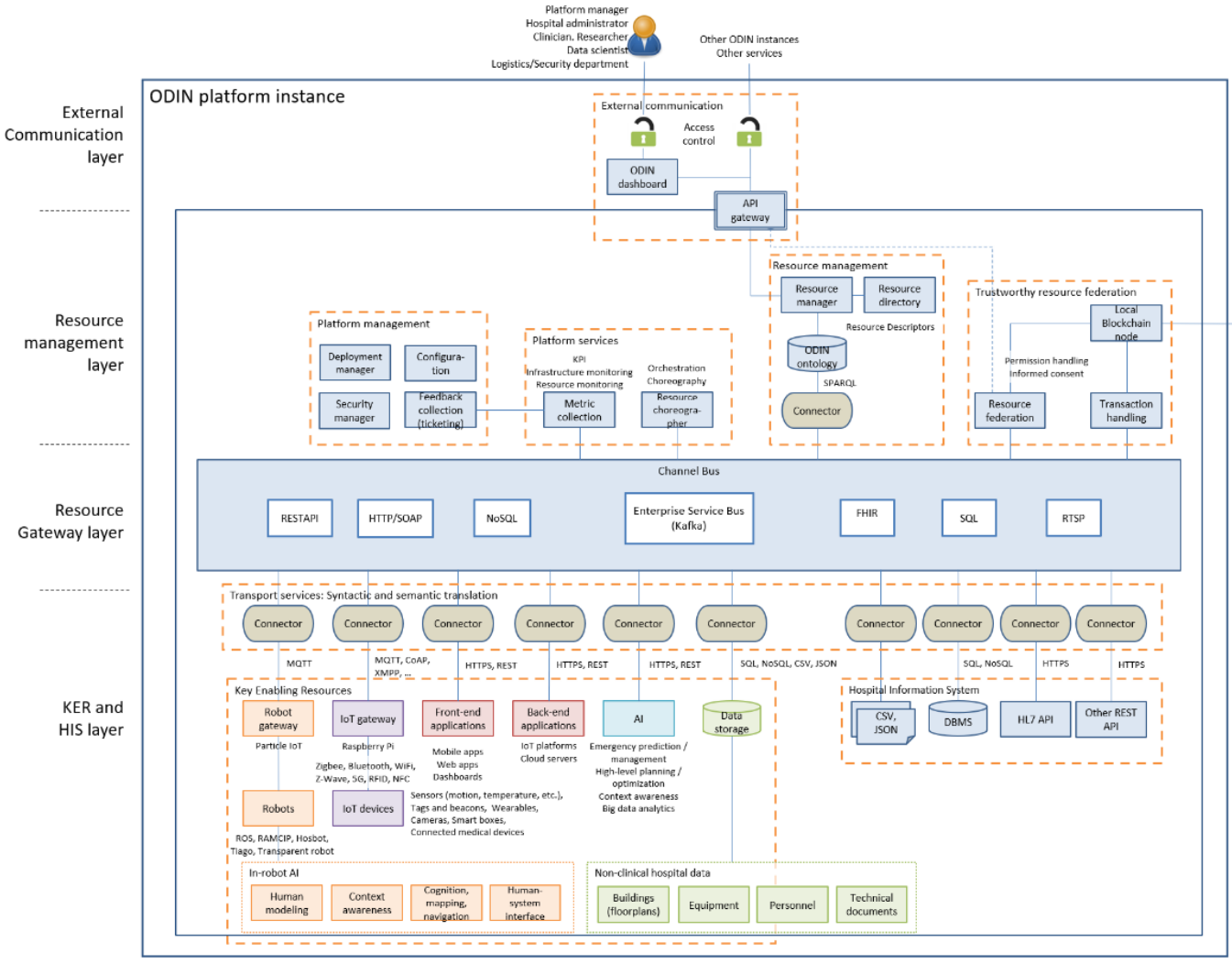

Figure 1, shown in detail in Figure 5. The following types of KERs are described:

- Robotic platforms
- IoT platforms
- AI algorithms

- Front-end and back-end services
- Data storage services

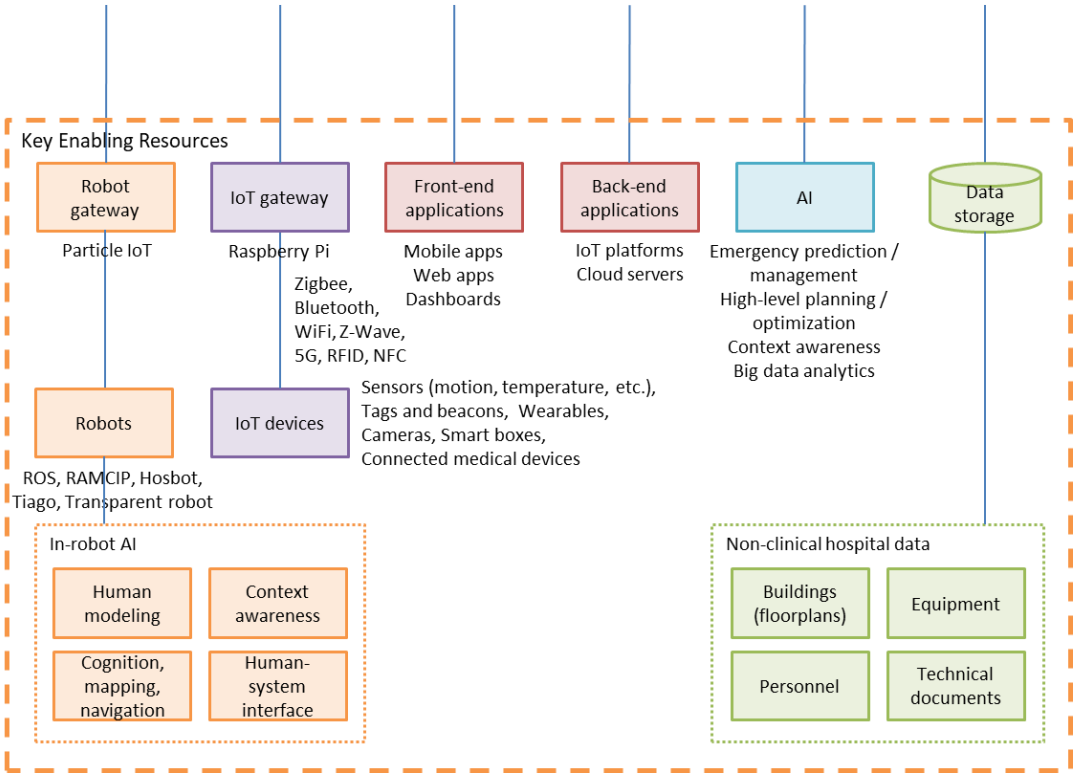

Figure 5: The ODIN Key Enabling Resources.

3.1.1 Robotic platforms

Robotic platforms support operations in the hospital that involve physical contact and interaction with people or with the building. They enable a wide range of applications for the hospital environment, such as clinical support (e.g. carrying patients) and logistics (e.g. distributing material in the hospital).

3.1.1.1 Robots

Table 3: Hosbot Robot.

HOSBOT Robot	
Introduction	HOSBOT is a robot that is used in ODIN. Robotic platforms support operations in the hospital that involve physical contact and interaction with people or with the building. They enable a wide range of applications for the hospital environment, such as clinical support (e.g. carrying patients) and logistics (e.g. distributing material in the hospital).
Place in the Architecture	KER layer

Specifications/Features	• An indoor robotic mobile platform intended for autonomous and collaborative indoor transportation and delivery of goods around the hospital. It also includes a configurable smart boxes system for enabling robotic transportation with automatic goods recognition through RFID and RTLS location technologies, and proximity sensors for enhanced context awareness.
Relevant Deliverable(s)	D5.3 Technology Integration report D2.3 ODIN platform catalogue
Prerequisites	Robot Gateway
Current status	First version ready and under real environment testing
How to Install	Pending to be published in the Wiki
Why to use this component	By utilizing HOSBOT in ODIN hospitals, healthcare facilities can benefit from increased efficiency in transportation and delivery processes, reduced human workload, improved inventory management through automatic goods recognition, and enhanced safety with the robot's context awareness capabilities

Table 4: Robot TIAGo.

Robot TIAGo	
Introduction	TIAGo is a robot that is used in ODIN. Robotic platforms support operations in the hospital that involve physical contact and interaction with people or with the building. They enable a wide range of applications for the hospital environment, such as clinical support (e.g. carrying patients) and logistics (e.g. distributing material in the hospital).
Place in the Architecture	KER layer
Specifications/Features	• TIAGo: service robot equipped with an 8-DoF arm, a RGBD camera and a speech synthesizer. It is also combined with a mobile differential base enabling moving and performing different helping tasks in the healthcare environment.
Relevant Deliverable(s)	D5.3 Technology Integration report D2.3 ODIN platform catalogue
Prerequisites	Robot Gateway
Current status	First version ready and under real environment testing.
How to Install	Pending to be published in the Wiki
Why to use this component	In an ODIN smarthospital, TIAGo can assist with tasks such as delivery of medication or supplies, patient monitoring, and providing information to patients and staff. Its arm, camera, and speech synthesizer enable it to perform a wide range of duties, making it a versatile and valuable asset in the healthcare environment. By utilizing TIAGo, ODIN smarthospitals can enhance productivity, improve patient care, and alleviate some of the workload on healthcare staff.

Table 5: Robot CERTHBOT.

Robot CERTHBOT	
Introduction	CERTHBOT is a Robotic Platform. Robotic platforms support operations in the hospital that involve physical contact and interaction with people or with the building. They enable a wide range of applications for the hospital environment, such as clinical support (e.g. carrying patients) and logistics (e.g. distributing material in the hospital).
Place in the Architecture	KER layer

Specifications/Features	• CERTHBOT: is a robotic platform developed as an assistant for patients. The robot handles emergencies by contacting nurses or caregivers when necessary. Monitors patient actions and provides reports about them. Reminds them of the medication schedule and assists them with the rehabilitation process by monitoring the procedure and alerting if some exercise is performed incorrectly. It should be noted that CERTHBOT is a descendant of RAMCIP • CERTHBOT
Relevant Deliverable(s)	D5.3 Technology Integration report D2.3 ODIN platform catalogue
Prerequisites	Robot Gateway
Current status	First version ready and under real environment testing
How to Install	Pending to be published in the Wiki
Why to use this component	By utilizing CERTHBOT in an ODIN smarthospital, healthcare providers can enhance patient care, improve patient safety, promote medication adherence, and support the rehabilitation process. The robot's monitoring, reporting, emergency handling, and assistance functionalities contribute to a comprehensive and supportive approach to patient care.

Figure 6: Examples of robotic platforms used in the ODIN project. From left to right: HOSBOT, TIAGo, RAMCIP, Pepper, Furhat, CerthBot.

3.1.1.2 Robot Gateway

Table 6: Robot Gateway.

Robot Gateway	
Introduction	In the ODIN project, e-robots are connected to the hospital environment via the Enterprise Service Bus (ESB). MQTT is used for intra-robot communication due to its lightweight and efficient messaging capabilities. Kafka is employed for intra-module communication, with a custom connector being developed to facilitate bidirectional information exchange between the two layers.
Place in the Architecture	KER layer
Specifications/Features	- e-robots in the ODIN project are connected to the hospital environment via the Enterprise Service Bus (ESB). - Communication occurs through two layers: intra-robots and intra-modules. - Intra-robot communication is based on MQTT, an OASIS standard messaging protocol for IoT. - MQTT is chosen for its lightweight publish/subscribe messaging, suitable for remote devices with limited network bandwidth. - The robot communication protocol is designed to be general and scalable, adaptable to specific use-case requirements.
Relevant Deliverable(s)	D5.3 Technology Integration report D2.3 ODIN platform catalogue
Current status	e.g., MQTT connectors ready
How to Install	Pending to be published in the Wiki
Why to use this component	The integration of the ESB, MQTT, Kafka, and the bidirectional message translation connector enhances communication efficiency, system scalability, and adaptability to specific use-cases, ultimately improving the overall performance and functionality of the e-robots within the ODIN project.

3.1.2 Internet of Things (IoT)

Internet of Things (IoT) devices allow monitoring hospital status and activity, such as measuring temperature, detecting presence/absence of objects, detecting motion, capturing images, etc. They are invaluable to support all types of use cases in hospitals, as they provide unobtrusive means of continuous monitoring.

3.1.2.1 IoT devices

Table 7: RTLS Devices.

RTLS	
Introduction	RTLS is charge of providing real time location for people or high value assets in the hospital.
Place in the Architecture	RTLS is inside IoT Ker category.
Specifications/Features	Currently the RTLS provides instant position for the assets tracked (x, y, z, plus time of the measure). Each position is published in a single topic.
Relevant Deliverable(s)	- D4.3 Implementation of Local CPS-IoT RSM v2
Prerequisites	RTLS require: - The appropriate connector to communicate with the ODIN platform ESB and report the data. - Knowledge on the Resource Descriptor technologies, to send the information required by the standard used. - Ontology branch related to IoT devices - Infrastructure installation with gateways and server
Current status	The RTLS is developed. A new version of the connector must be created with ontology included.
How to Install	Pending to be published in ODIN Wiki To learn about the standards used for the Resource Descriptor D4.3 and the standards web pages can be consulted.
Why to use this component	RTLS allows controlling position of things, automate processes upon position triggers and helps taking decisions. This KER really supports upper layers and other KER's such as robots.

Table 8: Transparent Robot.

Transparent Robot	
Introduction	The Transparent robot, developed by SSSA, is an environmental monitoring IoT-based platform designed as an add-on system to evaluate the quality of indoor spaces.
Place in the Architecture	KER IoT device.
Specifications/Features	• Environmental Monitoring: The Transparent Robot serves as an IoT-based platform specifically designed for

	environmental monitoring in indoor spaces. Its primary purpose is to evaluate the quality of the monitored environment. • Integrated System: This robot is considered an integrated system, meaning it combines various components to enable comprehensive context awareness. It incorporates a significant number of sensing components that contribute to its monitoring capabilities. • Modular Design: The Transparent Robot is designed as a modular device, allowing for flexibility in sensor connectivity. Different types of sensors can be connected to a microcontroller based on the specific monitoring requirements of the environment. This modular approach facilitates customization and adaptability to different scenarios.
Relevant Deliverable(s)	- D5.8 Technology Integration Periodic Report v2
Current status	Version Ready working with HOSBOT in testing environment
How to Install	Pending to be published in ODIN Wiki
Why to use this component	The Transparent Robot focuses on environmental monitoring, utilizes an integrated system with multiple sensing components, and offers modularity to accommodate diverse sensor connections based on monitoring needs.

3.1.2.2 IoT gateway

Table 9: IoT gateway.

IoT Gateway	
Introduction	MQTT devices are ready to join ODIN platform thanks to the integration of a Mosquitto server.
Place in the Architecture	IoT KER category.
Specifications/Features	MQTT devices can use Mosquitto server to share data through its own topics in MQTT and send them to Kafka thank to the connectors deployed. Exchange data using topics Import/Export data to Kafka
Relevant Deliverable(s)	- D4.3 Implementation of Local CPS-IoT RSM v2
Prerequisites	IoT MQTT devices require: - The appropriate connector to communicate with the Mosquitto server.

	- Knowledge on the Resource Descriptor technologies, to send the information required by the standard used. - Ontology branch related to IoT devices
Current status	Mosquitto is already deployed and all the robots and some AI use MQTT as a first integration with ODIN.
How to Install	Check D4.3. More information to be published into the Wiki.
Why to use this component	IoT components can use MQTT easy due to its reduced resource waste. This way they can integrate to ODIN platform

3.1.3 Artificial Intelligence (AI)

Artificial Intelligence (AI) algorithms allow advanced operations to be performed automatically, such as context awareness, prediction, optimizations, scheduling, etc. AI plays a central role in the ODIN system and in the overall effort to move towards smart hospitals. The AI algorithms in ODIN are considered as Key Enabling Resources, being handled similarly to other resources, i.e. undergoing semantic/syntactic translation, and representation in the ODIN ontology. This allows them to be discovered and used by other resources in custom workflows within or beyond the ODIN use cases. The following tables showcase a selection of notable features exhibited by the AI-based models known as KERs that are being developed within the ODIN project.

Table 10: RUC A-UC: Sleep disorder management.

RUC A – UC: Sleep disorder management	
Relevant Deliverable(s)	D6.6 – Development of the high-level AI-based models of planning, scheduling, and workflow modelling v1
Description	An AI-based model will be developed by PEN to automatically detect a sleep disorder from polysomnography (PSG) or pulse oximetry data. The AI model will detect the presence of a sleep disorder (e.g. sleep apnoea and/or insomnia) and will estimate the staging. The federated prediction can be enables so that an aggregated global model will be used exploiting data from multiple hospitals/sites.
Place in the Architecture	KER
Specifications/Features	Important parts of the AI-based sleep disorder management are the automatic classification and the disorder staging.
Prerequisites	PSG data (or pulse oximetry) and the federated learning pipeline to exploit multiple data sets
Dataset	The dataset is described in detail in Table 16 The first dataset contains around 200 up to 230 sleep apnoea patients that underwent PG and/or PSG recordings between 2007 and 2021. The second dataset contains 64 insomnia patients that underwent PSG recording between 2008 and 2010.

Current status	The federated learning pipeline has been developed and is ready to be deployed at CUB site and in the ODIN platform. The data anonymization is going to be finalized. The next step is to deploy the AI model based on the anonymized data and refine the model with prospective data in the upcoming months/years.
How to Install	Instructions will be presented at the next version of this deliverable.
Why to use this component	• Automatic sleep disorder detection and staging to minimize the error made by the subjective evaluation of the clinicians and the turnaround time to make the diagnosis. • Patients will receive a better diagnosis and consequentially treatment and prognosis.

Table 11: RUC A– UC: Automated patient inclusion system in the UCC.

RUC A– UC: Automated patient inclusion system in the UCC	
Relevant Deliverable(s)	D6.6 – Development of the high-level AI-based models of planning, scheduling, and workflow modelling v1
Description	An AI-based system will be developed by PEN. The system will enable an automatic patient inclusion in the Utrecht Cardiovascular Cohort (UCC) which is a prospective cohort study aiming at a uniform data collection for all cardiovascular patients. The system will exploit patient data from: • lab • diagnosis • agenda • medication • measurement • intervention • billing information in order to perform an automatic inclusion and replace the manual inclusion performed by the clinical staff.
Place in the Architecture	KER
Specifications/Features	Important parts in the AI-based system are the automatic inclusion (e.g. fuzzy matching) using structured and/or unstructured data.
Prerequisites	Patient data to train the AI-based system (ongoing collection performed by UMCU, with expected ending and sharing with PEN by September 2023).
Dataset	The dataset is described in detail in Table 17.

	The data employed for the automated patient inclusion method in the UCC includes age, medication, disease, measurements and other patient data acquired in the outpatient clinic. Data acquisition is ongoing and performed by UMCU. Data sharing for model deployment is planned for September 2023.
Current status	Prospective data acquisition is ongoing and led by UMCU. Data ecosystem and literature review and study performed by PEN.
How to Install	Instructions will be presented at the next version of this deliverable.
Why to use this component	• The automatic inclusion will overcome the limits and drawback of the manual process (e.g. patients that do not provide consent). • It is more sustainable and less time consuming. • The AI-based system avoids missing eligible patients and/or cardiovascular diseases.

Table 12: RUC A2-UC4: Early identification of patients at risk of malnutrition.

RUC A2 – UC4: Early identification of patients at risk of malnutrition	
Description	An AI-based system will be developed by FORTH. The TIAGo robot, using its RGB-D camera, will take an image of the patient's food tray: • before the patient eats his meal • after the patient finishes his meal The system will recognize the different types of food in the tray and estimate its volume in order to calculate the energy intake and the consumed macronutrients.
Place in the Architecture	KER
Specifications/Features	Important parts in the AI-based nutrition system are the food segmentation subsystem, the food classification subsystem and the food volume estimation subsystem.
Relevant Deliverable(s)	D6.6 – Development of the high-level AI-based models of planning, scheduling, and workflow modelling v1
Prerequisites	RGB-D camera of the TIAGo robot, WIFI connection.
Dataset	The dataset is described in detail in Table 18. MedGrFood database for building and training the AI-based models.

	The food image database that will be created from food images from the Geriatrics Unit of UCBM to re-trained the AI-based models.
Current status	All subsystems are developed. They need to be re-trained with the food images from the Geriatrics Unit of UCBM.
How to Install	Instructions will be presented at the next version of this deliverable.
Why to use this component	• Daily reports regarding the patients' food consumption will be available to clinicians. • Timely intervention to minimize complications caused by undernutrition. • Offer the clinicians the opportunity for an early intervention to diagnose and treat undernutrition, and minimize the risk for further hospitalization. • The elderly patients will reach the required energy intake to avoid weight and muscle mass loss, minimize the risk of aspiration and treat dysphagia.

Table 13: RUC A2 – UC4: Rehabilitation monitoring to prevent loss of mobility.

RUC A2 – UC4: Rehabilitation monitoring to prevent loss of mobility	
Description	An AI-based system will be developed that will be able to: • Monitor the patient while is executing the prescribed exercises. • Provide information regarding the correct execution of each exercise and the number of repetitions. • Give feedback to the patient and the medical staff.
Place in the Architecture	KER
Specifications/Features	Two different approaches are examined: i) a sensor-based and ii) a vision-based approach in order to identify the best suited for our research problem. In the sensor-based approach the rehabilitation activities are analyzed using IMU sensors to evaluate and monitor the development of patients undergoing physical therapy. In the vision-based approach the patient is monitored while is executing the prescribed exercises with the RGB-d camera of the TIAGo robot and information regarding the number of repetitions and the correct execution of the exercises is provided.
Relevant Deliverable(s)	D6.6 – Development of the high-level AI-based models of planning, scheduling, and workflow modelling v1
Prerequisites	RGB-D camera of the TIAGo robot, WIFI connection.
Dataset	The dataset is described in detail in Table 20

	• UCBM partners have provided FORTH the appropriate videos of participants executing the rehabilitation exercises for the vision-based approach. • A physiotherapist from the Geriatrics Unit of UCBM has performed each rehabilitation exercise twice while having attached the 5 IMU sensors. The collected data streams are used for the sensor-based approach.
Current status	AI-based models have been developed for both approaches. Further investigation is necessary to determine the optimal threshold for each exercise. We have to deploy the full set of exercises for both approaches to further enhance the analysis and provide more comprehensive results.
How to Install	Instructions will be presented at the next version of this deliverable.
Why to use this component	• In a clinic-based rehabilitation program, the availability of the trained physiotherapists/clinicians confines the rehabilitation treatment along with patients' other scheduled treatments. • Moreover, in cases where the physiotherapist is not available, the patient performs the prescribed rehabilitation exercises without supervision. In such cases, the lack of timely feedback and real-time supervision by a healthcare professional might result in false execution of the exercises, injuries and in long-term not achieving the expected results of the treatment. • Information regarding the number of repetitions, the correct execution of the exercises is provided along with the execution time.

Table 14: RUC A3 - UC5: Oxygen monitoring therapy to prevent hypoxia.

RUC A3 - UC5: Oxygen monitoring therapy to prevent hypoxia	
Description	A robotic platform with AI and IoT modules is developed. Specifically, the patient will be monitored with the RGB-d camera of the TIAGo robot while is receiving the prescribed oxygen therapy. With the proposed solution, the correctness of the therapy in terms of the correct positioning of the oxygen mask is monitored.
Place in the Architecture	KER
Specifications/Features	Face detection using the MTCNN face detector. Oxygen mask detection is based on building and applying a CNN accompanied with the appropriate layers for a multiclass (3-classes) classification problem
Relevant Deliverable(s)	D6.6 – Development of the high-level AI-based models of planning, scheduling, and workflow modelling v1

Prerequisites	RGB-D camera of the TIAGo robot, WIFI connection.
Dataset	The dataset is described in detail in Table 19. Training: 5129 images from individuals that were all members of FORTH (1254 images from participants without wearing the venturi oxygen mask, 1556 images from participants wearing correctly the venturi oxygen mask, 2319 images from participants wearing wrongly the venturi oxygen mask). Validation: The developed models will be validated on images of patients from the Geriatrics Unit of UCBM that will be collect in the purpose of the ODIN project.
Current status	The AI-based models regarding the correct positioning of the venturi oxygen mask are already developed. During the next period the model for the correct positioning of the nasal prongs will be developed. Also, both models need to be validated on images of patients from the Geriatrics Unit of UCBM.
How to Install	Instructions will be presented at the next version of this deliverable.
Why to use this component	In this UC AI capabilities able to monitor patients during the assigned oxygen therapy will be introduced to reduce the time of nurses dedicated to monitoring. This will help to maximize patients' compliance to oxygen therapy and its benefits, and to minimize the side effects due to the excess of oxygen or to an erroneous use, even in a setting like that of COVID-19 where the human presence of medical staff should be reduced for safety reasons.

