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

D3.10 ODIN Platform v1

Deliverable No.	D3.10	Due Date	28/02/2022
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
Dimitrios Giakoumis	CERTH	dgiakoum@iti.gr
Konstantinos Votis	CERTH	kvotis@iti.gr
Andreas Kargakos	CERTH	akargakos@iti.gr
Konstantinos Flevarakis	CERTH	kostisfl@iti.gr
Ernesto Iadanza	UoW	ernesto.iadanza@warwick.ac.uk
Pilar Sala	MYS	psala@mysphera.com
Pablo Lombillo	MYS	plombillo@mysphera.com
Alejandro Medrano	UPM	amedrano@lst.tfo.upm.es
Eugenio Gaeta	UPM	eugenio.gaeta@lst.tfo.upm.es
Daphne Plati	FORTH	daphni.plati@gmail.com
Fanis Kalatzis	FORTH	tkalatz@gmail.com
Luis Carrascal Crespo	INETUM	luis.carrascal@inetum.com
Antonio-Jesus Gamito-Gonzalez	INETUM	antonio-jesus.gamito@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
Pablo Lombillo	MYS	plombillo@mysphera.com
Pilar Sala	MYS	psala@mysphera.com

History

Date	Version	Change
04/02/2022	0.1	Initial draft containing table of contents and first skeleton content.
25/02/2022	0.2	Updated architecture diagram and skeleton content. Partner assignments.
31/03/2022	0.5	Collected input from partners.
08/04/2022	0.9	Collected input from partners and further contributions.
20/04/2022	0.17	Further updates and finalization of peer review version
29/04/2022	0.20	Modifications based on peer reviews
29/04/2022	1.0	Final version

Key data

Keywords	System architecture, components, flow of operations
Lead Editor	Ilias Kalamaras (CERTH)
Internal Reviewer(s)	Dimitrios I. Fotiadis, Daphne Plati (FORTH) Alejandro Medrano (UPM)

Abstract

D3.10 “ODIN Platform v1” presents the design of the first version of the ODIN platform. The deliverable focuses on the specification of the ODIN platform architecture, showing how the platform components are connected to each other. Each component of the architecture is described in detail, covering Key Enabling Resources, Hospital Information System, resource gateway, resource management and federation, high-level platform services and management, external communication, security aspects and the DevOps infrastructure. The deliverable includes sequence diagram showing how the platform components interact with each other to perform the core functions of the platform and to support the ODIN use cases. The deliverable also includes information on the deployment of the ODIN platform and the related infrastructure. Finally, the deliverable includes preliminary notes regarding system integration, documentation and validation, to be further elaborated in the next versions.

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

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 use cases. This is the first version of the deliverable, to be followed by two further versions. The scope of the current deliverable is mostly focused on the description of the architecture, its components and their deployment, while next versions will focus on system integration, 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 each component of the ODIN architecture, grouping the components in functional categories.
- Section 4 presents sequence diagrams for the main operations performed in the ODIN platform, and for example scenarios in relation to the ODIN use cases.
- Section 5 provides notes related to the integration of the platform. More details will be provided in future versions of this deliverable.
- Section 6 describes issues related to the deployment of the ODIN platform at the envisioned sites.
- Section 7 contains manuals for the use of the platform, targeted to end-users, deployers and developers. More details will be provided in future versions of this deliverable.
- Section 8 contains information about testing and validation of the platform components. At this stage, it only includes information about the strategy to follow for testing. Results of actual testing will be available in future versions of the deliverable.
- Finally, Section 9 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	D3.10 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	D3.1 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.

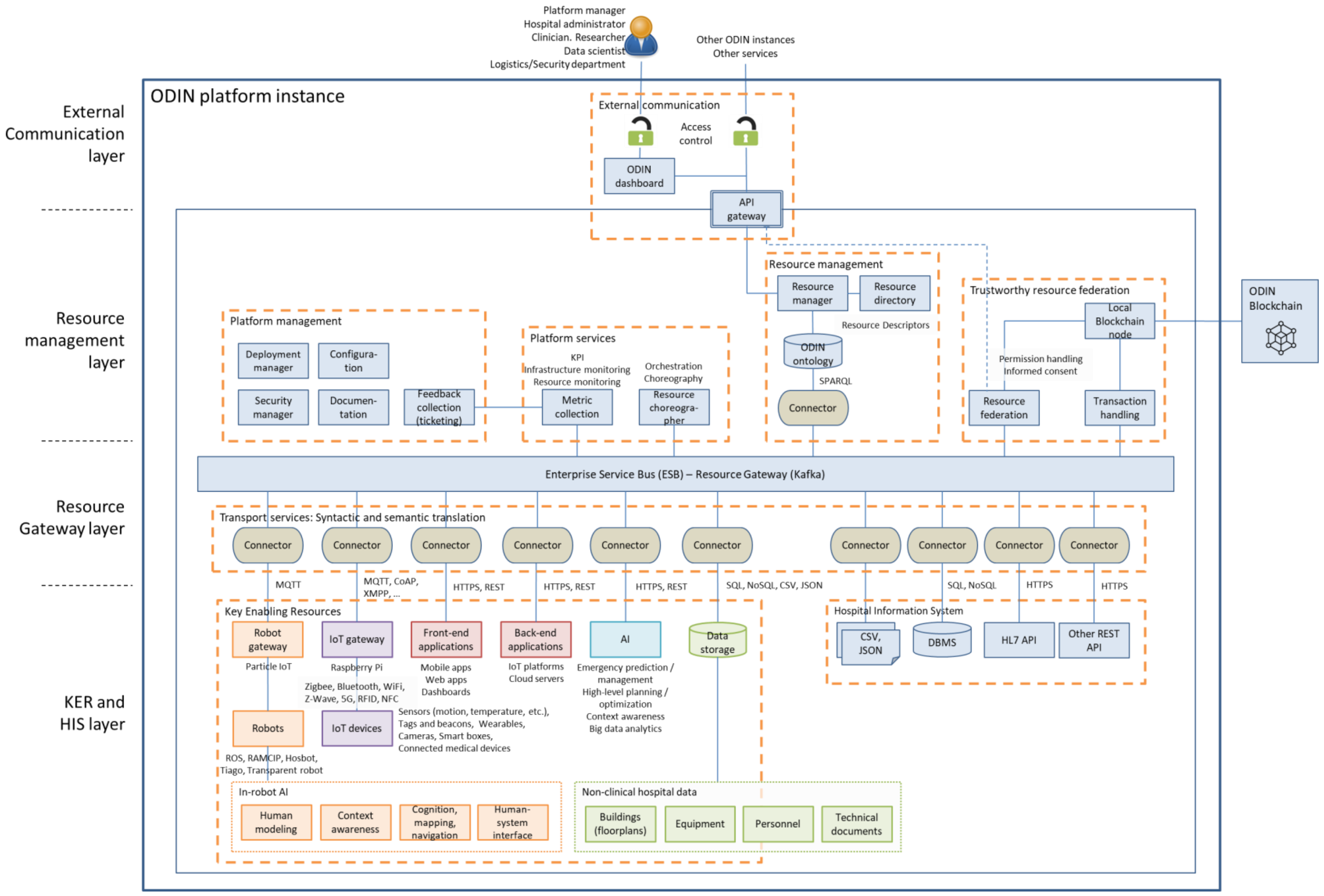


Figure 1: The architecture of the ODIN platform.

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. 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.

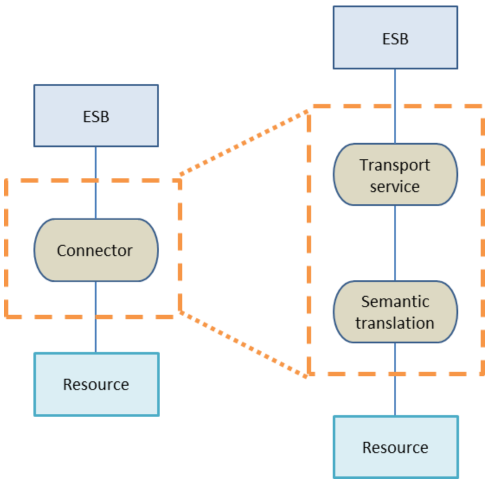


Figure 2: Each connector handles both semantic translation and transport to the ESB.

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.

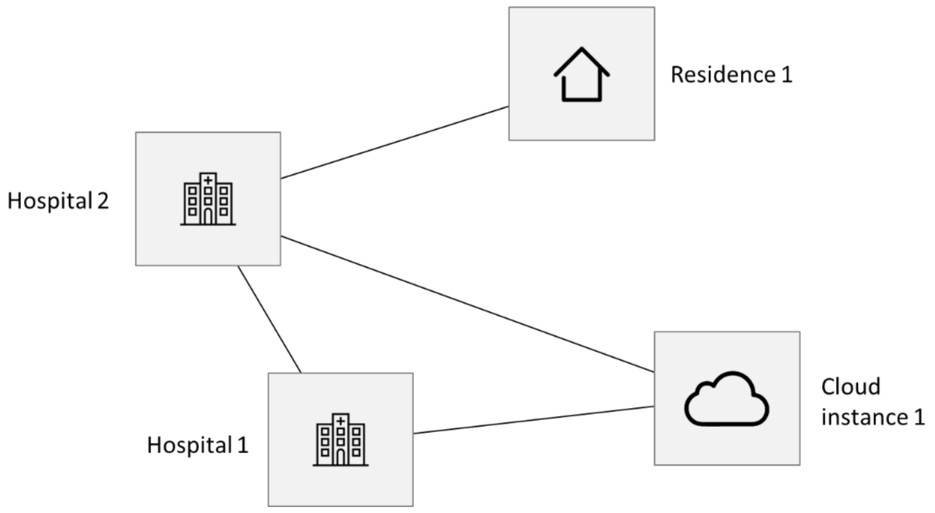


Figure 3: Connection of ODIN instances in a decentralized scheme.

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 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”.

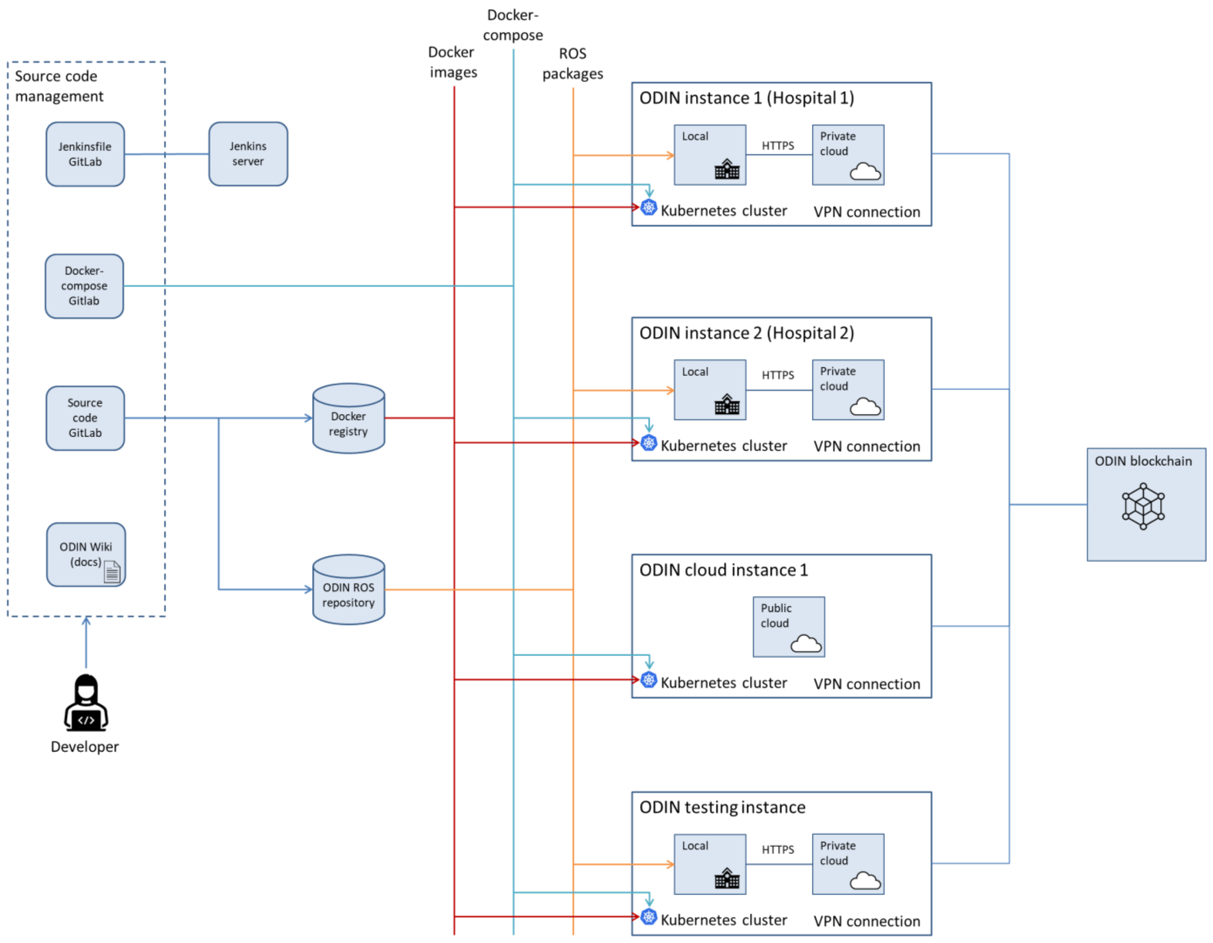


Figure 4: ODIN development and deployment infrastructure.

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. Section 3 provides 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
	Front-end and back-end services	3.1.4	WP3-WP6, WP10	Any in related tasks[1]
	Data storage	3.1.5	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	3.3.1	T4.3	D4.2, D4.2-D4.4
	Transport services	3.3.2	T4.3	D4.2, D4.2-D4.4
	Semantic translators	3.3.3	T3.2	D3.2-D3.3
Resource manage-ment	ODIN ontology	3.4.1	T3.2	D3.2-D3.3
	Resource manager	3.4.2	T4.1, T4.2	D4.1, D4.2-D4.4
	Resource descriptor	3.4.3	T4.2	D4.2-D4.4
	Resource directory	3.4.4	T4.2	D4.2-D4.4
Resource federation	Resource federation	3.5.1	T4.4	D4.5-D4.7
	Transaction handling	3.5.2	T4.4	D4.5-D4.7
	ODIN Digital Ledger Technologies	3.5.3	T4.4	D4.5-D4.7

[1] Front-end and back-end services are not related to a particular task, but may be already available by consortium partners or may be developed within the technical WPs as needed, or may be developed by external parties through open calls.

Platform services	Metric collection	3.6.1	T4.6	D4.2-D4.4
	Resource choreographer	3.6.2	T4.5	D4.5-D4.7
Platform manage-ment	Deployment manager	3.7.1	T3.1	D3.10-D3.12
	Access control manager	3.7.2	T3.1, T3.3	D3.4-D3.6, D3.10-D3.12
	Platform configuration	3.7.3	T3.1	D3.10-D3.12
	Platform documentation	3.7.4	T3.4	D3.7-D3.9
	Feedback collection	3.7.5	T3.4	D3.7-D3.9
External communi-cation	API gateway	3.8.1	T4.3	D4.2-D4.4
	Administration dashboard	3.8.2	T3.1	D3.10-D3.12
Horizontal aspects	Platform security	3.9	T3.3	D3.4-D3.6
	DevOps infrastructure	3.10	T3.1	D3.1, D3.10-D3.12

3 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 4 provides examples of how these components interact with each other to accomplish main tasks.

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

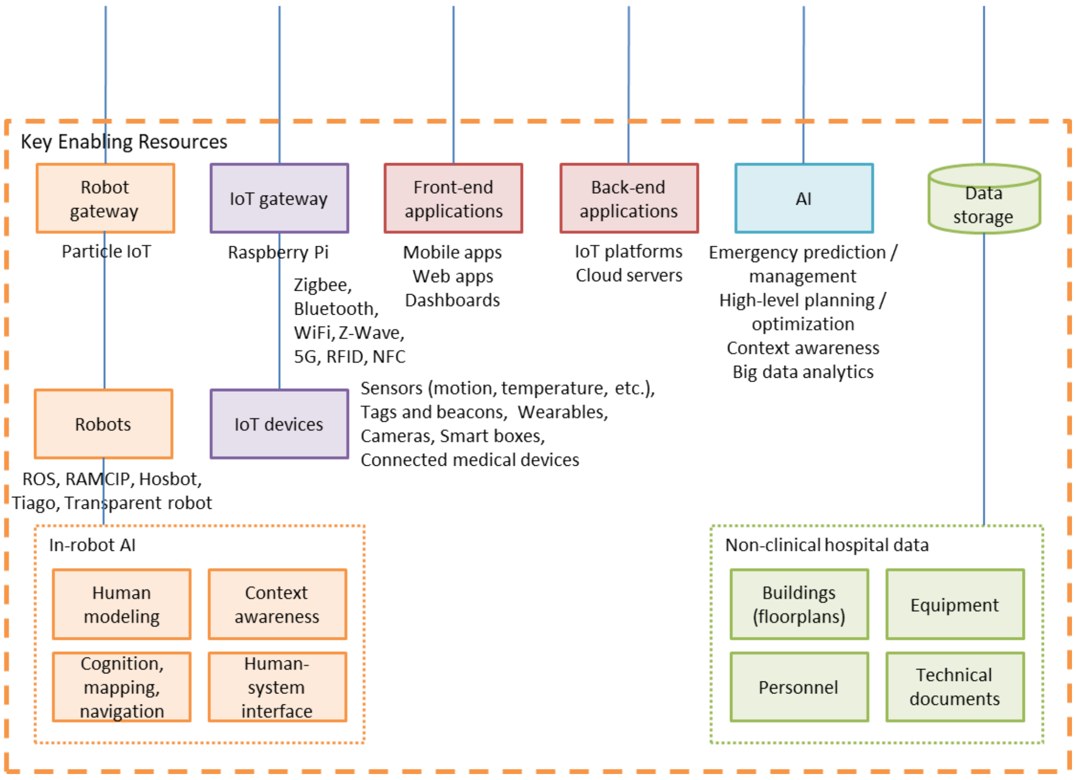


Figure 5: The ODIN Key Enabling Resources.

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

There are different robots that will be deployed for different tasks in hospital environments for the ODIN system (see Figure 6). They have been tested in step 2 of WP5, as reported in D5.7 "Technology Integration Periodic Report v1", and are listed as follows:

- HOSBOT: an indoor robotic mobile platform intended for 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, and proximity sensors for enhanced context awareness.
- TIAGo robotic platform: 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.
- RAMCIP: robotic platform developed to act as an assistant for patients. It handles emergencies by calling the caregivers when required, monitors and guides the patients in everyday tasks and handles video-talks with relatives.

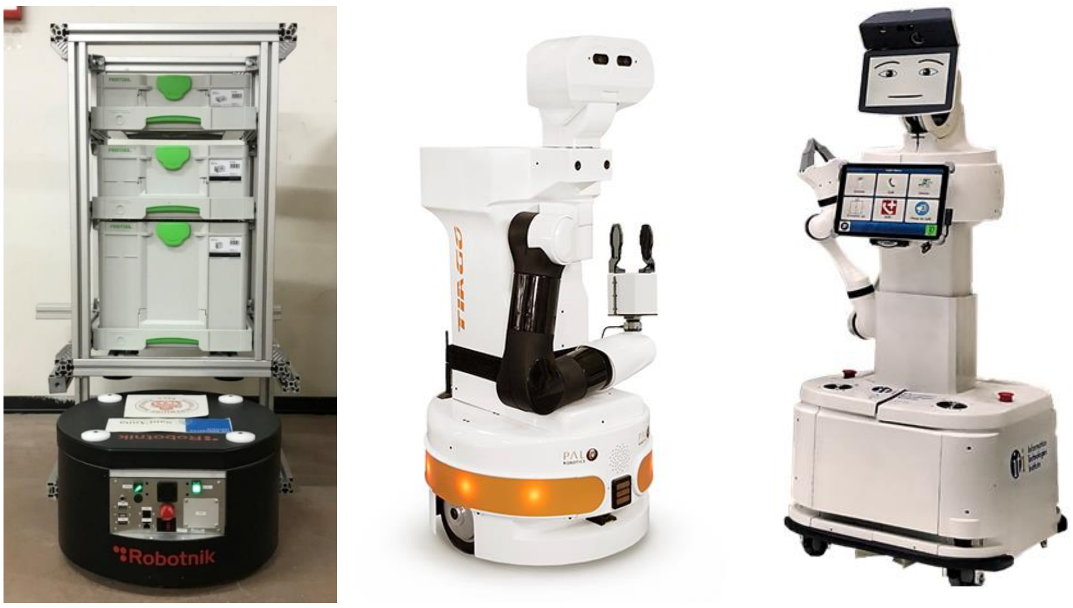


Figure 6: Robotic platforms used in the ODIN project. From left to right: HOSBOT, TIAGo, RAMCIP.

3.1.1.2 In-robot AI

The different robots used for the development of the project will integrate various sensors that will be used to provide the data required by the different algorithms responsible for the navigation, perception, modelling, and communication. These sensors include IMU's, cameras, lasers and/or encoders.

2D Lidars are used for navigation, localization and obstacle avoidance. The localization is performed by an algorithm called "AMCL" that is constantly matching a predefined map with the current laser data. Autonomous navigation is performed by using the "Move Base" open-source algorithm. Given a goal in a map, the selected mobile base will try to accomplish it while analysing dynamically the environment and avoiding any dynamic obstacles that might appear.

The cameras included in the different robots enable recognition algorithms based on AI that enable an automatic detection of predefined objects or shapes that can be used for identifying patients or useful objects used for certain tasks in the hospital.

The above AI algorithms run on the robotic platform itself and are distinguished from the AI algorithms that are used individually as KERs (Section 3.1.3). The latter are external algorithms that can be called by any resource via the main messaging bus, undergoing semantic/syntactic translation as described in Section 3.3. However, the in-robot AI consists of algorithms that run on top of the robotic platform to directly provide functionalities that are needed for the robot operation.

3.1.1.3 Robot gateway

Each of the *e*-robots developed within the ODIN project will be connected to the rest of the hospital environment throughout the Enterprise Service Bus (ESB) (Section 3.3.1). Information is shared following two different layers of communication. A first layer, *intra-robots*, will be based on MQTT. MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). After an overview of the current communication protocols, MQTT has been selected due to its performances (*i.e.,* lightweight publish/subscribe messaging for connecting remote devices, limited network bandwidth necessary). A second layer, *intra-modules*, will be instead based on Kafka. An "ad-hoc" connector is under development for bridging the overall information and translate messages between the two communication layers in a bidirectional way (*i.e.,* from *e*-Robots towards ESB and from ESB towards *e*-Robots). The robot communication protocol has to be intended as general and scalable in order to be modified to fit the information required by the specific Use-Case (i.e., images, robot mission details, stock of consumables, environmental room conditions).

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

IoT devices to be used within ODIN include mostly monitoring devices that capture the state of the hospital environment in real time. Representative IoT sensors that can be used as part of the ODIN use cases include the following:

- Environment sensors (temperature, humidity, luminosity, etc.)
- Activity sensors (motion sensors, cameras, beacons, CO2, etc.)
- Presence sensors (pressure sensors to detect fill status of smart drawers, door open/close sensors, etc.)
- Wearable sensors (e.g. smartwatches capturing the health status of an individual, such as heart rate, respiration rate, oxygen saturation, etc.)
- Smart medical devices (scales, blood pressure monitors, oxygen supplies, etc.)
- Fire sensors
- Smart appliances (electrical equipment, connected lightbulbs, smart plugs, etc.)

Apart from collecting measurements, specific IoT devices can operate as actuators, performing an action dictated by an external system. For instance, a lightbulb can receive a command to change its luminosity, or a wearable device can send a notification to the user.

IoT devices usually communicate with a device such as a mobile phone, or a PC, which collects and processes the measured information and transmits commands. Communication between IoT devices and data processing systems often passes through an IoT gateway, which is described below in Section 3.1.2.2.

3.1.2.2 IoT gateway

An IoT gateway is a physical device or software program that will serve as the connection point between the sensors and intelligent devices and the ESB, through the corresponding connector. It can also be used to pre-process data locally at the edge as well as to provide additional security.

The IoT gateway needs to support several protocols that could be used by the IoT devices deployed in the Hospital, such as BLE, Z-wave, Zigbee and others. In its simplest role, an IoT gateway will buffer communications between the sensors and ODIN platform.

The concrete IoT gateways to be used will depend on the final selection of the IoT resources to be deployed in the reference use cases. For instance, when deploying the RTLS system, the MYSPHERA Beacon v2.3 (BCNEW0203) will be used, that collects location data from the tag sensors via BLE and sends location information to the ODIN platform through an Ethernet link, which also provides power supply thanks to Power-over-Ethernet (PoE) technology.

Another potential multipurpose IoT gateway under consideration will be based on Raspberry PI, which provides an effective hardware base, highly modular, for tailoring it to the specific needs of the reference use case. The latest Raspberry Pi 4 Model B includes a high-performance 64-bit quad-core processor, hardware video decoding at up to 4Kp60, up to 8GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0, and PoE capability, with a guaranteed availability until 2026. Raspberry Pi is also able to support Kubernetes clusters.

The IoT gateway could include some sort of intelligence, creating an edge computing infrastructure. This is a feature that must be discussed and is not decided yet but this would let gateways take decisions with less latency. For example, including measures and analytics with rule-based decisions, the gateways could perform certain tasks on their own.

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.

In addition to the implementation of individual AI models for specific tasks, a Federated Learning (FL) pipeline will be created that will support the AI models developed within the ODIN platform. FL overcomes the main drawbacks related to AI in healthcare which are privacy and security. The FL infrastructure is based on a centralized server approach where the server, called “coordinator”, manages the FL framework. The actual task of training on local data is carried out at sites by components, which are called “workers”[2] and deployed at the locations of the data. The coordinator shares the base model and the training configuration with workers. At each site, a worker trains the model on local data and sends the model updates to the coordinator. Then, the coordinator aggregates the model updates from multiple workers and shares the updated model with workers for the next training round. This process continues until a desired level of performance, or a pre-configured number of training rounds is reached. It should be noted that, in case of multiple use cases that are simultaneously running, the system setup complexity increases. For a large number of use cases, this requires a workflow component that keeps track of the state of the system. In a nutshell, we need an orchestrator component that takes all this responsibility at the coordinator.

From the deployment perspective, the main requirements for an end-to-end pipeline are a centralized server to coordinate the orchestration, the support for different use cases in parallel, a monitoring of the FL pipeline, as well as requirements related to the healthcare settings. In fact, stringent regulations apply to enable access to hospital networks, mainly due to data privacy concerns. Figure 7 depicts the different components of the FL framework. The main components are:

1. The user interface provided to the user of the system. This web-based user interface supports all the operations of the FL framework. It should be noted that the user of the FL framework should also have access to an installable Python toolkit which supports all the basic operations like creating use cases, creating and uploading model artifacts by connecting to the coordinator APIs.
2. The coordinator that plays the role of an orchestrator, responsible for managing execution of FL framework for multiple use-cases in a seamless manner. The coordinator consists of admin service hosts which are a set of APIs responsible for use case management, like hosting a new use case, registration of workers and creating a new configuration. The coordinator is responsible to execute a job, defined as a single

[2] Not to be confused with the ODIN concept of “eWorkers”, which is the notion of enhanced hospital workers using the Key Enabling Resources. The term “workers” used in this section is a standard terminology used in Federated Learning and applies only in relation to this.

execution of a use case algorithm in the FL framework. Furthermore, it stores the base models for all the hosted use cases, together with their relevant metadata, such as versioning and provenance. These are the models shared with the workers for performing the training during the first round. The database is a Postgres-backed data store responsible for persisting all information. The configuration service is a set of APIs that allows to configure the training pipeline parameters like the number of training rounds, the optimizer and the model hyperparameters (learning rate and any other common configurations that need to be shared with the worker).

3. Lastly, the worker which is a component that executes tasks, contributing to model training in one or more use cases.

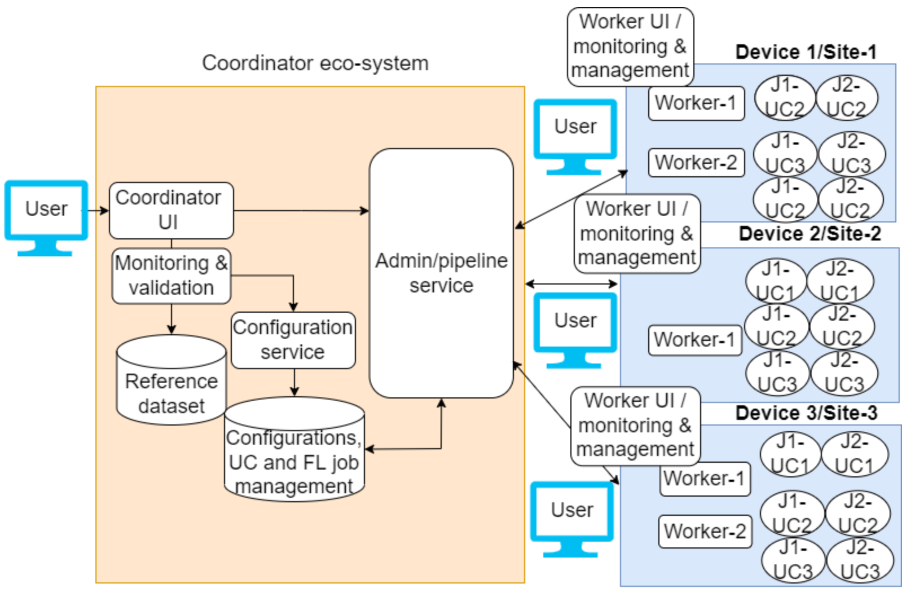


Figure 7: The Federated Learning architecture as AI-system in the ODIN platform.

The Federated Learning approach presented above fits well in the overall ODIN architecture, which aims to be decentralized. With respect to the ODIN architecture (Figure 1), worker and coordinator systems are regarded as AI KERs that connect to the ODIN platform through the ESB. In a typical example of FL, several hospitals may host workers as AI resources in their premises, while another hospital, or a cloud ODIN instance, may host the coordinator, also as an AI resource. The coordinator can communicate with the workers through the Resource Federation mechanisms (see Section 3.5).

An example Federated Learning setup using the ODIN infrastructure (ESB and resource federation) can be seen in Figure 8. There are two worker nodes installed in two different hospitals. Each node has access to the data of each hospital only. The AI service at each node has access to the hospital data through the ESB, following any syntactic/semantic transformations that are needed through the data connectors. The local dataset at each node is used to train a local AI model. Once the local models are trained, they are transmitted to the coordinator node to update the federated model. In this example, the coordinator node is implemented as a cloud ODIN instance, with a single AI KER holding the coordinating AI logic. The local models are transmitted to the remote coordinator through the resource federation functionalities. In other words, each hospital requests permission to use the remote coordinator node to retrieve and update the federated model. Once permission is granted, a channel is opened between the worker and coordinator nodes, passing through the API gateways of the corresponding ODIN instances. Once new data are obtained by each hospital, the federated model can be updated again by each local worker.

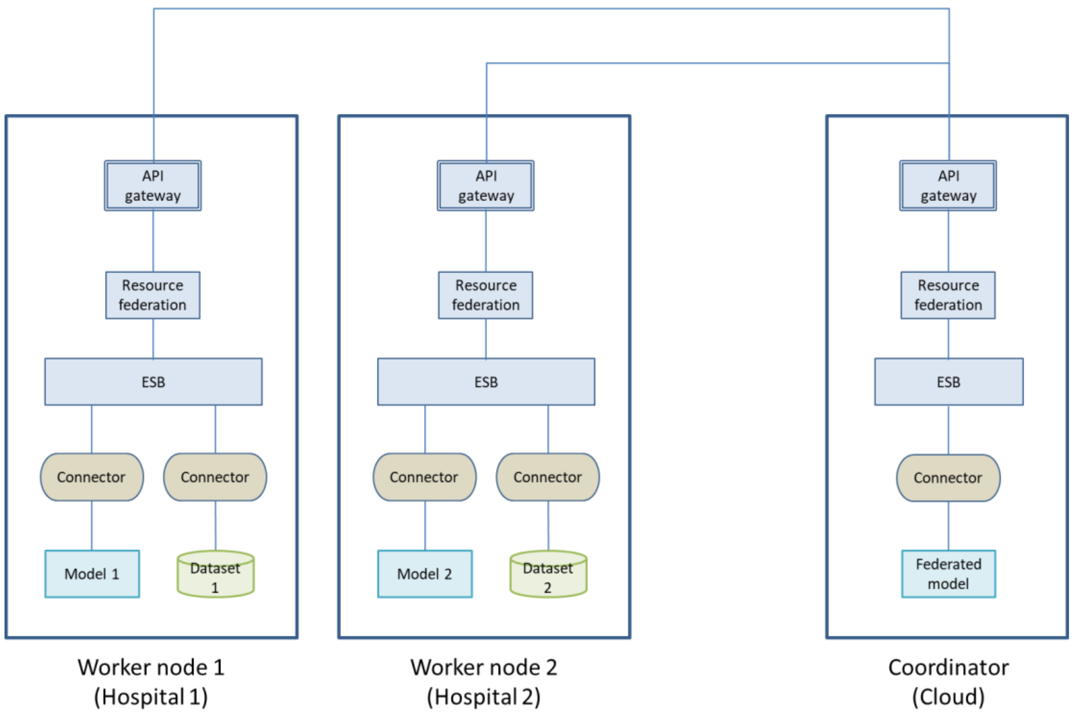


Figure 8: Example implementation of federated learning using the ODIN infrastructure.

3.1.4 Front-end and back-end services

The ODIN architecture aims to allow external technologies to be integrated in hospital workflows, in the form of Key Enabling Resources. Apart from the already mentioned key technologies (robotics, IoT, AI), external software services are also foreseen as resources that can add value to the system. Examples include already developed mobile applications for contact tracing or existing solutions for hospital inventory tracking. In order to include such resources in an ODIN instance, they need to pass through the same semantic/syntactic transformation mechanisms as all other resources.

Front-end applications are ones that provide a Graphical User Interface (GUI) for interaction with a human user, while back-end services are processes that run on a server and usually expose functionalities through and Application Programming Interface (API), that can be used by front-end applications. Examples of front-end applications include web-based IoT monitoring dashboards, big data analytics dashboards, mobile applications for contact tracing, etc. Such applications are usually accompanied by one or more back-end services that provide supporting functionality, such as data storage, computation, etc.

Consortium partners have already implemented front-end and back-end applications that can be integrated in ODIN, or may implement ones within the context of ODIN. API designs have started since the early days of the project and will be extended to satisfy the needs of the ODIN use cases. Furthermore, existing third party services may also be included if necessary, to avoid having to reinvent systems that already exist, e.g. existing hospital inventory management systems.

The back-end services to be integrated in ODIN will be mostly based on a REST API design. Back-end services may provide access to a database (DB) or to computational functionalities, such as data analytics and AI. Through the REST API, basic operations can be done onto ODIN databases (CRUD – Create, Read, Update, and Delete). Based on the actions required in each request, a different HTTP request is issued:

- GET: HTTP request when requiring information from the database
- POST: HTTP request when inserting a new entry in the database
- PUT: HTTP request when updating an already existing entry in the database
- DELETE: HTTP request when deleting and entry from the database

Depending on the request type, the data exchanged is either in the form of JSON (JavaScript Object Notation) objects or included in the URL (for GET and DELETE requests).

For security reasons, all requests are encrypted with the secure protocol TLS (Transport Layer Security). All actions requiring authentication need to have a security token included in their header. This token is acquired by the client by the login request.

For the development of necessary REST APIs, Node.js is considered which is extremely fast and scalable. Over Node.js, frameworks such as Loopback[3] can be used for a simplified access to the databases, supporting both relational and noSQL. The REST API allows an easier access to the database by all the applications in the cloud and the backend is the unique responsible for the complexity of data, the integrity and the protection. The users do not need to know how and where the data are stored but they have only the most suitable interface for inserting, updating, reading and deleting with security. Deployment of the services can be based on solutions such as the Payara Application Server[4] to allow seamless integration with external computational clients (any computational device that is supported by a programming language). The communications are based on HTTP (Hypertext Transfer Protocol), where also the REST architecture is residing.

The front-end applications and the back-end APIs are being developed in collaboration with the partners of the consortium who oversee their own application development in the ODIN ecosystem.

To integrate an existing service in ODIN, the proper way would be to connect the front-end and back-end parts separately to the ODIN platform, so that they communicate through the ODIN message bus. If such an approach is straightforward to implement, it will be followed. However, since existing external applications may not allow for extensive modifications to bring them to such a scheme, direct connection between the front-end and the back-end may be retained, considering the whole system as a single resource. This resource may be connected to the ODIN platform e.g. through its backend, to allow it to store/retrieve data or other information from the other KERs or ODIN components. These two alternative setups are depicted in Figure 9.

[3] LoopBack framework, https://loopback.io/

[4] Payara Application Server, https://www.payara.fish/

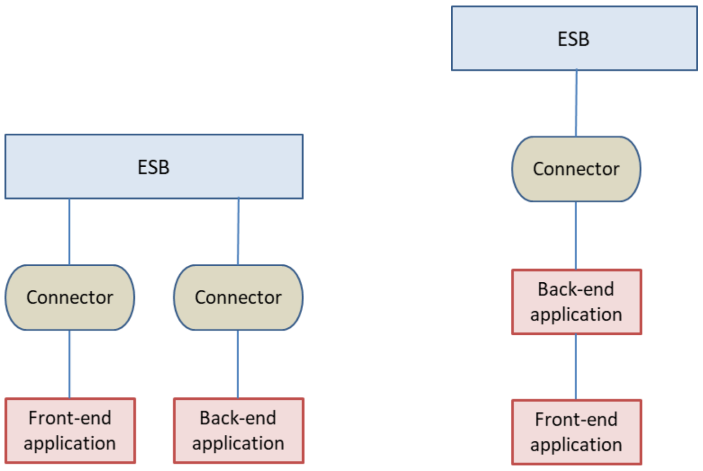


Figure 9: Different setups for connecting front-end and back-end applications. Left: As separate KERs connected through the ESB. Right: As directly connected services, with only the back-end connected to the ESB.

3.1.5 Data storage

Within the ODIN architecture, data are also considered an enabling resource. 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.

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.

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 10 depicts the HIS-related components within the ODIN architecture.

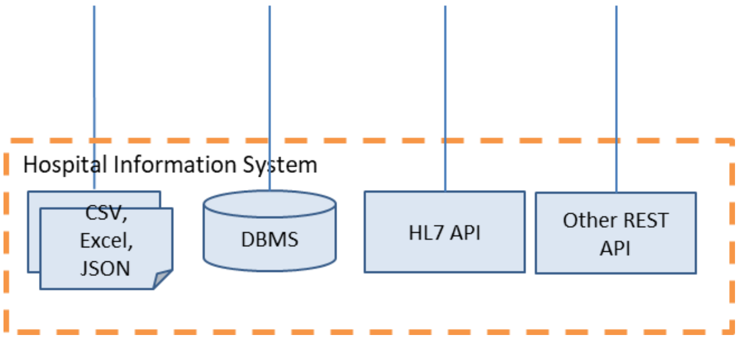


Figure 10: The Hospital Information System connected to the ODIN platform.

Figure 11 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.

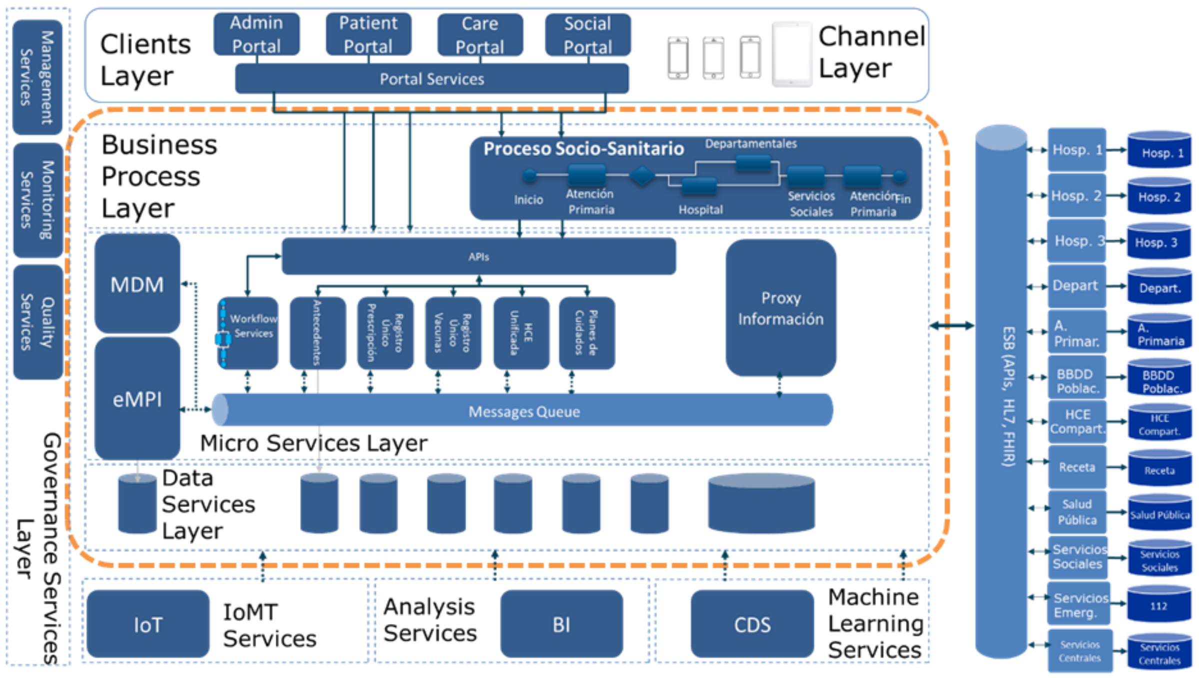


Figure 11: Internal details of a Hospital Information System.

3.3 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 12 depicts the ODIN Resource Gateway as it appears within the overall ODIN platform architecture (Figure 1).

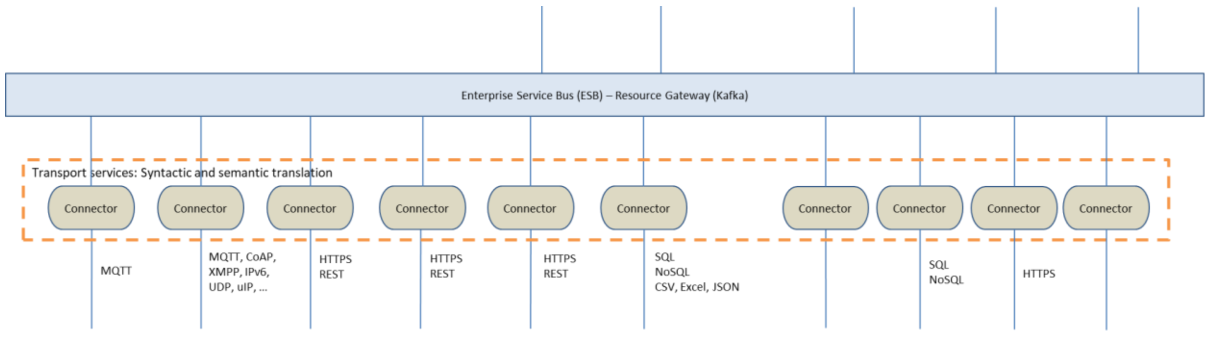

Enterprise Service Bus (ESB) – Resource Gateway (Kafka)
Transport services: Syntactic and semantic translation
Connector
MQTT
MQTT, CoAP, XMPP, IPv6, UDP, uIP, ...
HTTPS REST
HTTPS REST
HTTPS REST
SQL NoSQL CSV, Excel, JSON
SQL NoSQL
HTTPS

Figure 12: The ODIN Resource Gateway.

The Resource Gateway will be implemented following an Enterprise Service Bus (ESB) approach, as a framework for message-oriented middleware with a rule-based routing and mediation engine that provides an implementation of the Enterprise Integration Patterns to configure routing and mediation rules and being extended to incorporate semantic capabilities with the ODIN selected standard. This approach has as benefits higher adaptability, maintainability and scalability.

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

- the **Enterprise Service Bus**
- the **transport services**, for syntactic transformation of messages
- the **semantic translators**, for semantic transformation of messages

These modules are detailed in the following sections.

3.3.1 Enterprise Service Bus (ESB)

This is a message oriented middleware that connects all core components and resources, enabling their communication through a publish/subscribe approach. It is implemented using a global communication solution, most probably Kafka.

The ESB will be implemented to achieve high scalability and high throughput, it will be distributed, and will store every message sent to ensure reliability and high availability. The ESB will be the transport of the messages, and ODIN must define a canonical messaging language so that the components can communicate together. The messaging will be extendable to support future use cases.

The following requirements will be satisfied by the ESB:

- Enable the exchange of messages within components and resources in a secure way
- Allow for metric, aggregation and data availability to be processed
- Allow for stream processing to support real time or batch processing
- Integrate routing capabilities or priority queues
- Support for dynamic topic management
- Define the message format
- Define a messaging language that serves as canonical messaging system so that all platform components can communicate

The amount of resources connected to the ESB will be considerable. A specific strategy must be selected to create the topics where the messages will be published by resources. It must be discussed whether a general topic such as “Robots” is used, and the robots pick the messages and then filter them using an attribute from the message, or a more concrete topic strategy is used, such as having one topic per robot or resource, for example “Robot-ID”, where ID is created and registered when a new robot or resource is added to the platform. These decisions will be made in the context of T4.3.

3.3.2 Transport services

Each resource and component can have a different communication transport protocol. Some of the most important ones appear at the bottom of Figure 12: MQTT, CoAP, XMPP, HTTPS, SQL, NoSQL, FTP and lots more are just some of the protocols that must coexist. Thus, in order to connect to the ESB, the components need a Connector, which has a Transport Service translator, and a Semantic Translator.

Transport Service is in charge of adapting one communication protocol to the ESB protocol. For example, robots using MQTT publishing to a Mosquito server can publish their messages and then the Connector would read the messages from the Mosquito server and publish them into Kafka. This is the operation of the Transport Service.