Table 15: RUC B: Consumables management.

RUC B: Consumables management	
Description	An AI-based system will be developed that will be able to predict the number of stent devices that will be used in the Cardiology department.
Place in the Architecture	KER
Specifications/Features	Time series methods will be explored and methods for reducing the prediction error will also be examined.
Relevant Deliverable(s)	D6.6 – Development of the high-level AI-based models of planning, scheduling, and workflow modelling v1
Prerequisites	-

Dataset	SERMAS dataset. The dataset involves information related to the hospital's stent acquisitions and patients' episodes during the years 2021 and 2022.
Current status	The AI-models are under development.
How to Install	Instructions will be presented at the next version of this deliverable.
Why to use this component	The Cardiovascular Institute (ICV) in Hospital Clínico San Carlos in SERMAS, represents around the 48% of the total hospital expenditure in medical equipment and consumables. With the AI-based model

3.1.4 Datasets

Within the ODIN architecture, data are also considered as a KER. Hospitals possess data that are invaluable for their operation and can be exploited by the other enabling technologies (robotics, IoT and AI) to boost hospital operations further.

Patient-related data are generally stored in the databases of the Hospital Information System (HIS), which is discussed in Section 3.2. However, there are other types of data regarding hospital functionalities that may lie outside the HIS, in separate databases, which can provide essential information for the ODIN use cases. These include e.g., building-related data, such as floorplans, equipment databases, personnel information, technical documents regarding hospital operations, etc.

To include these data as resources in an ODIN instance, they are considered as KERs, and are connected to the main bus (ESB). This means that appropriate connectors need to be developed that have access to the raw data, in their original form (e.g. spreadsheets, CSV files, relational databases, etc.) and allow other KERs to retrieve them via the ESB. For example, such a connector could be setup to listen for requests towards a relational database, and upon such a request generate the appropriate SQL query, submit it to the connected DBMS, retrieve the results, transform them to the common bus format (FHIR) and publish them to the bus. The upon mentioned are described in depth in the Deliverable D3.3.

Apart from data that already exist in the hospitals and need to be connected to the ODIN platform, there may be the need for KERs to store data they generate. In this case, the ODIN platform provides databases that can be used to store arbitrary information. For instance, a deployed IoT platform may lack a data storage mechanism, so ODIN can provide a database for measurement storage. Or an AI algorithm may need to store trained models or intermediate outputs in a long-term storage. Such databases can be setup by the consortium partners and be connected to the ESB in the same manner that existing databases can be connected, in order to allow their access by the KERs.

The data storage facilities are designed as the repositories that will host every kind of information in the system and. These ODIN data repositories are composed from all the required DB schemas and will allow the basic CRUD (Create, Read, Update, and Delete) operations. In order to fit the different kinds and formats of the collected data more than one database engine are used, both relational and non-relational. The noSQL DB will be used only for the case of unstructured data since this type of DB technology is highly-scalable and provides better functionality to the management of unformatted data. The relational database will be normalized as required for a good practice and to avoid data redundancy. The databases will be modelled in a completely

transparent manner for the clients. When the performance is not enough then the resources can be increased without impact or effect on the applications.

With regard to relational databases, the following requirements are satisfied by the ODIN data storage mechanisms:

- The databases will satisfy all the requirements of the First Normal Form (1NF) i.e. no duplicated columns from the same table, creation of separate tables for each set of related data and identification of each row with a unique field (column of primary key).
- The databases will satisfy all the requirements of the Second Normal Form (2NF), i.e. they will be in 1NF and subsets of data applying to multiple rows of a table will be removed from that table and placed in separate tables, creating relationships between these new tables and their predecessors through the use of foreign keys.
- The databases will satisfy all the requirements of the Third Normal Form (3NF), i.e. they will be in 2NF and columns that are not dependent upon the primary key will be removed from the tables.
- No columns will contain lists of objects.

At the Tables below are presented the datasets that are used in the ODIN project:

Table 16: RUC A – UC: Sleep disorder management.

RUC A – UC: Sleep disorder management	
Relevant Deliverable(s)	D6.1 – The data model ecosystem for AI operations and modules implementation D6.2 – Data results interpretation and data integration services v1
Provider	CUB will provide to PEN the retrospective datasets to build an AI model for automatic sleep disorder detection.
Description	This dataset will be used for the AI-based system that will support the clinician to make a faster and more accurate diagnosis.
Place in the Architecture	KER
Features	Includes features regarding AI-based sleep disorder detection and staging based on PSG and pulse oximetry data.

Table 17: RUC A– UC: 4.2 Automated patient inclusion system in the UCC.

RUC A– UC: 4.2 Automated patient inclusion system in the UCC	
Relevant Deliverable(s)	D6.1 – The data model ecosystem for AI operations and modules implementation

	D6.2 – Data results interpretation and data integration services v1
Provider	UMCU will provide to PEN the prospective dataset to build an AI model for automated inclusion in the UCC.
Description	This dataset will be used for the AI-based system that will enable an automatic inclusion approach to replace the current manual screening and inclusion.
Place in the Architecture	KER
Features	Includes features regarding processing structured and unstructured data and an automatic inclusion algorithm (e.g. fuzzy matching).

Table 18: RUC A2 – UC4: Early identification of patients at risk of malnutrition.

RUC A2 – UC4: Early identification of patients at risk of malnutrition	
Provider	UCBM Geriatrics Unit will provide Forth the food image dataset in order to build the food image database, which is the key to create a highly accurate model for a dietary assessment system.
Description	This dataset will be used for the AI-based system that will recognize the different types of food dishes in the tray and estimate its volume to calculate the energy intake and the macronutrients consumed.
Place in the Architecture	KER
Features	Includes features regarding food categorization, characterization and nutritional analysis of food images.
Relevant Deliverable(s)	D6.1 – The data model ecosystem for AI operations and modules implementation D6.2 – Data results interpretation and data integration services v1

Table 19: Dataset for RUC A3 - UC5: Oxygen monitoring therapy to prevent hypoxia.

Dataset for RUC A3 - UC5: Oxygen monitoring therapy to prevent hypoxia	
Provider	FORTH recorded 26 participants 26 performing the requested head movements. The individuals that participated were all members of FORTH. The dataset contains 5129 photos of participants (1294 without wearing the oxygen mask, 1556 wearing the oxygen mask correctly, 2319 wearing the oxygen mask wrongly).

Description	This dataset will be used to build the AI-based models able to detect the correctness of the therapy in terms of the correct positioning of the oxygen mask (for both nasal prongs and venturi mask).
Place in the Architecture	KER
Features	This dataset consists of images of individuals wearing the venturi oxygen mask correctly, not wearing the oxygen mask and wearing the oxygen mask wrongly. Specifically includes 5129 images from individuals that were all members of FORTH (1254 images from participants without wearing the venturi oxygen mask, 1556 images from participants wearing correctly the venturi oxygen mask, 2319 images from participants wearing wrongly the venturi oxygen mask).
Relevant Deliverable(s)	D6.1 – The data model ecosystem for AI operations and modules implementation D6.2 – Data results interpretation and data integration services v1

Table 20: Dataset for RUC A2 – UC4: Rehabilitation monitoring to prevent loss of mobility.

Dataset for RUC A2 – UC4: Rehabilitation monitoring to prevent loss of mobility	
Provider	UCBM
Description	This dataset will be used for the AI-based system that will be able to: • Monitor the patient while is executing the prescribed exercises. • Provide information regarding the correct execution of each exercise and the number of repetitions. • Give feedback to the patient and the medical staff.
Place in the Architecture	KER
Features	In the context of the ODIN project, UCBM partners have provided FORTH the appropriate videos of participants executing the rehabilitation exercises for the vision-based approach. A physiotherapist from the Geriatrics Unit of UCBM has performed each rehabilitation exercise twice while having attached the 5 IMU sensors. The collected data streams are used for the sensor-based approach.
Relevant Deliverable(s)	D6.1 – The data model ecosystem for AI operations and modules implementation D6.2 – Data results interpretation and data integration services v1

3.2 Hospital Information System

The term Hospital Information Systems (HIS) refers to the component of health informatics that places focus largely on the administrative, financial, and clinical needs of hospitals. These systems augment the ability of healthcare professionals to coordinate care by providing a patient's health information and visit history at the place and time that it is required. So basically, HIS is designed to manage patients and their related information in a centralised way via electronic data processing and predict health status within the hospital environment. No doubt, it has as its aim to provide better healthcare service with precision accuracy. Figure 7 depicts the HIS-related components within the ODIN architecture.

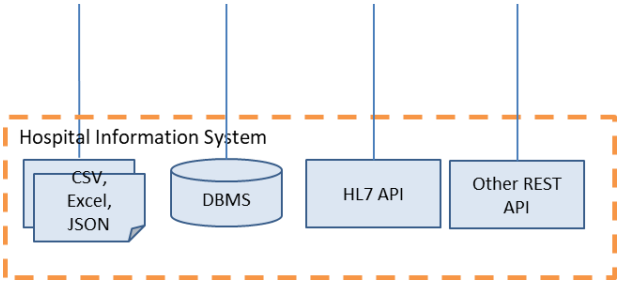

Figure 7: The Hospital Information System connected to the ODIN platform.

Figure 8 details the internal components of a HIS and its integrations with different applications, portals and devices. The way a HIS is implemented and how it can interact with external systems varies largely across hospitals, which makes its integration with the ODIN platform challenging. As an example, the way data are organized and made available to external parties could vary from complete HL7-compatible APIs to custom REST APIs, to databases with no available API, to plain data files (e.g. CSV, JSON, Excel sheets, etc.). To be able to interact with this diversity in hospital information systems, the ODIN platform puts emphasis in the implementation of appropriate connectors (syntactic and semantic transformation) that provide the necessary interface between the HIS and the ODIN ESB.

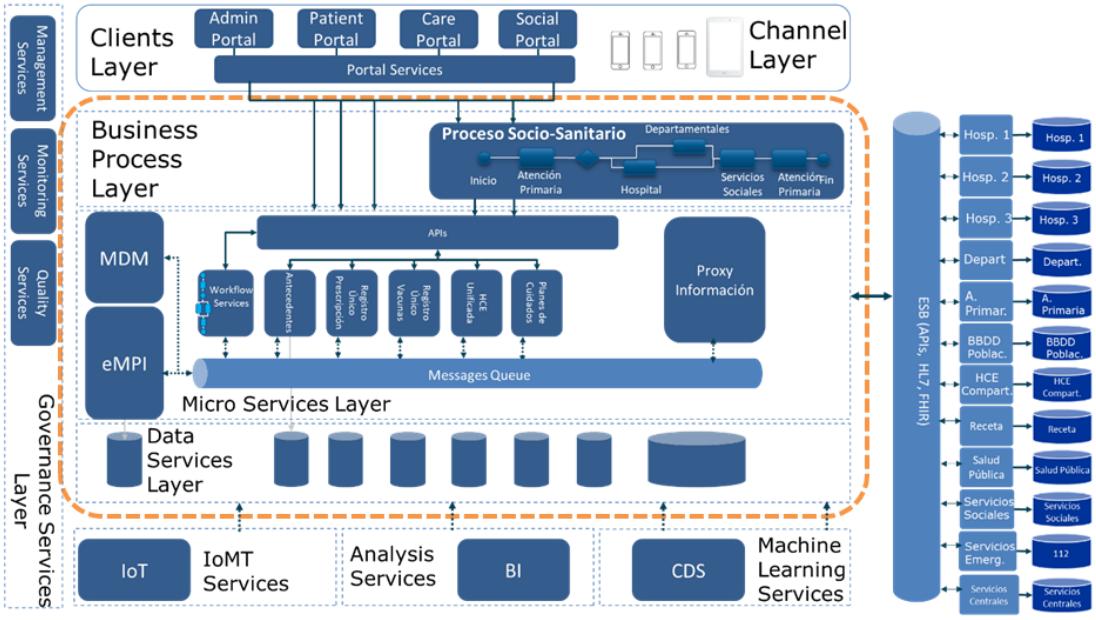

Figure 8: Internal details of a Hospital Information System.

4 ODIN platform components

This section describes each component of the ODIN platform in more detail, covering its main functionalities and how it interacts with other components. Section 7 provides examples of how these components interact with each other to accomplish main tasks.

4.1 Resource gateway

The Resource Gateway is the component that is in charge of managing communication to the ODIN upper layers, functioning as the only entry point and coordinating calls to the other modules. Having a single-entry point will avoid the management of large number of point-to-point connections that would increase the complexity, dependencies and maintenance time. Figure 9 depicts the ODIN Resource Gateway as it appears within the overall ODIN platform architecture

(

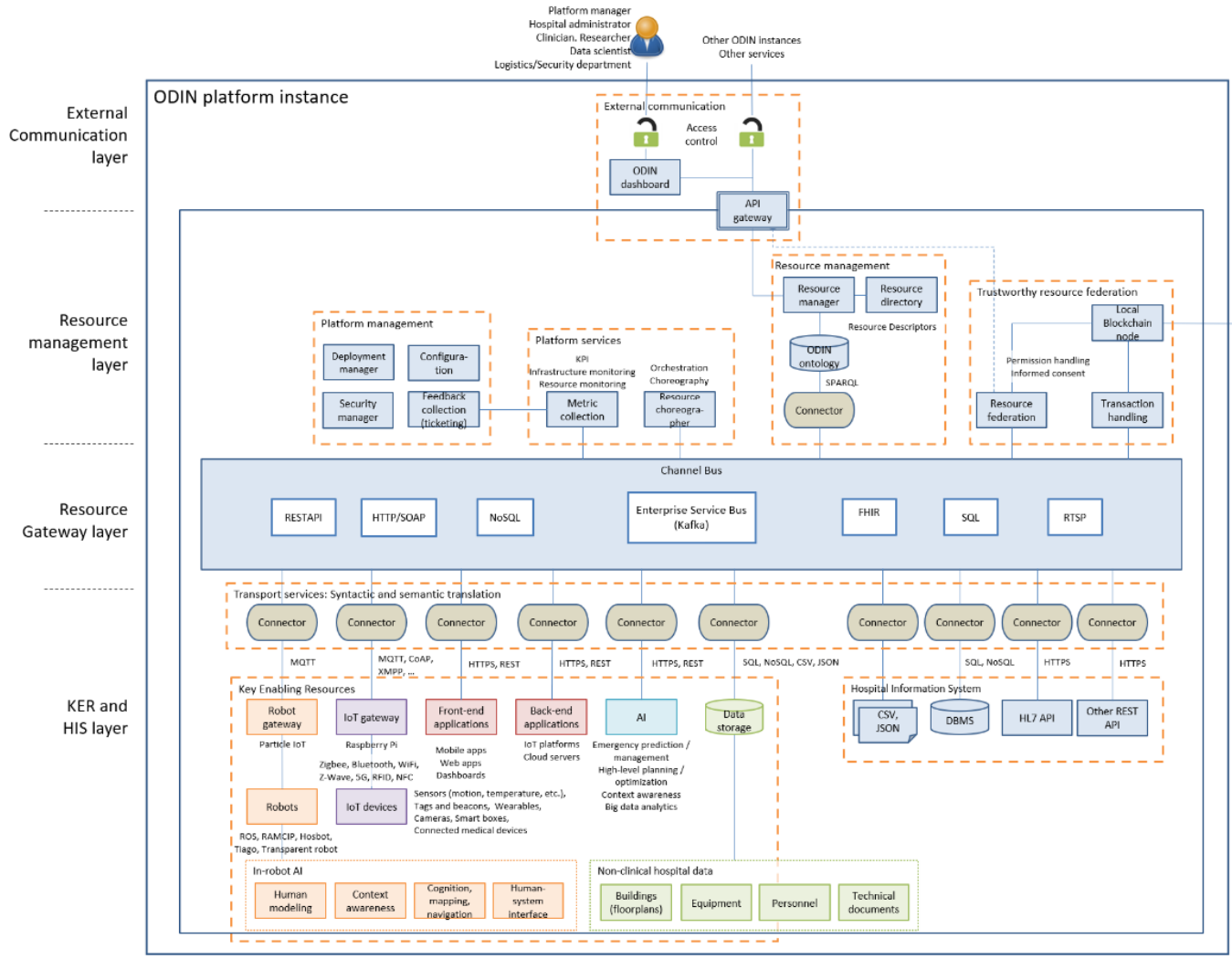

Figure 1).

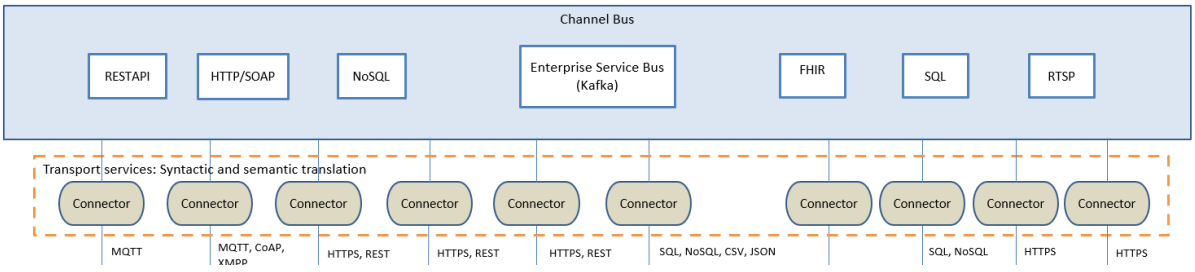

Figure 9: The ODIN Resource Gateway.

The Resource Gateway follows an Enterprise Service Bus (ESB) approach but, in addition, ODIN offers other kinds of communication channels to integrate more KER's and adapt to some requirements that do not fit an event message scheme. A KER registering into the platform will inform the supported interfaces, protocols and functions to the Resource Manager, using the Resource Descriptor attributes. This way, any service or KER will be able to query the Resource Manager and understand the supported communication channels of each KER.

Within the Resource Gateway we can identify three main modules needed to fulfil the requirements:

- the Channel Bus, which integrates previous ESB and other means of communication
    - Enterprise Service Bus, implemented with Kafka and supporting communication using messages. Specifications are defined using AsyncAPI. Will be used for core component communications such as registering, discovery, choreography management, but also for publishing information or receiving instructions.
    - Peer to peer channels
        - SQL endpoints
        - NoSQL endpoints
        - REST (i.e, Web of Things, OpenAPI)
        - HTTP/SOAP
        - Streaming over RTSP
        - Application level (FIHR, HL7)
        - Graphql
        - File based
- the **transport services**, for syntactic transformation of messages, implemented in the Connectors, using Apache Nifi, Apache Camel or Kafka Connect, depending on the availability of solutions.
- the **semantic translators**, for semantic transformation of messages, that will also be part of the Connectors.

4.1.1 Enterprise Service Bus (ESB)

Table 21: Enterprise Service Bus.

Enterprise Service Bus (ESB)	
Introduction	The ESB is the multi-channel communication component available in ODIN Platform. Main channel is asynchronous messaging based, implemented with Apache Kafka. Components connect to the ESB using Connectors and publish/consume messages from the ESB or connect among them using peer-to peer connections using Open API, gRPC or any supported protocol.
Place in the Architecture	Enterprise Service Bus
Specifications/Features	Asynchronous communication using Kafka Messaging Peer to peer connections (Open API, gRPC, SQL/No, Streaming RTSP)
Relevant Deliverable(s)	e.g D4.3 Implementation of Local CPS-IoT RSM Features v2
Prerequisites	Strimzi, Zookeper

Current status	Version 1 ready,
How to Install	To be published into wiki
Why to use this component	The ESB is the main communication core component to allow access form services and applications to the data and functionality shared by other KER's and services

4.1.2 Transport services

Table 22: Transport services.

Transport services	
Introduction	Connectors have the duty to adapt protocols from KER's to the platform, such as connecting MQTT to Kafka, but also perform semantic translations to convert messages from one system to another. For example, convert messages used in MQTT robotic platform into canonical messages used in the ESB. Currently exist several ways to create connectors, using Apache NIFI flows and also using Apache Camel connectors or ready to be used Confluent Connectors, but also they can be created from scratch using Producer/Consumer/Stream API or Apache Kafka Connect
Place in the Architecture	Connectors, bridging KER and services to the ESB
Specifications/Features	Protocol connectivity Semantic translation
Prerequisites	Apache Kafka Connect Apache NIFI
Relevant Deliverable(s)	D4.3 Implementation of Local CPS-IoT RSM Features v2
Current status	There are currently lots of connectors[1] supported but semantic translation may be performed in the KER side. For example, MQTT connector are available but just transports data, no message translation is performed. Connectors shall be provided by each KER provider.
How to Install	To be updated in the Wiki.

[1] https://nifi.apache.org/docs.html "Processors" section

Why to use this component	Without Connector a KER cannot join the platform. The connector make KER and Platform independently connected; therefore, KER development is not tied to the Platform logic and vice versa

4.2 Resource management

From a functional point of view, the co-existence of a wealth of diverse resources in a common platform is very challenging, taking also into account that the platform should support future additional resources. One important step to address these challenges is to create abstractions of the resources that adhere to a common agreed structure. The diverse individual resources can then align with these abstractions, and high-level operations can be performed that are agnostic of the low-level details of each resource, and are only concerned with the common characteristics of the abstractions.

The resource management components, depicted in Figure 10, are responsible for creating and managing these resource abstractions. The core component of this category is the ODIN ontology, which holds the abstractions and structure for all kinds of information used in ODIN, including information about resources. The other components of this category are responsible for registering new resources, aligning them to the ODIN ontology, and providing the necessary descriptors that will allow their discovery by other resources and their use in high-level workflows. These components are detailed below.

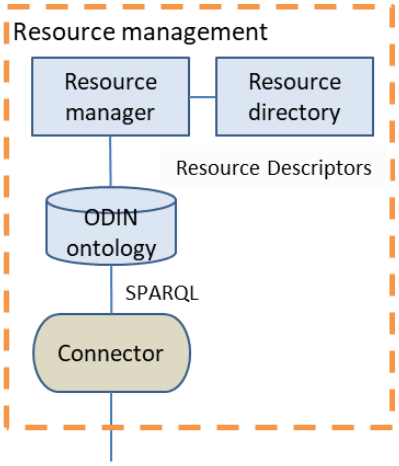

Figure 10: The resource management components.

4.2.1 ODIN ontology

Table 23: Odin ontology.

Odin ontology	
Introduction	A semantic ontology is a method for representing knowledge that is built on formal collections of terms. It is used to describe and portray a domain, or a specific area of interest, in a cohesive and unambiguous manner. The ODIN ontology is designed based on widely accepted models and provides a comprehensive data model for the ODIN open digital

	platform supporting services and Key Enabling Resources (KERs), which will be enhanced by the Internet of Things (IoT), Robotics, and Artificial Intelligence (AI) to empower workers, medical locations, logistics, and interactions with the territory.
Place in the Architecture	ODIN ontology is part of the resource management system, it a core component which regulates every data exchange in the ESB.
Specifications/Features	The Ontology provides data models for: • Physical space representation (like buildings, and rooms) • Organizational structures of the hospital • Hospital Logistics management • Clinical operations (like disease catalog, clinital processes, treatments, patient journey) • Robotic description, missions, and management • AI description • Dataset cataloguing • IoT sensing and actuation • Medical device cataloguing • Internal ODIN representation The ontology enables uses like: • KER uniform representation and directory • Standardised communications • KER discovery • Spatial reasoning • Logical reasoning • Automatic configuration Explainability (e.g. XAI)
Prerequisites	The component is only a model (OWL format), it does not require deployment. Resource Directory provides the main means for dynamic knowledge management (e.g. the current status of KERs).
Relevant Deliverable(s)	D3.3 Hospital Knowledge Base and ODIN semantic ontology v2
Current status	Version 2 is under development, continuous update and delivery is planned.

How to Install	The OWL file is used to understand the data model, through visualization such as Protégé, it is also used by the Fuseki component to perform inference.
Why to use this component	This component provides the basis for the datamodel used in the core components of ODIN. This means that the representation of KERs as well as all of the core messages will be mapable to the ontology classes. This allows the creation and interpretation of messages, as well as advanced semantic querying of the semantic statements with reasoning when needed.

4.2.2 Resource manager

Table 24: Resource manager.