There exist several solutions to cover the Transport services, such as Apache Camel components, or creating new ones using Apache Nifi or Apache Kafka using Streams and Connect. Apache Camel has the advantage that several components and connectors are already available to use.

Some of the requirements to be followed by the Transport Services are:

- Transform the messages from one protocol to the ESB protocol
- Log errors
- Allow a setup to control by message the configuration, such as the polling or publishing throughput speed.

3.3.3 Semantic translators

Every resource will need to semantically translate, also known as align, their data to the common ODIN language that is the ODIN ontology, and vice versa. This operation can be approached in several ways. Ideally ODIN resources are ODIN-native and no semantic translation is necessary. When this is not possible, because the resource is already developed and is using another data model, the next option is to employ tools for semantic translation.

The platform will include tools offering semantic translation services. These tools, some of which are listed in D3.2, are mostly based on R2RML; allowing for standardized (i.e. W3C Recommendation) description of the translation. The tools should be flexible enough to connect most of the resources whether they are static or streamed, and independently of the internal format the resources uses. The tools can be offered as support services, or instantiable as part of the resource sidecar for more controlled deployment and operations of the semantic translations.

3.4 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 13, 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.

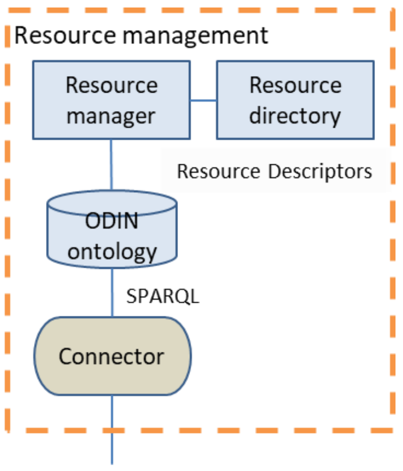


Figure 13: The resource management components.

3.4.1 ODIN ontology

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. Ontologies are the foundation of the Semantic Web, a World Wide Web extension created by the World Wide Web Consortium (W3C) with the purpose of allowing computers to enable network interactions. After a thorough review of the state of the art in terms of existing (and publicly accessible) ontologies related to health care, some ontologies were selected as a starting point to create the ODIN ontology. These cover many health-related knowledge domains, from hospital buildings to organizations, from diseases to treatments, from medical devices to automation and robotics, from artificial intelligence to the web of things.

Starting from these building blocks, the ODIN ontology was designed, whose main goal is to provide a data model for an open digital platform to support 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.

All of this corresponds to the establishment of an ontology capable of specifying any hospital structure through its classes and characteristics, allowing the ontology to be reused. It has now been expanded by the creation of individuals capable of expressing crucial points and pre-determined goals. The result is an ontology with 11 super classes with numerous child classes, such as Device, Domain, Element, ICD9, Interface, Measurement, Occupation, Organization, Sites of Care Delivery, Unit of Measure, WoT. It also includes many object and data properties, as well as several individuals.

To define medical devices, a new ODIN EMDN ontology was created, representing each and every device class in the new European Medical Devices Nomenclature (EMDN), to have any existing medical device available. This choice is, per se, a great achievement, considering that to the best of our knowledge there was no existing ontology ready to represent all the possible medical devices on the European market.

An overview of the ODIN ontology is described in Figure 14. For a detailed description and a complete guidance you can refer to Deliverable 3.2 “Hospital Knowledge Base and ODIN semantic ontology v1”.

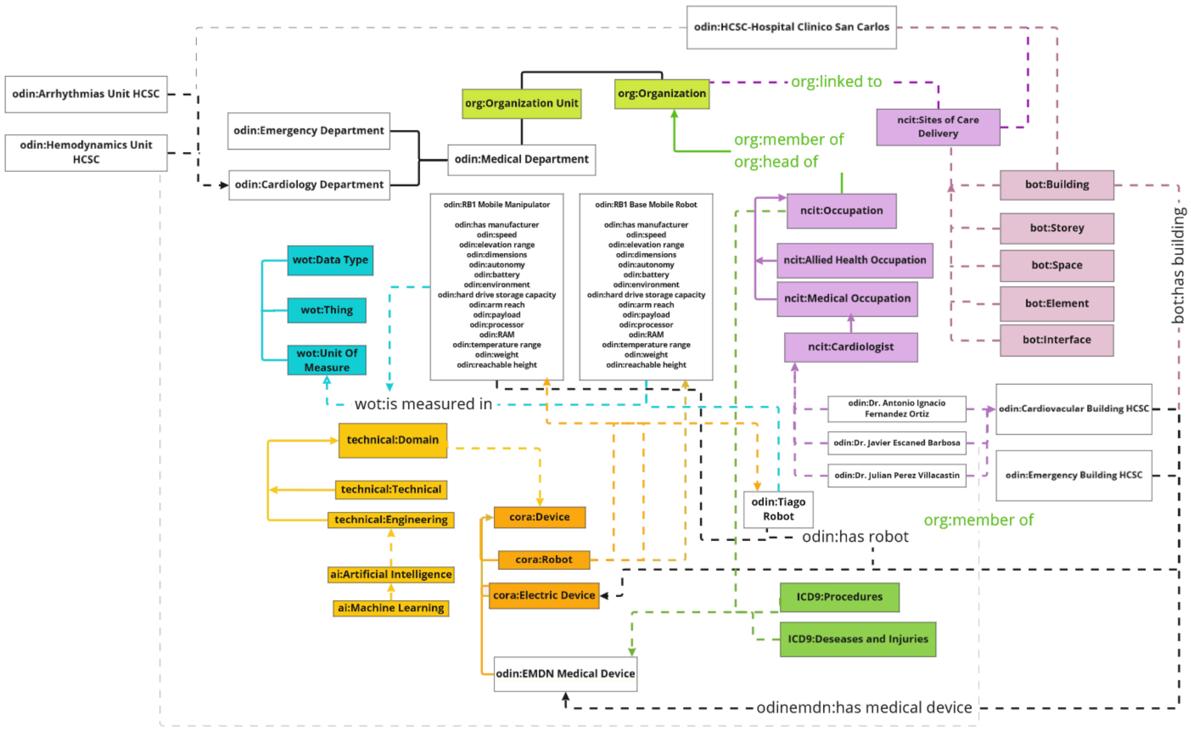


Figure 14: ODIN Ontology overview.

3.4.2 Resource manager

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 RM employs the Resource directory, described in Section 3.4.4, to store the Resource Descriptors and the current status. Through the RM, extended functionality can be offered, for example resource discovery (see Section 3.4.4), which can be used by other ODIN components to automatically collect all compatible resource information and build:

- Metric reporting
- Interactive Choreographer elements to build workflows graphically
- Integrated dashboards were information and access is easily accessible according to the role of the user
- Integral privacy consistency, ensuring that user information and rights are protected and enforced across all resources
- Security consistency, ensuring the resources behave within established parameters.

3.4.3 Resource descriptor

The different resource types that will be managed by the ODIN platform are varied and heterogeneous. Therefore, the Resource Descriptor provides the abstraction layer that is necessary for a homogeneous description of data. It will specify the data structures to support multi-domain exchange of information. This includes the Thing Descriptor that provides common interfaces and will be aligned with the semantic models defined.

The information that the Resource Descriptor manages can be, but are not limited to, the following:

- Semantic Resource Description: What the resource is, and its semantic properties (such as location).
- Resource Services: The services that are offered by the resource.
- Resource Federation: A description of the static properties of the resource with regards to the capabilities it can offer to a federated network. The static properties describe the capabilities of the resource like its availability for remote operation. In contrast, the dynamic properties would be managed through the Resource Federation component (see Section 3.5), which can manage the permissions and conditions under which sharing, and transactions are approved.
- Resource Privacy, Security and Trust: Information pertaining to the PST properties, such as: the services enabling and ensuring GDPR compliance and rights; integrity verifiers, security protocols employed, default role access; signature of resource developer, as well as signature of CA which would provide a higher degree of local trust to the Resource as a component.
- Metric reporting: The types of metrics and KPIs that can be collected with respect to this resource.
- Resource health: The endpoints to verify the current operational status of the resource.
- Resource User Interfaces (UIs): User-operated interfaces to interact with the resource itself, with special emphasis on Web interactions.
- Resource administration: Administrative interfaces for changing the configuration or behaviour of the resource itself.
- Resource documentation: Links to different levels of documentation. For example, user manuals for users, administration manual for system administrators, API documentation for developers.
- Resource deployment: How the resource is deployed, and where.
- Resource communications: A list of required network operations with a description of use. By default, resources will not be allowed to access external networks. This list allows system administrators to review the need and allow this access if needed.

3.4.4 Resource Directory

The Resource Directory is the repository of resources, where all the available Resource Descriptors are stored to allow 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. It must be pointed out that the Resource Directory may exist on hospital premises, but also on the cloud as there may be different resources on each deployment.

Handling such information in an accessible way and, keeping the relations of the objects aligned is another desired feature. Thus, an approach like a Resource Description Framework should be followed, where the information is stored using triplets of subject-predicate-object, where the subject is the resource, the predicate is an attribute type and object is the attribute's value. The Resource directory must also be able to receive queries and answer in a short time.

Taking this into account, there exist some interesting solutions for the Resource Directory implementation, such as Apache Jena[5] and Fuseki[6], both having SPARQL support to query the information stored in triplets. Moreover both are also open source which is another necessary requirement.

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.

3.5 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 15.

- The Resource Federation component handles permissions for sharing resources, after receiving resource sharing requests from the API gateway.

[5] Apache Jena, https://jena.apache.org/

[6] Apache Jena Fuseki, https://jena.apache.org/documentation/fuseki2/

- 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.

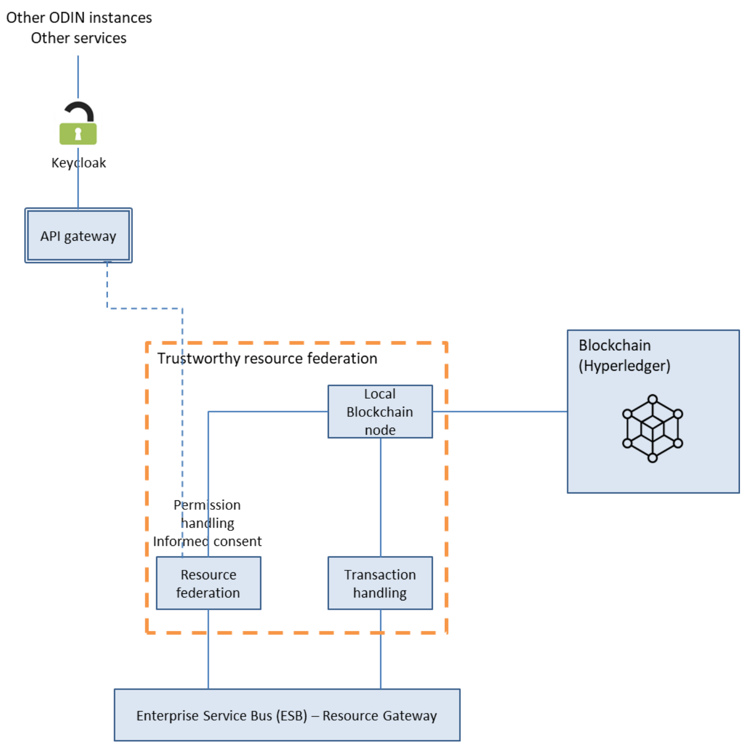


Figure 15: Components involved in resource federation.

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”. The sequence of operations between the components involved during resource federation can be found in Section 4.1.9.

3.5.1 Resource federation

The Resource Federation (RF) component handles permissions when a remote instance requests access to a local resource. The request is first made towards the local API gateway. The latter inquires the Resource Federation component about whether the requested resource is sharable and whether the requesting instance has permission to use this resource.

The information about whether the remote instance has the appropriate permission is stored in the ODIN blockchain. The RF component retrieves this information from the blockchain and, in case of granted permission, informs the API gateway that permission is granted. The API gateway then makes the local resource API visible to the remote instance, and communication between the two instances can begin.

In case permission is not already granted, i.e. the local instance has not given consent for the remote instance to use this resource, the request is directed to the hospital administrators through the RF graphical user interface (RF GUI). In this case, a notification appears in the RF GUI, informing the end user that the specific remote instance has requested access to the specific local resource. The end user can choose to grant or revoke permission, or to inquire more information from the remote instance, through communication between the administrators, as to the purpose of resource sharing and the targeted application and use. Once a final decision is made, the RF component stores it in the blockchain, so that it cannot be inadvertently manipulated in the future. If permission is granted, the API channel between the two resources opens for communication; otherwise, a message is sent to the requester that permission is not granted. In further requests from the same remote instance, the information already stored in the blockchain is used to automatically grant or revoke access, unless a different kind of request is made (e.g. for a different resource), at which case the end user is again notified.

From an implementation point of view, the information regarding the granted permissions could be encapsulated in the request itself during resource communication, in the form of federation-specific message headers, although this decision has not been made at this stage of the project. Implementation details will be clarified throughout the activities of T4.4 “Digital Ledger Technologies Resource Federation and management framework”.

The RF component is also connected to the Enterprise Service Bus, to make its functionalities visible to other components of the ODIN platform, in case they need access to the available resource federation permissions.

3.5.2 Transaction handling

The Transaction Handling (TH) component logs all transactions between remote resources in the ODIN blockchain. Logging transactions is important to ensure the auditability and traceability of resource sharing operations, which increases the trustworthiness of resource federation. After permission to use a local resource has been granted to a remote ODIN instance, any API request and response transmitted between the involved API gateways is stored in the ODIN blockchain.

Since all requests and responses between remote ODIN instances pass through the API gateway, it is the latter that informs the Transaction Handling component that an API request has been made and a response has been sent. The TH component stores only metadata associated with this transaction, such as the timestamps of request and response, the specific API endpoint (URL) used, the number of bytes transferred, etc.

The TH component is also connected to the Enterprise Service Bus, to make transaction auditing available to other components of the ODIN platform, or to the local resources themselves. This enables the use of the transaction auditing mechanism even between local resources, in cases, e.g. that operations need to be applied to sensitive hospital data. Auditing such transactions in an immutable manner can ensure that the parties involved are accountable for their actions.

3.5.3 ODIN Digital Ledger Technologies (Blockchain)

The ODIN Blockchain is the place where permissions and transactions involved in resource federation are stored. The Blockchain as a technology has certain advantages over other database management systems that make it appropriate for trustworthy federation of data and resources:

- Auditability
- Traceability
- Immutability
- Accountability

- Non-repudiation

Blockchain is based on the concept of a Distributed Ledger, which is an encrypted record of all transactions and information performed across all participants in the blockchain, acting as a multi-party database, with no central authority. Information in a blockchain is stored in a growing chain of blocks, where each block contains a hash of the previous block. This makes modifications to individual blocks difficult, ensuring security-by-design. It also ensures immutability of the stored information, as the chain of blocks is only growing (i.e. information is only appended). In a decentralized system, there are many peer nodes participating, each maintaining a copy of the complete blockchain, which increases trust between the involved parties.

The blockchain used in ODIN will be implemented based on open-source solutions, such as Hyperledger. The ODIN blockchain will be a private blockchain, meaning that no external party will be able to participate to it, unless with the proper permissions. The participating nodes will be primarily cloud-based nodes, forming a blockchain network that is hosted in the ODIN cloud. However, individual ODIN instances may participate in the network through a local blockchain node, if they wish and if they possess the necessary computational resources. This will make them part of the blockchain network, maintaining a copy of all involved transactions and participating in the enhancement of trust between entities.

3.6 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 16. 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.

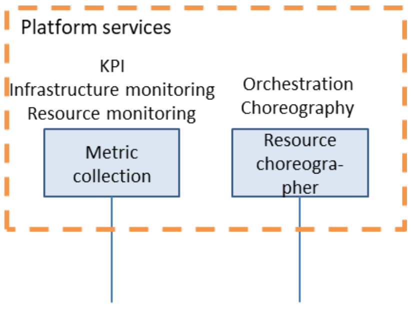


Figure 16: High-level platform services.

3.6.1 Metric collection

The metric collection component collects information from the resource usage and generates relevant KPIs for the end-users. The architecture of the metric collection component, depicted in Figure 17, is already described in D4.1 “CPS-IoT Resource Management System Specification” and the candidate technologies for its implementation are provided in D4.2 “Implementation of Local CPS-IoT RSM Features v1”.

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).

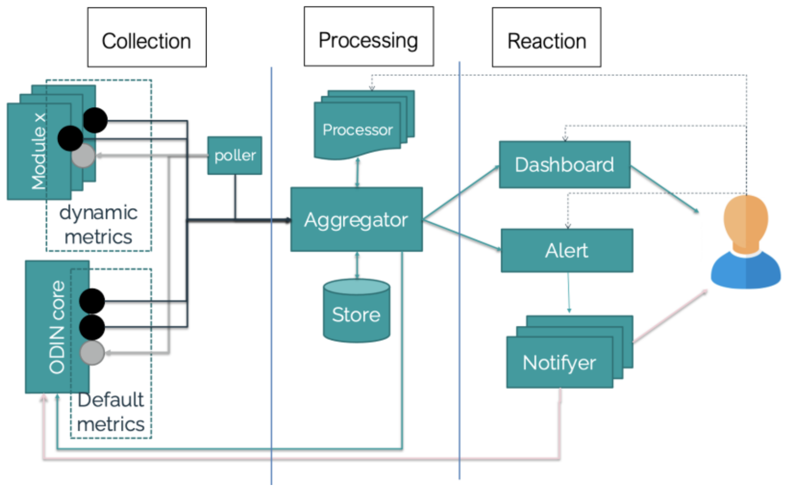


Figure 17: Metric collection architecture.

At platform level, the metric collection component does not communicate only with the ESB to listen for metrics written on the bus, but it also allows to the containers that wrap the ODIN resources to pull metrics such as CPU usage, memory and bandwidth consumption, as well as other platform metrics, such as ESB message rates, directly into the aggregator component of the metric collection. These are considered default metrics, will be registered and collected by the system by default.

The metric collection component will support 3 patterns of data injection:

- “Pull”: In this use case, the collector extracts metrics from a different resource. In many cases, the metrics are exposed by the services themselves, such as Kubernetes.
- “Push”: In this case, the service is responsible of calling the REST-API of the metric collection component that collects the metrics.
- “ESB”: In this case, a bridge is listening for a specific metric topic in the ESB. Also in this case, the metric is pulled by the collector asynchronously.

3.6.2 Resource choreographer

The Resource Choreographer is an important core component of the platform. It can receive events, and other type of inputs that trigger new events or tasks in an automated manner over the resources and services available on the platform. Moreover it can be programmed without coding, just designing workflows using Business Processing Model Notation or Decision Model Notation, both used to define paths of actions that must be followed by a group of services to achieve an objective.

Those objectives are high-level business use cases of interest for the hospitals. For example, the Resource Choreographer could have a workflow specifying that when an alarm is received because a department runs out of an item, some robots are called to replenish it, taking the item from the main warehouse. The Resource Choreographer is very important as it lets the hospitals create new use cases defining new workflows or importing new ones from other hospitals as a resource, creating events and tasks using the available resources represented by their KER abstraction.

Currently there exist several solutions to implement this component, such as Kogito[7], Drools[8] or JBPM[9]. The important thing is to define the messages to be exchanged by the workflow system with the components so the orchestration can be achieved.

It must be pointed out that this kind of service management is called orchestration, where a central unit dictates what to do, but although the approach is centralised, that does not mean that every action or step must be decided by the Resource Choreographer. It is expected that lots of interactions are managed among the resources, services and components by themselves, which is the opposite strategy, called choreography. More details about the Resource Choreographer component can be found in D4.5 "Implementation of Advanced CPS-IoT RSM Features v1".

3.7 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 18, 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.

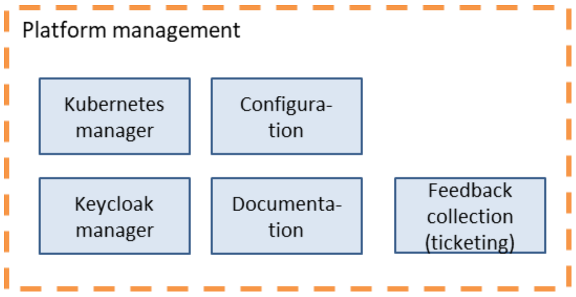


Figure 18: Platform management components.

3.7.1 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

[7] Kogito, https://kogito.kie.org/

[8] Drools, https://www.drools.org/

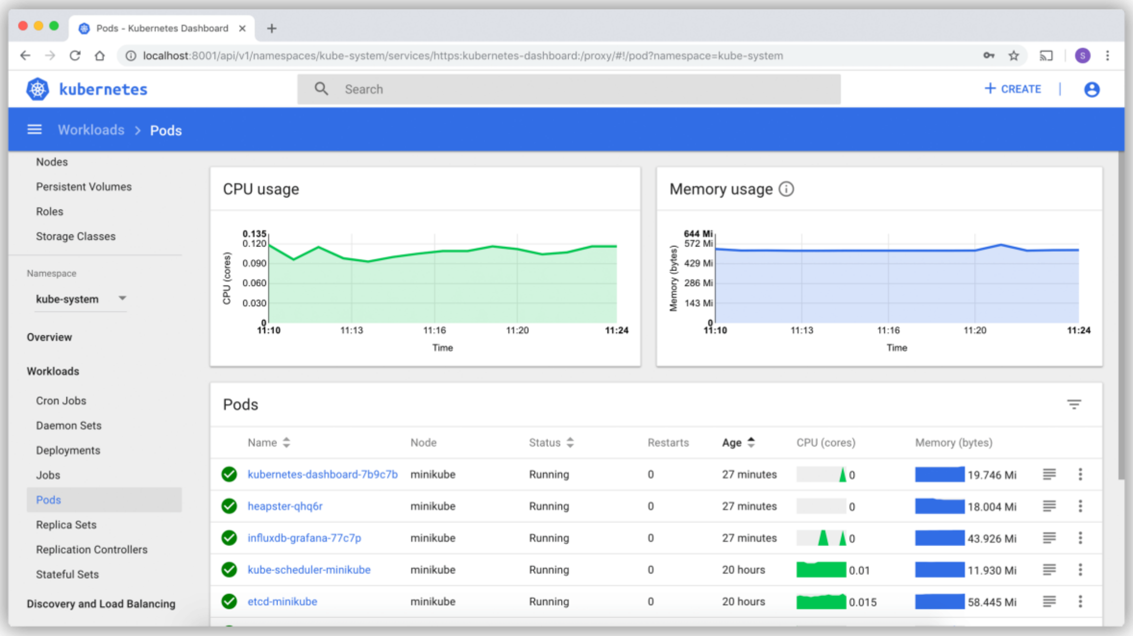
[9] JBPM, https://www.jbpm.org/

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

The Deployment Manager component will be implemented based on existing solutions for managing Kubernetes deployments, such as the Kubernetes dashboard (Figure 19). These provide comprehensive graphical user interfaces for managing and monitoring the deployments, starting/stopping and scheduling services, and monitoring their operation.



Pods

	Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
✓	kubernetes-dashboard-7b9c7b	minikube	Running	0	27 minutes	0	19.746 Mi
✓	heapster-qhq6r	minikube	Running	0	27 minutes	0	18.004 Mi
✓	influxdb-grafana-77c7p	minikube	Running	0	27 minutes	0	43.926 Mi
✓	kube-scheduler-minikube	minikube	Running	0	20 hours	0.01	11.930 Mi
✓	etcd-minikube	minikube	Running	0	20 hours	0.015	58.445 Mi

Figure 19: Screenshot of the Kubernetes dashboard[10].

3.7.2 Access control manager (Keycloak manager)

Access control is a critical component of data security that determines who has permission to access and use company information and resources. Through authentication and authorization, access control policies verify the identity of users and ensure appropriate access to company data. Access control can also be applied to limit physical access to hospital wards, buildings, rooms, and data centres.

Access control identifies users by verifying various login credentials, which can include username and password, PIN, biometric scans, and security tokens. Many access control systems also include multi-factor authentication, a process that requires multiple authentication methods to verify a user's identity. 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.

[10] Image from https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

There are four main types of access control. Organizations typically choose the method that best suits their specific security and compliance requirements. Here are the four access control models:

1. Discretionary access control (DAC): In this method, the owner or administrator of the system, resources or protected data sets the policies for the users who are allowed access.
2. Mandatory access control (MAC): In this non-discretionary model, people are granted access based on an examination of the security attributes. A central authority determines access rights based on different security levels. This pattern is common in government and military circles.
3. Role-based access control (RBAC): The RBAC grants access based on defined business functions rather than the identity of the individual user. The goal is to provide users with access only to data deemed necessary for their role within organizations. This widely used method relies on a complex combination of role assignments and permissions.
4. Attribute-based access control (ABAC): In this dynamic method, access is based on a set of attributes and environmental conditions, such as time of day and location, assigned to both users and resources.

Another important aspect of access control is the ability to share identities among different physical subsystems, the Single Sign-On (SSO) concept. SSO, the so-called "single authentication", indicates in IT an authentication system that practically always takes place in the same way:

1. The user logs in only once on his/her workstation.
2. In this way the user gets access to all computers and services (including the cloud) for which he/she is authorized locally as long as he/she remains in the same location.
3. As soon as the user logs off from his/her workstation, all access rights expire. This occurs after a predetermined period of time or when the user manually performs a single sign-out or sign-off.

SSO is therefore an access system for multiple applications associated with each other, but independent of each other, to which the user must register only once, instead of entering their access codes in each software application individually. Thanks to their ease of use, Single Sign-On systems are used both in the private sphere (web applications and private cloud) and in the professional sphere (applications used within the company and intranet portals). SSO is usually used with the OpenID connect[11] standard and the OAuth2[12] protocol.

[11] OpenID connect, https://openid.net/connect/ , Last access Dec 2021

OpenID is an open standard for authentication used for over a billion accounts such as Google, WordPress and PayPal. The most recent version of the system is called OpenID Connect (OIDC) and represents a combination of OpenID and OAuth2. If used with the Single Sign-On procedure, the users need an OpenID account which they obtain from a so-called OpenID Identity Provider (e.g. Google). With this account (and its URL), the users access all Internet sites that also support OpenID. In the meantime, the reliable Identity Provider transmits a "token" to its website as proof of the user's identity.

One use for SSO via OpenID is for travelling patients. When a patient is registered using the SSO of their reference hospital (Identity Provider) they could still use these credentials in another hospital in the region (or nation) when it is trusted by the Identity Provider and the patient. This is the technology with which “log in with Facebook” and “continue with Google” works on 3rd party websites.

Unlike OpenID, OAuth2 is an authentication and authorization protocol. The main difference is that instead of authenticating themselves on a website, the users delegate a so-called client that accesses the website with an Identity Provider token. The advantage is that the users do not have to transmit their data to the respective website.

Here the appropriate metaphor is “housesitting”: if as the owner of the house (as a user) you give the house keys to a friend (the client), he is authorized to enter the house (the website). OAuth2 is used for example when you want to import friends from your Facebook account to another service without transferring your data.

The ODIN platform will use Single Sign-On (SSO) with OpenID Connect (OIDC) / OAuth2 for the platform subsystems that share the same user realms. OpenID support enables authorization decoupling, leaving that task to Keycloak[13], which is already configured and working in the platform. There are other authorization options like LDAP or open-for-all user registration with local authentication. Through these protocols and standards, the ODIN platform will be able to provide the needed user groups and link them with the resource they can use feeding from legacy accounts and policies existing in the hospital.

Keycloak is an open-source software product that enables SSO with Identity Management and Access Management for modern applications and services. This software is written in Java and supports the identity federation protocols, by default OpenID Connect (OIDC) / OAuth2 as well as LDAP. It is licensed by Apache and supported by Red Hat[14].

[12] OAuth2, https://oauth.net/2/ , Last access Dec 2021

[13] Keycloak, https://www.keycloak.org , Last access Dec 2021

[14] Red Hat, https://www.redhat.com/en , Last access Dec 2021

From a conceptual perspective, the purpose of the tool is to facilitate the protection of applications and services with little or no encryption. Keycloak allows an application (often referred to as a Service Provider) to delegate its authentication.

3.7.3 Platform configuration

ODIN is a microservice-oriented platform, and as such there is bound to be many different aspects managed in as many different ways as there are microservices. Defining a standardized configuration mechanism for the whole platform may be counterproductive. So one approach is to have each micro service self-manage their configuration and have external service manage the adaptation of that configuration to the common platform. This is known as the sidecar pattern in the service mix microservice architecture. ODIN already enables this kind of configuration for common aspects, such as communication, security, metric collection, but for the rest of the configuration, such as individual resource configuration, this will be managed in another way.

We assume that 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.

In future versions of the ODIN platform, depending on the experience with modules deployed for ODIN v1, configuration aspects may be formalised further. A possibility could be to offer a configuration file system, similar to administrative interfaces, so that modules would register their configuration in a common configuration filesystem that can be accessed by system administrators and tools alike. Alternatively, a configuration API management could be offered, allowing modules to describe the APIs used to configure them, thus enabling a central uniform service for configuration of all compatible modules. In either case, the framework would allow for support tools to access and provide module-specific configurations remotely. Fortunately there are many solutions addressing precisely the distributed configuration of microservices, such as Apache Zookeeper[15], microconfig[16], or etcd[17].

3.7.4 Platform documentation

The ODIN platform offers full access to the documentation related to itself plus the KER's available in the platform. The platform documentation component aims to aggregate resources

[15] Apache Zookeper, https://zookeeper.apache.org/

[16] Microfonfig, https://microconfig.io/

[17] Etcd, https://etcd.io/

of many types, such as text, images, videos and other means, with an organized format that is searchable. The component takes into account that some kinds of documentation can be discoverable, such as the one provided by the KERs, let it be new robots, new API Services or other kinds of resources.

The component will be integrated into ODIN's dashboard (see Section 3.8.2), at least as a link, and will be available for most users, depending on their roles, so that they do not get overwhelmed by the amount of information.

User guides, developer guides and Deployment Manuals will be available using this component and the information related to the KER's. We may expect some integration from some Wiki tools such as DocuWiki[18], Read the Docs[19] and tools like Swagger[20] for API documentation. It also is expected to have integration with video platforms where tutorials may be published.

The documentation component will be connected with the ESB to get KER related documentation from the Resource Directory, so that when a new KER is published, the documentation gets available in the portal.

3.7.5 Feedback collection (ticketing)

The Feedback Collection component is used to collect feedback from the end users regarding the use of the ODIN platform. All the issues, bugs, comments, and enhancements the users want to communicate to the ODIN's support team, must be forwarded using the Feedback Collection component. This component handles all the inputs from users, and helps the ODIN support team to organize the priority tasks, perform the follow up and status of each bug/issue and offer a good support to the users.

The Feedback Collection component will be integrated into the platform using an open source solution. The first option would be to have the solution integrated and accessible from the user platform dashboard, and from there jump to the tool. Another option is to create a minimum UI on the platform dashboard and integrate the API from the feedback, which would be a little bit of overhead.

Relevant requirements for the Feedback Collection component include the following:

- Friendly user interface and interaction, as it is going to be used by non-technical users
- Mobile and web support
- Multiple language support

[18] DocuWiki, https://www.dokuwiki.org/dokuwiki

[19] Read the Docs, https://readthedocs.org/

[20] Swagger, https://swagger.io/

- Ticketing support
- Live chat
- Integration API so it can be integrated easily with the platform, especially with the main UI, configuration management, and for example the KPI system for reporting
- Metric support to check quality of service
- Report generation
- Knowledge base support
- OpenID or OAuth2 support
- (Optional) Workflows, branding and mail support

There are several options to be considered for the Feedback Collection implementation, such as Helpy[21], Jira Service Management[22], FreeScout[23] or Zammad[24] among others. The decision will be made throughout activities of T3.4 and reported in next versions of this deliverable as well as deliverables D3.7-D3.9 "Technical Support Plan and Operations".

3.8 External communication

This group of components support the access of the ODIN platform from the end users or external applications. As depicted in Figure 20, 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 Section 3.9.

[21] Helpy, https://helpy.io/

[22] Jira Service Management, https://www.atlassian.com/software/jira/service-management

[23] FreeScout, https://freescout.net/

[24] Zammad, https://zammad.com/en

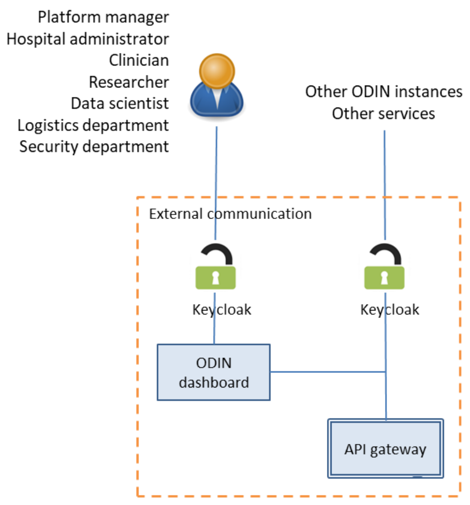


Figure 20: Components for external communication.

3.8.1 API gateway

In a microservice architecture that implements a digital platform, scalability and resilience are key aspects that must be supported. The client apps usually need to consume functionality from many microservices. If that consumption is performed directly, the client has to handle multiple calls to microservice endpoints and authorize directly the consumers. If an application is an entire platform, as the ODIN case, and contains many microservices, handling so many endpoints from the client apps can be a very difficult to manage.

In this case, having an intermediate level or an intermediary (Gateway) can solve the issue (Figure 21). Without an API Gateway, the client apps must send requests directly to the microservice owners and that raises problems, such as:

- Coupling: without the API Gateway pattern, client apps are coupled to the internal microservices. The client apps need to know how the multiple areas of the application are decomposed in microservices. When evolving and refactoring the internal microservices, those actions impact maintenance because they cause breaking changes to the client apps due to the direct reference to the internal microservices from the client apps. Client apps need to be updated frequently, making the solution harder to evolve.
- Too many round trips: a single page/screen in the client app might require several calls to multiple services. That can result in multiple network round trips between the client and the server, adding significant latency. Aggregation handled in an intermediate level could improve the performance and user experience for the client app.

- Security issues: without a gateway, all the microservices must be exposed to the "external world", making the attack surface larger than if you hide internal microservices that are not directly used by the client apps. The smaller the attack surface is, the more secure an application can be.
- Cross-cutting concerns: each publicly published microservice must handle concerns such as authorization and SSL. In many situations, those concerns could be handled in a single tier so the internal microservices are simplified.[25]

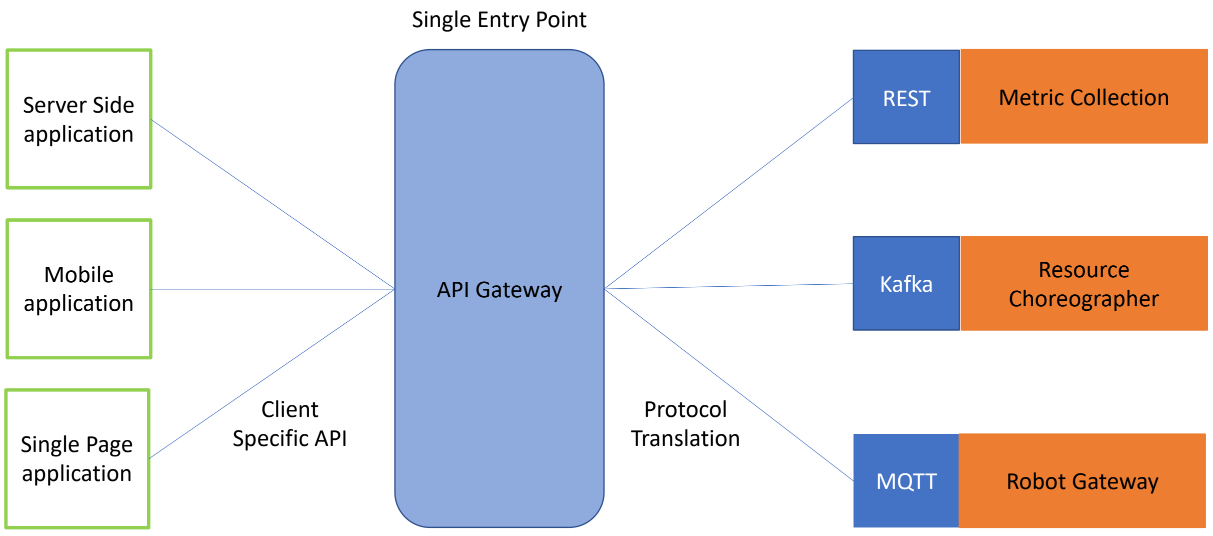


Figure 21: API Gateway pattern in microservices architecture.

Within ODIN, by using container and orchestration technologies in conjunction with an API Gateway, it is possible to offer multiple features. This API Gateway configuration will provide very important features described by the following design patterns:

- Reverse proxy. The API Gateway offers a gateway routing to redirect or route requests (usually HTTP requests) to the endpoints of the internal microservices. The gateway provides a single endpoint or URL for the client apps and then internally maps the requests to a group of internal microservices. This routing feature helps to decouple the client apps from the microservices but it is also convenient when modernizing a monolithic API by positioning the API Gateway in between the monolithic API and the client apps. In this configuration, new APIs can be added as new microservices while still using the legacy monolithic API until it is split into many microservices in the future.

[25] The API gateway pattern versus the Direct client-to-microservice communication, https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern , Last Access March 2022.

- **Request aggregation**. As part of the gateway, multiple client requests (usually HTTP requests) targeting multiple internal microservices into a single client request can be aggregated. This pattern is especially convenient when a client page/screen needs information from several microservices. With this approach, the client app sends a single request to the API Gateway that dispatches several requests to the internal microservices and then aggregates the results and sends everything back to the client app. The main benefit and goal of this design pattern is to reduce chattiness between the client apps and the backend API, which is especially important for remote apps out of the data centre where the microservices live, like mobile apps or requests coming from Single-Page Applications (SPA).
- **Gateway offloading.** One of the core features offered by the API Gateway is offloading functionality from individual microservices to the gateway. This pattern simplifies the implementation of each microservice by consolidating cross-cutting concerns into one tier. This is especially convenient for specialized features that can be complex to implement properly in every internal microservice, such:
    - 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.

3.8.2 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 (Section 3.6.1), resource choreographer (3.6.2), deployment manager (3.7.1), access control manager (3.7.2), platform configuration (3.7.3), platform documentation (3.7.4), and feedback collection (3.7.5); 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 Section 3.7.4.

3.9 Platform security

As already described in D3.4 "Privacy Security and Trust report v1", 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 v1. The full detailed analysis and implementation of these features, as well as design for advanced privacy, security and trust features will be 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 grant by using standards for network security. Such mandatory standards in ODIN will be TLS[26], HTTPS[27] as well as IPSec[28] and VPN[29]. 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[30] and MongoDB[31] databases by also using community edition technologies such as Percona[32] for data encryption and Vault[33] for encryption key management.

Advanced trust features of the ODIN platform will be based on the Resource Descriptors (Section 3.4.3) 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 3.7.2), blockchain services, and the public key management schema, such as X509[34] and hash functions[35], 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.).

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

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

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

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

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

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

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

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

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

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

3.9.1 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 3.5). 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.

3.9.2 Execution environment v1

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

- Firewall configuration not allowing external communications (especially outgoing) outside the Kubernetes cluster.
- 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.
- Network segmentation, i.e. configuring resource modules in their own network, which separates systems into subnets with unique security controls and protocols.

3.9.3 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[36] will be used to check container vulnerability as well as protection against the OWASP top 10 vulnerabilities[37].

3.9.4 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.

3.10DevOps infrastructure

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)

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

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

- 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 6, more information 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.

3.10.1 ODIN ROS repository

To support the needs of robotics partners working on the ODIN platform, an ODIN ROS repository will be created. This repository 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 repository involves two steps, described below.

3.10.1.1 Provision machines

The first step involves the provisioning of the virtual machines. The ODIN ROS repository requires three different kinds of machines (shown in Figure 22):

- 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

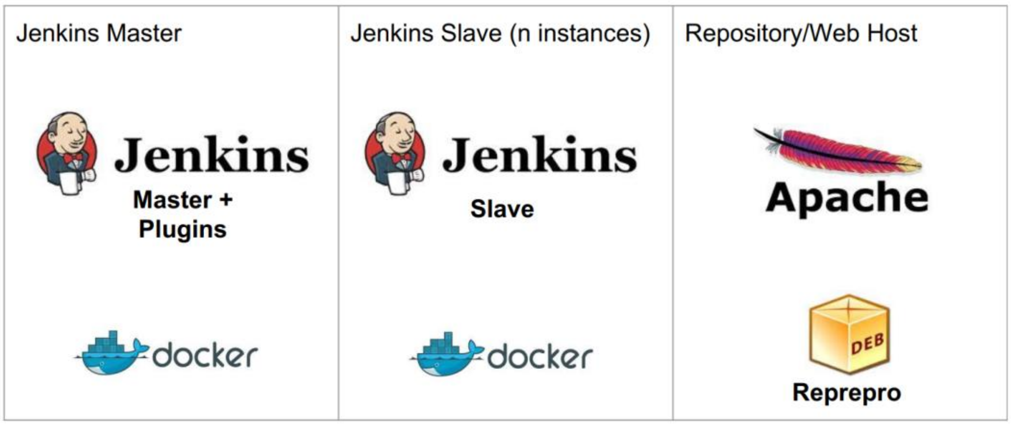


Figure 22: The ODIN ROS repository infrastructure.

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.

3.10.1.2 Generation of Jenkins jobs

After the ODIN ROS repository is set up, the generation of Jenkins jobs will start 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 will 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 will be utilized since they offer isolation and provide faster performance than Virtual Machines (VMs).

3.10.2 Kubernetes cluster

The base for the installation of a new ODIN Platform instance in a hospital environment will be Kubernetes. Kuberentes[38] 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 Figure 23, it consists of a set 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.

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

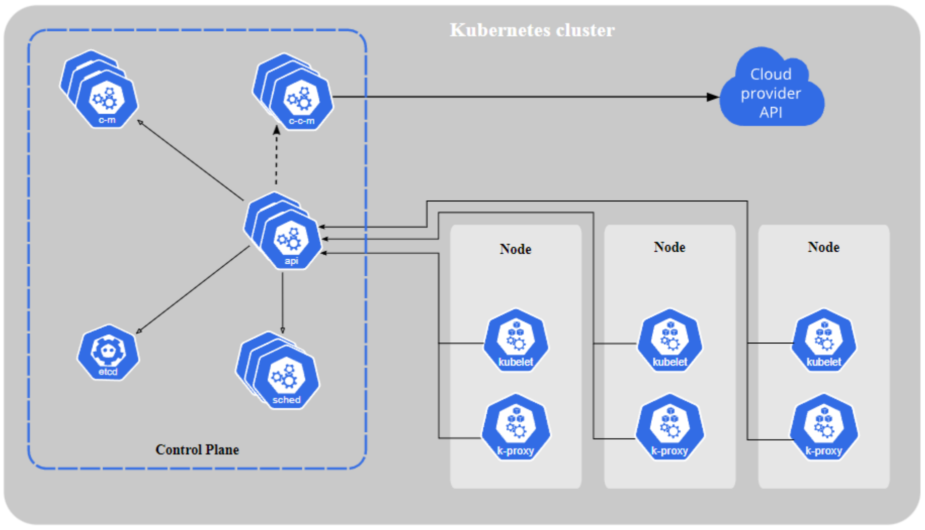


Figure 23: Kubernetes cluster[39].

In a new ODIN platform installation at a deployment site, Kubernetes will be the first component to be installed. As depicted in Figure 24, 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.

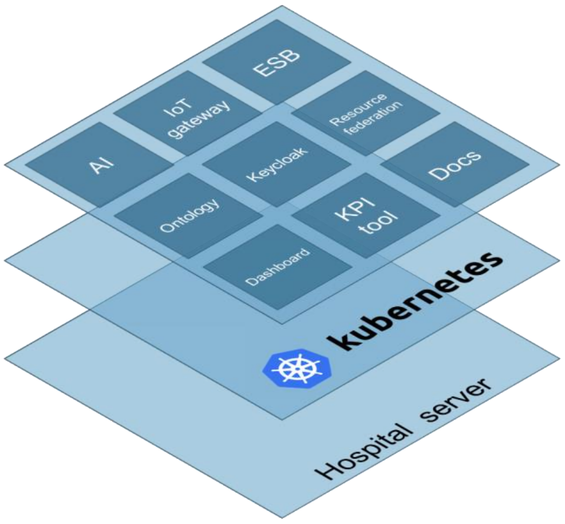


Figure 24: Kubernetes as a substrate for the deployment of ODIN platform components.

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

4 Operation flow

After the description of each separate component of the ODIN platform in Section 3, this section describes how these components communicate with each other to implement the main functionalities of the ODIN platform. The flow of operations is described in the form of sequence diagrams. Each component or actor is represented by a vertical line with time directed downwards. Calls between components are represented by arrows between them.

4.1 Core procedures

This section covers individual procedures for performing core functionalities of the ODIN platform, outside the context of a specific use case scenario.

4.1.1 User authorization

Figure 25 depicts the operation flow for user authorization. The interaction is between the end-user and the Keycloak service, which communicates with an OpenID/LDAP provider to handle authorization.