Resource manager	
Introduction	The Resource Manager (RM) is the hub for resource registration, querying and update[1]. At the same time, it is the connection to the ODIN platform and the middle component between the ontology and the Resource Directory.
Place in the Architecture	The Resource Manager is in the upper layer (above the ESB) of the ODIN architecture.
Specifications/Features	The main functionalities of the Resource Manager are: - Ensuring that the resources been registered follow the structure defined by the Resource Descriptor. - Ensuring that the information contained follows the ontology. - Additionally, it is the focal point to consult about the available resources and extract the information stored in the Directory.
Prerequisites	It needs the ESB to be available to communicate with the KERs that want to register and get information. On the other hand, the ODIN ontology is necessary.
Relevant Deliverable(s)	D4.3 Implementation of Local CPS-IoT RSM Features v2
Current status	The technology used to deploy the Resource Manager is NiFi and more specifically NiFi flows. Additionally, the Resource Manager depends on the ontology to determine the structure of the Reosurce Directory. Currently an instance of NiFi is deployed, the flows are still to be defined depending on the KERs to be integrated.
How to Install	All details about NiFi and Fuseki deployments will be available in the Wiki
Why to use this component	The Resource Manger registers all the information about the KERs integrated in the ODIN platform such as: the information they provide, how to interact with them (protocols, topics, security measures, etc) in a unified way.

4.2.3 Resource descriptor

Table 25: Resource descriptor.

Resource descriptor	
Relevant Deliverable(s)	- D4.1 CPS-IoT Resource Management System Specification - D4.3 Implementation of Local CPS-IoT RSM Features v2
Introduction	The Resource Descriptor is the element that homogenises data description. It provides a common scheme to describe everything that integrates the ODIN platform in terms of the supported data types, formats, protocols, and structures.
Place in the Architecture	The Resource Descriptor is within the Resource Manager that is in the upper layer (above the ESB) of the ODIN architecture.
Specifications/Features	It describes how the data is organized. To have a structured digital directory to look for the ODIN platform's components[1] available. The standards that can be used are: Web of Things, AsyncAPI, OpenAPI and FHIR.
Prerequisites	It doesn't need any other components to be deployed.
Current status	The standards have been defined and the KERs can be registered using them.
How to Install	No installation necessary
Why to use this component	It provides with an standard structure of data description to be able to exchange it between different systems ensure it is legible regardless of the origin.

4.2.4 Resource Directory

Table 26: Resource Directory.

Resource Directory	
Relevant Deliverable(s)	- D3.3 Hospital Knowledge Base and ODIN semantic ontology v2. - D4.1 COS-IoT Resource Management System Specification. - D4.3 Implementation of Local CPS-IoT RSM Features v2.
Introduction	The Resource Directory is the repository of resources, where all the available Resource Descriptors are stored.

Place in the Architecture	The Resource Directory is within the Resource Manager that is in the upper layer (above the ESB) of the ODIN architecture.
Specifications/Features	The Resource Directory enables resource discovery, it allows resources to find information about other resources.
Prerequisites	The ODIN ontology is necessary and the Resource Descriptor and the Resource Manager technologies.
Current status	The technology used to implement it is Fuseki. An instance of Fuseki has been deployed and the latest version of the ontology has been imported.
How to Install	All details about Fuseki deployment are available at "INETUM components deployment" To be uploaded in wiki.
Why to use this component	It allows other resources or components understand which capabilities and resources the platform has. For example, if a moving robot enters a room, it could query the Resource Directory to understand which resources the room has, such as lights, equipment, or other resources.

4.3 Resource federation

Resource federation is one of the core objectives of the ODIN platform, allowing resources to be shared across ODIN instances, enabling thus the implementation of the platform in a decentralized manner. An ODIN instance, e.g. a hospital, can designate some of its resources as sharable, making them findable by other ODIN instances. The other ODIN instances can view these remote resources as if they were local and use them within their own operations and workflows.

An example could be a hospital possessing high-end computing power sharing its AI services with other hospitals which lack the relevant infrastructure, or a hospital collecting information from remote sensors in other hospitals, in order to better manage material exchange in cases of crisis.

The components involved in resource federation are depicted in Figure 11.

- The Resource Federation component handles permissions for sharing resources, after receiving resource sharing requests from the API gateway.
- The Transaction Handling component logs all transactions between remote resources to the ODIN Blockchain.
- The ODIN Blockchain is the place where all permission requests and transactions are stored in an immutable manner.
- The Local Blockchain Node is the (optional) blockchain node installed in the ODIN instance premises.

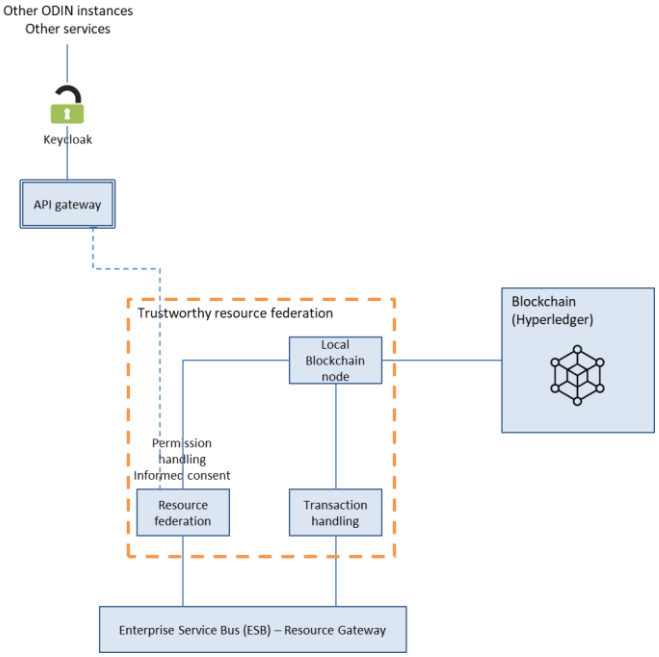

Figure 11: Components involved in resource federation.

In the following sub-sections, each of these components is described in more detail. A more detailed description of resource federation and the involved components can be found in D4.5 “Implementation of Advanced CPS-IoT RSM Features v1”.

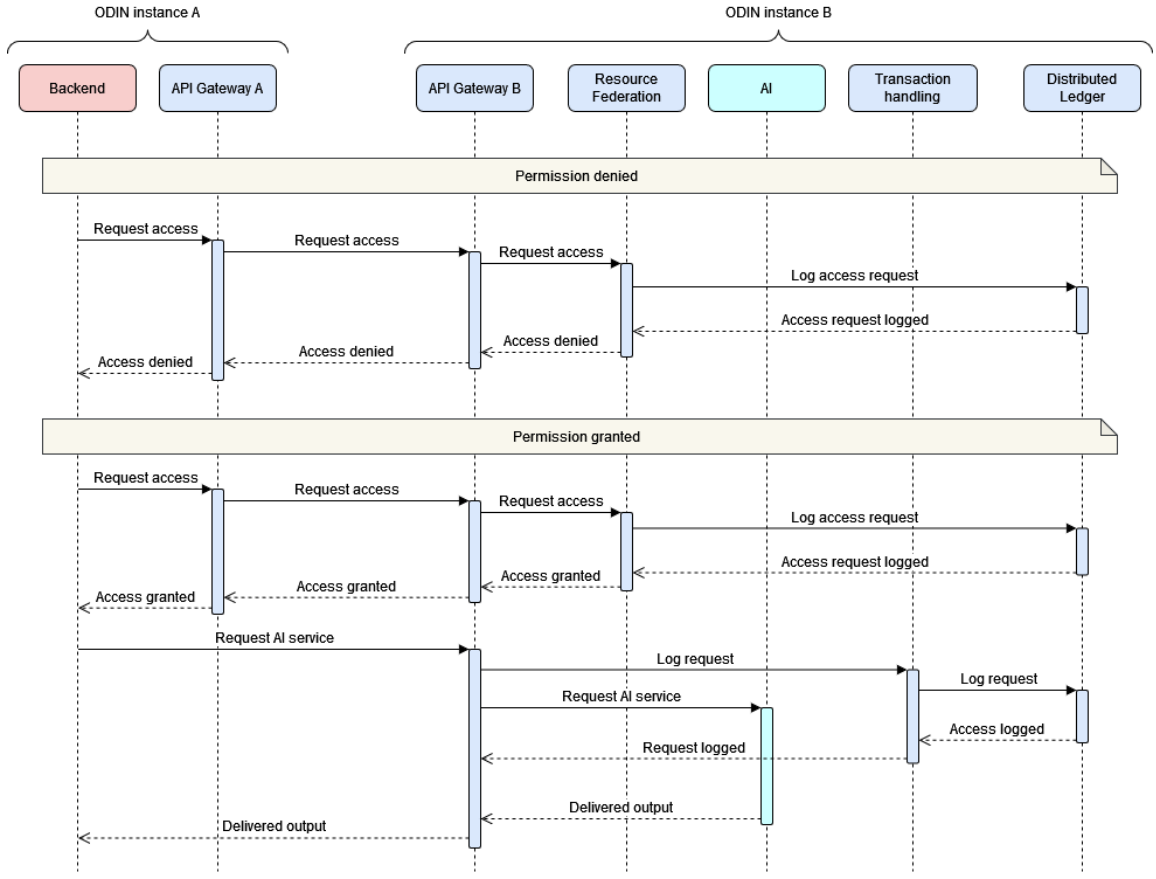

Figure 12: Resource federation and transaction auditing sequence diagram.

4.3.1 Resource federation

Table 27: Resource federation.

Resource federation	
Introduction	Brief introduction of the component
Place in the Architecture	Platform Services
Specifications/Features	DLT components in the ODIN platform provide the infrastructure necessary for secure, transparent, and efficient record-keeping, processing, and transfer of data and assets in a decentralized manner.
Prerequisites	No prerequisites
Relevant Deliverable(s)	e.g., D4.6 Implementation of Advanced CPS-IoT RSM Features v2
Current status	Version 1 ready
How to Install	This information will be available after the final release of the Resource Federation component
Why to use this component	By using DLT components such as consensus algorithms, data storage systems, cryptographic algorithms, network communication protocols, and smart contract execution engines, ODIN can create a decentralized network that enables secure and transparent record-keeping, processing, and transfer of data and assets.

4.4 Platform services

Building on top of the so far described infrastructure, higher-level platform services can be built that can send or use information to/from the resources and facilitate hospital operations. The ODIN platform in its current form provides two such services, namely metric collection and resource choreography, as shown in Figure 13. Both of these services are connected to the ESB in order to collect information from the resources or send instructions to them, providing high-level resource-related operations to the end-users. More high-level platform services may be added through open calls or in future extensions of the platform. The metric collection and resource choreography components are described in more detail below.

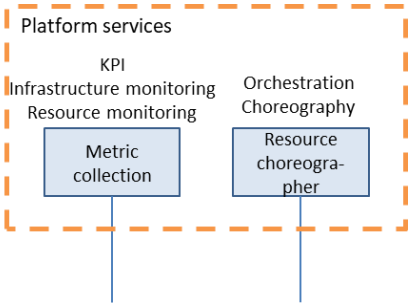

Figure 13: High-level platform services.

4.4.1 Metric collection

Table 28: Metric collection.

Metric collection	
Introduction	Metrics are produced by the different modules deployed in the ODIN platform. These metric sources need to be registered so that independently of the data injection method, the metrics are aggregated at a central point: the aggregator. Once aggregated, the metrics are stored, and they may also be post-processed to generate new metrics from them. The system also implements a reaction system that can display the metrics in real time in a dashboard to a user; or the system can process real time metrics to identify situations from the metrics which need to trigger notifications. Different notification methods may be used, depending on whether the reaction needs to be performed by a human or it can be handled automatically by a module (e.g. AI, choreographer).
Place in the Architecture	Platform Services, Metric Collection
Specifications/Features	• Register Metric producer • Pre-registered platform, and KER, metric producers • Metric collection, via pull or push (where needed) • Metric aggregation and processing • Metric storage • Threshold Triggering • Alerting Dashboard, visualization of metrics in real time
Prerequisites	The component is deployable on Kubernetes, metrics producers need to be registered; automatic metric producers are registered once they are detected.
Relevant Deliverable(s)	D4.6 Implementation of Advanced CPS-IoT RSM Features v2
Current status	Version 1, based on Prometheus and Grapahana are ready. V2 under development, better integration with ODIN ESB, automatic metric producer registration, default dashboards, and subjective metric management (questionnaires).
How to Install	The component is bundled in a helm chart, no parameters are needed.
Why to use this component	This component is useful to monitor the health of the platform, its individual components, KERs, and the system as a whole. The system is also capable of managing non-

	technical metrics, such as KPIs, allowing for continuous, real-time monitoring for decision making.

4.4.2 Resource choreographer

Table 29: Resource choreographer.

Resource choreographer	
Introduction	The Resource Choreographer, implemented using Kogito, can orchestrate Platform's resources and services to create new applications and cover new use cases. It can define workflows that define events, tasks and states to drive platform resources and accomplish new business applications
Place in the Architecture	Resource Choreographer
Specifications/Features	Define a workflow to control states, events, functions, define rules and logic Communication through events using Kafka Support for microservice orchestration using Open API, Async API, gRPC, Support for BMNP, DMN with visual editing.
Prerequisites	Kubernetes/Docker/Quarkus
Relevant Deliverable(s)	D4.6 Implementation of Advanced CPS-IoT RSM Features v2
Current status	Under development. Workflows can be created as needed and integrated to the platform.
How to Install	To be developed
Why to use this component	The Resource Orchestrator can define the logic to maintain new applications without modifying the underlaying microservices. Using visual editor can help non-technical staff, creating new applications.

4.5 Platform management

The platform management components are responsible for managing the ODIN platform itself and ensuring the proper operation of the other components of the ODIN platform. The components in this category, depicted in Figure 14, include the deployment manger, the access control manager, the platform configuration component, the platform documentation management, and the feedback collection components. These are described in detail next.

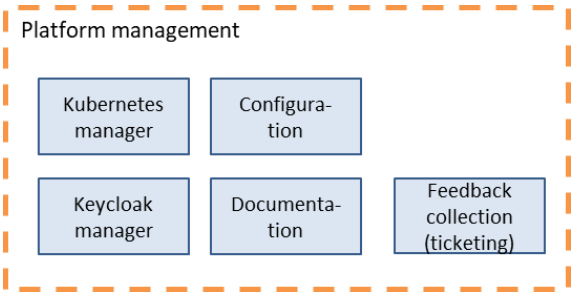

Figure 14: Platform management components.

4.5.1 Deployment manager (Kubernetes manager)

The Deployment Manager component is used to manage the overall ODIN platform deployment at a specific ODIN instance (e.g. a hospital). The ODIN platform is deployed as a set of microservices running on top of a Kubernetes cluster, which orchestrates them. Each ODIN component roughly corresponds to a microservice which is managed by Kubernetes. The Deployment Manager component allows the system administrator to start/stop and monitor the deployed services.

The Deployment Manager provides at least the following functionalities:

- Deploy (start) a microservice, i.e. an ODIN platform component, such as the ESB or the metric collection component
- Stop the execution of an ODIN component
- View the logs of a deployed ODIN component, for troubleshooting
- Manage the cluster resources allocated to the deployed microservices

Rancher is a container management platform which makes the adoption of Kubernetes simple to use and easy to scale. It is an open-source platform built on Kubernetes that disentangles and simplifies the management of Kubernetes by using a complete software stack. It aims to strength the cloud and DevOps in order to accelerate the digital transformation by bringing more products to the market. Rancher is especially useful in three kind of use cases.

- Multi-Cluster Management for Edge Deployments. Rancher enables DevOps teams to connect to hundreds of different clusters with the press of a button. Rancher makes it possible for the organizations to scale their multi-cluster management to edge location.
- Accessibility for Hybrid and Multi-Cloud Deployments. Multi-cloud architectures are made of applications that run on different clouds. This architecture is making the use of different cloud-strategies a necessity, thus making Kubernetes work in both large- and small-scale projects. Rancher is installable on any certified Kubernetes cluster and is being widely used by a vast amount of different enterprises and cloud-native companies.
- Centralized visibility for DevOps teams. By offering a single entry-point into Rancher and the Kubernetes ecosystem, Rancher enables developers to quickly deploy centralized applications while letting operators have full administrative control. Through Rancher, DevOps teams can access monitoring, alerting, logging, security scanning, service mesh and persistent storage. They can also provision, inspect, import and manage thousands of clusters on multi-cloud and edge scenarios.

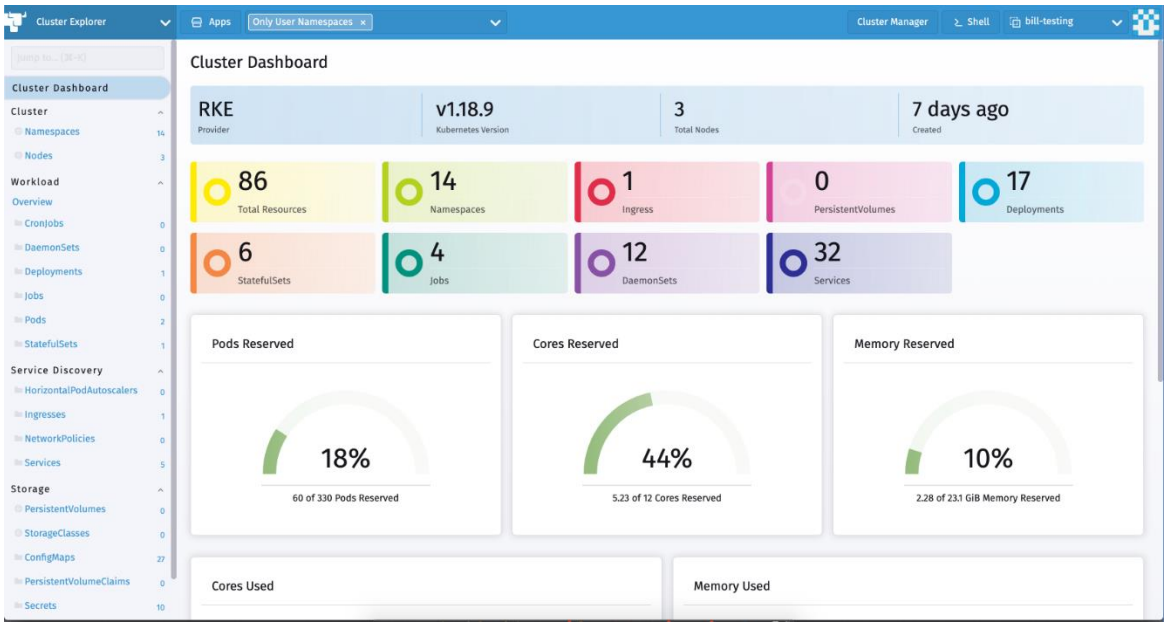

Figure 15: Screenshot of the Rancher dashboard.

Those kind of use cases are can be associated to the needs of the ODIN project. The ODIN platform will be built upon Kubernetes with the goal to be distributed to a lot of different Hospitals around the world. Therefore, a container management platform with the capabilities of Rancher will help to accelerate the management and ease the maintenance of different Platform clusters around the world.

4.5.2 Access control manager (Keycloak manager)

Table 30: Access control manager (Keycloak manager).

Access control manager (Keycloak manager)	
Introduction	It determines who has permission to access and use the available information and resources. Through authentication and authorization.
Place in the Architecture	The Access control manager is in the highest level of the architecture at the External Communication layer.
Specifications/Features	Access control identifies users by verifying various login credentials.
Prerequisites	It doesn't depend on other components. It needs a infrastructure in place to deploy Keycloack.
Relevant Deliverable(s)	D3.4 privacy Security and Trust report v2
Current status	There is a deployment of Keycloack in the testing infrastructures.
How to Install	All details about Keycloack deployment are available at "INETUM components deployment" (https://cbmlbox.ics.forth.gr/index.php/s/Bd6BfSWEnE7Stpq)

Why to use this component	The access control manager protects the information an resources within the ODIN platform from the usage of non-authorized personal. Once the user has been authenticated, the access control authorizes the appropriate access level and certain permitted actions associated with the user's credentials and IP address

4.5.3 Platform configuration

Table 31: Platform configuration.

Platform configuration	
Introduction	Many modules deployed over the ODIN platform will offer web interfaces for system administrators to manage the configuration of the said module. This could become unmanageable when the number of modules and respective interfaces increases. To address this, ODIN proposes a central administrator panel which is composed of links to each interactive resource administrative interface; the resource just needs to register its administrative interface(s) so that it appears in this panel making the overall system administration easier.
Place in the Architecture	Platform Management, configuration
Specifications/Features	Central Dashboard for accessing configuration utilities of platform, its components and registered KERs. Centralized service for maintaining configuration information, naming, and providing distributed synchronization.
Prerequisites	This component is one of the first components to be deployed, no pre-requisites
Current status	Version 1 under development, based on web bookmark manager such as homer[2].Version 2 will provide distributed configuration of microservices, based on popular solutions such as Apache Zookeeper[3], microconfig[4], or etcd[5].
How to Install	Helm chart

[2] https://github.com/bastienwirtz/homer

[3] https://zookeeper.apache.org/

[4] https://microconfig.io/

[5] https://etcd.io/

Why to use this component	Quickly access configuration darshboards from a central point of entry. Manage all the platform configuration centrally.

4.6 External communication

This group of components support the access of the ODIN platform from the end users or external applications. These components are the API gateway, which handles access from external applications, and the ODIN dashboard, which handles access from the end users. All communication passes through security mechanisms, which are described in 2.

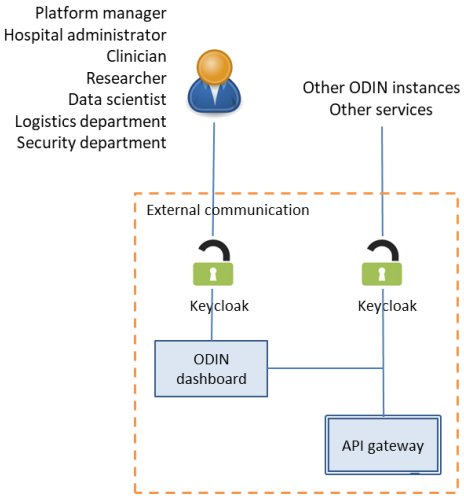

Figure 16: Components for external communication.

4.6.1 API gateway

Table 32: API Gateway.

API Gateway	
Introduction	The API Gateway opens the Platform and some of its functionalities to be accessed from the outer boundaries of the platform. It exposes resources to be consumed by external applications. In the last version TYK will be used as API Gateway
Place in the Architecture	API Gateway
Specifications/Features	Reverse Proxy API aggregation Authentication and authorization Service discovery integration Response caching

	Retry policies, circuit breaker, and QoS (Quality of Service) Rate limiting and throttling Load balancing Logging, tracing, correlation Headers, query strings, and claims transformation IP whitelisting. Automatic API publishing
Prerequisites	Tyk Gateway may need other components (https://tyk.io/docs/apim/open-source/)
Relevant Deliverable(s)	D4.3 Implementation of Local CPS-IoT RSM Features v2
Current status	Under development TyK is already available but its Connector to the platform is under development
How to Install	https://tyk.io/docs/tyk-oss/ce-kubernetes/
Why to use this component	The API gateway enables KER/Platform to expose functionality to external applications without comprising security, availability and reliability.

4.6.2 Administration dashboard

The ODIN administration dashboard provides the end-user interface to all ODIN components. While the API gateway provides a programming interface for external applications to communicate with the ODIN platform, the administration dashboard provides a graphical user interface (GUI) through which the end-users can recall and have access to the functionalities of the platform, such as the previously mentioned metric collection (4.4.1), resource choreographer (4.4.2), deployment manager (4.5.14.5.1), access control manager (4.5.2), platform configuration (4.5.3), platform documentation and feedback collection (6.4); as well as other GUI that KERs and other components may provide.

The end-users of the ODIN platform belong roughly to the following roles:

- Platform administrator
- Hospital administrator
- Clinician
- Hospital personnel
- Clinical researcher

Through the ODIN administration dashboard, each user role has access to different components of the ODIN platform. For example, a hospital administrator can have access to the Resource Choreographer to design new applications, the clinician can have access to monitoring interfaces of devices installed in patient's rooms, hospital personnel can have access to alerts produced by the KERs, and the platform administrator can have access to the deployment management components.

The ODIN administration dashboard collects the graphical user interfaces of all ODIN platform components in a common landing page. Most of the ODIN components will be accompanied by a GUI through which they can be managed by end-users. These interfaces will be accessible through the global administration dashboard, according to the roles' access rights.

Moreover, the ODIN administration dashboard will collect the interfaces of the KERs themselves. Certain kinds of KERs, such as front-end applications, IoT monitoring dashboards, etc., possess a GUI that allows an end-user, e.g. a clinician, to use them. During resource registration, a KER can register its GUI in the resource descriptors, in the same manner as it can register any exposable API, i.e. by specifying its address and possibly meta-data regarding the GUI's description and functionalities. The ODIN administration dashboard can then discover which resources have an accompanying GUI and provide access to it through the main landing page. Communication with the KER-specific GUIs always passes through the API gateway, which handles authorization and routing. These KER GUIs become dynamically available to the overall GUI as resources are registered/unregistered from the ODIN platform.