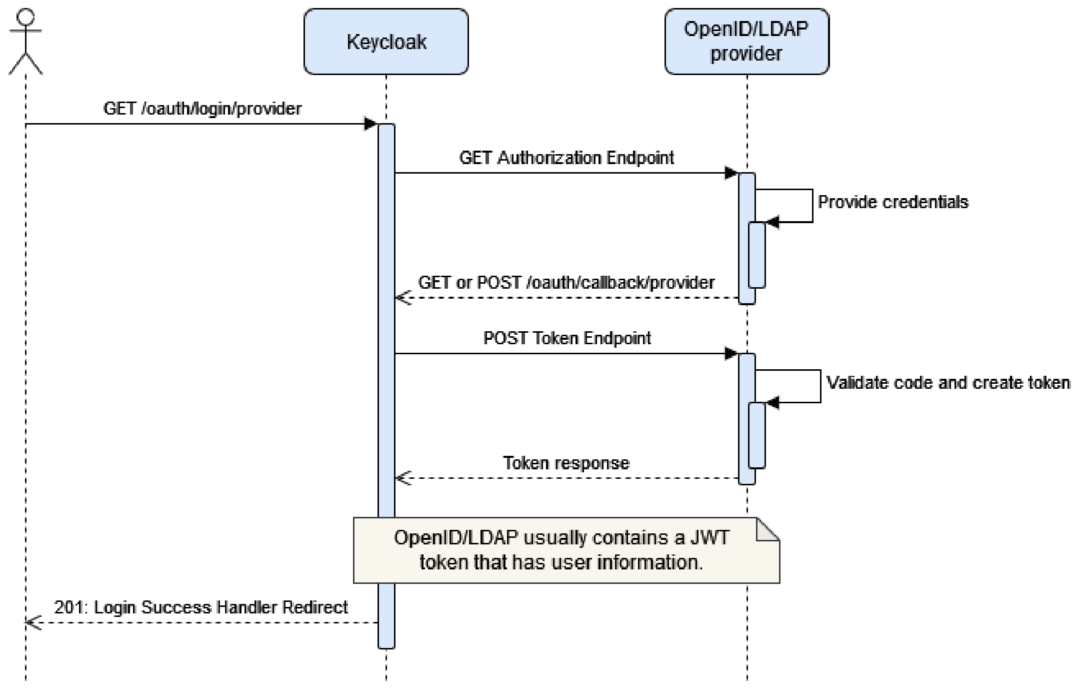

Keycloak
OpenID/LDAP provider
GET /oauth/login/provider
GET Authorization Endpoint
Provide credentials
GET or POST /oauth/callback/provider
POST Token Endpoint
Validate code and create token
Token response
OpenID/LDAP usually contains a JWT token that has user information.
201: Login Success Handler Redirect

Figure 25: User authorization sequence diagram.

4.1.2 User creation (by the administrator)

Figure 26 depicts the process for creating a new ODIN user. This process describes the creation of the user by the administrator, while Section 4.1.3 describes self-registration.

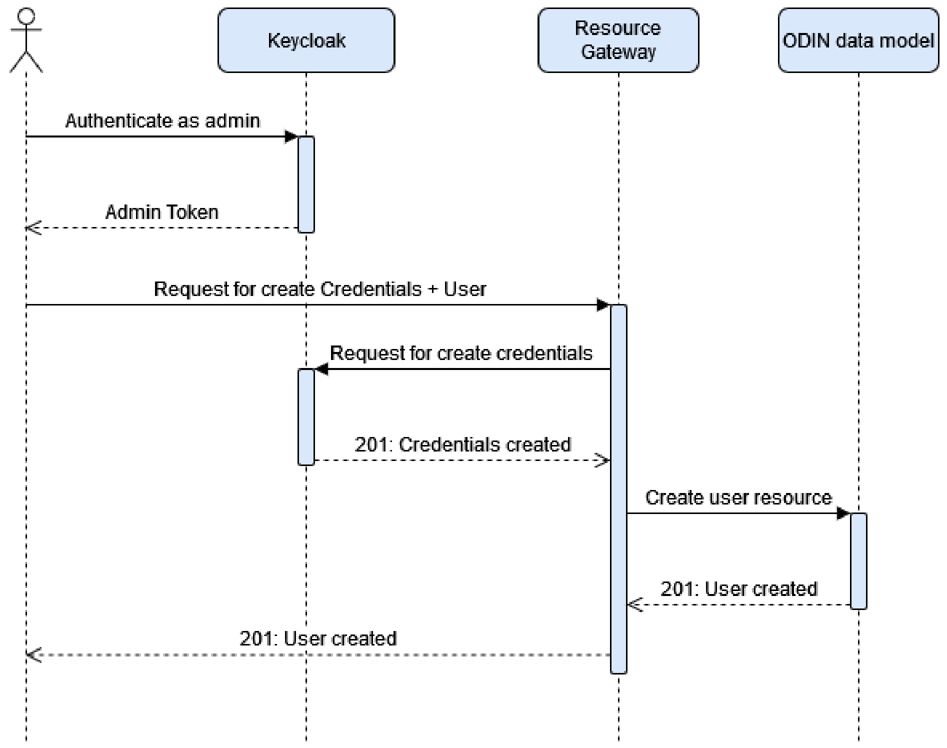


Figure 26: User creation (by an administrator) sequence diagram.

4.1.3 Self-user registration

Figure 27 depicts the process for self-registration by end users. Users can also be created by the administrator, as described in Section 4.1.2.

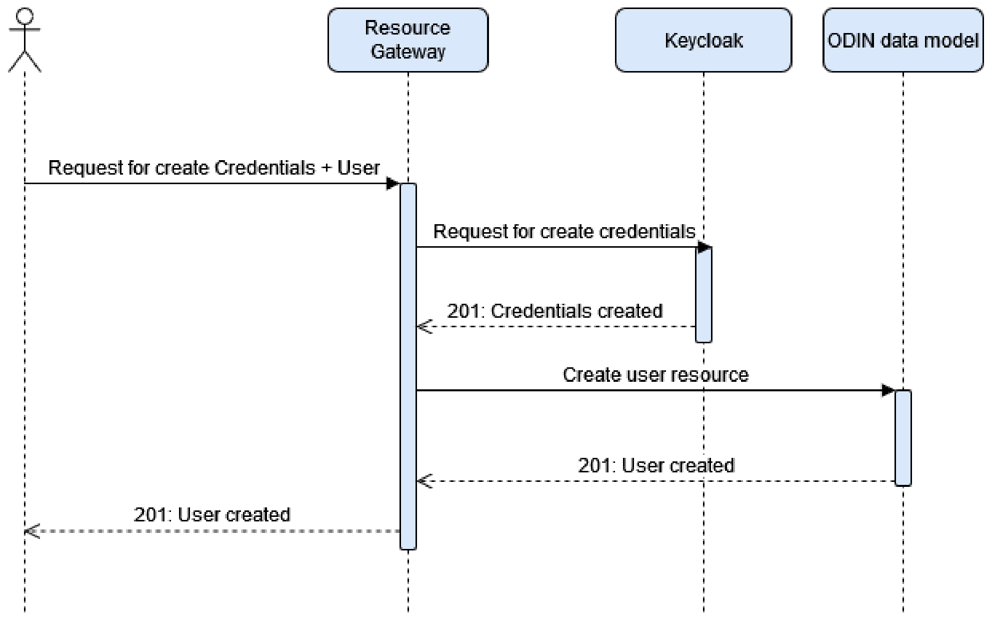


Figure 27: Self-user registration sequence diagram.

4.1.4 Resource registration

Any time a new resource is registered, a client will have to be authenticated in order to use the Resource Manager service. This client can be either a front-end for manual registration, or the resource itself, or a sidecar representing the resource in the ODIN platform.

The Resource Manager will check the permissions to allow for the registration to proceed, as well as validate the resource descriptor for security risks and configurations. Finally it will check the resource is not already registered before adding the resource descriptor to the Resource Directory, completing the registration. Figure 28 depicts the resource registration workflow.

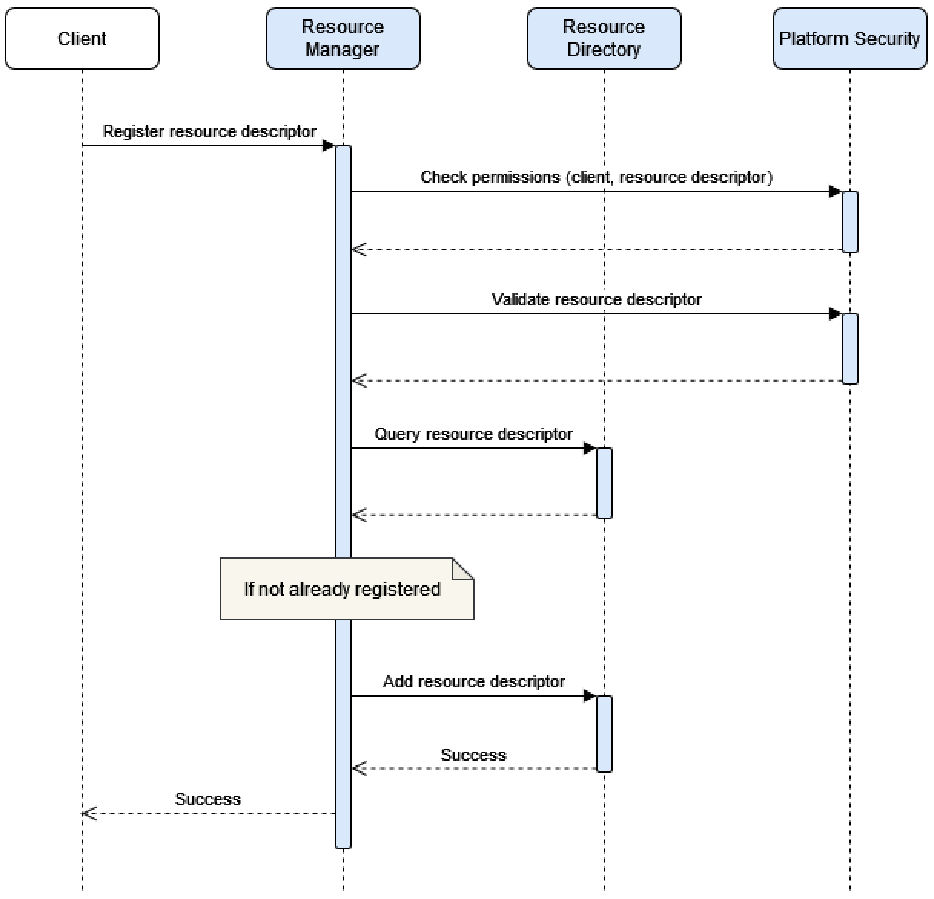

Client
Resource Manager
Resource Directory
Platform Security
Register resource descriptor
Check permissions (client, resource descriptor)
Validate resource descriptor
Query resource descriptor
If not already registered
Add resource descriptor
Success
Success

Figure 28: Resource registration sequence diagram.

4.1.5 Resource communication

The sequence diagram of Figure 29 shows how communication between two resources is performed, emphasizing the different stages and transformations that information passes through to reach the destination. In this example, an IoT device publishes measurements to the main bus (ESB), while a back-end service has subscribed to the bus, so it receives the measurements. The measurements from the IoT device are sent to the IoT gateway (e.g. a Raspberry PI device), which is connected to the ODIN platform. To publish the information to the bus, the measurements pass through semantic translation and protocol transport. On the other hand, for the back-end service to read the measurements, the reverse steps are taken, performing protocol transport and semantic translation, before the transport service calls the back-end REST API to handle the measurement.

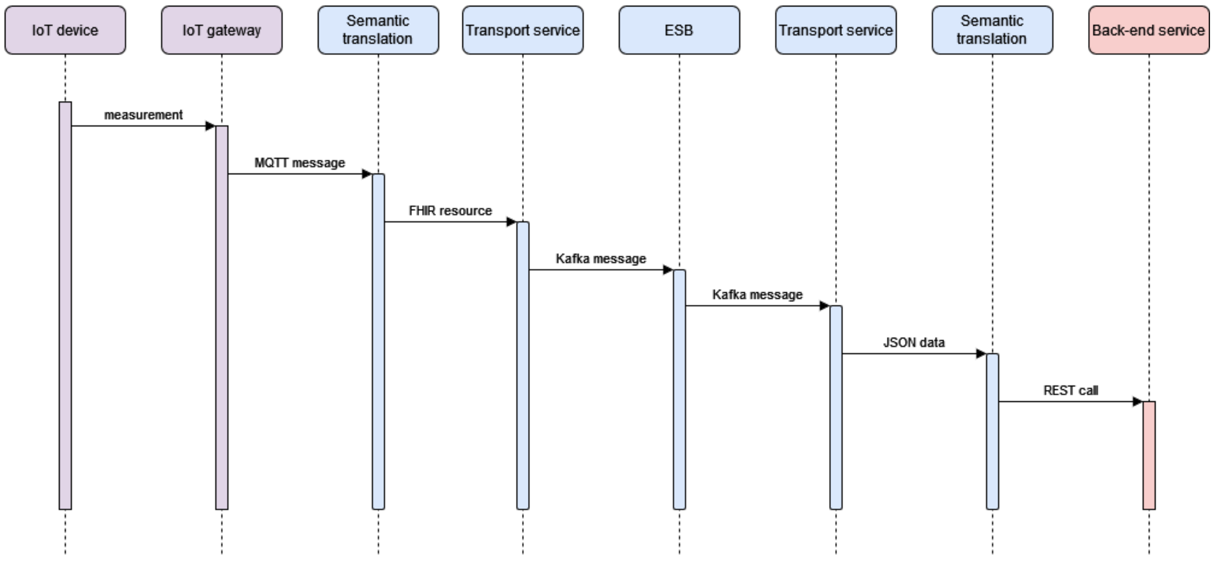


Figure 29: Resource communication workflow.

4.1.6 Communication between front-end and back-end resources

Communication between a front-end resource and a back-end resource can happen in two ways in the ODIN architecture. In case the front-end application is tightly coupled to the back-end application, with the two being difficult to conceptually separate, they can communicate directly. In this case, they may as well be considered as a single resource connected to the ODIN platform.

However, there may be cases where the front-end is loosely coupled with the back-end, with each of them being a separate application. In other words, the front-end may be designed in such a way that can use multiple back-ends, or the back-end is designed to be used by multiple front-end applications. In this way, the front-end and the back-end are considered as separate resources that need to communicate through the main service bus, undergoing the same resource abstraction mechanisms as with any other resource. Figure 30 depicts such an example, with the front-end application requesting historical data from a back-end service, with communication passing through the ESB, in a publish-subscribe manner.

A particular example of communication between front-end and back-end services is the communication interface between the AI-based services and user interfaces. Within ODIN, this communication is built on the Java programming language using the Java SE technology (Jakarta EE) supported by Jersey RESTfull web-services framework and the Jackson Java library for the processing of the data structures. The Jersey framework implements the JSR-311 and JSR-339 Java specifications for RESTfull web-services. The HTTP POST method is used to provide the requested data structures in the JSON format; the method returns OK (code: 200) in case there is no error followed by the representation of the resource and returns NOT FOUND (code: 404) or BAD REQUEST (code: 400) in case of an error.

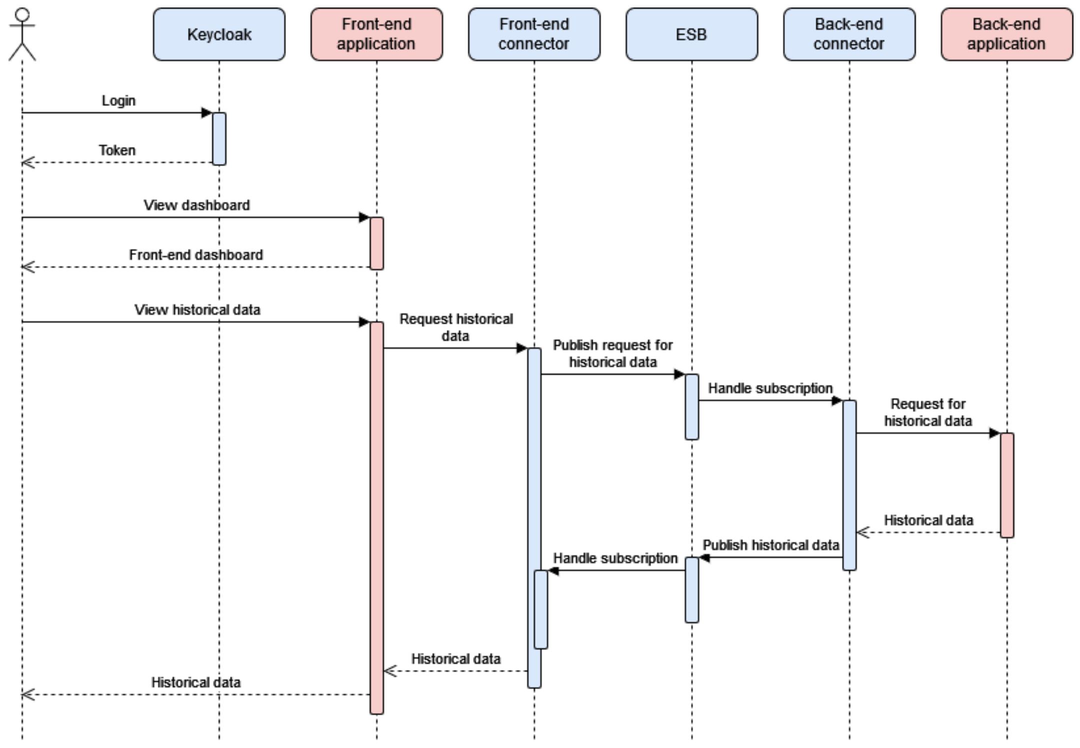


Figure 30: Sequence diagram for the communication between a front-end and a back-end resource.

4.1.7 Communication with the HIS

Communication of the ODIN platform with the Hospital Information System (HIS) is essential to retrieve patient-related data and provide functionalities that are useful to the hospitals. Communication with the HIS is challenging, because different hospitals may implement their HIS differently. As an example, the way data are stored and made available for use may significantly differ across hospitals, ranging from hospitals with fully-developed data retrieval APIs, to hospitals with DBMSs for managing data, to hospitals keeping their records in CSV files. The ODIN platform needs to adapt to all circumstances. This adaptation is mainly handled by the HIS connector components, which provide the bridging functionality between the HIS and the main service bus (ESB). Figure 31 presents examples of communication between the ESB and the HIS, in different HIS data storage variants: HIS with API, HIS with DBMS, HIS storing data in CSV files. From the point of view of the ESB (and, in turn of other resources), all cases are handled in the same manner: whenever a request for data is made, the subscribed connector is called and the requested data are published back to the ESB.

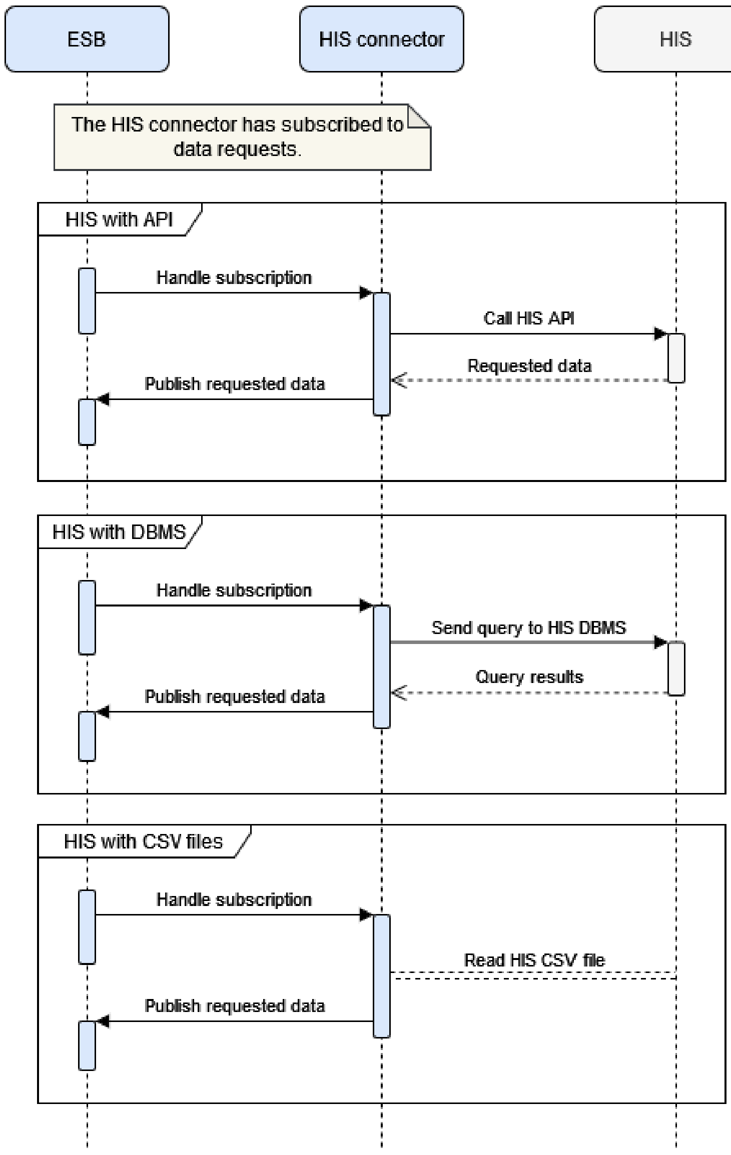


Figure 31: Sequence diagram for the connection of HIS variants to the ESB.

4.1.8 High-level application creation

The ODIN platform allows end-users to create custom high-level applications to support their use cases, using the available resources. This is accomplished using the Resource Choreographer component. A new high-level application can be designed from scratch using the Resource Choreographer's interface, as depicted in Figure 32, or by submitting an existing workflow from a file, as depicted in Figure 33.

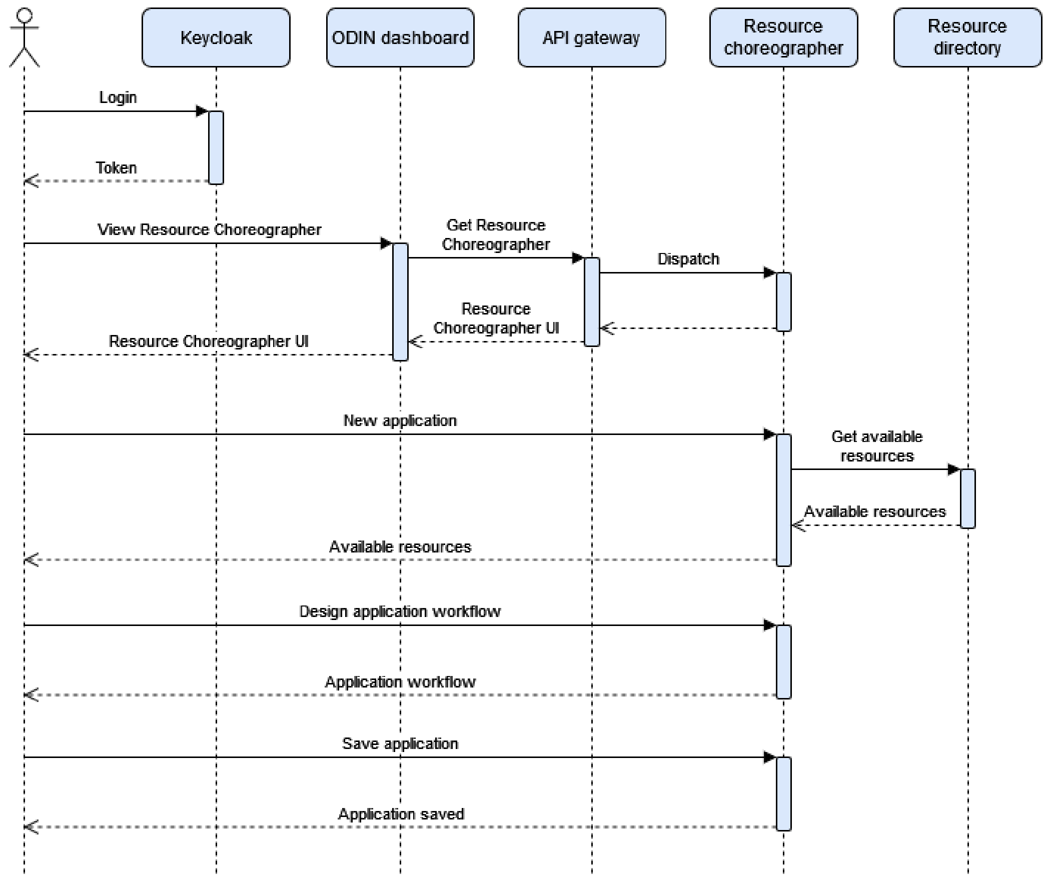


Figure 32: High-level application creation sequence diagram.

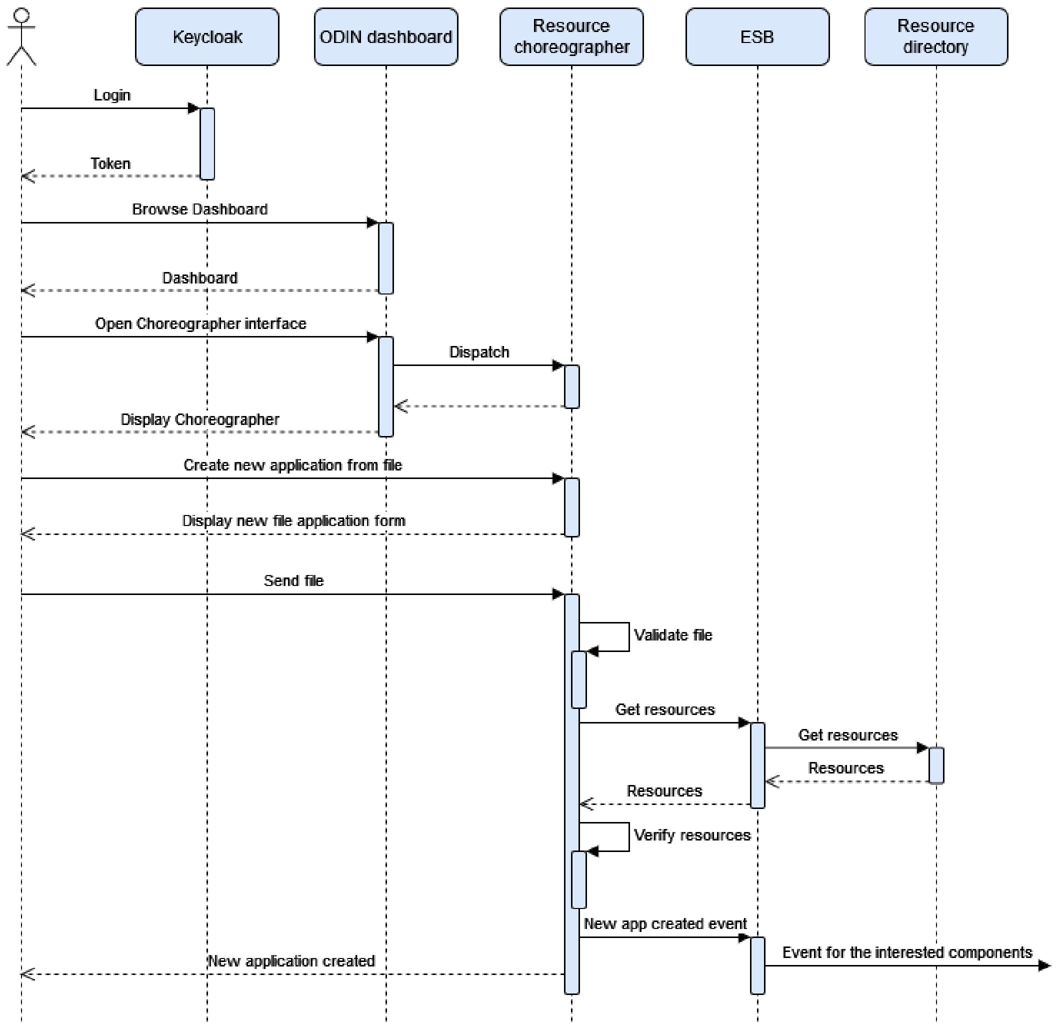


Figure 33: Sequence diagram for new high-level application creation from file.

4.1.9 Resource federation

Figure 34 depicts the flow of operations involved in resource federation and transaction handling. In this example, a back-end service of ODIN instance A requests to use an AI service from ODIN instance B. Two options are displayed, with permission denied or granted. In case permission is granted by the Resource Federation component, the two remote resources can communicate with each other, with all transactions being logged by the transaction handling component.

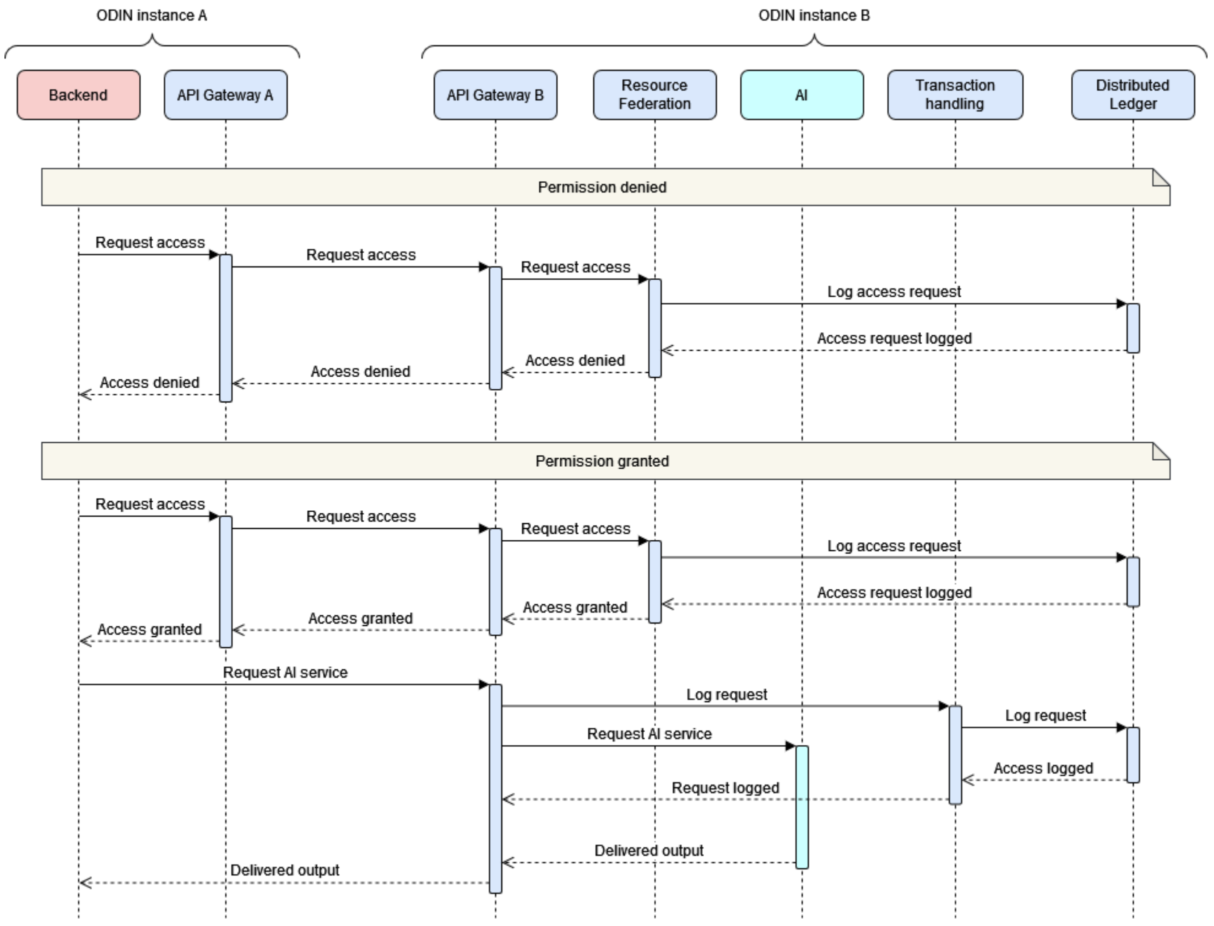


Figure 34: Resource federation and transaction auditing sequence diagram.

4.1.10 Metric collection

The metric collection components are responsible for collecting metrics and KPIs from the use of resources. Metric collection can be performed in one of the following ways:

- Pull type, where the metric collection component asks the resource for the metric (Figure 35)
- Push type, where the resource itself asks the metric collection component to store the metric (Figure 36)
- ESB type, where the resource publishes the metric to the ESB and the metric collector subscribes to the message (Figure 37)

After metrics are collected, they may be used to generate alerts and notify the end-users, in case the metric values are outside the normal or expected range, as depicted in Figure 38.

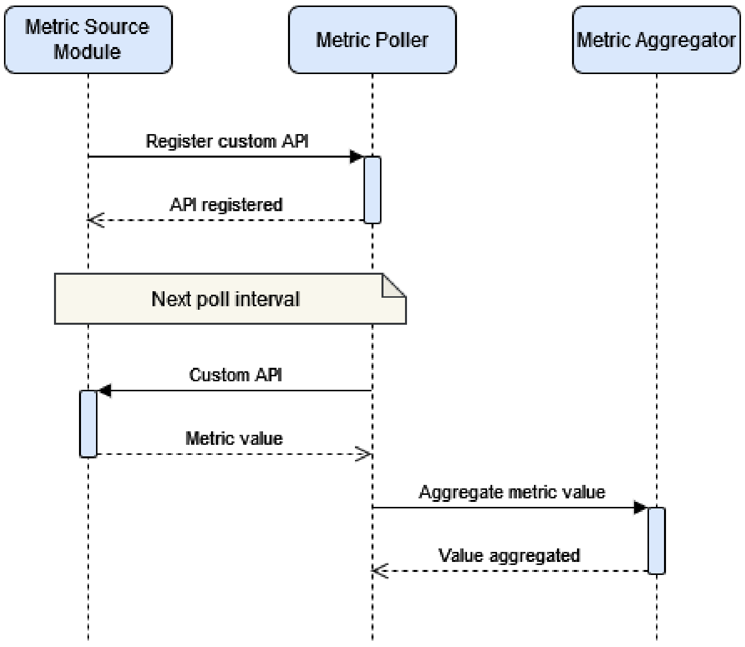


Figure 35: Pull type metric collection sequence diagram.

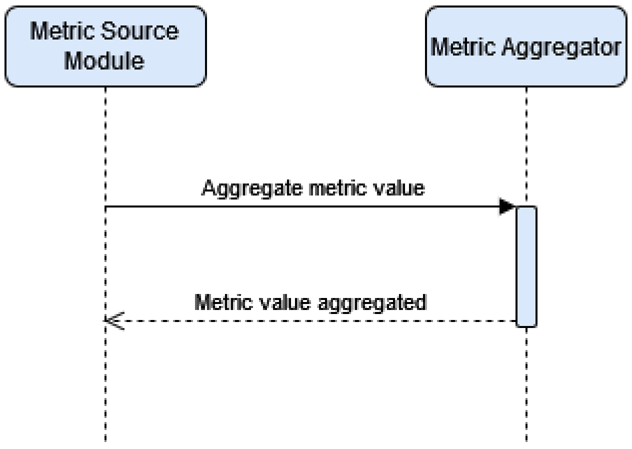


Figure 36: Push type metric collection sequence diagram.

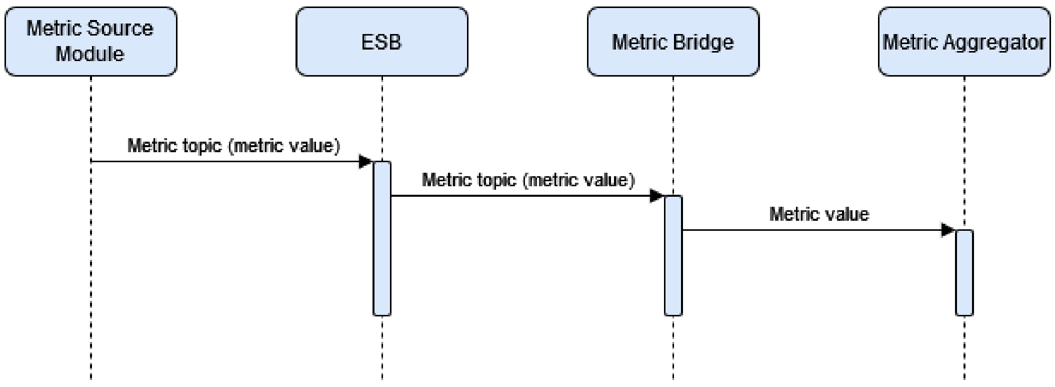


Figure 37: ESB type metric collection sequence diagram.

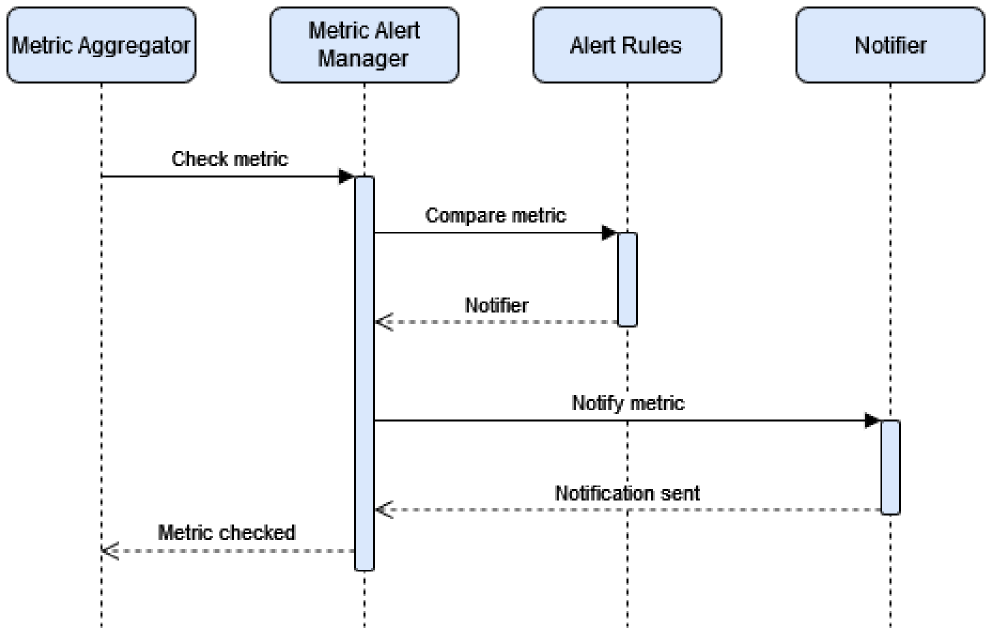


Figure 38: Metric collection alert sequence diagram.

4.1.11 Feedback collection

Feedback Collection is very UI-coupled, so in this case, users will interact directly with the tool available for collecting bugs and issues that will appear in the Platform Dashboard. Everything will be protected by the security provided by the platform. Figure 39 depicts the sequence diagram for feedback collection. Here, “Component Interested in Ticket Event” could be the Metrics system for example.

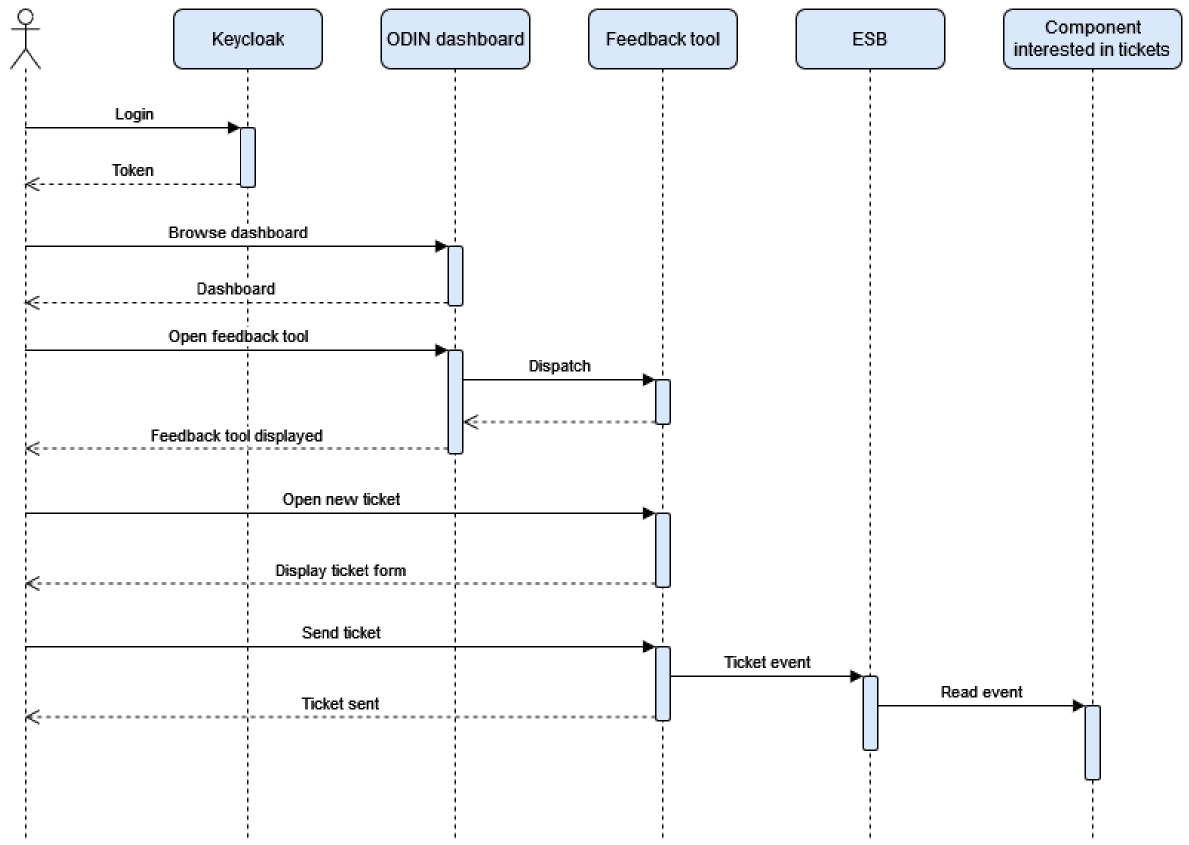


Figure 39: Feedback collection sequence diagram.

4.2 Examples from the ODIN use cases

The sequence diagrams presented in this section depict the flow of operations between resources and the components of the ODIN platform to accomplish example tasks related to the seven ODIN use cases.

4.2.1 UC1: Aided logistics support

Use case 1 (UC1) of the ODIN project is related to logistics support. The example depicted in Figure 40 is about a robot collecting towels from the storage room and carrying them to a patient room. To make the scenario more challenging, the shortest path to follow is crowded, so that the robot needs to communicate with other resources in order to find another route. The diagram of Figure 40 has been simplified: all communication between resources are depicted as being made directly between resources, but in reality all such communication passes through the ESB and the semantic translation and transport service steps.

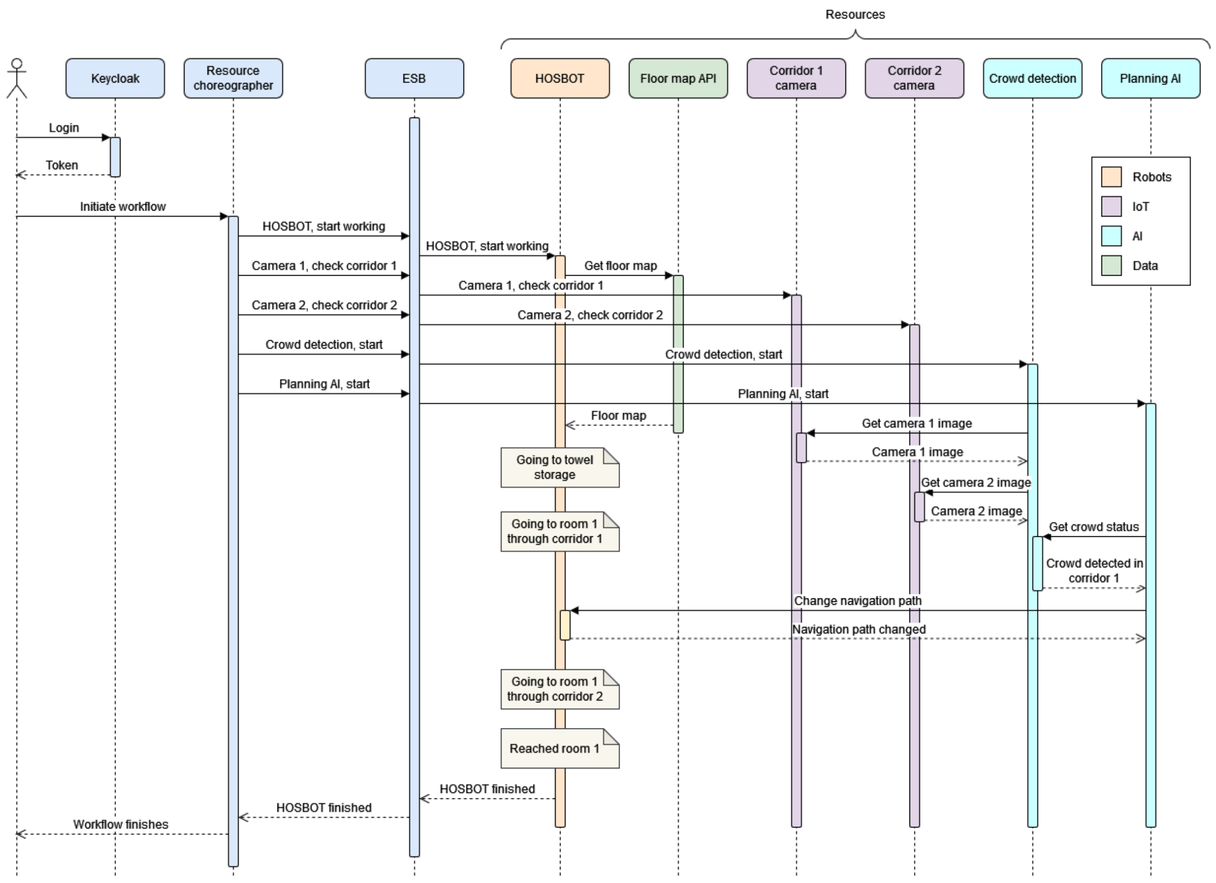


Figure 40: Example sequence diagram for UC1 – Aided logistics support.

4.2.2 UC2: Clinical engineering, medical locations, real-time management

Figure 41 depicts an example sequence diagram related to UC2 "Clinical engineering, medical locations, real-time management", showing the calls to the ODIN platform components that support the services that provide the main functionalities.