Finally, the ODIN administration dashboard will provide access to all available documentation about the specific ODIN instance. This includes documentation of the ODIN platform components, as well as documentation of each deployed KER, as described in 3.

Some representative screenshots of the ODIN platform dashboard are shown in Figures Figure 17-Figure 20. Figure (Figure 17) shows the landing page of the dashboard, where the deployment administrator can have an overview of the number, types and locations of the resources deployed. From the navigation pane at the left, the user can navigate to all main parts of the ODIN platform, such as resource management, choreography, the ODIN ontology, etc.

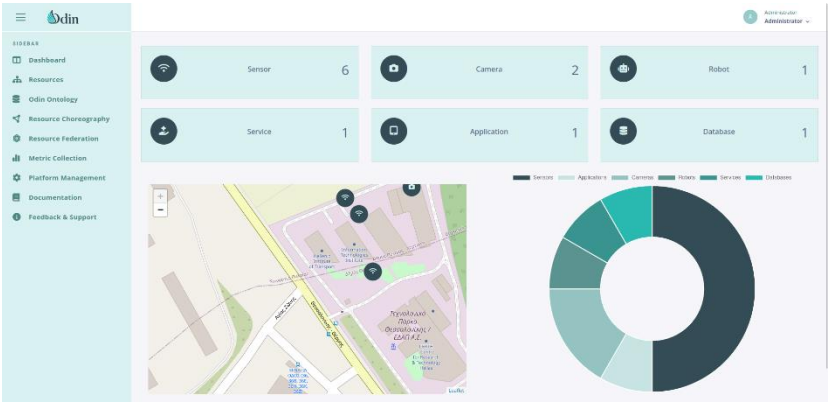

Figure 17: The landing page of the ODIN platform.

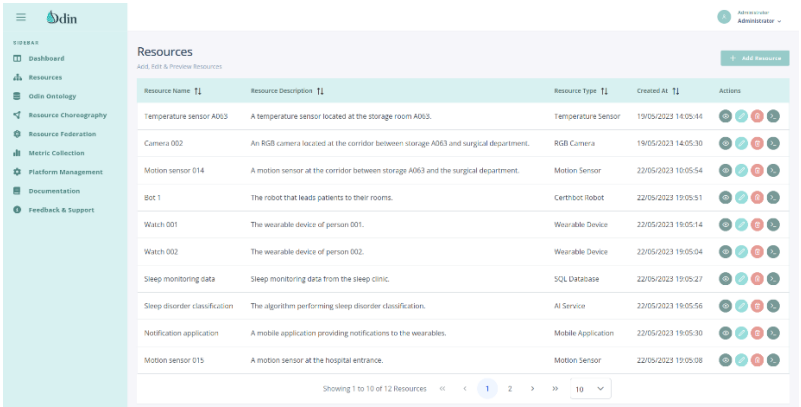

Figure 18: Resource management page.

As an example, clicking on the "Resources" link, the user can manage the registered resources, which are listed as shown in Figure (Figure 18). For each resource, its name, a short description and its type are listed for quick reference. The user can add a new resource by clicking on the "Add Resource" button, leading to a dialog as shown in Figure (add_resource.png, "Adding a new resource."). Here the user enters the basic information for a resource, including its name and description, as well as its type, which is a reference to the corresponding semantic class in the ODIN ontology. Depending on the resource class selected, different properties become available, as defined in the Resource Descriptor. The user can edit them as shown in Figure (edit_resource.png, "Updating the details of a Resource Descriptor."), updating the relevant information in the Resource Directory.

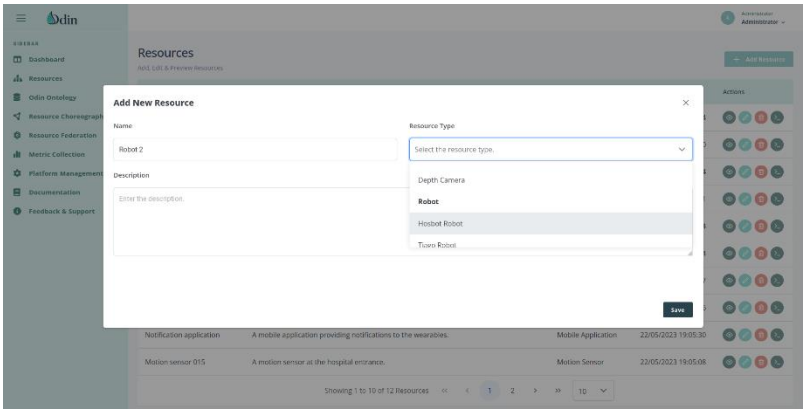

Figure 19: Adding a new resource.

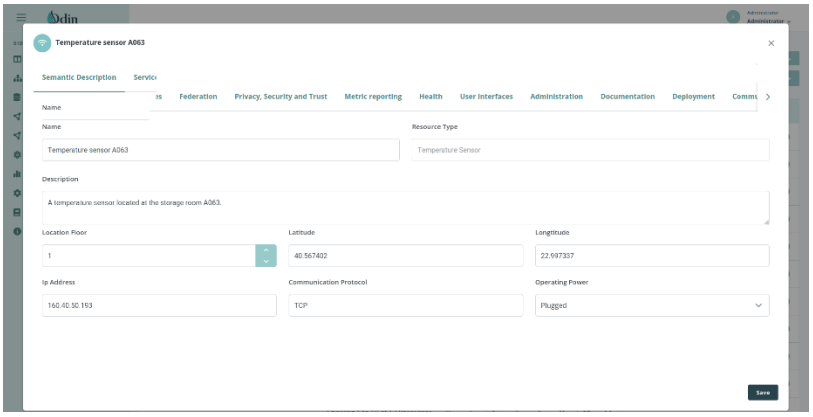

Figure 20: Updating the details of a Resource Descriptor.

5 Platform security

This section describes the tools and components that are used for the development and the security of the platform.

As already described in D3.5 “Privacy Security and Trust report v2”, the objective of security is to provide Confidentiality, Integrity and Availability:

- To provide Confidentiality at platform level, two main features will be implemented: encryption in transit and encryption at rest.
- To provide Integrity, a system of trust will be provided, ensuring both executables and data are unaltered.
- To provide Availability, the platform will rely on the underlying redundancy feature of container orchestration technologies such as Kubernetes.

This section offers an overview of the security analysis and countermeasures for ODIN platform v2. The full detailed analysis and implementation of these features, as well as design for advanced privacy, security and trust features are reported in D3.5 “Privacy Security and Trust report v2”.

Encryption in transit means that data that are in transit across the network needs to be confidential. This confidentiality can be granted by using standards for network security. Such mandatory standards in ODIN will be TLS[6], HTTPS[7] as well as IPSec[8] and VPN[9]. Encryption at rest refers to data that are physically stored in a database. Generally, encryption at rest is included in cloud infrastructure and in paid versions of most common database technologies. However, we can provide such features for the ODIN data layer at least for community editions of MySQL[10] and MongoDB[11] databases by also using community edition technologies such as Percona[12] for data encryption and Vault[13] for encryption key management.

Advanced trust features of the ODIN platform will be based on the Resource Descriptors (Section 0) that will include signatures of developers, promoters, as well as platform validators, to ensure the executables behind the resources as well as the data they produce and services they offer can be trusted; or at least the level of trust can be established so system administrators can gauge the risks. Leveraging access control (described in Section 4.5.2), blockchain services, and the

[6] Transport Layer Security, https://datatracker.ietf.org/doc/html/rfc8446, Last access March 2022

[7] HTTPS, https://datatracker.ietf.org/doc/html/rfc2818, Last access March 2022

[8] IPSec, https://datatracker.ietf.org/doc/html/rfc6071, Last access March 2022

[9] VPN, https://en.wikipedia.org/wiki/Virtual_private_network, Last access March 2022

[10] MySQL, https://www.mysql.com, Last access March 2022

[11] MongoDB, https://www.mongodb.com, Last access March 2022

[12] Percona Server MySQL and MongoDB, https://github.com/percona, Last access March 2022

[13] Vault, https://www.vaultproject.io, Last access March 2022

public key management schema, such as X509[14] and hash functions[15], trust can be maintained in the whole processing chain, expanding on integrity.

For availability, gateway offloading offering load balancing, ACLs, filters, quotas (rate limiting, thresholding, throttling), high-availability design, extra bandwidth (rate limiting, throttling), and service replication provided by orchestration technologies will address a high degree of availability of sensible services. Another security strategy that could be implemented is the lack of availability, in particular remote network availability should not be granted by default, ensuring modules are not able to just circumvent firewall policies from within. All external communications will be disabled by default, and only enabled by the module explicitly requesting access (either outgoing or incoming), which will then need to be approved by system administrators.

For a deep protection against Distributed Denial of Service (DDoS), the network security measures of the following subsections will be supported by the platform whenever possible and evaluated as necessary after an asset risk analysis (STRIDE, DREAD, etc.).

5.1 Beyond security

The security components of the platform are the cornerstone of two other important areas for ITC in hospital environments: Privacy and Trust. Although these have already been stated as part of the security mechanisms to be implemented in the ODIN platform, an expansive view has to be applied in order to properly address these issues in the future implementations of the ODIN platform.

Trust is mainly reliant on Blockchain technology, supporting federation of the platform services (see Section 4.3). The main purpose of this infrastructure is to build trust around the connections between instances, and in particular the trust of smart contracts by which different parties specify the conditions for access to different resources.

Privacy needs to address GDPR, as well as other regulations stipulating legal and ethical access or disclosure of certain aspects (such as data or services). Compliance with all these regulatory frameworks can quickly get out of hand, especially in heterogeneous and dynamic environments like the one the ODIN platform is managing. Thus, the platform will offer a simplification over the management and compliance with regulatory frameworks by offering blueprints for resources to comply. Each blueprint will define the required services for a particular regulation. For example, the GDPR blueprint will include services for users to exercise their rights to erasure or correction, or auditing services for ensuring encryption at rest. When a resource is compliant with a particular regulation, it will provide the services corresponding to its blueprint, enabling all the features the said regulation ensues. The trust of this compliance can be extended through the certification by an expert certification authority which will analyse the blueprint implementation, and if the audit is positive, the blueprint declaration will be signed, indicating to the platform and system administrators the increased level of trust this provides.

[14] X509 Public Key Infrastructure, https://datatracker.ietf.org/doc/html/rfc5280, Last access March 2022

[15] Hash Functions, https://csrc.nist.gov/projects/Hash-Functions, Last access March 2022

5.2 Execution environment v1

The following components and configurations were specified for version 1 of the ODIN platform to offer a minimal set of security features:

- Firewall configuration not allowing external communications (especially outgoing) outside the Kubernetes cluster. Already implemented in the current deployment.
- All public facing endpoints, i.e. all the gateway endpoints, will be protected with valid certificates, and HTTP Strict Transport Security (HSTS) will be enabled ensuring connections are forced to employ encrypted channels. Work in progress to obtain certificates from a valid authority and not self-signed.
- Network segmentation, i.e. configuring resource modules in their own network, which separates systems into subnets with unique security controls and protocols. Already implemented.

5.3 Secure DevOps v1

Before running any module, this module needs to be compiled and tested. The use of DevOps in ODIN development ensures this process is automatized, and thus amongst the automatic tests in this process, it is imperative that some security audit is performed. Container-based security will be provided with the integration of security-based pipelines into the DevOps systems. Tools like DefectDojo[16] will be used to check container vulnerability as well as protection against the OWASP top 10 vulnerabilities[17].

5.4 Intrusion Detection System (IDS) support

The living ODIN platform instances need to be constantly analysed and tested against unexpected data breaches and security violations. IDS will be used to provide these security features:

- Anti-virus and anti-malware software that detects and removes viruses and malware.
- Endpoint scrutiny that ensures network endpoints (desktops, laptops, mobile devices, etc.) do not become an entry point for malicious activity.
- Scans with web security tools that detect web-based threats, block abnormal traffic, and search for known attack signatures.
- Execution of tools that prevent spoofing by checking if traffic has a source address consistent with the origin addresses.

[16] DefectDojo, https://github.com/DefectDojo/django-DefectDojo, Last access March 2022

[17] OWASP top 10, https://owasp.org/www-project-top-ten/, Last access March 2022

6 DevOps infrastructure

This section describes the tools and components that are used for the development of the platform. It clarifies platform's security and the DevOps infrastructure as well as the ODIN ROS buildfarm.

The components described so far are the components that are deployed in a typical ODIN instance and provide the ODIN functionalities to the end-users. Besides the components comprising the ODIN platform itself, the ODIN DevOps infrastructure provides the architectural components necessary to achieve continuous development, integration and deployment of ODIN components in the ODIN instances. These components provide functionalities targeted to the developers of the ODIN components, rather than the end-users, so they are not directly visible to the end-users of the platform. However, they are essential to the developers to facilitate the development of new components and update to new versions, as well as to speed up the troubleshooting and issue fixing, when malfunctions arise.

The DevOps infrastructure used in ODIN consists of the following components, which have mostly been described in detail in D3.1 "Operational framework":

- Source code manager (D3.1, Section 2)
- Docker registry (D3.1, Section 5.3)
- ODIN ROS repository (not in D3.1, covered below)
- Pipeline orchestration server (Jenkins) (D3.1, Section 8)
- Kubernetes cluster (D3.1, Section 8, more information below)
- Platform Documentation (D3.8, table below)

Here, information is provided regarding the ODIN ROS repository, which was is not covered in D3.1, as well as some more information about the Kubernetes clusters that will be used for service deployment.

6.1 Docker Registry

GitLab, by default has its own Docker Registry. Thus, it will serve as ODIN's docker registry. A dedicated project will used as docker registry for all the images created by the developers in ODIN. Developers can also use their individual repositories as container registries for testing purposes and migrate their images to the main container registry afterwards. That simplifies the work for the developers because there is no need for different accounts in different domains. The container registry will be available in the above domain:

https://registry.gitlab.odin-smarthospitals.eu

I first instance of the container registry has already been implemented in the address: odin-gitlab.iti.gr:5050.

6.2 Kubernetes cluster

The base for the installation of a new ODIN Platform instance in a hospital environment will be Kubernetes. Kubernetes[18] is an open-source system for managing containerized applications across multiple hosts. It provides basic mechanisms for deployment, maintenance, and scaling for the ODIN applications.

The ODIN Kubernetes cluster will enable resource and platform components deployment and communication with each other. As seen in **Error! Reference source not found.**, it consists of a s et of nodes that run containerized ODIN applications. Every cluster has at least one working node.

The working nodes will host the Pods. A Pod represents a set of Docker containers in an ODIN cluster. The management of the worker nodes and Pods inside the cluster is performed by the control plane. The control plane is the container orchestration that exposes the API and interfaces to define, deploy and manage the lifecycle of containers.

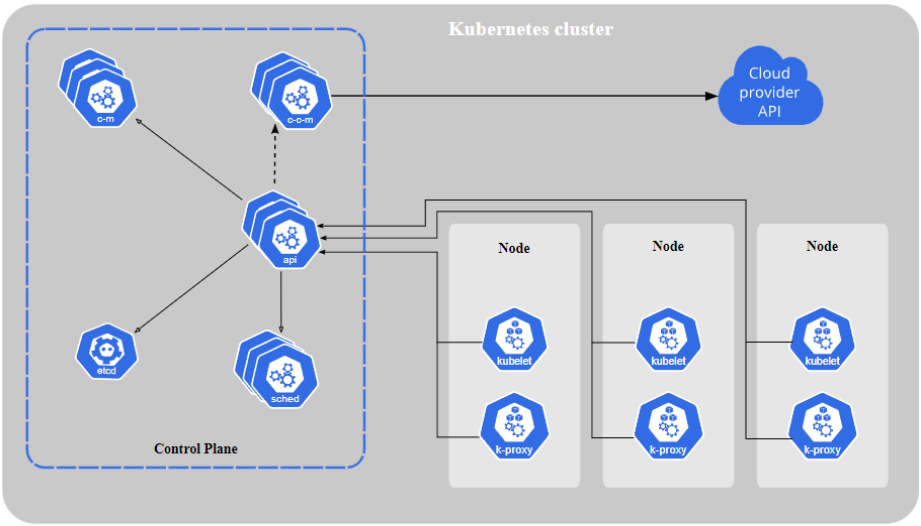

Figure 21: Kubernetes cluster[19].

In a new ODIN platform installation at a deployment site, Kubernetes will be the first component to be installed. As depicted in Figure 22 Kubernetes will act as a substrate on top of which all components of the ODIN platform will be deployed, drawing from their corresponding images available at the ODIN Docker registry. Kubernetes will facilitate the installation of components when they become available and the communication with each other.

[18] Kubernetes, https://kubernetes.io/

[19] Image from https://kubernetes.io/docs/concepts/overview/components/

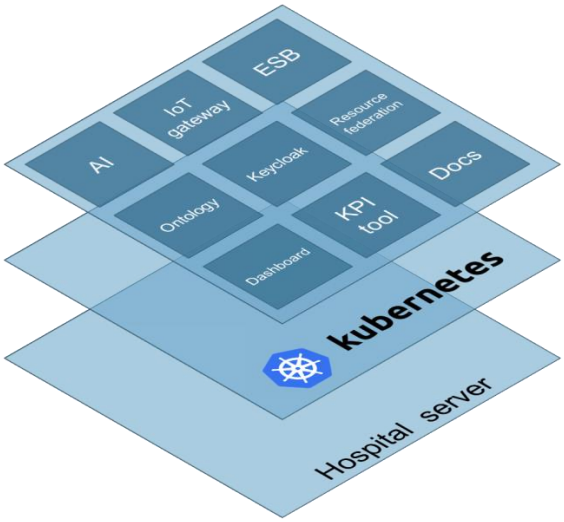

Figure 22: Kubernetes as a substrate for the deployment of ODIN platform components.

6.3 ODIN ROS buildfarm

To support the needs of robotics partners working on the ODIN platform, an ODIN ROS buildfarm has been created. This buildfarm is an important infrastructure that will support the ROS ecosystem of the ODIN platform. It provides features such as building of source and binary ROS packages, continuous integration, testing and analysis. The main coordination is done by a Jenkins instance and the goal is that most of the infrastructure is modular and reusable. The setup of the ODIN ROS buildfarm has been done in two steps, described below:

6.3.1 Provision machines (THL)

The first step involves the provisioning of the virtual machines. The ODIN ROS repository requires three different kinds of machines (shown in Figure 23):

- A Jenkins Master, orchestrating the execution of various jobs
- Multiple Jenkins slaves performing the actual builds
- A web server providing file hosting for end-result products

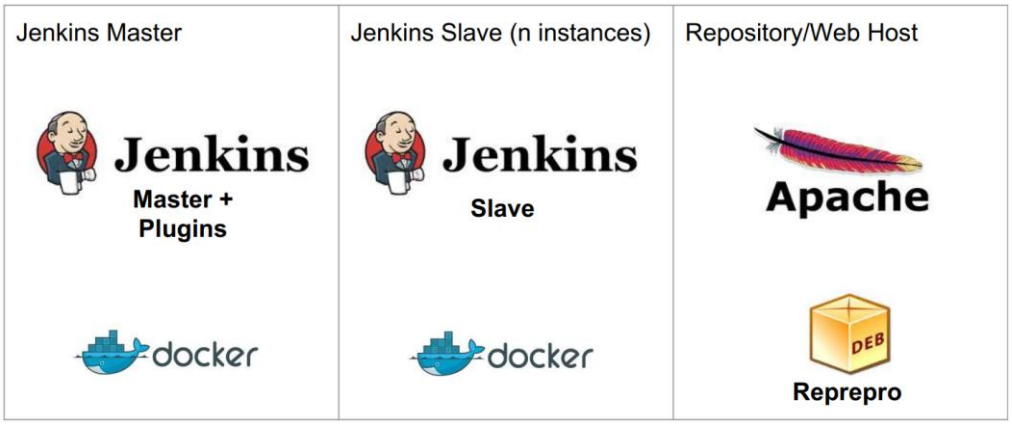

Figure 23: The ODIN ROS repository infrastructure.

At the beginning, there will be an empty Jenkins master, several slaves, and a repo which are waiting to be utilized. The slaves as well as the repository host will automatically keep their configuration up-to-date based on the configuration. The provisioned ODIN ROS repository will be able to run jobs for multiple ROS distributions.

6.3.2 Generation of Jenkins jobs (THL)

After the ODIN ROS repository is set up, the generation of Jenkins jobs starts for the ROS packages of the ODIN robotics partners. The job generation happens in two steps. First, a set of administrative jobs needs to be generated and then the Jenkins master will run special jobs which will generate the actual jobs performing all the builds. The result of running the job generation instructions will be a Jenkins Master configured with a set of jobs. Those jobs perform the desired build automatically. Whenever the ROS distro database changes, Jenkins will automatically reconfigure the jobs with new / updated / removed repositories and packages. Manual interaction will be only necessary to synchronize built packages into the main repository. For running the jobs, Docker containers are utilized since they offer isolation and provide faster performance than Virtual Machines (VMs).

The high-level view of the processes running on the ODIN ROS buildfarm is the following:

- Get the source code: e.g. `git clone`
- Install the required dependencies using `rosdep`
- Build a package: `CMake, make, make install`
- Run the tests: `make test`
- Create a Debian package: `apt-src build`
- Generate documentation with `rosdoc_lite`

6.3.3 Job Types (THL)

In more detail, in the ROS build farm there are running three important types of jobs:

- devel jobs which build and test ROS packages within a single repository.
- release jobs which generate binary packages.
- doc jobs which generate the API documentation of packages and extract information from the manifests.

6.3.4 Apt Repositories

The artifacts of the release jobs (the source and binary packages) are hosted in apt repositories. When the packages are newly built, they go to the “building” repo. This is automatically synced with the “testing” repo which can be accessed via the “testing” distributed mirror (aka ros-shadow-fixed mirror). In the official ROS apt repos, the “testing” repo is manually synced periodically (roughly every month) to the “main” repo which is accessed by the “main” mirror (this is used whenever there is a call with “`sudo apt install ros-*`”). The following figure illustrates the different apt repositories:

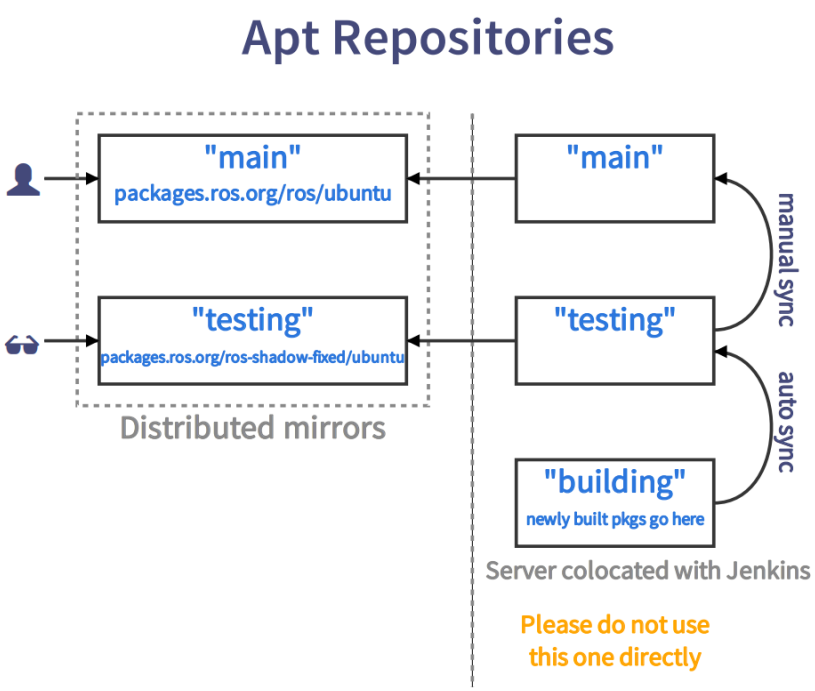

Figure 24: Official ROS Apt repos and their sync process.

The ODIN ROS buildfarm is following the official ROS repo logic and has also a "testing" and "main" apt repos, although the "main" is not foreseen to be synced often through the ODIN project.

Currently, the Jenkins Buildfarm along with the Apache WebServer is running in a Kubernetes cluster. To achieve this, some work was needed, related to i) the shared storage, ii) handling the docker commands related to host paths after finalizing the work on i), iii) synchronizing effectively different reprepro pods accessing the same (shared) volumes and iv) handling **private** repos through a user-friendly process. These changes are described in detail below.

6.4 Platform Documentation and feedback collection

The Platform Documentation component provides stakeholders with comprehensive documentation for the Platform and its KERs. It serves as a valuable resource, offering clear instructions and guidelines. The Feedback component allows users to provide input, report issues, and seek support.

Table 33: Platform documentation.