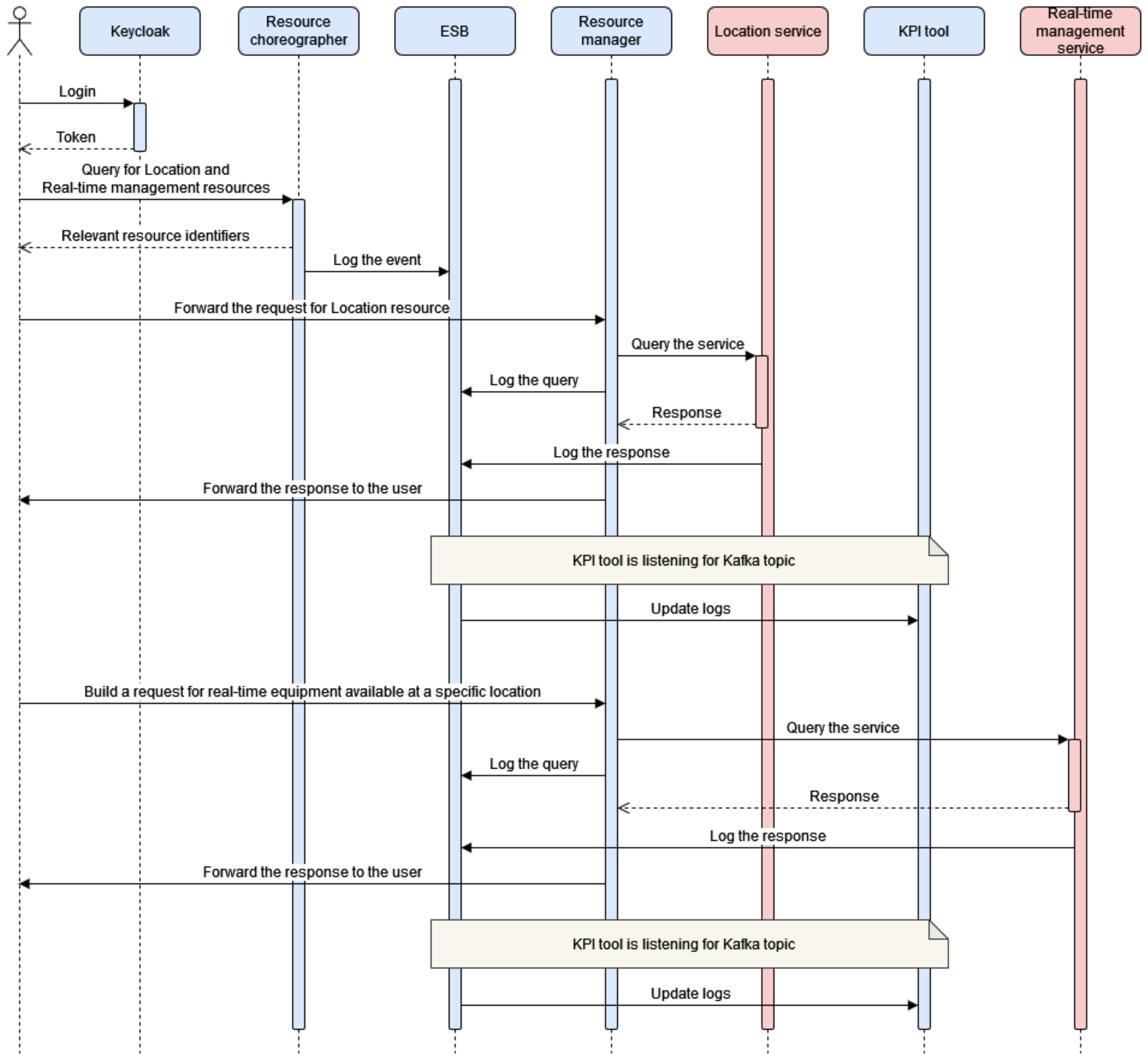

Keycloak
Resource choreographer
ESB
Resource manager
Location service
KPI tool
Real-time management service
Login
Token
Query for Location and Real-time management resources
Relevant resource identifiers
Log the event
Forward the request for Location resource
Query the service
Log the query
Response
Log the response
Forward the response to the user
KPI tool is listening for Kafka topic
Update logs
Build a request for real-time equipment available at a specific location
Query the service
Log the query
Response
Log the response
Forward the response to the user
KPI tool is listening for Kafka topic
Update logs

Figure 41: Example sequence diagram for UC2 – Clinical engineering, medical locations, real-time management.

4.2.3 UC3: AI-based support system for diagnosis

UC3 is related to the use of AI to support diagnosis systems, e.g. to optimize the diagnostic workflow in cardiovascular outpatient clinic. In such a case, AI will replace manual screening of patients to assess eligibility for accessing a cardio management system. The ODIN partners will leverage Federated Learning (FL) in the ODIN platform, as an AI service, and will enable the AI-support system for UC3. The FL pipeline will be part of the AI services of the ODIN architecture, depicted in Figure 1. Hereby, a sequence diagram for the FL service is provided in Figure 42.

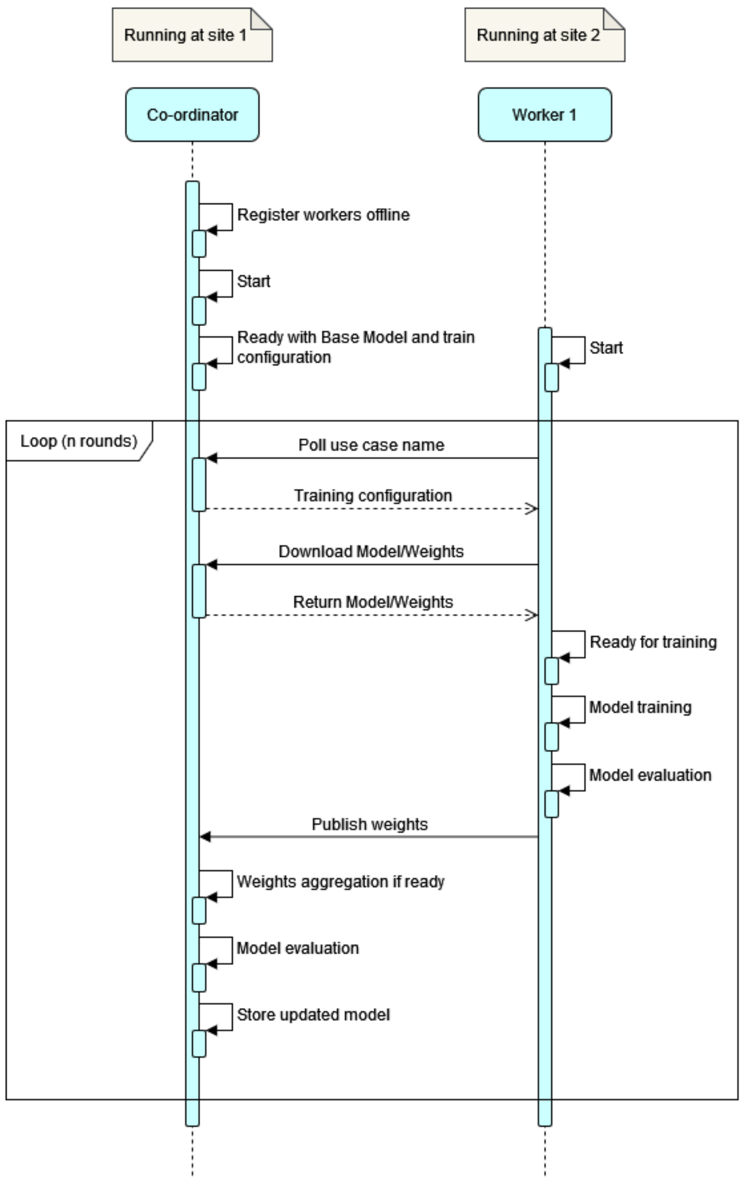


Figure 42: A sequence diagram depicting the federated learning pipeline using a polling-based approach, to be used in the context of UC3 - AI-based support system for diagnosis.

4.2.4 UC4: Clinical tasks and patient experience

UC4 deals with usage of the ODIN KERs to support clinical tasks, such as transportation of blood samples, monitoring food assumption and performing rehabilitation exercises. Robotic platforms play a large role in such scenarios, and they need to communicate with other types of resources in order to properly navigate the hospital and be notified in cases of emergencies. Figure 43 depicts an example scenario.

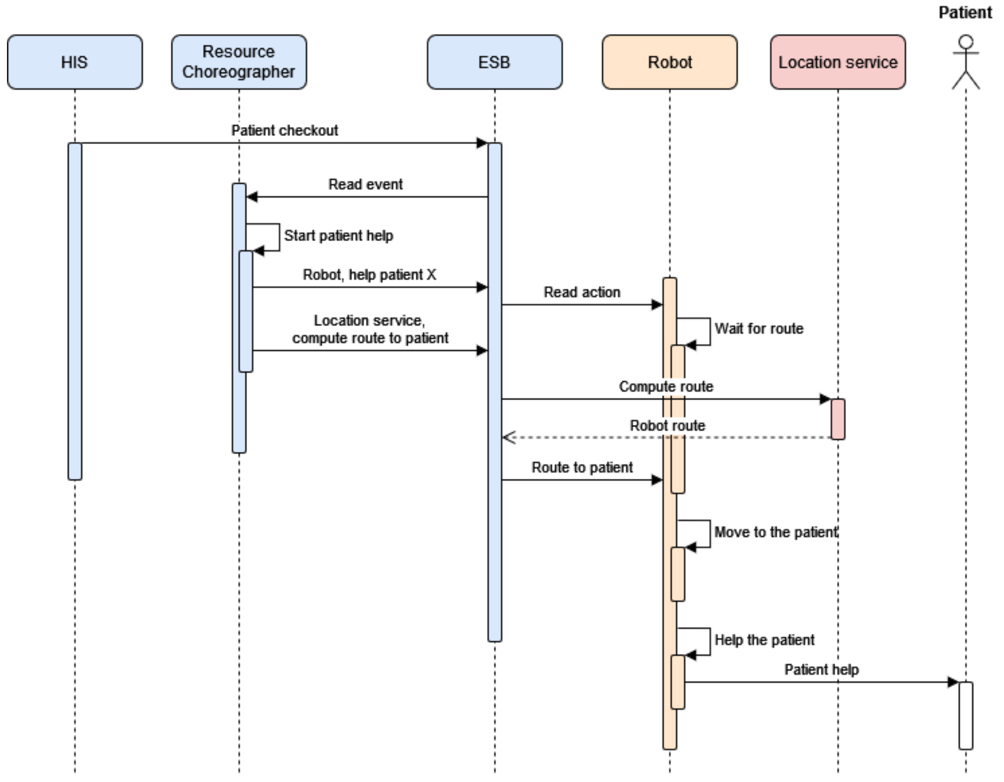


Figure 43: Example sequence diagram for UC4 - Clinical tasks and patient experience.

4.2.5 UC5: Automation of clinical workflows

UC5 deals with the use of the KERs to automate clinical workflows. An example is automated monitoring of oxygen therapy, i.e. monitoring a patient as he/she follows the oxygen prescription and providing warnings or advice in case the prescription is not followed properly. In the example of Figure 44, a robotic platform, an IoT sensor (oxygen meter) and an AI-based recommendation engine cooperate to provide the necessary monitoring functionality, coordinated by a high-level workflow defined in the Resource Choreographer.

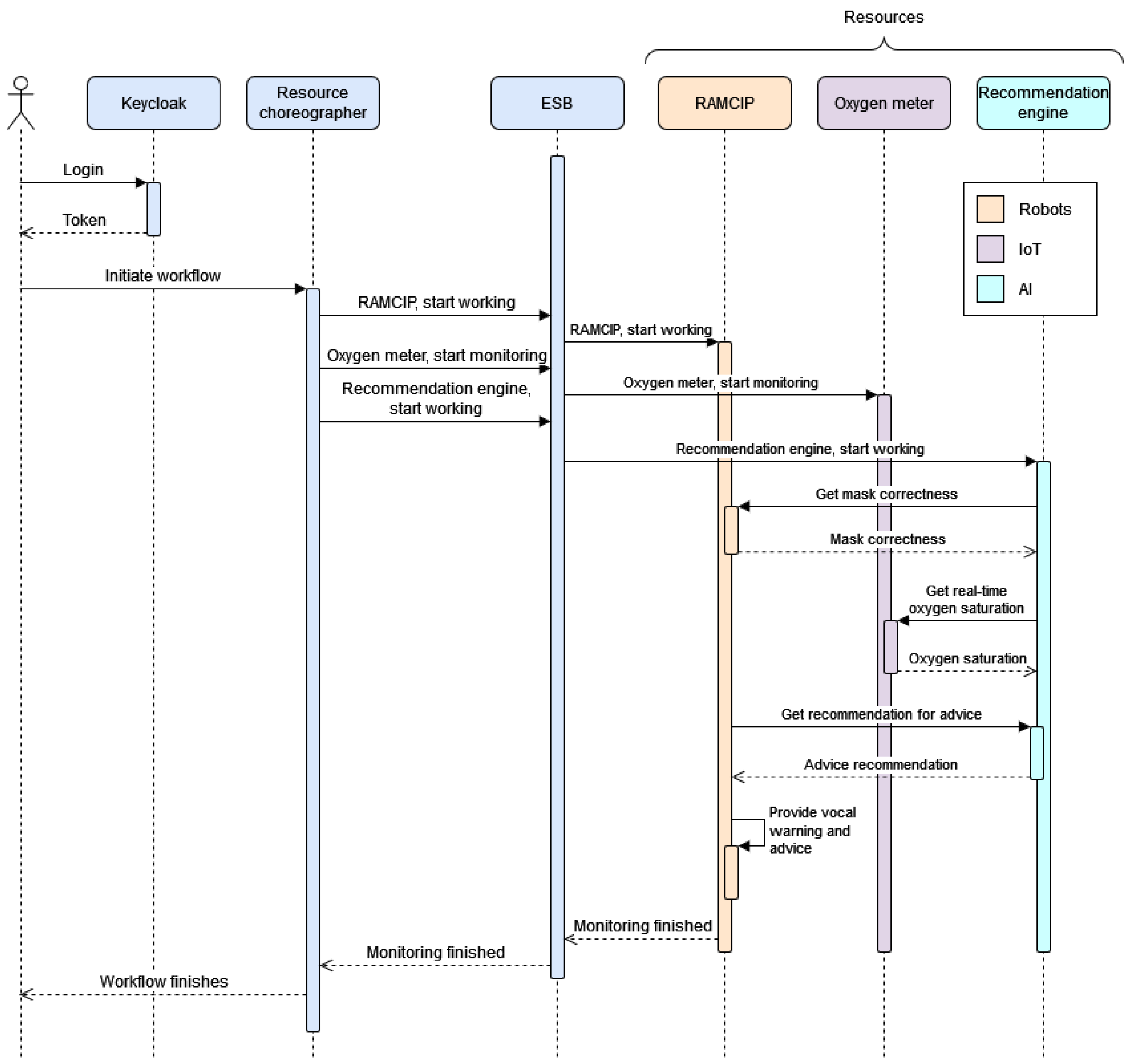


Figure 44: Example sequence diagram for UC5 - Automation of clinical workflows.

4.2.6 UC6: Inpatient remote rehabilitation

UC6 deals with remote rehabilitation of patients, so it relies on devices and sensors that monitor the health status of the individual. Figure 45 depicts an example of how a wearable device can be used within the ODIN architecture to monitor a patient.

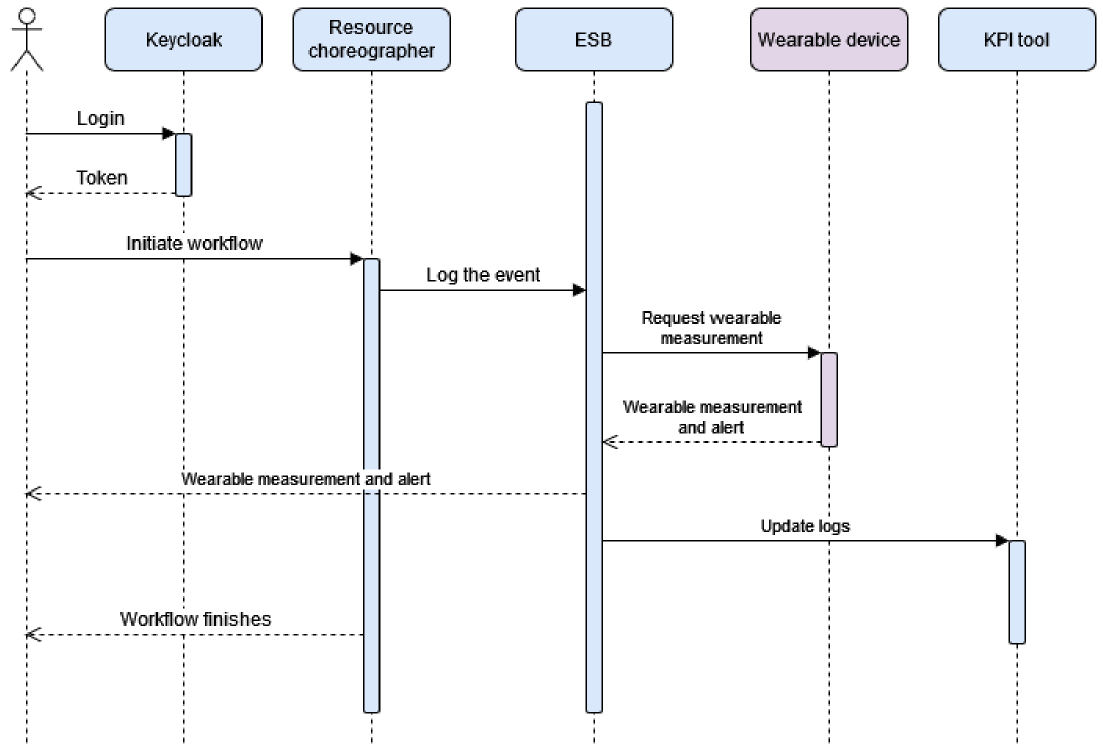


Figure 45: Example sequence diagram for UC6 - Inpatient remote rehabilitation.

4.2.7 UC7: Disaster preparedness

UC7 deals with disaster preparedness. In case of a disaster, such as natural disaster, a new wave of COVID-19 or any other circumstance that would alter the normal performance of the centre, ODIN will be able to predict which safety rules have to be adopted, which devices will be needed and estimate how many qualified professionals are needed to handle the situation. It will also check that the available stock is enough to satisfy the needs of the area that the centre covers. Basically, ODIN will be able to extract conclusions from circumstances either given as an input by the staff or predicted by ODIN. Figure 46 presents an example of a disaster management scenario.

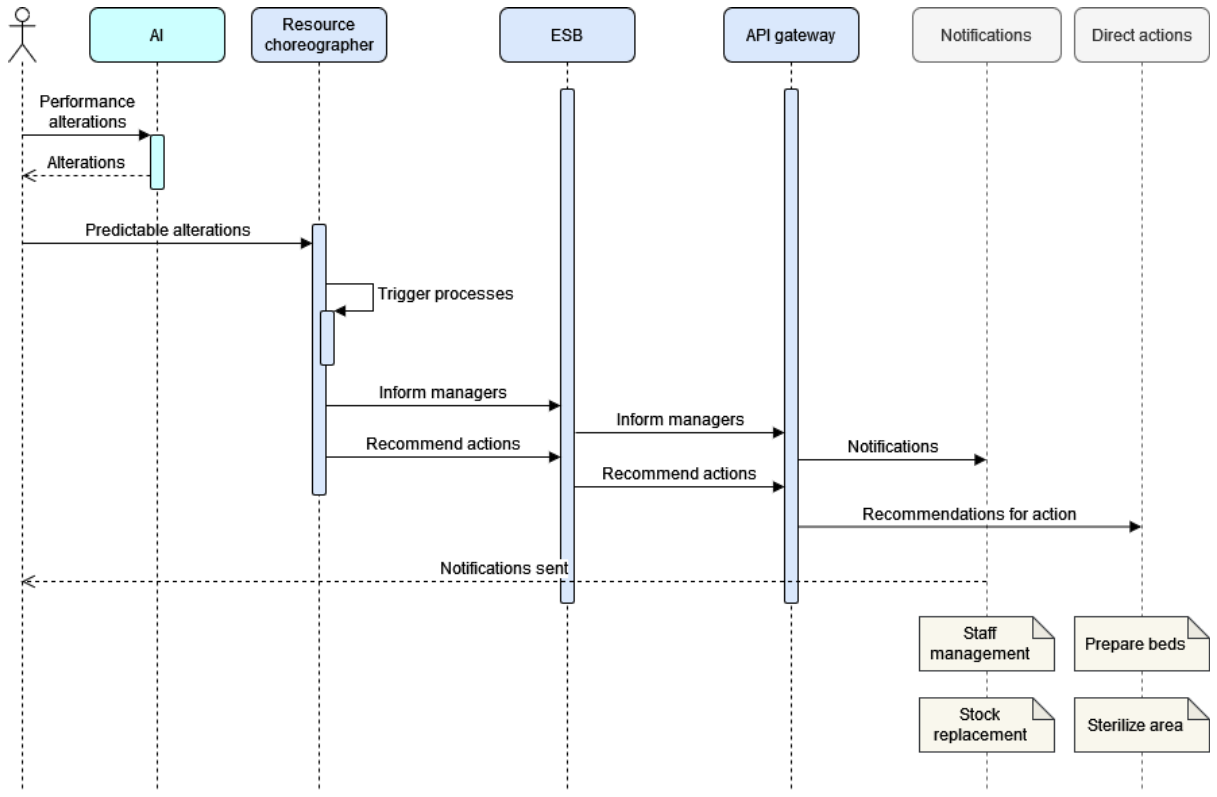


Figure 46: Example sequence diagram for UC7 – Disaster preparedness.

5 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 Section 3.10.2). 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[40].

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

- Development and testing of each component of the ODIN platform
- Distribution of each component as a dockerized application
- Specification of each component's API, i.e. how its functionalities can be accessed by other components
- Definition of how all components are composed together to form the complete ODIN platform (i.e. the global "docker-compose" file).

At this stage of the project, integration is at a very early stage. Individual components are under development, so they are not in the form of dockerized applications yet, and their APIs are not specified yet. The completeness status of each component and their APIs will be described in future versions of this deliverable. In this version, templates to use for API specification and notes regarding continuous development and integration are included, as preparation activities.

To specify the Application Programming Interface (API) of each ODIN component, each component will be described in terms of its functionalities it offers. These functionalities are usually expressed as a set of programming functions (or methods) that can be called by other components. The specification of these methods for each component will be documented using the template of Table 3.

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

Table 3: Template for component function API specification.

Component	*Component name*		
Function name	*Name of function to specify*		
Description	*A short description of what the function does*		
Inputs *(the inputs the function accepts)*	Name	Type	Description
	The name of the first input	*The type of the first input (i.e. number, string, boolean, etc.)*	*A short description of the input*
	The name of the second input	…	…
Outputs *(the possible outputs of the function)*	Name	Type	Description
	A name for the first output	*The type of the first output (i.e. number, string, boolean, etc.)*	*A short description of the output*
	A name for the second output	…	…

As an example, Table 4 specifies the API of the "add resource" function of the "Resource manager" component.

Table 4: Example component function specification API.

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.

The API specification will be facilitated by commonly-used API specification systems, such as Swagger[41], which also allows the direct use of the API from its web interface.

ODIN will follow a continuous development and integration (CI/CD) approach, with components being integrated once available and modifications being made once problems arise or new functionalities become necessary. The steps involved in the ODIN CI/CD pipeline have been detailed in D3.1 "Operational framework".

[41] Swagger, https://swagger.io/

6 Platform deployment

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

6.1 Deployment types

The ODIN architecture depicted in Figure 1 corresponds to a single ODIN instance, i.e. a specific installation of the ODIN platform at a specific site. The core components of the ODIN platform are the same regardless of the characteristics of the target site. However, the deployed KERs connected to the core platform may be quite different in different target environments. The following types of deployment sites are foreseen in ODIN.