Platform documentation	
Introduction	The Platform Documentation component offers the documentation of the Platform and its KERs to the stakeholders.
Place in the Architecture	Documentation component + connector

Specifications/Features	Wiki edition and document publishing Integration of KER documentation during KER registering or update Other: https://js.wiki/
Prerequisites	A Kubernetes cluster PostgreSQL Database
Relevant Deliverable(s)	D3.8 Technical Support Plan and Operations v2 and D3.7 Technical Support Plan and Operations v1
Current status	Wiki is available but ESB connector is under development
How to Install	Wiki.js can be installed as standalone app in a server, using docker, Kubernetes or other ways https://docs.requarks.io/install/kubernetes. The installation of the connector is still pending.
Why to use this component	This component holds all the documentation of the platform and provides a way to maintain in a semi-automated way the documentation provided by the KER's.

Table 34: Feedback collection.

Feedback collection (ticketing)	
Introduction	The Feedback component allows to receive request and inputs from the users and open callers to attend bugs, problems and other matters related to support. Main component is supported by Trello.
Place in the Architecture	Feedback component
Specifications/Features	Brief list of the component's specs/features
Prerequisites	No installation for Trello is needed, just an account.
Relevant Deliverable(s)	D3.8 Technical Support Plan and Operations v2 and D3.7 Technical Support Plan and Operations v1
Current status	e.g., Version 1 ready, under development, etc.
How to Install	No installation required for Trello.
Why to use this component	This component offers control to the support team over the stakeholders' requests and problems.

7 Platform integration

Platform integration refers to the composition of all the already described components in the complete ODIN platform. To achieve this, each component needs to specify its API (Application Programming Interface), which is its exposed interface to other components of the system. Each component needs also to be developed and available in a format that can be easily connected to other components.

Within ODIN, system integration is facilitated by the adoption of a microservice approach. Each component of the ODIN platform is considered as an isolated service, implemented as a Docker image, which can be deployed in any system having an implementation of the Docker containerization facility. Moreover, each component is considered as a microservice, orchestrated by a Kubernetes platform, which lies at the bottom of the ODIN platform deployment (see 6.1). The Kubernetes substrate handles workload allocation and monitoring across the available processing power, and orchestrates the deployed components, according to composition instructions, in the form of Helm charts[20].

Under these considerations, integration of the ODIN platform consists of the following steps:

7.1 Component Distribution

As described in the previous paragraph every software component in ODIN will be distributed as dockerized application. Therefore, each component will be associated with its corresponding Docker image. A docker image is made of a set of instructions to create a docker container. Those instructions or commands are written in a file called Dockerfile. Inside a Dockerfile, the developer specifies all the necessary dependencies to run its software, thus making the produced images lightweight and fast. In order to run a docker container you need a Docker Image. In ODIN every Docker Image will be uploaded to the specified docker registry described in (3.10.1).

Since a lot of applications in ODIN will consist of more than one container, Helm Charts will used to deploy them. Helm Charts are YAML files that contain all the information needed to deploy docker containers in Kubernetes environments. Helm Charts' declarative nature is making the configuration of multi-container Kubernetes applications a straightforward procedure, easy to read, understand and reconfigure in different parts of the development cycle. After the creation of the helm chart, the developer can run multiple docker containers with just one command. A helm chart folder should be available in the GitLab repository of every component in ODIN.

7.2 API Specification

- Specification of each component's API, i.e. how its functionalities can be accessed by other components

[20] Helm, https://helm.sh/

Component	Resource manager		
Function name	Add resource		
Description	Adds a resource to the resource catalog.		
Inputs	Name	Type	Description
	resource name	string	The name of the resource to add.
	resource type	string	The type of the resource to add, taken from the ODIN data model.
Outputs	Name	Type	Description
	success	boolean	Whether the operation finished successfully.
	error	string	An error message in case of an unsuccessful operation.

7.2.1 Resource manager

Component	Resource Descriptor		
Function name	KERs registration based in the ontology definition.		
Description	Endpoint for KERs to register themselves in the platform via a Kafka topic. The information will be contained in a JSON document.		
Inputs	Name	Type	Description
	resource name	string	The name of the resource to add.
	resource type	string	The type of the resource to add, taken from the ODIN data model.
	resource ID	string	Unique identifier.
	resource functionality	string	The KERs functionality.
	Topic to register	string	The topic to which KERs send the registering information. Value = odin.core.resourcemanager.joined
	Standard used	string	WoT, FHIR, AsyncAPI or OpenAPI
	API Gateway	JSON	API specification
	Resource Federation	Boolean	Can the resource be used for Resource Federation
	Documentation	URL	Path to KERs' documentation

	File	JSON	The file with the full information about the KER following the defined standard
	Name	**Type**	**Description**
Outputs	success	boolean	Whether the operation finished successfully.KER was registered and saved in the Resource Directory.
	error	string	An error message in case of an unsuccessful operation.

7.2.2 Resource federation

Resource federation will have two components, access component and resource component, to handle the two types of requests. Bellow there are two tables, one for each component's API, that presents how each of them can be accessed by other components.

Component	Resource component		
Function name	Add resource		
Description	Adds a resource to the resource catalog.		
Inputs	**Name**	**Type**	**Description**
	sender	string	The sender's ID, whether this is a name, a company name, a department or a combination of the above.
	reciever	string	The reciever's ID, whether this is a name, a company name, a department or a combination of the above.
	resource	string	The name of the resource to add. Also, if the sender wants can add more information here(e.g. a description).
	timestamp	Timestamp	The time and date in timestamp format.
	Name	**Type**	**Description**
	success	boolean	Whether the operation finished successfully.

Component	Access component
Function name	Add access request
Description	Adds a request to the request catalog.

Inputs	Name	Type	Description
	sender	string	The sender's ID, whether this is a name, a company name, a department or a combination of the above.
	reciever	string	The reciever's ID, whether this is a name, a company name, a department or a combination of the above.
	request	string	The ID of the item that the sender is requesting to access(e.g. the name of an AI model). Also, if the sender wants can add more information here(e.g. a description).
	timestamp	Timestamp	The time and date in timestamp format.
Outputs	**Name**	**Type**	**Description**
	success	boolean	Whether the operation finished successfully.
	error	string	An error message in case of an unsuccessful operation.

7.2.3 Metric collection

Check the Prometheus[21] and Grafana[22] interfaces.

7.2.4 Resource choreographer

Resource Choreographer will forward orders to other components that will be listening to its messages published in the Resource Choreographer control topic.

Component	Resource Choreographer
Function name	Process data
Description	Endpoint for KERs must listen to get instructions from the Resource Choreographer topic odin.core.resourcechoreographer.control. Information will be shared in JSON format with Structured CloudEvent specification[23]. Inputs will define data content

[21] https://prometheus.io/docs/prometheus/latest/querying/api/

[22] https://grafana.com/docs/grafana/latest/developers/http_api/

[23] https://github.com/cloudevents/spec/blob/main/cloudevents/bindings/kafka-protocol-binding.md

Inputs	Name	Type	Description
	resource ID	string	The ID of the resource to receive the instruction.
	function type	string	The function that must be performed over the data. Any of the functions published by the KER
	Input data	string	Source to get input data
	Output data	string	Output where to send result data
	Error topic	string	Topic where to send message in case of error
	ProcessID	String	ID of the process involved with this function
Outputs	**Name**	**Type**	**Description**
	Output data	-	Depends on the input data and the function
	error	string	An error message in case of an unsuccessful operation.

Component	Resource Choreographer		
Function name	Disconnect		
Description	Function to disconnect a KER from the platform. Published into topic odin.core.resourcechoreographer.control		
Inputs	Name	Type	Description
	resource ID	string	The ID of the resource to receive the instruction.
	function type	string	Disconnection function
Outputs	Name	Type	Description
	error	string	An error message in case of an unsuccessful operation.

Component	Resource Choreographer
Function name	Perform Evacuation plan. Published into topic odin.core.resourcechoreographer.control

Description	Function to perform an evacuation plan depending on the type of disaster		
Inputs	Name	Type	Description
	resource name	string	All
	Evacuation type	string	Type of evacuation to perform
Outputs	Name	Type	Description
	error	string	An error message in case of an unsuccessful operation.

Component	Resource Choreographer		
Function name	New KER registered/updated, KER disconnected or kicked.		
Description	The RC will listen for new KER's joining the platform in the topic odin.core.resourcemanager.control		
Inputs	Name	Type	Description
	resource ID	string	The ID of the resource registered.
	function type	string	New Ker registered/disconnected/kicked/updated
	data	string	KER's Resource Descriptor
Outputs	Name	Type	Description
	Output data	-	Depends on the input data and the function
	error	string	An error message in case of an unsuccessful operation.

7.2.5 Deployment manager (Kubernetes manager)

As described in (4.5.1) Rancher is used as Kubernetes manager. Rancher is a third-party application that provides API specification in their official site[24]

[24] https://ranchermanager.docs.rancher.com/v2.5/pages-for-subheaders/about-the-api, last accessed, Jan 2023

7.2.6 Access control manager (Keycloak manager)

Keycloak gives authorization and authentication mechanisms based in standards such as OpenIDConnect/OAuth2.0 or SAML2.0.[25]

7.2.7 API gateway

The API Gateway will expose externally the KER's and ODIN functionalities. That API will depend on the exposed functionalities and KER's so will not be specified here. On the other hand, the API Gateway will listen to changes in the platform, such as new KER's exposing their API, updating them or terminating it's exposition to external clients.

Component	API Gateway		
Function name	Publish API/Update API/ unpublish API		
Description	The API Gateway listens into odin.core.resourcemanager.control to be aware of new KER's publishing and updating the API exposed.		
Inputs	Name	Type	Description
	resource ID	string	The resource's ID publishing or updating documentation
	function type	string	New Ker registered/disconnected/kicked/updated
	Input data	string	Optional KER's Resource Descriptor
Outputs	Name	Type	Description
	Output data	-	Depends on the input data and the function
	error	string	An error message in case of an unsuccessful operation.

7.3 Component Composition

As already mentioned in section 6.2 every new instance of the ODIN platform in a hospital environment will be installed upon Kubernetes. Therefore, a new Kubernetes cluster will be generated. To collect and install all the different components in the cluster Helm[26] charts will be used. In that way, Helm simplifies the configuration and the installation of different components in the cluster.

[25] https://www.keycloak.org/docs/latest/securing_apps/index.html

[26] Helm, https://helm.sh/docs/, last accessed Jan 2023

To deploy an application in Kubernetes you need to use multiple YAML files such as pods, services, deployments etc. A helm chart collects all those different files and makes the installation procedure effortless, since if everything is in place, just one command can do the job. Moreover, to modify helm charts without having to modify every single one YAML file in the helm chart the developer can just use a single file called values.yaml. In this file the developer has the option to adjust the values of the different files to achieve the maximum optimization.

Even though the simplest way to deploy Helm is to create a single chart that holds the entire application, there is also the option to use a chart with dependencies to other charts. This is called the umbrella chart. By using this approach, multiple charts can be deployed with the use of the umbrella chart. In Figure 25 we can see the basic structure of the files inside a helm chart.

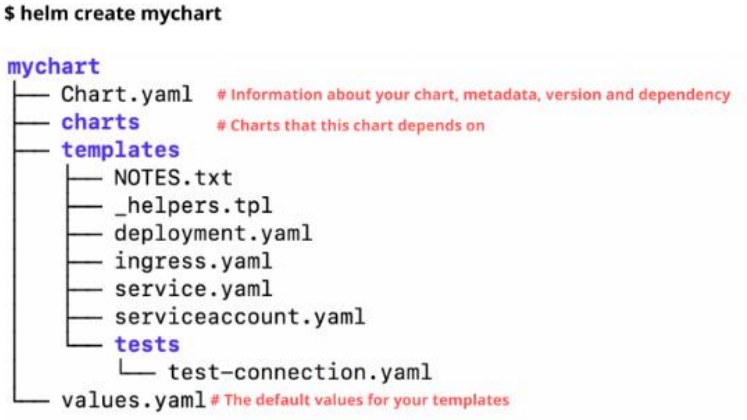

Figure 25: Simple helm chart file structure[27].

Components in ODIN will be deployed through helm charts. Every component will be packaged using its own helm chart. Umbrella charts can be used to package one or more components together. The ODIN platform in its latest version will be installed using Helm. During the third year of the project most of the components of the platform have a working version ready, the focus will turn through the packaging of each component with helm to enable integration between different components.

[27] Image from https://razorops.com/blog/introduction-to-helm-3-the-package-manager-for-kubernetes/

8 Platform deployment

This section covers issues regarding the deployment of the ODIN platform at the actual end sites envisioned in ODIN.

8.1 Execution environment

In this section, we delve into the execution environment of the ODIN platform. We examine the interconnectivity between various ODIN instances using Virtual Private Network (VPN), explore the scenarios where private cloud Virtual Machines (VMs) are employed, analyse the characteristics of the Platform's cloud instance, and shed light on the configuration of the testing instance that has been implemented to facilitate the development process.

8.1.1 VPN connection

This section describes issues related to VPN connection between ODIN instances, which can provide an extra level of security within the ODIN ecosystem. The configuration of a VPN among the hospitals participating in ODIN provides a secure channel for communication across instances, safeguarding ODIN against unauthorized use. It should be noted that this level of security, being part of the Privacy, Security and Trust facilities, acts complementary to resource federation (Section 4.3) in establishing trust among ODIN instances: the former provides secure channels of communication, while the latter provides auditability, traceability, and non-repudiation.

8.1.2 Private cloud VMs

In case a hospital does not have enough processing power to host a complete ODIN instance, private cloud VMs will be used to remotely host the ODIN platform or parts of it. As described in 6.2 the base of an ODIN platform instance will be a Kubernetes cluster. Therefore, the same kind of infrastructure must be provided in the private cloud for a homogeneous operation. To accomplish that goal several VMs would need to be provisioned to create a functional Kubernetes cluster. The number of VMs and the resources allocated for each VM will vary depending on the hospital needs and the workloads that will run in the cluster.

A minimum Kubernetes cluster would include:

- A master node. This is where the backplane and etcd services will run. No workloads are recommended to be run in this type of nodes. Optionally another node can be provisioned to provide high availability.
- Two worker nodes. This type of nodes is where the workloads are executed. Most likely more than just two nodes will be necessary for the ODIN platform. Adding more nodes to a running cluster is feasible without much hassle.

Additionally, it is also recommendable to have a management platform for the cluster such as Rancher[28]. That would increase the number of necessary VMs.

[28] Rancher, https://rancher.com/

Nevertheless, where regulations permit the use of public cloud services, it is an option to be considered, although traditionally hospital managers have been very reluctant to deploy IT services in the public cloud.

Most major cloud infrastructure providers offer a managed Kubernetes cluster service (EKS[29], AKS[30], GKE[31], OKE[32]). This can be a very interesting option that offers hospitals flexibility if they don't have the resources or the staff to deploy and manage the cluster. In addition, these providers also offer the option to connect to their services via a site-to-site VPN or use a direct connection (AWS Direct Connect[33], Azure ExpressRoute[34], Google Cloud Interconnect[35], Oracle Cloud Infrastructure FastConnect[36]) which makes connectivity a lot faster than through a VPN.

8.1.3 ODIN cloud instance

Apart from the hospital and residence ODIN instances, which will be deployed in actual locations, there will be also ODIN instances deployed in public cloud VMs, hosting public resources to be used by other ODIN instances, such as AI, back-end services and data storage.

Because of this particularity, the fact that the instance is executed in public cloud, most hospitals will be hesitant to employ this deployment option. However, in the hospital of the future, where digital services and appliances will be the forefront of management tasks, the availability and scalability of the infrastructure for executing the platform may outweigh any privacy and trust concern, especially if these are addressed both by the cloud provider and the platform. Thus, we expect the deployment option of public cloud to increase over time, and being mindful of this option for the ODIN platform ensures this future use.

Initially public cloud instances will offer supporting services, hosting less privacy-critical resources (i.e. HIS) or location specific resources (i.e. IoT, Robotic systems, infrastructure databases), in favour of externalizable resources. One clear example is the case of AI, where the need for specialized infrastructure may be required. In this case, the hospital may externalize the provision of AI services and applications to a company which offers state-of-the-art hardware and maintenance, as well as upgrades to the latest technologies when available; services which the hospital may not have the expertise or the funding to implement. Through the ODIN platform, only the infrastructure needs to be externalized, as the resources would be managed transparently through the federation features of the platform.

Another example for public cloud deployed instances, at least in the short term, is to support instances where the focus would be to offer public resources. Some examples of these resources may be anonymized datasets or non-sensitive services (e.g. unit conversions, market valuations).

[29] Elastic Kubernetes Service, https://aws.amazon.com/eks/

[30] Azure Kubernetes Service, https://azure.microsoft.com/en-us/services/kubernetes-service/

[31] Google Kubernetes Engine, https://cloud.google.com/kubernetes-engine

[32] Oracle Kubernetes Engine, https://www.oracle.com/es/cloud-native/container-engine-kubernetes/

[33] https://aws.amazon.com/directconnect/

[34] https://azure.microsoft.com/en-us/services/expressroute/

[35] https://cloud.google.com/hybrid-connectivity

[36] https://www.oracle.com/cloud/networking/fastconnect/

Within this scope, there may be interest to offer more health-related resources; in particular, national or regional agencies may want to offer public KERs to the hospitals within their jurisdiction. For example, medication agencies may offer eLeaflet information, learning material and dose calculators for hospitals to access and incorporate into their own workflows.

With increasing trust in public cloud providers, maybe with the emergence of cloud providers specialized in healthcare services, we might expect hospital operations to progressively migrate to public cloud deployments. However current hospitals with low resources, such as those in third world countries, war zones, under-populated regions or campaign/temporal hospitals, could view the public cloud deployment of ODIN platform as the most affordable, and maybe only, option in the short term. Regardless, the management of this option should not be more complex than other options thanks to the standardization of resource management and federative features of ODIN.

8.1.4 ODIN testing instance

With a view to testing the integration of the different components of the platform, a specific ODIN instance will be provided in the cloud. The different development teams will have access to this instance, although such access will not be publicly open, but it will require a specific VPN connection. This ODIN testing instance will resemble an instance deployed in hospital premises. It will be available during the course of the project, and it will be decommissioned at the conclusion.

8.2 Current status

At the moment of this deliverable, there is one instance deployed in the cloud, for testing purposes. It follows the architecture shown in figure 25. This testing instance has been deployed in a way that resembles hospital premises with a highly available architecture.

We can see that the server's structure is replicated within three availability zones. The availability zones represent three different data centres of the cloud provider, with individual electric supplies and geographically separated. This way, we can ensure the ODIN's platform availability if any of those data centers would go down.

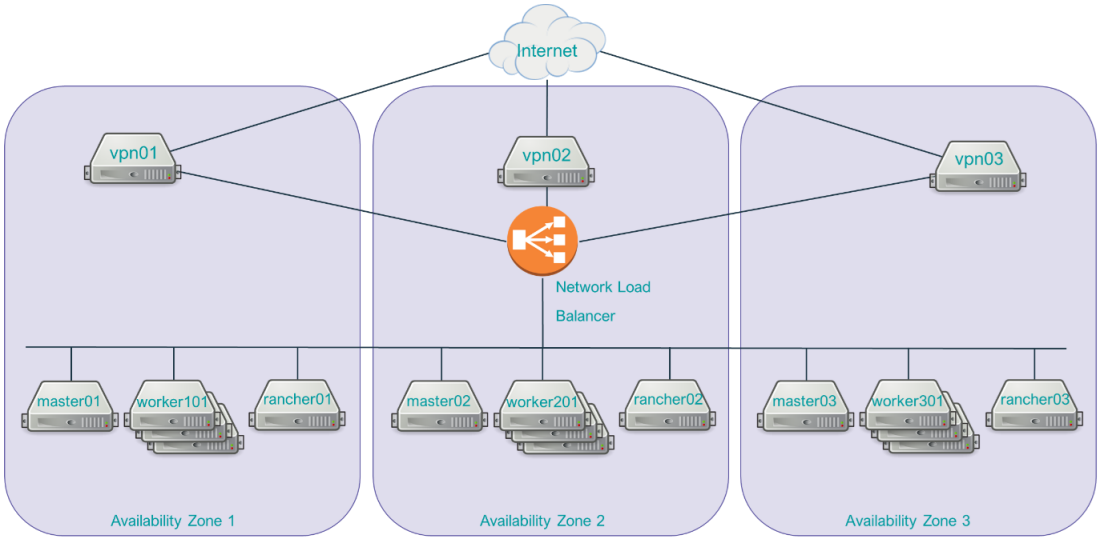

Figure 25: Cloud instance architecture.

Although cloud services have been used, there is no direct access to the platform from the Internet. To access the testing instance a VPN connection, through one of the vpn servers, is necessary. These vpn servers are placed on the public subnet, just for services exposed on the Internet. This security mechanism has been implemented using OpenVPN. This will be the public network.

In addition, we have the private network that is divided in two other subnets: Front-End, a private segment where platform services will be published, by means of a load balancer, to users connected through the VPN. And, finally, Back-End, also a private segment with servers and other internal services, where the Kubernetes cluster is placed. This cluster will run the different components of the ODIN platform. The Kubernetes distribution that has been chosen is RKE2, that focuses on security and compliance.

None of these subnets has direct access to the Internet. If any component deployed on the Back-End subnet needs to access an external resource on the Internet, an HTTP proxy and a Socks proxy have been deployed with a whitelist of just the specific resources that are strictly necessary. Both proxies are deployed on the public subnet with Internet access. The communications between the different subnets are controlled with firewall rules, that allow only the essential traffic that is strictly necessary.

The central piece of the platform is the Enterprise Service Bus provided by Kafka. To deploy it on Kubernetes the Strimzi operator has been used, that creates a secure Kafka cluster by default.

Finally, Kubernetes is managed with Rancher that has been deployed on an independent cluster using the K3S distribution.

8.3 Deployment requirements

Although it is still early to define all the necessary requirements to deploy the platform in a production environment because a lot of the components are still in the development phase, a first approximation of the requirements will divide the platform in two different versions: the Light version, and the Full version. This could be used as a reference for the pilots in order to acquire all the necessary equipment to install the platform on their side.

The **Light version** of the platform, is used to define the system requirements to deploy basic components of the platform (e.g., ESB, resource management, specific KER connectors, operating dashboard, security mechanisms) in the pilot's sites, that can support the needs of the RUCs, without including all advanced functionalities of the platform, or without supporting a very large number of simultaneously connected KERs. In this case the light version system requirements for a high availability deployment on a 3-node Kubernetes cluster are the following:

- 2 CPU cores
- 24 GB RAM
- 1 TB disk

As mentioned in (8.1) cloud support will be provided when needed, in order to extend these requirements, if a hospital cannot offer higher processing power availability.

The **Full Version** of the platform consists of the complete set of components, (e.g., including semantic interoperability, resource choreographer, resource federation, KPI collection, etc.). In addition, the requirements of the full version of the platform should take into account the possibility of a large number of diverse KERs being interconnected. A different Kubernetes setup is deemed appropriate for this version, with several nodes available, in order to ensure the scalability of the solution. The recommended deployment requirements for the full version are the following:

- 3 master nodes:
    - 2 CPU cores
    - 8 GB RAM
    - 20 GB disk
- 3 worker nodes:
    - 4 CPU cores
    - 16 GB RAM
    - 20 GB disk
- For the working nodes that require GPUs
    - 1 Nvidia Tesla T4 GPU with 16 GB dedicated GPU RAM
    - 4 CPU cores
    - 16 GB RAM
    - 120 GB disk

In all the above, the CPUs are considered 64-bit. A Kubernetes node can be installed on a physical or a virtual machine.

9 Platform verification and validation

Part of the objectives of deliverables D3.10-D3.12, i.e. this and the next versions of this deliverable, is to include results regarding the validation of the platform. Validation of the ODIN platform is important to ensure that the developed system fulfils the specified requirements and is of high quality. This series of deliverables deals with verification and validation of the platform up to the system testing phase. The validation testing phase will be conducted through user acceptance tests and KPI reporting during the pilot evaluation, which is the focus of WP7.

The approach to follow for system testing has been detailed in D3.1 “Operational framework”, including unit testing and integration testing. At the current stage of the project, component development and definition of unit tests is ongoing. Therefore, the unit tests and integration/system testing scenarios are not yet specified, nor are any validation results available. These will be reported in future versions of this deliverable. In the current version, details regarding the procedure to follow during unit and integration testing are provided, which are not covered in D3.1.

There are several systems designed to help developers in writing unit tests in a variety of programming languages, as well as in executing tests and collecting the results. These have been reported in D3.1 “Operational framework”, Section 4.1, and the developers of the ODIN components are free to select the ones that best fit their workflow. The unit tests created by ODIN developers throughout the course of the project will be gathered and reported in the future versions of this deliverable, so that they are accessible by all technological partners.

9.1 Testing framework

The testing framework defines the key performance indicators (KPIs) that will be used in order to evaluate the ODIN platform and its individual components. It will also define the tools and the procedures that will be used for assessment in the ODIN project. It clarifies what will be tested, by whom and what should be the expected output. In ODIN the validation of each component is been interpreted as the conformity between the components and Platform to the user requirements. The concepts that will be used from the framework will be the following:

- Requirements
- Specification
- Unit Test, Verification Test, Validation Test
- Non-regression assessment
- Verification (compliance to specification)
- Validation (compliance to requirements.)
- Stakeholders: specifiers, developers, end users, commissioners
- Integration: At interface (pre and post conditions, parameters) and in data (data model)

9.2 Phases and tests types

In the ODIN projects there will be five main categories of Tests Unit, functional, Integration, Acceptances and Performance tests.

9.2.1 Unit Tests

Unit testing is an automated procedure that ensures the component's functionality. The tests will be specified by the developers of each component. The developers then can run those tests in each updated version of their components, so they will be in accordance with the DevOps methodology.

Unit testing is defined as a test executed by developers in a laboratory environment in order to present the program's ability to meet the requirements set in the design of the specification. This kind of testing can be run on individual units or to bundle of individual units. During this testing the focus is set on the unit, rather than the general system.

In order to run unit tests in ODIN, it is important to determine what we understand as a unit in the ODIN platform. ODIN platform as defined in chapter 2 uses a bus-oriented microservice architecture where each component is responsible for a single task. The architectural design has divided the work in four different layers each one consists of different KER's HIS's and a bundle of individual components. These components can be databases, APIs, registries and services in general. The aforementioned components can be characterized as units and unit testing will be applied to the classes, methods and internal interface of those units.

In ODIN "unit" is defined as the smallest testable part of each developed component. Unit testing should be an automated procedure written by using a unit test framework, in accordance with the programming language that each developing team is making use of. (e.g., unittest for Python.).

Unit test is taking place before (or in parallel with) uploading code into repositories in order to make sure that each unit has the appropriate behaviour. The components that have passed all the test should be the only components to be committed. Especially for the major methods, functions, classes(units), internal APIs of each component a single test should be implemented. In that way, the above tests will be the first thing to run when calling the main program, in order to ensure that there are not unexpected errors

Some simple unit testing guidelines are mentioned below:

- Unit test means that each test tests exactly one thing.
- Each test method is a test.
- Each test method should have on assert (Pass/Fail).
- Input and output data should be deterministic.
- Failed tests should have clear and unique error messages.

In order to further improve the software quality a test for each bug that occurred in the developing process is recemented with a bug statement number to automate bug tracking and accelerate bug fixing.

In conclusion, unit test covers a single functionality of each unit. Units with more than one functionality should consider matching those functionalities with tests. The accepted components will be the components which have passed all their unit tests.

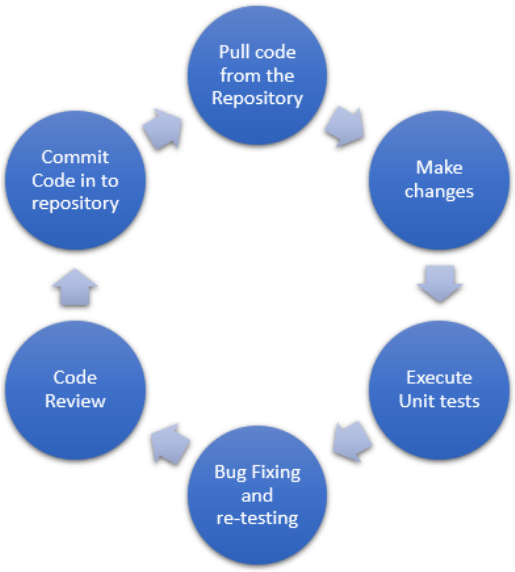

Figure 26: Unit Testing Lifecycle.

9.2.2 Functional Tests

Functional Testing is another method that can be used in order to guarantee the system's functionality. In this testing phase the system, an appropriate input will be given to the system in order to evaluate if the expected requirements are fulfilled.

In general, functional testing is a black box method that does not focus on the platforms source code. This phase can be used in order to determine the clearly the functions that need to be performed by the platform.

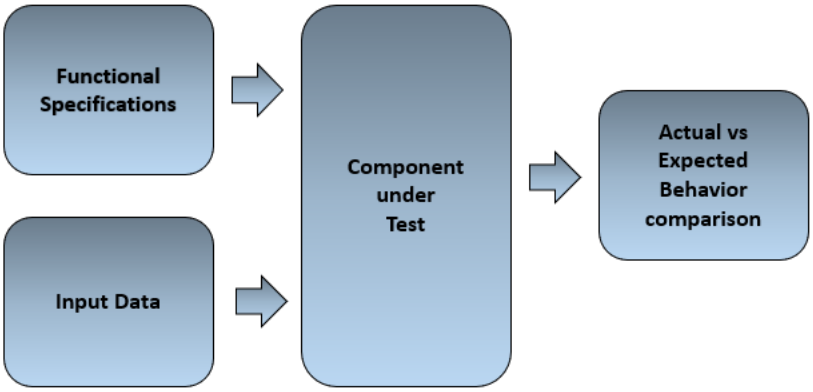

Figure 27: Flow of functional Tests.

9.2.3 Integration Tests

This testing phase is focusing on the component's interaction with each other in order to provide higher-level functionality, in opposite to unit testing that is used for individual components. As a consequence, integration testing is about runtime factors such as compatibility between interfaces, services and dependency resolution. In a nutshell, integration tests are used to evaluate the collaboration of multiple units.

In ODIN integration can be easily defined by simply following the architecture of the ODIN platform. Closed set of components such as Platform Management could be the first area of integration testing. Scaling up the architecture, connection between different sets of components (e.g., Platform management and Platform services) can be tested. Following this pattern, the next phases should focus on the collaboration of sets of components that are part of different layers of the platform architecture end finally the whole Platform itself should be tested.

Following unit testing, integration testing is focusing in the phase of guaranteeing that the internal part of each component has the appropriate behaviour. This can be done by testing the responses of the component's APIs, or by creating some methods that are exchanging expected data to ensure that with a specific input the output is as expected. There is a variety of different tools and frameworks for integration testing focusing on different development environments that can be adopted, since there is a variety of different components in ODIN. The main idea is to develop tests for certain use-case or tool and deliver a corresponding template.

Table 35: Odin integration test template.

ODIN component							
Tool/use case under test							
Test ID	Test	Description	Input	Outputs	Associated Tool/Use case	Test Result	Comment
#	Integration	Description of the function, integration, module or tool to be tested	Inputs for integration test	Expected outputs	Tool: Jenkins, Maven, Git Use case: Enabler #, Use case #	Pass/Fail	If failure, explain why
# #							

9.2.4 Acceptance Testing

In ODIN Validation is accomplished with Factory Acceptance Testing (FAT) and Site Acceptance Testing (SAT). Factory acceptance testing is taking place before the deployment of the system to the production environment by the vendor. FAT aims to ensure that the created platform or system meets the expected specifications and regularity obligations. After that, the system can be installed in a real-world production environment where the site acceptance testing will be performed.

The test users in FAT, which often are business experts or end-users, are not focusing on minor issues or software crashes, since that is a responsibility of the previous testing phases (unit testing, functional testing, integration testing). In FAT, a real-world and real-time and real-settings environment is simulated and performs as the final verification of the system's required functionality and appropriate operation. If the platforms behave as expected and no major issues occurs during FAT, the same level of stability is anticipated in a Production environment.

Site Acceptance testing is taking place on pilot site after the completion of FAT. To evaluate the operational readiness testing of the overall solution, Operational Acceptance Testing (OAT) is used.

SAT, is taking place in a production environment and serves as a final verification of the system's functionality and performance. It aims to measure the systems operational readiness in a production environment. It covers key quality attributes of the already installed system such as functional stability, portability, and reliability, as well as procedures for disaster recovery, end-user training, maintenance and security by interfering with the already installed platform.

In ODIN platform, the testing strategy is built in a sense of beta testing. The already tested pilot trials, should be tested after by the end users and vendors to judge if the system can support day to day usage and if the system is sufficient and correct for business usage. The test scenarios, in our case can be structured around the pilot trials. The general approach is to provide guidelines to the users on how to operate the platform and get feedback if the requirements are met.

The strategy can be described with the following steps:

- Set up of the operational environment for testing and connection of the interfaces
- Performance of the necessary actions with the desired input
- Deployment of the tests in pilot site environments after verifying the delivery of the expected outputs in real time
- Proceed to disaster recovery, training of end users and provide maintenance and security guidelines after ensuring the user/vendor criteria is met.

Figure 28: High level Acceptance Testing diagram.

There are plenty of different tools that are providing complete solutions for UAT. In ODIN Gitlab is used for developing software and Jenkins for continuous integration. These two tools can be the main choice for testing as well. Gitlab enables issue and bug tracking, and can be used along with Jenkins to execute tests. Other solutions could include software like Jira and Zephyr for Jira tool, which can be integrated with a variety of automation tools such as Jenkins, Selenium, Appium and SmartBear. qTest platform and Rally software, by providing KPI tracking and reporting, high scalability and issue Tracking are popular solutions in the field of testing. In such a big consortium as the consortium in ODIN, which includes so many different partners, it is no straightforward to oblige on a unified framework for the execution of the required test methodology, because it is not easily applicable for all the different technologies included. Therefore, using more than one tool is probably required although it is not mandatory.

9.2.5 Performance Testing

Performance Testing is a testing method that revolves around responsiveness, stability and speed of the platform under predefined conditions. It is the testing phase that can provide useful results about the overall scalability and reliability of the system's performance. There are a lot of different methods for performance testing. Apart from the methods, measurements and goals should be clarified. An early definition of performance testing to the evaluation from user perspective is

described by Vokolos and Weyuker[37]. This test method is more about reducing the performance bottlenecks rather than concerning bugs.

There are numerus methods for performance testing covered in literature[38]. A brief description of the methods is as follows:

- Load Testing: The application's ability to perform under expected user's loads. It aims to identify the bottlenecks before the production phase.
- Stress Testing: Identifies the limits of the system as it is put under extreme loads.
- Endurance Testing: It estimates the behaviour of the system over a long time period, as it has to sustain continuous load.
- Spike Testing: Producing unexpected changes in user loads in order to test the application as spikes are usual causes of failure.
- Configuration Testing: It tests the system with different combinations of system and hardware to evaluate the functional requirements.

Performance testing being performed last in the testing pipeline after the completion of acceptance testing. Therefore, if we consider the final version of the ODIN platform ready, operational and deployed on the pilot sites, that is the time when performance testing will take place. It will include members from integration team in collaboration with the developers and the stakeholders, who have better understanding of the scenarios where performance is critical and vital in order to produce optimal solutions.

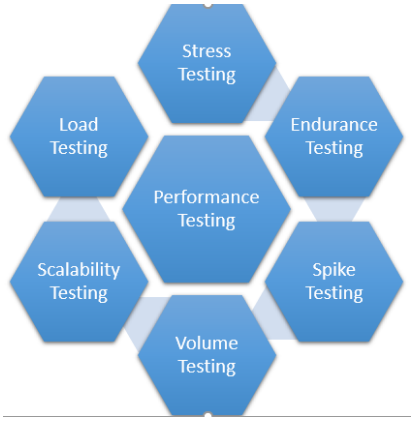

Figure 29: Performance testing lifecycle.

Load, stress, endurance, spike testing is not mandatory for all the software components developed in ODIN, due to the different nature and requirements of every component. A suggested solution is to provide tests for different software components that are offering the most used functionalities of the platform. In any case, the developing and integration teams could apply those tests to their software even if they are not essential for the project, to optimize it.

[37] Vokolos, F. I., & Weyuker, E. J. (1998, October). Performance testing of software systems. In Proceedings of the 1st International Workshop on Software and Performance (pp. 80-87).

[38] Mustafa, K. M., Al-Qutaish, R. E., & Muhairat, M. I. (2009, December). Classification of software testing tools based on the software testing methods. In 2009 Second International Conference on Computer and Electrical Engineering (Vol. 1, pp. 229-233). IEEE.

Configuration test should apply to every major component in ODIN in order to find the best balance between performance and scalability in both software and hardware environment.

Some tools suggested for performance testing such as LoadNinja, Apache Jmeter for Java apps, Gattling, Webload, SmartMeter, LoadRunner etc. Gitlab uses k6, an open-source tool for evaluating the system's performance under load. All of those tools providing virtual users or machines to manually load the system and inspect its behaviour in a 'real-world' and 'real-time' heavy load scenario.

9.3 Test Methodology

There are a lot of different development methodologies to determine development lifecycle and testing. Two of the most well-known are the waterfall model and the V-model. The waterfall model clarifies a set of fixed steps in downstairs structure. The main steps of this methodology are user requirements, architecture and technical specifications, unit implementation and testing, system integration and maintenance. The V-shape method is an extension of the waterfall method. It makes a direct association of each step with a corresponding testing phase to evaluate the results of the development. Those methods have their weaknesses where in a hospital environment such as the environment that ODIN will take place can produce a lot of difficulties. Waterfall method postpones testing and V-shape method is an inflexible method to apply in the architecture of ODIN. A method that brings security in earlier stages and enables flexibility and innovation is essential for the project. Such method is the DevSecOps, which is selected as that will be applied to the ODIN project for testing around the software cycle.

The previously described methodologies placing security and testing at the end as an individual and separate procedure. In a platform such as ODIN the growing security threats can occur in various phases of the development lifecycle, which requires a more active approach detecting and solving security vulnerabilities prior to the deployment of the software in the production environment. Thus, shifting security to the left with the adoption of DevSecOps practices is used as testing methodology in ODIN.

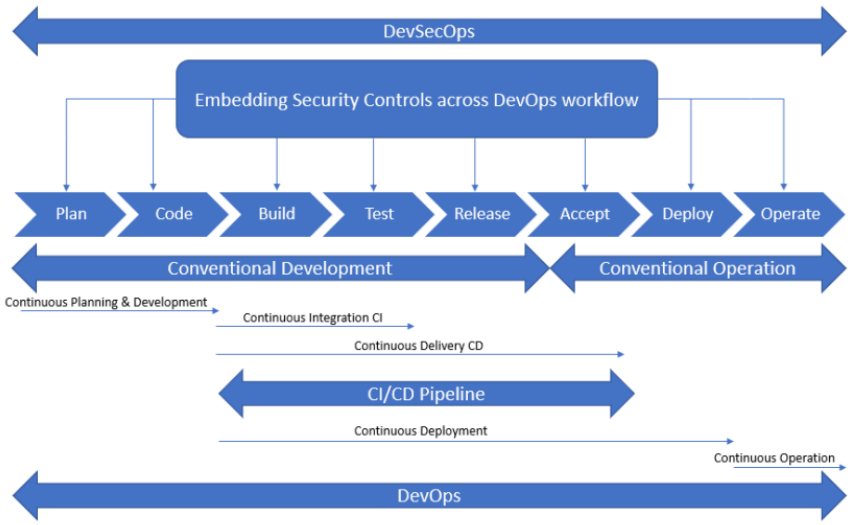

Figure 30: DevSecOps Methodology.

The general activities and responsibilities of the ODIN testing and integration methodology are presented in the table below

Table 36: Testing activities and responsibilities.

Level of Testing	Activities	Test environments	Testing Frequency	Responsible		
				Writing test cases	Providing test data	Running tests
Unit	Select test cases, Write automated test cases	Developer's environment, Continuous integration infrastructure	Creation of the test before/during development. Automated tests run continuously when component is built on the CI infrastructure.	Developer	Developer	Component/ Unit provider
Functional	Select test cases based on the requirements. Prepare demos with test data and run them	Developer's environment, CI infrastructure	Create tests whenever a new functionality is introduced. Run tests continuously when adding the functionality.	Developer, Integration team	Developer, integration team	Developer, Integration team
Integration	Select test cases, Manage unit dependencies, Write automated tests, Prepare non-automated test cases	CI Infrastructure	Automated tests run continuously when binding software components together. Manual testing each time a new component is introduced.	Integration Team	Integration Team	Integration Team
Factory/Site Acceptance	Define test cases according the pilots' trials. Prepare test data for a real time case scenario. Run integration tests of the platform. Identify observations and track issues.	Hospital/ Pilot Site environment.	Tests run on the integrated/ production platform which will be used whenever a trial is validated	Integration team, Pilot Site stakeholders	Integration team, Pilot Site stakeholders	End users/ vendors/ pilot site stakeholders

	Acceptance test review,					
Performance	Design test cases for scalability, stress, load, endurance and extreme unlikely scenarios Run the tests along with integration team and pilot site stakeholders Define the boundaries that the trials cannot perform at their best	Factory / Pilot Site environment	After the application has passed all test levels, validate the scenarios in which the designed application has the desirable performance	Developers Integration team, Pilot site stakeholders	Developers Integration team, Pilot site stakeholders	Developers Integration team, Pilot site stakeholders

9.4 Test Scenarios

This segment outlines several test scenarios for the components of the platform and their integration, focusing on two distinct types of tests: functional and integration tests. These tests are applicable in the current phase of the project, which involves the development of the second version of the ODIN platform. However, it is worth noting that the final version of the platform, the third version, will require additional testing in the form of acceptance and performance tests.

9.4.1 Functional Test scenarios

The following section presents functional test scenarios for the various components of the platform, each of which is accompanied by a brief description, the evaluation criteria utilized, and the corresponding pass/fail results of the test.

9.4.1.1 User creation

Table 37: User creation functional test.

№	Test	Description	Evaluation criteria	Results
1	*Simple user creation*	Create user in the internal database for use in different roles, with standard credentials.	The created user must be able to login and access the assigned resources within their role	Fail / Pass
2	*SSO*	Synchronize user base and roles with existing user provider (via, SAML, or OpenID).	An existing user must be able to login with their SSO credentials, and access assigned resources within their role	Fail / Pass

3	*User rejection*	Only created users are able to access the system.	Non-existent users credentials, erroneous credentials, are rejected.	Fail / Pass
4	*Authentication*	Only users with the correct role may access resources allowed to their role	Access from a user with Role which does not allow access to a specific resource is rejected	Fail / Pass

9.4.1.2 Resource registration

Table 38: Resource registration functional test.

№	Test	Description	Evaluation criteria	Results
1	*Resource registration*	A New resource is registered, it is added to the available list of resources	Difference of the list of resources (and their properties) before and after registration	The only difference is the added resource
2	*Resource unregistration*	A Resource is unregistered, it should be removed from the list of available resources	Difference of the list of resources (and their properties) before and after unregistration	The only difference is the removed resource
3	*Resource querying*	Resources may be queried, to find available resources which comply with a specific criteria.	All the elements of the result of the query of with a specific criteria, comply with said criteria.	Fail / Pass

9.4.1.3 Resource communication

Table 39: Resource communication functional test.

№	Test	Description	Evaluation criteria	Results
1	*Send and receive Raw Data*	Subscribe to a test topic (two clients, one publisher and one consumer). The publisher sends raw data to the topic and the consumer receives the data.	The consumer receives the data.	Pass / Fail
2	*Send data and filter them (not passing the filter)*	A publisher subscribes to a test topic and a consumer subscribes to the filtered test topic. The publisher sends raw data that does not pass the filter threshold to the test topic, and the consumer does not receive the data.	The consumer does not receive any data.	Pass / Fail
3	*Send data and filter them*	A publisher subscribes to a test topic and a consumer subscribes to the filtered test topic. The publisher sends raw data, that pass the threshold	The consumer receives the data.	Pass / Fail

	(passing the filter	of the filter, to the test topic and the consumer receives the data.		
4	*Create an alert with a preconfigured rule*	A rule is created on the rule engine that specifies two test topics (topic1 and topic2). One publisher client subscribes to topic1 and one publisher client subscribes to topic2. One consumer client subscribes to the test alert topic topic3. The publisher clients send data that trigger the rules. The rule engine creates an alert to the topic3. The consumer client receives the alert.	The consumer receives the alert.	Pass / Fail

9.4.1.4 Peer-to-peer resource communication

Table 40: Peer-to-peer resource communication functional test.

Nº	Test	Description	Evaluation criteria	Results
1	*Send request and receive answer*	Using Swagger or Postman send a request to a given API with the right format. Receive an answer in the specified form defined by the API	The answer is correct	Pass / Fail
2	*Send bad request and receive the answer (getting an error)*	Using Swagger or Postman send a request to a given API with bad format. Receive an answer in the specified form defined by the API for the given error	The error answer is correct.	Pass / Fail

9.4.1.5 Resource discovery

Table 41: Resource discovery functional test.

Nº	Test	Description	Evaluation criteria	Results
1	*Push request*	A request should be sent by the Resource Discovery to the Resource Directory	Send data to verify that the the resource discovery receives them and stores them	Fail/Pass
2	*Store requests*	Store resource discovery requests to the Resource Directory	Run a query to verify the data exists on the RD	Pass / Fail

The Resource Manager (RM) is the hub for resource registration, querying and connection to the ODIN platform. The RM will offer a set of services regarding resources, and their management. Specifically, the RM organizes its operations around the concept of the Resource Descriptor which is a semantic representation of the resource itself as well as the potential interoperability interfaces it offers. More information on this element can be found in Section 3.4.3.

The RM will manage the local privacy, security and trust of the resources being connected to the ODIN instance. Through a series of automatic security verification procedures, integrity verifications, and trust mechanisms, it will determine the trust category and security clearance

the resource has access to. More information about Security procedures can be found in Section 3.9.

The Resource Directory enables resource discovery, i.e. it allows resources to find information about other resources. Resource discovery will involve extracting from the Resource Directory and visualizing the relevant resource information based on an accurate, semantically-rich and user-friendly definition of characteristics that capture features and relations in full detail. This will build on the available ontologies representing the various knowledge domains relevant in the application and knowledge graphs technologies to support reasoning across data representation domains. This enables the exploration and selection of patterns in resource distribution and availability, building on the definition of domain specific representations of data according to domain ontologies and standards.

9.4.1.6 High-level application creation

The Resource Choreographer helps to design and execute High-level applications using current ODIN KER's.

Table 42: High level application creation functional test.

№	Test	Description	Evaluation criteria	Results
1	*Run a workflow*	Load a workflow into the Resource Choreographer that uses 1 or more KER. Trigger the workflow to start it. The Resource Choreographer sends the messages to the required topic. The Ker's read the topics and behave as expected	The workflow is executed as expected	Pass / Fail

9.4.1.7 Resource federation

Table 43: Resource federation functional test.

№	Test	Description	Evaluation criteria	Results
1	*Push requests*	The DLT has to have an operating API to receive messages.	Send data to verify that the API receives them and stores them in the ledger	Pass / Fail
2	*Store requests*	Store logs with access or resource requests to the DLT	Run a query to verify the data exists on the ledger	Pass / Fail
3	*Retrieve specific request*	Retrieve a log with access or resource request that is stored in the ledge	Provide a specific hash ID of the log to query the log	Pass / Fail

9.4.1.8 Metric collection

Table 44: metric collection functional test.

№	Test	Description	Evaluation criteria	Results
1	*Registration of metric*	A new metric is registered in the system	The new metric is registered, and can be read in the system	Pass / Fail
2	*Visualization of Metrics*	The metrics are visualized in a dashboard	Metrics are visualized in the dashboard, and are updated according to the refresh parameters and the value of the metric itself.	Pass / Fail

9.4.2 Integration Test scenarios

As described in 9.2.3 integration tests will be used when two or more platform components are used together. In this section, we introduce some testing scenarios based on the Reference Use Cases (RUCs) defined in the project.

9.4.2.1 RUC B1, UC1: Aided logistics support

Hardware Integration

Hardware integration (see D5.8) refers to connecting robotic components to the ODIN platform and their pilot use cases, following the steps of the hardware integration plan. The plan consists of 5 steps: 1) Hardware selection, 2) Single testing components, 3) Testing robotic platforms, 4) Full hardware integration, and 5) Deployment of ODIN demonstrators (see Figure 31).

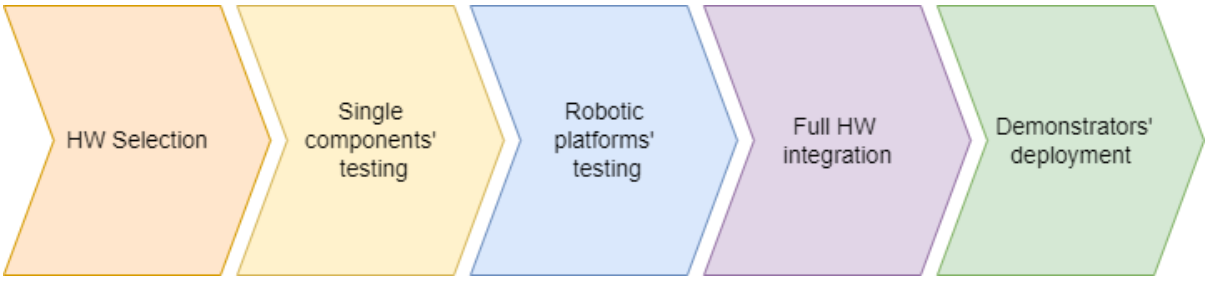

Figure 31: Main steps of HW integration plan.