- Hospital: This is the main target deployment site. As a deployment site, a hospital is a challenging environment, where diverse KERs need to be deployed, including robotic platforms, sensors, smart medical devices, front-end and back-end applications, AI algorithms for decision making, etc. In a hospital, the ODIN platform needs to be connected with the Hospital Information System to get access to hospital data and enable automation of existing hospital procedures.
- Residence: The architecture of the ODIN platform allows it to be installed in other facilities, such as individual residences (homes) of patients, e.g. allowing remote monitoring and rehabilitation. In a residence environment, the focus is mostly on the installation of monitoring IoT devices, e.g. motion sensors, connected blood pressure monitors, etc., and perhaps on in-house robotic assistance, if necessary. Residence instances will probably depend on hospital- or cloud-based instances to process the collected data and provide assistance to the residents.
- Cloud: These instances are installed in cloud virtual machines, with the purpose of offering software functionalities that are shared among other ODIN instances (e.g. public AI APIs), or to offer computationally expensive operations to ODIN instances that do not possess the necessary computational power. Back-end and front-end applications, AI algorithms and databases are among the kinds of resources that can be deployed in cloud ODIN instances. Access to these resources by other ODIN instances is accomplished through the resource federation mechanisms described in Section 3.5.

6.2 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 3.5) in establishing trust among ODIN instances: the former provides secure channels of communication, while the latter provides auditability, traceability and non-repudiation.

6.2.1 Overview

A virtual private network (VPN) extends a private network through encrypted connections over a public network and provides to its users access as if their computing devices were directly connected to the same private network [1]. Benefits of such a network include increased flexibility, functionality, and security. A VPN enables an organization to provide to its members access to internal resources that are inaccessible on the public network while enforcing its internal policies (security, QoS).

A typical VPN is established through a virtual point-to-point connection using tunnelling protocols that deliver and separate data streams over the same infrastructure or dedicated circuits. Typical tunnelling protocols include GRE, L2TP, MPLS, PPTP and IPsec. From a user perspective, the resources available within the private network can be accessed remotely [3].

Security features of a typical VPN connection include:

- Data confidentiality, using encryption protocols such as IPsec and SSL/TLS.
- Data integrity, using specialized methods such as TCP checksums and Authentication Headers (AH) or Encapsulating Security Payload (ESP) in IPsec.
- Authentication mechanisms (e.g., digital certificates) that are used to set up the tunnel.

6.2.2 Classification

VPN solutions can be classified according to several characteristics. Several are summarized in Figure 47 and are the following [4]:

- The deployment scenario used (i.e., access or site-to-site)
- The model used by the shared infrastructure (i.e., overlay or peer)
- The provisioning method (i.e., customer or provider)
- The OSI layer on which the VPN connectivity is established (i.e., Layer 2, Layer 3, or Layer 4).

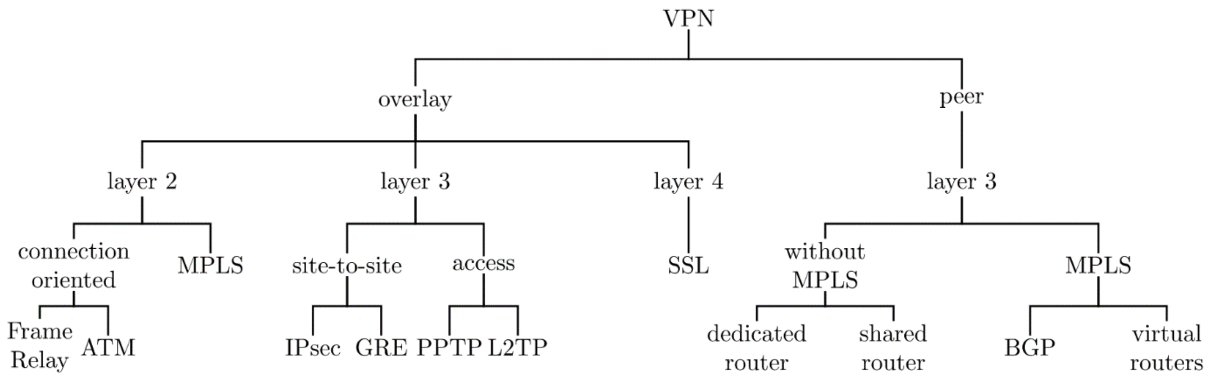


Figure 47: Classification of VPN solutions [5].

6.2.2.1 Deployment scenarios

VPNs can be deployed in two main scenarios [4]:

- Access VPN (named also “remote VPN” or “virtual dial in”): This deployment virtualizes a dial-up connection and connects a single user to an organization network through systems such as ISDN, PSTN, cable modem, wireless LAN by using protocols such as PPTP and L2TP.
- Site-to-site VPN: This deployment virtualizes leased line and connects multiple remote networks with one another by using tunnelling protocols such as IPsec, GRE and MPLS. Site-to-site VPNs are usually deployed in two ways [6]:
    - Intranet VPN where all the interconnected networks belong to the same organization.

- Extranet VPN where the interconnected networks belong to multiple organizations.

6.2.2.2 Models

Regarding the role of the shared infrastructure, VPNs use two models (shown also in Figure 48) [4]:

- In the *overlay model*, the infrastructure is unaware of the VPN solution and merely offers the IP connectivity service (i.e., packets sent between VPN gateways traverse the network without the infrastructure knowing that they are VPN packets). This has the benefits of data privacy since the infrastructure is unaware of the VPN connection.
- In the *peer model*, the gateways inside the infrastructure participate to the creation of the VPN and interact for the VPN connectivity. This has the benefit of better routing performance and scalability.

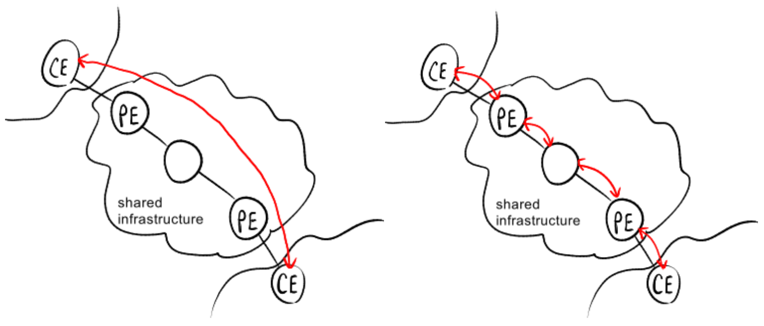


Figure 48: VPN models. Left: Overlay model [17]. Right: Peer model [18].

6.2.2.3 Provision

Regarding the way VPNs are provisioned, there are two main methods (also shown in Figure 49):

- In the *customer provision*, the users create and manage the VPN by themselves, and tunnels are set up between Customer Edges (CE).
- In the *provider provision*, the VPN is provided and managed by the provider of internet connectivity, and tunnels are set up between Provider Edges (PE).

The customer provisioned VPNs cannot be peer models because the provider cannot be aware of VPNs self-created by the customer.

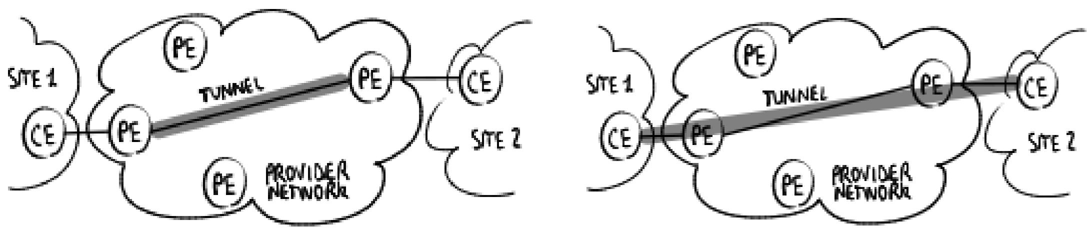


Figure 49: VPN provision types. Left: Provider provision [19]. Right: Customer provision [20].

6.2.2.4 OSI Layers

VPN connectivity can be at different OSI layers:

- In Layer 2, Ethernet frames are exchanged in the VPN through different methods:
    - Virtual Private LAN Service (VPLS): This method virtualizes a LAN and terminals are connected as if they were in the same LAN.
    - Virtual Private Wire Service (VPWS): This method virtualizes a leased line (over a packet-switching network).
    - IP-only LAN-like Service (IPLS): This method virtualizes an IP network, but only IP (ICMP and ARP) packets are allowed.
- In Layer 3, IP packets are exchanged in the VPN.
- In Layer 4, TCP or UDP (usually with SSL for security) connections are established in the VPN.

6.2.3 Protocols

There are several tunnelling protocols that have been developed to support VPN solutions. Some of them are included below:

- The Point-to-Point Protocol (PPP) is a Layer 2 protocol used in point-to-point connections (e.g., dial-up, ISDN) to encapsulate protocols at upper layers.
- The Secure Socket Tunnelling Protocol (SSTP) by Microsoft uses the PPP protocol to tunnel traffic through an SSL/TLS channel (SSTP was introduced in Windows Server 2008 and in Windows Vista Service Pack 1).
- The Generic Routing Encapsulation (GRE) is a protocol to encapsulate any protocol (including IP and other protocols at lower layers) into IP. Also, a version of GRE (named "enhanced GRE") was designed for PPP encapsulation in the PPTP protocol.
- The Layer 2 Tunnelling Protocol (L2TP) is a protocol that tunnels any Layer 2 protocol (e.g., PPP) into IP. L2TP was originally designed for provider provisioned access VPNs, and was standardized by the Internet Engineering Task Force (IETF).
- The Point-to-Point Tunnelling Protocol (PPTP) is a protocol to tunnel the PPP protocol into IP. PPTP was originally designed for customer provisioned access VPNs, and it was developed by major operating system vendors. However, it is regarded as obsolete due to well-known security issues.
- The Internet Protocol Security (IPsec) was initially developed by the IETF for IPv6, which was required in all standard-compliant implementations of IPv6 before RFC 6434 made it only a recommendation [7]. This protocol uses encryption, encapsulating an IP packet inside an IPsec packet. De-encapsulation happens at the end of the tunnel, where the original IP packet is decrypted and forwarded to its intended destination.
- Transport Layer Security (SSL/TLS) can tunnel an entire network's traffic (e.g., in the OpenVPN project) or secure an individual connection. Several vendors provide remote-access VPN capabilities through SSL. An SSL VPN can connect from locations where IPsec runs into trouble with Network Address Translation (NAT) and firewall rules.
- WireGuard is a VPN protocol that uses UDP to tunnel traffic. It was designed with the goals of ease of use, high speed performance, and low attack surface [8]. It aims for better performance and more power than IPsec and OpenVPN, two common tunnelling

protocols [9][10]. In 2020, WireGuard support was added to the Linux and Android kernels [11].

- The Internet Key Exchange volume 2 protocol (IKEv2) was created by Microsoft and Cisco and is used in conjunction with IPsec for encryption and authentication. Its primary use is in mobile devices, whether on 3G or 4G LTE networks since it is effective at re-joining when a connection is lost.

6.2.4 Multi-cluster configuration and VPN connectivity in Kubernetes

In ODIN, several hospitals will require running the ODIN platform on their premises and thus the management, automation, configuration, flexibility, and scalability of the ODIN platform is of utmost importance. To address these challenges, the popular framework of Kubernetes has been selected to be used to manage containerized environments and workloads.

In the ODIN architecture (Figure 1), the interconnection between the ODIN platform instances is shown. This translates to establishing a multi-cluster solution in Kubernetes that will enable the seamless integration, monitoring, re-configuration, rollout, and rollback of the ODIN platform services across the hospitals. Unfortunately, the issue of having multiple interconnected Kubernetes clusters is currently complex and there is no one-solution-fits-all situation. Several projects are aiming to address this need with each one having its own benefits and drawbacks.

There are two large families of solutions currently developed to tackle the Kubernetes multi-cluster configuration issue [12]. The first family revolves around extending the fundamental Kubernetes building blocks (e.g., the API server or the Kubelet component, etc.) and thus enable easy configuration of a multi-cluster scenario. The second family is focused on creating network connectivity between clusters so that applications and services within clusters can communicate with each other [13].

6.2.4.1 Multi-cluster Control Plane

The first family of solutions has worked on supporting and extending the core Kubernetes primitives for multi-cluster use cases to enable a centralized management plane for multiple clusters.

6.2.4.1.1 Dedicated API Server

The official Kubernetes Cluster Federation (KubeFed[42]) project represents an example of this approach, which “allows you to coordinate the configuration of multiple Kubernetes clusters from a single set of APIs in a hosting cluster” [14]. KubeFed extends the traditional Kubernetes APIs with new semantics for expressing which clusters should be selected for a specific deployment.

[42] KubeFed, https://github.com/kubernetes-sigs/kubefed

6.2.4.1.2 Virtual kubelet-based approaches

Virtual Kubelet (VK)[43] is a "Kubernetes kubelet implementation that masquerades as a kubelet to connect Kubernetes to other APIs" [15]. Initial VK implementations model a remote service as a node of the cluster used as a placeholder to introduce serverless computing in Kubernetes clusters such as Azure Container Instances (ACI), Amazon Web Services (AWS) Fargate and others. Lately, VK has gained popularity as a multi-cluster solution, since a VK provider is able to map a remote cluster to a local cluster node. Several open-source projects, including Admiralty[44], Tensile-kube[45], and Liqo[46], adopt this approach. This approach has several advantages compared to a dedicated API Server. These include no need for extra APIs configuration, transparency to the applications, flexible integration of remote cluster resources in the default scheduler's availability and decentralized governance.

6.2.4.2 Network Interconnection Tools

The second family of solutions include the networking interconnection aspects of multi-cluster topologies. This approach stems from the need for pods to communicate with pods on other clusters and services seamlessly. Inter-cluster connectivity can be provided by extensions of the Container Network Interface (CNI), the component responsible for cluster connectivity, or by dedicated tools. The important design choices for interconnection tools involve mainly three aspects: (1) interoperability with different cluster configurations, (2) compatibility between network parameters used in the other clusters, and (3) a systematic manner for connecting services exposed on all clusters.

6.2.4.2.1 CNI-provided interconnection

Cilium Cluster Mesh[47] represents an example of a CNI implementing multi-cluster interconnection. More precisely, Cluster Mesh extends the capacity of the popular Cilium CNI to "federate" multiple Cilium instances on different clusters. Cilium supports pod IP routing across numerous Kubernetes groups via tunnelling or direct routing without requiring any gateways or proxies. Moreover, it promotes transparent service discovery with standard Kubernetes services and CoreDNS. The main drawback of approaches like Cluster Mesh is the strict dependency on a given CNI, namely Cilium since this must be adopted in all clusters. Also, Cilium requires pod Classless Inter-Domain Routing (CIDR) uniqueness across clusters.

[43] Virtual Kubelet, https://github.com/virtual-kubelet/virtual-kubelet

[44] Admiralty, https://admiralty.io

[45] Tensile-kube, https://github.com/virtual-kubelet/tensile-kube

[46] Ligo, https://liqo.io

[47] Cilium Cluster Mesh, https://docs.cilium.io/en/v1.11/concepts/clustermesh/

6.2.4.2.2 CNI-Agnostic Interconnection

Submariner[48] enables direct networking between Pods and Services in different Kubernetes clusters, either on-premises or in the cloud. It is open-source and designed to be network plugin (CNI) agnostic. Also, it offers cross-cluster Layer 3 connectivity using encrypted (or unencrypted) connections. Submariner has a centralized architecture based on a broker that collects information about cluster configurations and sends back parameters to use. Submariner does not support services having endpoints spread across multiple clusters (multi-cluster services). Instead, it provides a more straightforward mechanism for discovering remote services, having all the back-end pods in the exact location. Currently Submariner supports two tunnelling protocols, namely IPsec and WireGuard.

Skupper[49] is a Layer 7 multi-cluster interconnection service. It enables secure communication across Kubernetes clusters by defining an ad-hoc virtual networking substrate. Unlike Submariner and Cilium, Skupper does not introduce a cluster-wide interconnection but just for a specific set of namespaces. Skupper implements multi-cluster services in namespaces exposed in the Skupper network. The inter-cluster communication is secured by mutual TLS (mTLS). When a service is exposed, Skupper creates endpoints, making them available on the entire cluster.

Another category of network-related solutions is the mesh VPNs [16]. These tools create a virtual subnet where the worker nodes (independently of location) are deployed. This enables the deployment of clusters with nodes placed in arbitrary environments. From the cluster's perspective, it is one normal Kubernetes cluster, and it is not aware that its nodes are placed in different locations. Simple management is enabled via node selectors. Popular solutions in this category include Nebula[50], Tailscale[51], Twingate[52], Netmaker[53], Kilo[54] and others.

6.2.4.2.3 Service Mesh

Service mesh frameworks are dedicated infrastructure layers to simplify the management and configuration of microservice-based applications. Service meshes introduce a sidecar container as a proxy to provide multiple features (i.e., secure connections with mutual TLS, circuit breaking, canary deployments). Some of the most popular service mesh architectures (e.g.,

[48] Submariner, https://submariner.io

[49] Skupper, https://skupper.io

[50] Nebula, https://github.com/slackhq/nebula

[51] Tailscale, https://tailscale.com

[52] Twingate, https://www.twingate.com

[53] Netmaker, https://github.com/gravitl/netmaker

[54] Kilo, https://kilo.squat.ai

Istio[55], Linkerd[56]) have multi-cluster support to enable and interconnect multi-cluster microservices applications. The interconnection among different clusters uses a dedicated proxy to route traffic from the mesh of one cluster to another. Similarly, Istio and Linkerd can create an ad-hoc mutual TLS tunnel across clusters and provide primitives to expose services across the clusters, enabling features such as cross-cluster traffic splitting. In general, multi-cluster support in service-mesh frameworks provides a wide range of features. However, they require manual configuration steps and several new specific APIs to configure to set everything up.

6.2.4.3 Considerations for the ODIN platform

Most of the afore-mentioned solutions require public IP addresses to the cluster nodes to work properly. This requirement needs to be considered since most hospital settings have strict network and security policies and usually outside access is allowed only through VPN connections. This complicates the solution of the multi-cluster configuration and needs to be taken into consideration for the integration (e.g., port-forwarding, split tunneling, other). However, besides the VPN complexity, there is the bigger issue of lack in repeatability, i.e., a highly customized solution that will be developed for one hospital, may not work for another.

Furthermore, the afore-mentioned solutions can support usage scenarios where two services in different hospitals can transparently communicate directly. However, in the ODIN project, a simpler solution that seems adequate is to have different ODIN instances that do not communicate directly but through gateways/proxies (see Section 3.8.1). This solution will simplify the hospital integration requirements and is generic enough to support different VPN solutions that may be considered. An example of such solution would be for each hospital to have its own VPN Gateway that all the related ODIN traffic would go through. In each hospital, a VPN client could be created that connects with a VPN server hosted in the cloud through the VPN Gateway. With this approach, each hospital has its own VPN Gateway that can be tailored to its internal security and network policies and the communication traffic between the ODIN services in the different hospitals is routed through the VPN server in the cloud and the VPN Gateways. The solution to adopt for the ODIN platform will be decided through the activities of WP3 and reported in future versions of this deliverable.

6.3 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 Section 3.10.2, 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

[55] Istio, https://istio.io

[56] Linkerd, https://linkerd.io

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[57]. That would increase the number of necessary VMs.

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[58], AKS[59], GKE[60], OKE[61]). 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[62], Azure ExpressRoute[63], Google Cloud Interconnect[64], Oracle Cloud Infrastructure FastConnect[65]) which makes connectivity a lot faster than through a VPN.

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

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

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

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

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

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

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

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

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

6.4 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). 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.

6.5 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.

7 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".

7.1 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.

7.2 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 "docker-compose" file 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.

7.3 User manual

The ODIN user manual will contain information and instructions about the use of the ODIN platform by the end users. The types of users foreseen in ODIN are presented in Table 5, 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 5: 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 0. 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.

8 Platform 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. The template presented in Table 6 will be used to report the developed unit tests.

Table 6: 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 7 provides an overview of the plan for integration/system testing to be followed within ODIN.

Table 7: 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 6.5)
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 Conclusion

This deliverable is the first 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 focuses on the specification of the ODIN architecture and its components, since, up to this stage of the project, the effort has been to clarify the platform architecture in order to guide component development.

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 versions of this deliverable (D3.11 and 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 and integration. Specific activities that will lead to the implementation of the ODIN platform and subsequent reporting in D3.11 and 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

The APIs of the developed components will be reported in D3.11 and D3.12, offering a concrete specification of the implemented functionalities, while the components themselves will be available in the relevant repositories. The scenarios used for unit and integration tests will be also reported, as well as the test results. The documentation accompanying the developed components will be gathered to compile the ODIN manuals for the developer, deployer and end-user. The deliverable will act as a guide for the development of a minimum ODIN instance to be initially deployed at the pilot sites during the next months, as part of WP7 activities.

As a roadmap towards the next versions of the ODIN platform implementation, Figure 50 depicts a timeline of the ODIN platform versions throughout the project duration. At the current point in 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.

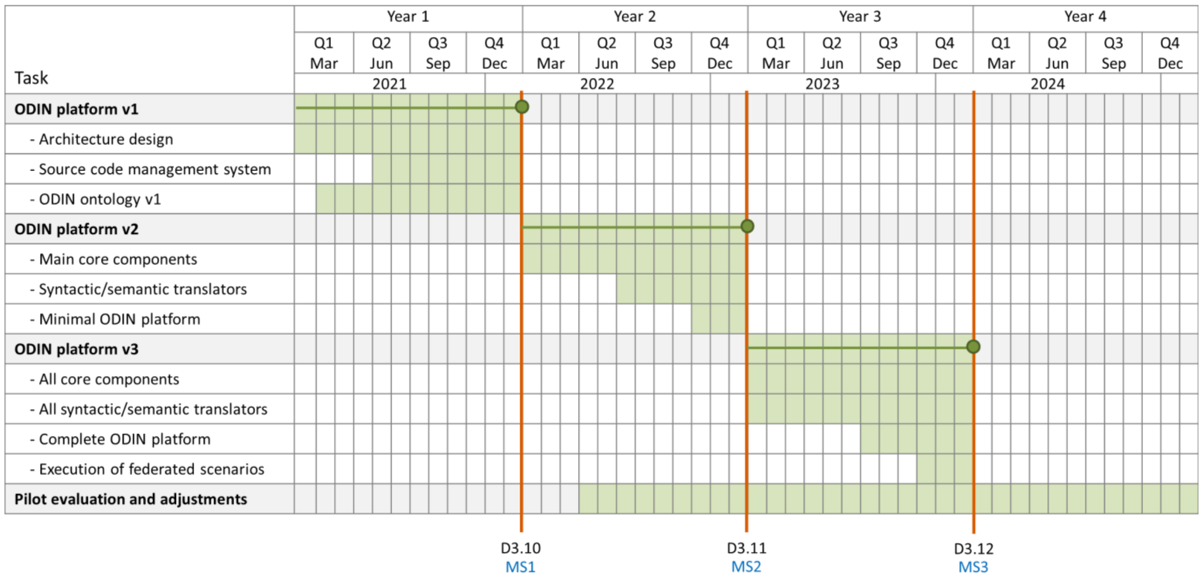


Figure 50: Timeline and characteristics of the ODIN platform versions.

During the second year, activities will focus 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.

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