The HOSBOT and Transparent Robot components are connected to the ODIN platform through a network system with a server and routers. Currently, only the Transparent Robot is fully connected, while other modules are connected for testing purposes. The connection of WP5 KERs is in progress at step 3, with more details provided in D4.3 and D4.6. The hardware integration for the SERMAS use case involving HOSBOT and Transparent Robot platforms is described, but it is subject to change as additional components may be added. The hardware integration status is at step 4, with testing underway. UCBM tests have shown robustness and functionality, but official validation will occur at SERMAS. The design, integration, and testing of HOSBOT have undergone iterations, and the hardware integration phases and final version are depicted in **Figure 32** and **Figure 33**.

Figure 32: Main hardware progresses of the HOSBOT platform.

Figure 33: Final version of the HOSBOT rack.

Software Integration

This section describes the integration of the robotic components and technologies with the ODIN platform as part of their pilot use cases. The software integration plan has six steps (see Figure 34), including 1) Revision of architecture, 2) Creation of package skeleton and dummy nodes, 3) Implementation of real nodes, 4) Initial testing on hardware, 5) Full software integration and testing, and 6) Deployment of the final version of the demonstrators.

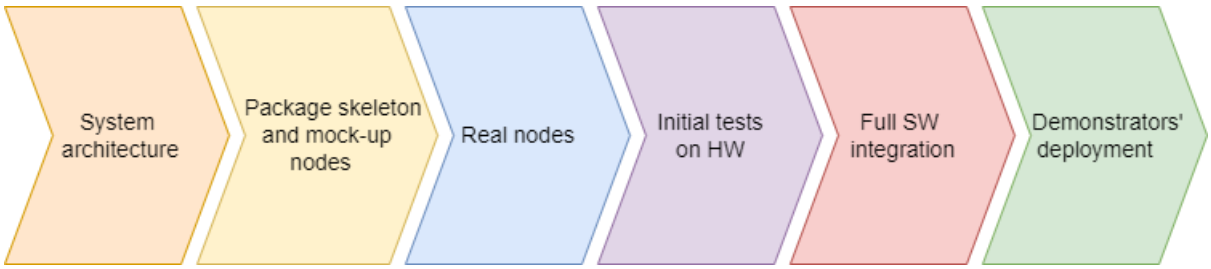

Figure 34: main steps of the software integration plan.

The software integration of the robotic components with the ODIN platform involves connecting ODIN's main software components, including human modelling, context awareness, social navigation, robotic cognition, and human-robot interaction and communication. Depending on the use case, a variety of software modules will be used to ensure successful demonstrations. The robotic software components will interact with the ESB layer of the ODIN architecture and communicate with other modules via the Kafka communication protocol. A connector has been developed to bridge the MQTT network protocol used by the robotic platforms with the Kafka Enterprise Service Bus. The integration of the software connection to the ODIN platform started with the *Transparent Robot* and has been successfully extended to other systems such as HOSBOT. The shared data between the different modules will be stored and post-processed to answer the needs of hospitals. The validation of the connection has been done on the transparent robot, as it has a continuous stream of data during its operation.

The progress on software integration for each pilot site demonstrator is provided based on the steps described previously. However, the integration process uses an iterative methodology, and additional software components may be used in the future if necessary to meet the requirements of the ODIN robotic use cases.

The SERMAS robotic demonstrator at RUC B2 has completed the integration of all software modules into the overall system and reached Step 5 in the software integration process. The HOSBOT platform collects various software modules developed by different partners, such as the robotic navigation module of RB-1 base, mapping and control modules, *Smartbox* software, HMI, social navigation algorithms, fleet management system, and integration with the ODIN platform. The real-time localization system is still missing, but the software integration has already been addressed. The *Transparent Robot* and *Proximity capacitive sensor* will be mounted as add-on elements with their software modules and connections to the ODIN platform for connecting with other devices.

9.4.2.2 RUC B2, UC2: Clinical engineering, medical locations, real-time management

SERMAS scenario, RUC B2

Robot: Hosbot

In this scenario the HOSBOT robot is taking the task to automatically deliver consumables in the hemodynamic room when requested. After the completion of the delivery the robot is responsible of registering the used goods to the platform. During this procedure the robot will provide continuous feedback to platform in order to check if everything is running appropriately.

1. At first, all the resources needed for the completion of the workflow of the robot will be registered in the platform.
2. The resources will be loaded to the Resource Directory.
3. When the right trigger occurs (for example, a notification arrives saying some service lacks of X fungible), the resource choreographer will evaluate them and it will provide the appropriate configuration of the workflow.
4. The Resource Choreographer will check fungible availability against the Resource Directory.
5. The Resource Choreographer will check robot availability and sends the task.
6. The HOSBOT will get his position using the RTLS.
7. The HOSBOT will compute route to the destination.
8. The HOSBOT start its navigation towards the destination to get the consumables.
9. The HOSBOT arrives to destination and the staff puts the fungibles into the smart boxes.
10. The HOSBOT (smartbox) updates fungibles data into the Resource Directory.
11. The HOSBOT updates charging status.
12. The HOSBOT computes the route to the new destination.
13. The HOSBOT navigates to the destination.
14. The HOSBOT reaches the destination and staff retrieves the fungibles.After the delivery of the consumables, HOSBOT will provide information about its location, charging status, and update the consumables' location.

Relevant deliverables: (D5.8 Technology Integration Periodic Report v2, D7.1) Pilot studies Use Case Definition and Key Performance Indicators.

9.4.2.3 RUC A1, UC3: AI-based support system for diagnosis

UMCU scenario (hypothetical), RUC A1

AI-based support system for diagnosis: In this scenario IoT technology will be used to locate diagnostic modalities (CT-scans, Echo's, MRI, etc.) and their availability within the hospital using sensors and tags. The data deriving from the IoT devices will be incorporated in a model that predicts the most efficient diagnostic pathway for the individual patient, in this case this regards patients that are eligible for a carotid endarterectomy (CEA).

1. At first, the specified resources needed for the execution of the workflow will be installed in the hospital and linked to the ODIN platform.
2. Then, the data of the IoT technology together with the patient data and information on availability of physicians (EHR data), will be loaded into the ODIN platform,
3. The ODIN platform will process all data and determine the most efficient diagnostic pathway for the individual patient. The platform will take into account all data. Additionally, predefined alternative diagnostic pathways for this specific patient population will be taken into account, meaning that, in some cases, diagnostic modalities can be interchanged based on availability.

4. The most effective diagnostic pathway is then presented to the healthcare professional in the electronic health record.
5. The healthcare professional decides to accept (or not) the proposed trajectory provided by the ODIN platform and plans accordingly.

Relevant deliverables: (D5.8 Technology Integration Periodic Report v2, D7.1) Pilot studies Use Case Definition and Key Performance Indicators.

9.4.2.4 RUC A2, A2.1, A2.2 UC4: Clinical tasks and patient experience

CUB scenario, RUC A2

Clinical tasks and patient experience – sleep disorder management.

In RUC A, AI models for automated sleep disorder detection are deployed and enabled in the ODIN platform. This will support the clinicians to make a faster, more accurate and objective diagnosis. The data description, visualization and model are described in deliverables D6.1, D6.2 and D6.6. The AI model for this RUC will be integrated as KER in the platform so that once the detection is made, the output is published (sleep disorder and potentially sleep staging), after having received a certain data input (polysomnography data).

Another KER provided and exploited by this RUC is federated learning. The latter will be integrated in the ODIN platform so that it can be leveraged by different use cases. However, the sleep disorder management will represent the first use case to test federated learning in the ODIN platform. The federated learning pipeline is based on pre-configured training rounds and Docker containers for the coordinator and the workers. The main actors of the FL pipelines are the user (researcher and/or clinician at hospital sites), that navigates through all the FL operations via a user interface, the coordinator which execute and orchestrate the pipeline, and the worker that performs the model training at local sites (e.g., the hospitals). The coordinator consists of subcomponents which are application programming interfaces (APIs) to configure the number of training (configuration server), a postgres data store to save all the information, a software development kit for the user, an aggregator for model updates aggregation and a gateway API. In the ODIN platform, federated learning is integrated as a KER where the coordinator can be hosted, while the worker will be deployed by the pilot sites for enabling specific RUCs (e.g., sleep disorder management).

At each training round, the coordinator exposes each participating worker. The worker executes this task and submits the results back to the coordinator. When all the participating workers have sent their weights of the differentiable ensemble, the coordinator aggregates the results by using a federated aggregation technique and updates the database so that the new tasks are available for the workers for the next round.

To test this integration, the AI model is registered using the Resource Description and the output published via the connector to the ESB. A similar approach will be applied when testing the integration of federated learning. Once model weights are received from the workers and the aggregation is performed, the coordinator will call the connector in order to publish the data back to the ESB.

UCBM scenario 1, RUC A2.1

Clinical tasks and patient experience – Meal delivery and monitoring of food consumption to prevent undernutrition.

In RUC A2.1 two robots (TIAGO, CERTHBot) are used to deliver food to the patient and to monitor the patients' food consumption in order to prevent undernutrition. The system will provide information regarding the food intake (calories, micronutrients based on food images taken by the TIAGO robot. The CERTHbot will monitor the patients will is consuming his food and will send information regarding the time he needs. While executing their workflow the two robots will provide continuous feedback to the platform to validate if everything goes as planned.

At first, in the hospital site, after the admission and screening step, the patients with dysphagia and undernutrition will be diagnosed. After that, the ODIN platform will be responsible to execute correctly the steps of treatment and monitoring. The following checkpoints will verify the accurate execution of those steps by the Platform:

1. All the resources needed for the completion of the workflow will be registered to the platform.
2. Those resources will be loaded to the resource choreographer. Based on them, the resource choreographer will provide the appropriate configuration of the workflow.
3. The robots will start executing the workflow. After patient recognition, robots will send their location to platform to validate if they recognized the correct patient. TIAGO Robot will also send back information about its meal supply to validate if it delivered the food to the patient.
4. In the next step of monitoring by both TIAGO and CERTHBot, the platform will check if the alerts are sent correctly, by examining the feedback of the robots.
5. In the end, the platform will check if the robots have returned to their bases, signalling the successful completion of the workflow.

UCBM scenario 2, RUC A2.2

Clinical tasks and patient experience – Rehabilitation monitoring to prevent loss of mobility.

In RUC A2.2 the TIAGO robot is used to monitor the elderly patients in the Geriatrics Clinic while are executing the prescribed rehabilitation exercises in order to avoid loss of mobility. While executing its workflow the TIAGO robot will provide continuous feedback to the platform to validate if everything goes as planned.

The ODIN platform will be responsible to execute correctly the steps of monitoring. The following checkpoints will verify the accurate execution of those steps by the Platform:

1. All the resources needed for the completion of the workflow will be registered to the platform.
2. Those resources will be loaded to the resource choreographer. Based on them, the resource choreographer will provide the appropriate configuration of the workflow.
3. The robots will start executing the workflow. After patient recognition, the TIAGO will send its location to platform to validate if they recognized the correct patient. TIAGO Robot will also send back information about the correctness of the executed exercises and the estimated time for completing the exercises.
4. In the next step of monitoring, the platform will check if the alerts are sent correctly, by examining the feedback of the robot.

5. In the end, the platform will check if the TIAGO robot has returned to its bases, signalling the successful completion of the workflow.

9.4.2.5 RUC A3, UC5: Automation of clinical workflows

UCBM Scenario, RUC A3

Automation of clinical workflows – Monitoring of oxygen therapy to prevent hypoxia complications.

In RUC a3 the ODIN platform is used to monitor the patients during their prescribed oxygen therapy in order to prevent hypoxia complications. This step comes after the admission and screening step and the patient evaluation step in the RUC a3 scenario. TIAGO robot is the responsible robot for the completion of the scenario. TIAGO robot will monitor the correctness of the oxygen therapy and patient's compliance to prevent hypoxia complications in the Geriatrics Clinic in UCBM pilot. Specifically, the proposed solution will be able to monitor the correctness of the therapy in terms of the correct positioning of the oxygen mask i.e. if the oxygen mask is worn correctly, for both venturi and nasal prongs mask.

After the first steps done by the Pilot site, TIAGO robot is activated to execute the patient monitoring scenario. There will be several tests during the scenario that will be running in order to evaluate if everything is working correctly.

1. At first, the specified resources needed for the execution of the workflow will be registered in the platform.
2. Then those resources will be loaded to the resource choreographer. The resource choreographer will evaluate the given resources and will provide the appropriate configuration of the workflow based on them.
3. With the appropriate configuration of the workflow in place, the TIAGO robot is ready to start the monitoring procedure. During this procedure, the robot will continuously provide feedback to the platform to validate if everything is running as expected.
4. Based on the feedback, the platform will check if the selected patient has identified correctly by the robot.
5. During the patient monitoring, the platform will check if the alerts are sent correctly based on their feedback (e.g., incorrect use of the mask).
6. Finally, the platform will check by the coordinates of the robot to identify if it has returned to its starting base.

9.4.2.6 UC6: Inpatient remote rehabilitation

UMCU scenario (hypothetical)

Inpatient remote rehabilitation: This scenario involves patients that could benefit from post-operative remote monitoring of blood pressure. A workflow will be established that incorporates remote monitoring of patients with mitigated remaining risks and trains (in-app) both nurses and patients to use and adhere to the blood pressure monitoring protocols. Sequence of operations: The patient leaves the hospital with the home-monitoring device and e-Health app. The patient records blood pressure according to the protocol (3 to 5 times a day). Then, the device returns the measurements to the ODIN platform which interacts with the EHR. A physician or nurse will then verify the measurements.

The ODIN platform will be used to store the data from the e-Health technology, process the data, and to send alarms to the designated health care professional regarding abnormal post-operative blood pressure measurements.

1. At first, the specified resources needed for the execution of the workflow will be installed in the hospital and the technology will be linked to the ODIN platform.
2. All registered data in the e-Health app will be loaded into the ODIN platform.
3. The data is processed by the ODIN platform, based on prespecified protocols.
4. ODIN platform interacts with the EHR to store the outputs.
5. Based on the output, alarms/notifications are sent to the treating physician/nurse monitoring the blood pressure in case of abnormalities.
6. Additionally, the platform sends reminders to the treating physician/nurse at prespecified time points (to be determined) to check all blood pressure measurements.

Relevant deliverables: (D5.8 Technology Integration Periodic Report v2, D7.1 Pilot studies Use Case Definition and Key Performance Indicators.)

9.4.2.7 RUC C, UC7: Disaster preparedness

RUC C is currently focusing on a solution based on computer vision algorithms to tackle evacuation risks, forgotten objects that could be dangerous, patients falling and COVID measures been followed (wearing masks or keep a minimal distance). More information about this RUC can be found in deliverables D6.1, D6.2 and D6.4.

The computer vision algorithm will be integrated in the platform as a KER. It will receive as input the security cameras video stream, process it, extract the relevant features and output an alert when necessary.

To test the integration of these solution in the platform the AI will be registered using the Resource Descriptor and the outputs of this algorithm will be published to the ESB in case of alert.

Relevant deliverables: (D7.1 Pilot studies Use Case Definition and Key Performance Indicators.)

9.4.3 Testing Environment

Each test phase will be executed in a different test environment. As an example, local environments can be used for unit testing as the development's teams are us them to develop their components. The environments written below can be considered for the testing phases towards the validation of the ODIN Platform

- Local System: Developers' local machine environment. (Unit testing)
- Development premises: Developers' local machines along with GitLab and Jenkins Testing environment.
- Staging: Testing environment provided INETUM (see 8.1.4)
- Pilots: The implementation of the RUC scenarios will also serve as validation test for the developed architecture.

Table 45: Template for unit test reporting

Field	Value
Test Case ID	Unique identifier of the test case
Summary	A brief description of the test case
Sub-system or interface to test	The system, sub-system or interface under testing
Related Requirements	The system requirement(s) validated by the specific test case
Objective	The test case goal
Assignee	The partner(s) to perform the test case
Preconditions	Potential prerequisites for the test case to be available for execution, i.e. test data requirements or previous test cases that should be executed first
Target Platform	The platform that is targeted by this test case
Test Data	The actual test data that should be used for this test case during execution
Actions	This is the test procedure, the actions taken to execute the test case
Expected Results	The expected outcome of the test case execution
Side-effects	The potential side-effects that this test case has, e.g. modification of existing data

Integration and system testing involve testing components of the system in combination. This makes it more challenging than unit testing, since it involves combining components developed by different developers, by different organizations. Table 46 provides an overview of the plan for integration/system testing to be followed within ODIN.

Table 46: Integration/system testing plan.

Testing activities	• Specify test scenarios • Manage dependencies between components • Write automated test cases • Prepare manual test cases
Test environment	• CI/CD infrastructure (see Section 8.1.4)
Testing frequency	• The automated tests run automatically whenever updates are made in the involved components. • Manual tests are performed at least at major releases of the ODIN platform.
Responsible	• For writing test cases: Component developers • For providing test data: Integration team • For running tests: Integration team

9.4.4 Tools for testing and Monitoring

This section aims to introduce several software tools and infrastructure that are suitable for testing and monitoring purposes. All of the listed tools have already been deployed as part of the ODIN DevOps infrastructure. It is important to note, however, that each component of the platform is unique and may require different testing methods. Therefore, developers are encouraged to utilize tools of their choosing and are not limited to the tools presented here

GitLab

GitLab[39] is a web-based, open-source platform that can be used for source code management. Gitlab offers open and private repositories along with issue tracking and wikis. It is a platform developed for the needs of the DevOps methodology, which means that it enables developers to handle all the aspects of a project, from project planning to source code management, monitoring and security. GitLab is also designed to helm teams work together in collaboration projects. Teams can shorten product lifecycles and enhance productivity to achieve greater value for the customers. GitLab uses a role-based strategy for addressing permissions to provide evaluation access on individual components depending on the access rights.

GitLab can be also used to enable security in the ODIN project. Gitlab in collaboration with Jenkins are used for facilitating and automating procedures of DevSecOps[40]. Security testing is applied within the CI pipeline raising developers' awareness of vulnerabilities in their code, in order to prevent security issues earlier in the project pipeline.

Jenkins

Jenkins server[41] is also a DevOps tool used for CI/CD jobs. Jenkins is using agents to run different jobs for different pipelines.

Jenkins as a very powerful automation tool can enhance the Testing procedure by providing automated tests in every part of the development lifecycle. As an example, a new push to a GitLab repository can trigger a Jenkins job to test if the pushed code is functional. Therefore, it helps developers to debug their code on time.

Kubernetes

A Kubernetes[42] cluster will used to for the deployment of the platform. Thus, every platform component will be deployed upon this cluster and therefore will be integrated and connected with the other components of the platform. This is making Kubernetes the base of the platform where all the different integration tests between the different components will take place.

[39] Gitlab website. https://about.gitlab.com/ , last accessed Feb 2023

[40] Gitlab. (2021, 1 June). GitLab is setting the standard for DevSecOps. https://about.gitlab.com/blog/2021/06/01/gitlab-is-setting-standard-for-devsecops/

[41] Jenkins Documentation https://www.jenkins.io/doc/, last accessed Feb. 2023

[42] Kubernetes documentation https://kubernetes.io/docs/home/, last accessed Feb. 2023

10 ODIN platform manuals

There are three primary roles involved in and ODIN platform implementation:

- The developer, who develops the ODIN platform components;
- The deployer, who installs the ODIN platform at a particular ODIN instance, e.g., a hospital;
- The user, who uses the ODIN platform and the functionalities that it offers.

The ODIN manuals provide documentation targeted at all the above roles. At this stage of the project, the manuals are not yet compiled, so the sections below contain only a sketch of the kind of information they will contain. The complete manuals will be available in future versions of this deliverable, in combination with deliverables D3.7-D3.9 “Technical Support Plan and Operations”.

10.1 Developer manual

The ODIN developer manual will contain information targeted at the developers of the ODIN platform and its components. This information will cover mostly how to use the ODIN DevOps infrastructure during development, to facilitate the application of a Continuous Integration / Continuous Delivery (CI/CD) pipeline.

The developer manual will cover the following steps of the CI/CD pipeline:

- Source code management
- Building executables and release versions
- Testing software
- Software release
- Component deployment
- Operation monitoring
- Pipeline orchestration and automation
- Security-related guidelines
- How to document components and APIs

The starting point for the compilation of the developer manuals will be the information provided in D3.1 “Operational framework”, specifically the content of the “ODIN guidelines” sub-sections in each of the main sections of that deliverable. This information, which so far has been targeted to the developers within the ODIN consortium, will be enriched with more specific instructions, screenshots and examples to be followed by future developers of the ODIN platform.

10.2 Deployment manual

The ODIN deployment manual will contain information and instructions about the deployment of the ODIN platform in an ODIN instance. The deployment manual will contain instructions mainly for the following steps:

- Installation of the DevOps infrastructure, i.e. mainly Kubernetes.

- Installation of the ODIN platform package, containing all necessary components for the platform operation. A corresponding "Helm chart" will contain all necessary instructions for the images to be drawn and how to set them up.
- Individual instructions for the deployment of the ODIN platform components in isolation.
- Installation and use of the ODIN dashboard and the ODIN platform management components, which allow monitoring the operation of the ODIN deployment.

10.3 User manual

The ODIN user manual will contain information and instructions about the use of the ODIN platform by the end users and will be published in the corresponding ODIN wiki. The types of users foreseen in ODIN are presented in Table 47, along with the type of information that will be available in the corresponding manuals, covering the parts of the platform that each user type is exposed to and how to make the most of them.

Table 47: ODIN end users and related manual information.

End user type	Description	Manual information
Platform manager	An IT-oriented person or team that manages the ODIN platform installation and monitors its operation. This user also is responsible for registering new resources in the platform.	How to monitor the ODIN installation and use the platform management, configuration and security components, as well as the resource management components.
Hospital administrator	A decision-making person or team working in the hospital that is responsible for directing the kinds of operations and scenarios that are important and need to be implemented by the ODIN platform. This is the user that designs new scenarios and high-level business logic to be carried out by resources.	How to use the Resource Choreographer, ODIN ontology and resource directory to design new high-level applications, and how to manage permissions for resource federation.
Clinical personnel	Medical doctors and nurses that are assisted by the KERs to perform daily clinical operations in the hospital.	How to use the available KERs in daily activity, to support clinical tasks.
Supporting personnel	Non-clinical personnel working at the hospital, e.g. logistics or security departments.	How to use the available KERs in daily activity, to support hospital management tasks.
Patient	The patients in the hospital, receiving the services of the clinical personnel, supported by the KERs.	How to use the available KERs to be aided during their hospital stay.

It should be noted that, although the core ODIN components are specified as in the architecture, the KERs that will be available at each ODIN instance will differ. Hence, KER-related documentation will be targeted at the KERs that are available in the particular ODIN instance.

Such dynamic documentation will be available through the platform documentation component, as mentioned in Section 6.4. The documentation to be provided in the current deliverable or in deliverables D3.7-D3.9 will contain information about common KERs that are found in the pilot sites of the ODIN project.

11 Conclusion

This deliverable is the second version of a series of deliverables that cover a wide range of topics within the ODIN project:

- ODIN architecture and components
- System integration
- Developer, deployer and user manuals
- System validation

The current version of the deliverable focusing on the specification of every of every component of the ODIN platform by providing info about every component. The deliverable provides a general overview of the ODIN platform and points to relevant deliverables to produce more details. It is also introducing platforms integration and validation as well as manuals about the use of the platform.

The compilation of the ODIN platform architecture has been the product of several trials, recommendations and suggestions by the ODIN consortium, and reflects in the best way, up to this point, the vision of the ODIN project: to develop a platform that can connect together diverse and distributed hardware and software resources to support hospital operations.

The ODIN architecture and the design of its core components aim towards the above goal, by enabling the following key functions:

- Resource abstraction, by creating syntactic and semantic resource representations that are agnostic of vendor-specific details;
- Resource communication, by allowing resources to communicate with each other to perform complex tasks;
- Resource federation, by allowing resources of one organization to be shared among other organizations in a trustworthy manner.

This deliverable aims to specify how the above key functions are implemented within ODIN, which components are responsible for their implementation and how they communicate with each other. The deliverable can be used as a map of the ODIN platform and as a guide to its components, clarifying the position of each component in relation to the other components.

The presented architecture is subject to changes during the course of the project. There are specific details that have not yet been decided and have been reported in the component descriptions. Discrepancies from the current version and necessary modifications will be reported in the next version of this deliverable (D3.12). However, the current version of the architecture is mature enough to be used as a starting point to guide component development and system integration in the next phases of the project.

Apart from any changes in the architecture itself, the next versions of this deliverable will focus on details about the implementation and validation of the ODIN components. The next phase of the project will be devoted to development, integration and Pilots. Specific activities that will lead to the implementation of the ODIN platform have been reported to current deliverable (D3.11) and will be updated to the next reporting in D3.12 involve the following:

- Finalization of the DevOps infrastructure implementation
- Development of individual ODIN components, both core platform components and KERs

- Definition and execution of unit tests for the individual components and integration test scenarios
- API specification and documentation of the developed components

As a roadmap towards the next versions of the ODIN platform implementation, Figure 35 depicts a timeline of the ODIN platform versions throughout the project duration. At the first year of the project (year 1), the first version of the ODIN platform has been developed, which focuses mostly on the preparation for component development and integration. In particular, the first version of the ODIN ontology has been implemented, as described in D3.2, which allows the description of a wide range of entities. The source code management system has also been developed to coordinate component development.

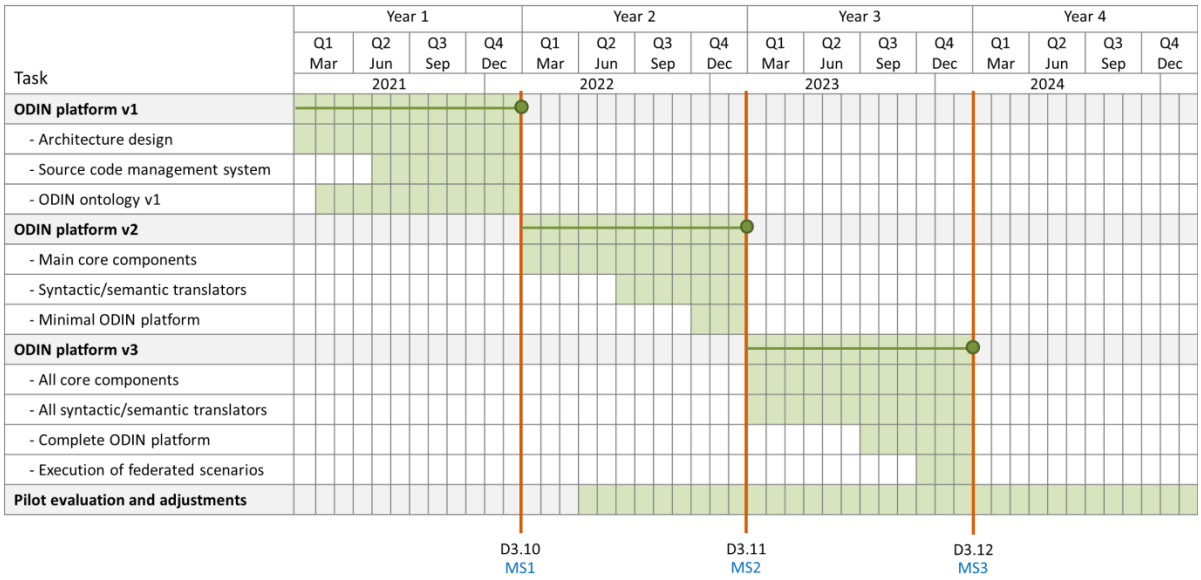

Figure 35: Timeline and characteristics of the ODIN platform versions.

During the second year (year 2), activities have been focused on development and integration of the main core platform components, including the Enterprise Service Bus, resource management components and resource federation. In parallel, syntactic and semantic transformation components will be developed at least for the KERs involved in representative use case scenarios (RUCs). The goal for the second version of the ODIN platform is to offer a minimal integrated ODIN platform that can be installed in pilot sites and support representative use cases.

During the third year, component development and integration will continue, focusing on the integration of high-level components such as the choreographer, as well as on the finalization of all relevant syntactic/semantic translators. The third version of the ODIN platform aims to cover the complete set of functionalities described in this deliverable. The third-year activities also include the execution of federated scenarios involving interaction across hospitals. Adjustments and modifications to the second and third versions of the ODIN platform will take place during the pilot evaluation phase, up to the end of the project, to fix issues and improve the platform functionalities. In this year also the validation of the project with testing will take place taking advantage of the use of ODIN Components in Pilot's sites.

References

[1] Schmidt, M.T., Hutchison, B., Lambros, P. and Phippen, R., 2005. The enterprise service bus: making service-oriented architecture real. IBM Systems Journal, 44(4), pp.781-797.

[2] "What Is a VPN? - Virtual Private Network - Cisco." https://www.cisco.com/c/en/us/products/security/vpn-endpoint-security-clients/what-is-vpn.html (accessed Mar. 15, 2022).

[3] "Virtual Private Networking: An Overview | Microsoft Docs." https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/bb742566(v=technet.10) (accessed Mar. 15, 2022).

[4] "Computer network technologies and services/VPN - Wikibooks, open books for an open world." https://en.wikibooks.org/wiki/Computer_network_technologies_and_services/VPN (accessed Mar. 15, 2022).

[5] "File:VPN solution classification.svg - Wikimedia Commons." https://commons.wikimedia.org/wiki/File:VPN_solution_classification.svg (accessed Mar. 15, 2022).

[6] "RFC 3809 - Generic Requirements for Provider Provisioned Virtual Private Networks (PPVPN)." https://datatracker.ietf.org/doc/html/rfc3809#section-1.1 (accessed Mar. 15, 2022).

[7] "RFC 6434 - IPv6 Node Requirements." https://datatracker.ietf.org/doc/html/rfc6434 (accessed Mar. 15, 2022).

[8] "WireGuard: fast, modern, secure VPN tunnel." https://www.wireguard.com/ (accessed Mar. 15, 2022).

[9] B. Preneel and F. Vercauteren, Eds., Applied Cryptography and Network Security, vol. 10892. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-93387-0.

[10] B. Dowling and K. G. Paterson, "A Cryptographic Analysis of the WireGuard Protocol," 2018, pp. 3–21. doi: 10.1007/978-3-319-93387-0_1.

[11] "WireGuard VPN makes it to 1.0.0—and into the next Linux kernel | Ars Technica." https://arstechnica.com/gadgets/2020/03/wireguard-vpn-makes-it-to-1-0-0-and-into-the-next-linux-kernel/ (accessed Mar. 15, 2022).

[12] "Understanding Multi-Cluster Kubernetes | Ambassador Labs." https://www.getambassador.io/learn/multi-cluster-kubernetes/ (accessed Mar. 15, 2022).

[13] "Simplifying multi-clusters in Kubernetes | Cloud Native Computing Foundation." https://www.cncf.io/blog/2021/04/12/simplifying-multi-clusters-in-kubernetes/ (accessed Mar. 15, 2022).

[14] "kubernetes-sigs/kubefed: Kubernetes Cluster Federation." https://github.com/kubernetes-sigs/kubefed (accessed Mar. 15, 2022).

[15] "virtual-kubelet/virtual-kubelet: Virtual Kubelet is an open source Kubernetes kubelet implementation." https://github.com/virtual-kubelet/virtual-kubelet (accessed Mar. 15, 2022).

[16] “8 Use Cases for Kubernetes over VPN: Unlocking Multicloud Flexibility | by Alex Feiszli | ITNEXT.” https://itnext.io/8-use-cases-for-kubernetes-over-vpn-unlocking-multicloud-flexibility-3958dab2219f (accessed Mar. 15, 2022).

[17] “File:VPN overlay model.svg - Wikimedia Commons.” https://commons.wikimedia.org/wiki/File:VPN_overlay_model.svg (accessed Mar. 15, 2022).

[18] “File:VPN peer model.svg - Wikimedia Commons.” https://commons.wikimedia.org/wiki/File:VPN_peer_model.svg (accessed Mar. 15, 2022).

[19] “File:Provider provisioned VPN.svg - Wikimedia Commons.” https://commons.wikimedia.org/wiki/File:Provider_provisioned_VPN.svg (accessed Mar. 15, 2022).

[20] “File:Customer provisioned VPN.svg - Wikimedia Commons.” https://commons.wikimedia.org/wiki/File:Customer_provisioned_VPN.svg (accessed Mar. 15, 2022).

Appendix A Acronym glossary

Acronym	Definition
ABAC	Attribute-Based Access Control
ACI	Azure Container Instances
AH	Authentication Header
AI	Artificial Intelligence
API	Application Programming Interface
APR	Address Resolution Protocol
AWS	Amazon Web Services
BLE	Bluetooth Low Energy
CA	Certificate Authority
CE	Customer Edge
CI/CD	Continuous Integration / Continuous Delivery
CIDR	Classless Inter-Domain Routing
CNI	Container Network Interface
CoAP	Constrained Application Protocol
CPS	Cyber-Physical System
CPU	Central Processing Unit
CRUD	Create - Read - Update - Delete
CSV	Comma-Separated Values
DAC	Discretionary Access Control
DAT	Developers Acceptance Testing
DB	Database
DBMS	Database Management System
DREAD	Damage, Reproducibility, Exploitability, Affected users, Discoverability
DSL	Domain-Specific Language
EMDN	European Medical Devices Nomenclature
ESB	Enterprise Service Bus

ESP	Encapsulating Security Payload
FHIR	Fast Healthcare Interoperability Resources
FL	Federated Learning
FTP	File Transfer Protocol
GDPR	General Data Protection Regulation
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HIS	Hospital Information System
HSTS	HTTP Strict Transport Security
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPLS	IP-only LAN-like Service
Ipsec	Internet Protocol Security
ISDN	Integrated Services Digital Network
IT	Information Technologies
ITC	Information Technology Center
JBPM	Java Business Process Model
JSON	JavaScript Object Notation
KER	Key Enabling Resource
KPI	Key Performance Indicator
L2TP	Layer Two Tunneling Protocol
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LTE	Long-Term Evolution

MAC	Mandatory Access Control
MPLS	Multiprotocol Label Switching
MQTT	MQ Telemetry Transport
NAT	Network Address Translation
OASIS	Organization for the Advancement of Structured Information Standards
OIDC	OpenID Connect
OSI	Open Systems Interconnection
PE	Provider Edge
PIN	Personal Identification Number
PPP	Point-to-Point Protocol
PPTP	Point-to-Point Tunneling Protocol
PST	Privacy, Security and Trust
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RAM	Random Access Memory
RBAC	Role-Based Access Control
REST	Representational State Transfer
RF	Resource Federation
RFC	Request for Comments
RFID	Radio Frequency Identification
RGBD	Red Green Blue Depth
RM	Resource Manager
ROS	Robot Operating System
RSM	Resource Management System
RTLS	Real-Time Location System
SPA	Single-Page Applications
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SSL	Secure Sockets Layer

SSO	Single Sign-On
SSTP	Secure Socket Tunnelling Protocol
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
TCP	Transmission Control Protocol
TH	Transaction Handling
TLS	Transport Layer Security
UAT	User Acceptance Testing
UDP	User Datagram Protocol
UI	User Interface
URL	Uniform Resource Locator
USB	Universal Serial Bus
VK	Virtual Kubelet
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network
VPWS	Virtual Private Wire Service
XMPP	Extensible Messaging and Presence Protocol

Appendix B

B.1 ODIN ROS buildfarm job types

B.1.1 Devel jobs

This type of jobs is running when there are changes in the `source` entry of our package(s), written in the `distribution.yaml` of each desired ROS version (after a PR was made from running `bloom-release`). The code is cloned from the relevant URL/branch and then each package is built and tested in isolation (with `catkin_make_isolated`). The following figure summarizes the process:

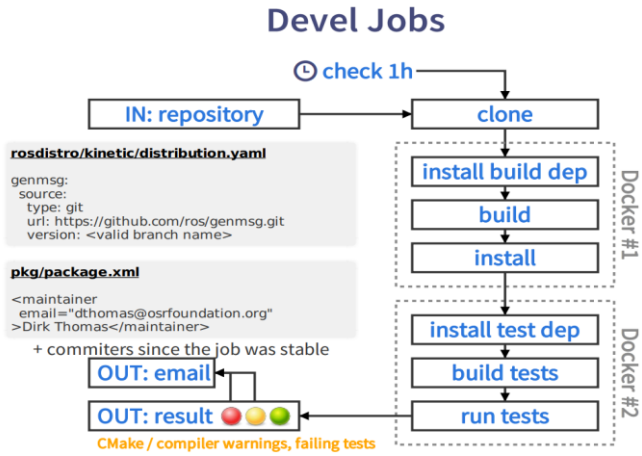

Figure 36: Devel Jobs process followed in the ODIN ROS buildfarm.

B.1.2 Release Jobs

The release jobs are further categorized into Source and Binary jobs. The Source jobs are triggered by new releases in the release repository of the package(s) (done by `bloom-release`), while the Binary jobs are triggered by the successful source jobs. The following figure is descriptive:

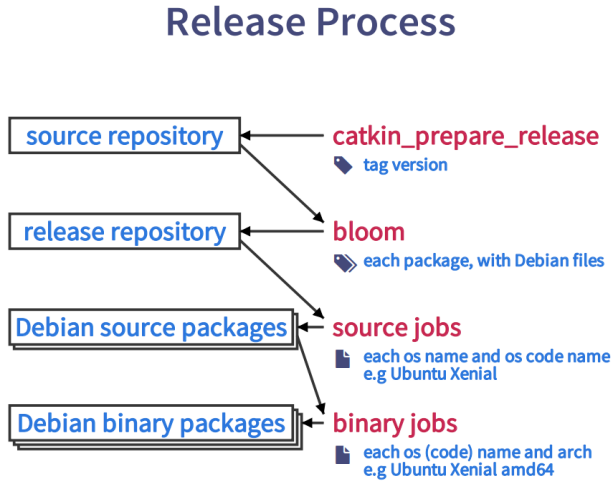

Figure 37: The Release Process followed in the ODIN ROS buildfarm.

B.1.3 Source Jobs

Source jobs use a Debian tool called `gbp-buildpackage` to build Debian source package(s) found in the relevant release repository branch. The following figure presents the process:

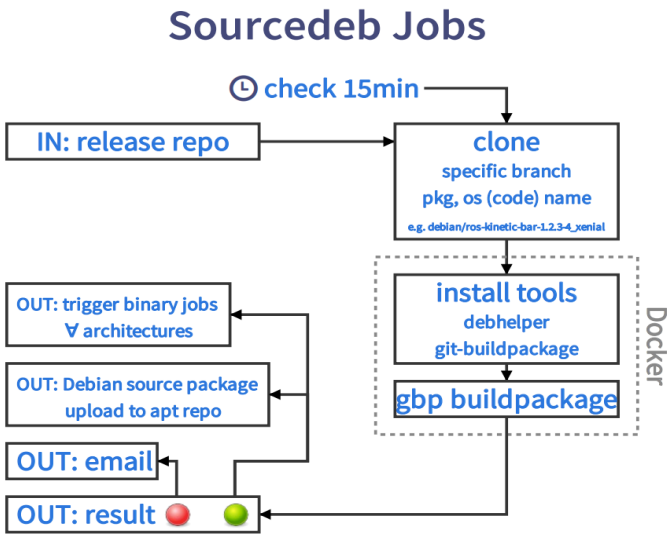

Figure 38: Sourcedeb Jobs running in the ODIN ROS buildfarm.

B.1.4 Binary jobs

Binary jobs utilize the Debian source package (built from the Source jobs) and create a Debian binary package via a Debian tool called `dpkg-buildpackage`. The following figure presents the process:

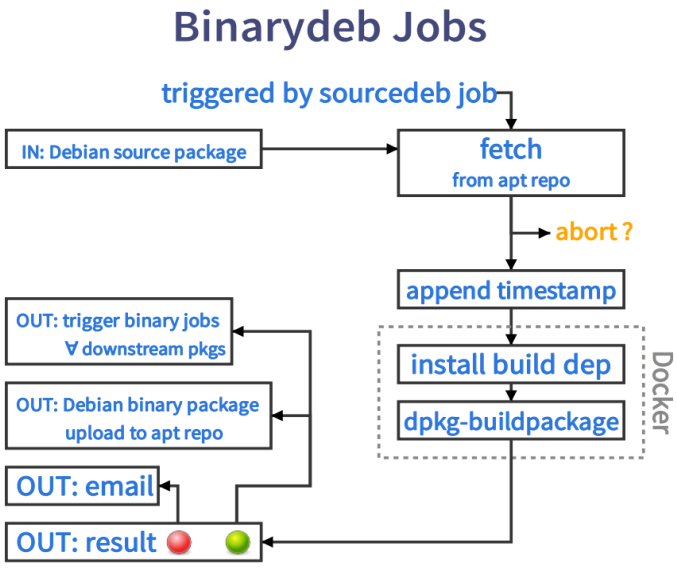

Figure 39: Binarydeb Jobs running in the ODIN ROS buildfarm.

B.1.5 Doc jobs

Doc jobs generate the API docs and meta information for each package by using a tool called `rosdoc lite`. The following figure presents the process:

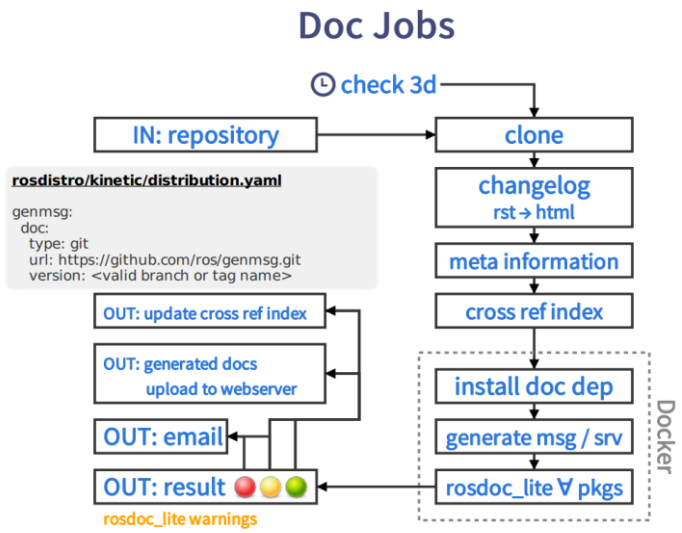

Figure 40 Doc Jobs running in the ODIN ROS buildfarm.

B.2 ODIN ROS buildfarm Apt repositories

B.2.1 Shared Storage between Pods (Jenkins and Apache)

Jenkins Pods (with Docker-in-Docker (dind) and reprepro) need to make changes that persist, either to the Jenkins internal state (e.g., by creating/deleting new/old Jenkins projects) or to the public APT repo contents (i.e. editing the `/var/repos` volume). To achieve this, we followed a design that is shown in the following figure.

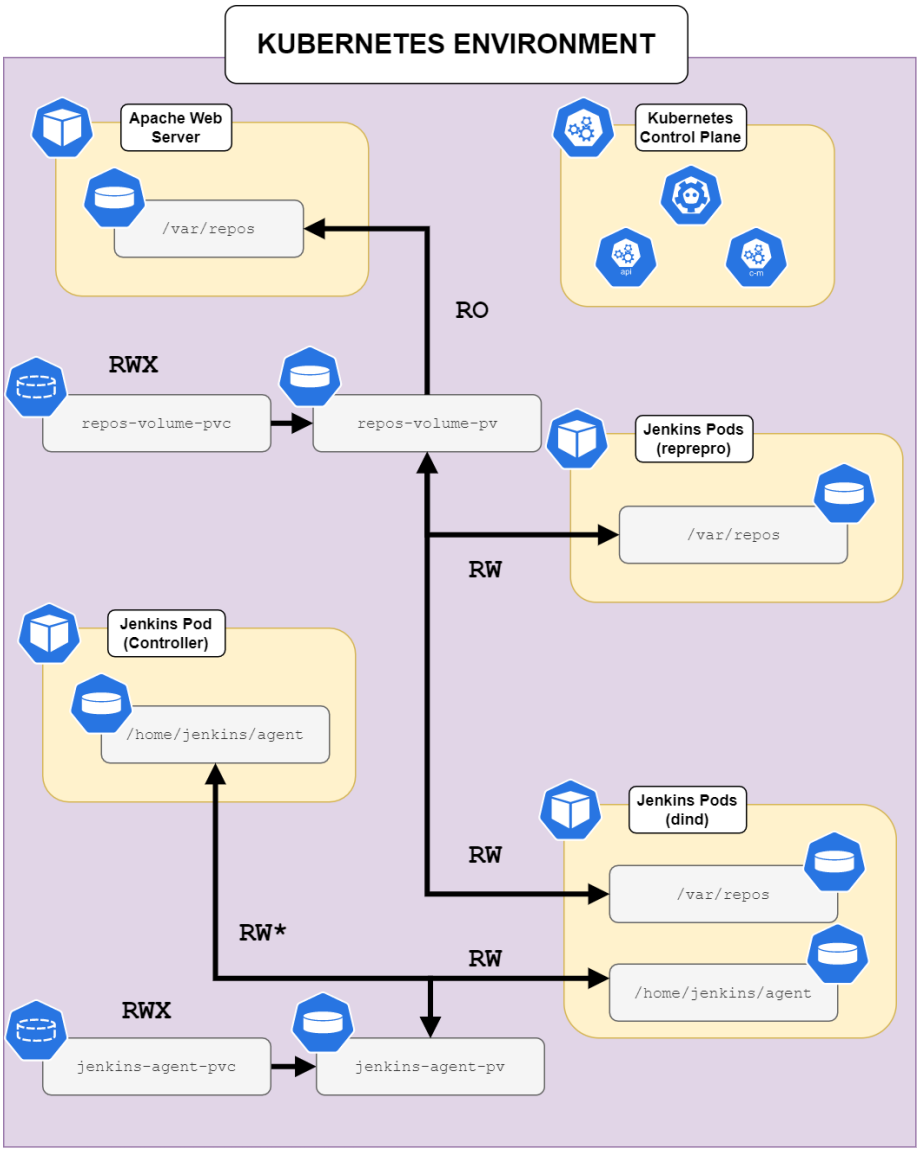

Figure 41: Design of the ODIN ROS Buildfarm in Kubernetes.

These changes led us to modify the official Jenkins Helm chart. Some observations related to the above figure are presented below.

For the ROS buidfarm to work on Kubernetes, some volumes needed to be shared between multiple pods. These volumes were:

- The `/var/repos` stored in a volume, across the pods running in our cluster. This path is shared between the following pods:
  - The `apache` pod is a web server that serves the contents of the `/var/repos` directory in the `repo.thlrnd.com` address.
  - The `dind` pods have outputs that usually need to be saved in the `/var/repos` directory.
  - The `reprepro` pods need access to this directory to create new content which will be shown by the apache pod.
- The `workspace` volume. This is done since there are jobs that run Groovy System scripts. These scripts need to run on the master node (the `Jenkins controller`) and

these scripts usually change files in paths below the $WORKSPACE directory. Such jobs are e.g. the -reconfigure jobs.
- All the shared volumes were created in an **NFS server**, also deployed in the cluster with a helm chart.

B.2.2 Modification of Docker commands

Since the Jenkins pods were running in Jenkins-agnostic Kubernetes nodes, the `docker` commands linking a host path as a volume, needed to be changed. Thus we changed the `docker run` commands to a sequence of `docker create, docker cp , docker start`, and `docker rm` commands. The main issue behind these changes was that the `docker run` commands would usually locally bind some paths from the host with the -v parameter. This could not work anymore since the provisioned machines in the cluster wouldn't have these paths. Thus, a necessary change was first to create the container, then copy all the necessary files needed for the commands to run successfully, then start the container, then copy the outputs of the commands, and finally remove the container.

B.2.3 Implementation of a locking mechanism that is NFS-safe

This change was necessary since the pods that used the `reprepro` program for changing the APT repo in the `/var/repos` volume, should synchronize. We adopted the `flufl.lock` Python package that ensures proper synchronization (it does this by creating new files based on a `lock` file, which are unique for each process - via its PID, timestamp among other mechanisms - the `open()` system call is atomic in POSIX systems). The non-Kubernetes implementation didn't have such an issue, since the `reprepro` program is in the `repo` machine and changes made through this one program (although different processes) could be synchronized effectively using traditional synchronization techniques.

B.2.4 Ability to work with private repos/packages

With the changes implemented, the user flow is the following:

- The users that want the ODIN ROS buildfarm to build their packages, need to create credentials for their repos in `buildfarm.thlrnd.com`. By doing so, they acquire a `keyID`.
- They use the `keyID` in the YAML file. This change is then used by the buildfarm to fetch the GIT repo and build the package.

B.2.5 Removal of the SSH agents and servers

Since everything was running in the same cluster and a pod could be created easily to write to specific (shared) paths (e.g. `/var/repos),` the original SSH functionality was deemed useless and was removed from the Jenkins buildfarm code (`ros_buildfarm`), both in the server deployment and in the Jenkins build steps.