pharaon

PILOTS FOR HEALTHY AND ACTIVE AGEING

Grant Agreement: 857188

D4.6 Service orchestration support tools - Final

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 857188.

Document Information

Deliverable number:	**D4.6**
Deliverable title:	Service orchestration support tools - final
Deliverable version:	1.0
Work Package number:	WP4
Work Package title:	Technology Support Tools
Due Date of delivery:	M54 (May 2024)
Actual Date of delivery:	M54 (May 2024)
Dissemination level:	Public (PU)
Type	Other (O)
Editor(s):	Andrej Grguric (ENT), Miran Mosmondor (ENT)
Contributor(s):	Miran Mosmondor (ENT) Andrej Grguric (ENT) Stefania D'Agostini (ENG) Alex McDonald (ICE) Carolin Wübbe (ASC)
Reviewer(s):	Mariana Camacho (SCMA)
Project name:	Pilots for Healthy and Active Ageing
Project Acronym	PHArA-ON
Project starting date:	01/12/2019
Project duration:	60 months
Rights:	PHArA-ON Consortium

Document history

Version	Date	Beneficiary	Description
0.1	16.2.2024	ENT	Table of Contents proposal and initial content considerations for each chapter
0.2	29.3.2024	ENT	Initial content about data transfer orchestration tools (Section 6)
0.3	19.4.2024	ENT	Completed overview of open-source data orchestration tools (Section 6.1)
0.4	2.5.2024	ICE	Added/updated content of Section 5.1 Service Registry in Pharaon and Section 5.3 Service Orchestration Tools
0.5	10.5.2024	ENT	Updated container registry in Pharaon in Section 5.2
0.6	16.5.2024	ENG	Add content about FIWARE and Orion Context Broker (Section 6.3)
0.7	21.5.2024	ENT	Added content about International Data Spaces (IDS) in Section 6.2.1
0.8	24.5.2024	ENT	Added content to related to architectural considerations of using IDS in Pharaon (Section 6.2.2)
0.9	27.5.2024	ENT	Added initial description of sharing data between Pharaon platforms and technologies using open-source IDS components (Section 6.2.3)
0.10	28.5.2024	ENT	Updated introduction, terminology section, overall chapter improvements.
0.11	29.5.2024	ENT	Updated introduction, relations and terminology sections, overall document improvements.
0.12	30.5.2024	ASC	Added description and usage of Matomo in pilots (Section 4 Monitoring Orchestration Tools)
0.13	03.06.2024	ENT	Update of Executive Summary, Progress beyond the state-of-the-art and play and Conclusion
0.14	05.06.2024	ENT	Overall minor improvement to all chapters, final wrap up
0.15	06.06.2024	SCMA	Internal review
1.0	07.06.2024	ENT	Deliverable final updates

PM efforts per beneficiary having contributed to the deliverable document[1]

#	Partner	PM effort in this deliverable
16	SCMA	0.00
27	ASC	0.01
28	ENT	0.60
32	ICE	0.02
45	ENG	0.20
TOTAL	Pharaon Consortium	0.83

Acknowledgement: This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 857188.

[1] Efforts for contributing to the preparation of document that describes the work done

Executive Summary

This deliverable reports on the work carried out within the task T4.2 “Service orchestration and customization support tools” of the fourth work package (WP4) of the PHArA-ON (in further text "Pharaon") project.

This deliverable is the third one and final within task T4.2. that provided technological support for orchestration and customization activities in the Pharaon project. We defined Pharaon service orchestration as the processes, procedures, guidelines, and tools implemented to manage multiple platforms, technologies (both internal and external to Pharaon, the latter onboarded via WP6 open calls), services and APIs. In the process, tight coordination has been established with task T4.3 “Service composition support tools” (parallel deliverables D4.7, D4.8 and D4.9 which report on the T4.3 activities regarding the developed ICE Orchestration tool). Also, other WP4 and cross-technical WP level cooperation has been conducted primarily within the SWAT1 task force lead by the project’s TM (Technical Manager).

The main achievements of task T4.2 have been already documented in previous versions of this deliverable, namely D4.4 and D4.5, which presented a comprehensive approach to service orchestration in Pharaon, state of the art technologies and tools that were used (like OpenAPI, GraphQL), collaborative efforts on API design and management, OpenAPI-based interactive and always up-to-date documentation of the Pharaon REST API, container and monitoring orchestration tools used in pilots, and other.

This version of the deliverable focuses on reporting on additional activities carried out within task T4.3 that were defined after the second project review and the definition of the Interoperability plan in June 2023. The main goal was to help and contribute to the project goals of defining, developing and adapting the enablers for information sharing and development of an extendable, multi-layer, and versatile interoperability environment (defined within WP3’s description of work). To help support achieving these goals, the work reported within this deliverable focused on the usage of open-source components and frameworks that enable interoperable and trusted data exchange and data-based services among various stakeholders. One of such is International Data Spaces (IDS), a virtual space that provides a standardized framework for data exchange, based on common protocols and formats, as well as secure and trusted data sharing mechanisms. Within this task, the analysis of IDS in Pharaon’s overall architecture has been made, and several open-source IDS components have been analysed and deployed in the Pharaon staging environment. Also, the usage of FIWARE open-source components has been analysed within the scope of this deliverable.

This deliverable report also presents updates to the set of provided orchestration tools and processes described in previous deliverables, for example improved monitoring orchestration tools used in certain pilots. Other work related to this task that was completed in the previous versions of the deliverable is summarized in this document to provide reference for all T4.2 activities in a single final document.

Progress beyond the state-of-the-art and play

Deliverables within this task documented the support activities related to Pharaon API design, documentation and orchestration. Previous versions of the deliverable described *the design first* approach for new Pharaon APIs, presented the OpenAPI standard adopted within Pharaon and reported on GraphQL as orchestration layers for downstream APIs. These deliverables also reported on the container orchestration tools and analysed logging and monitoring orchestration for the Pharaon environments. The latest version of the deliverable goes further with analysis and usage of orchestration tools that enable interoperable and trusted data exchange and the creation of data spaces in Pharaon. The EU's vision for the common European data space, defined within the European data strategy, is based on the principles of fundamental European values, and envisages the sharing of data within the EU and across sectors. Common European data spaces will facilitate trustworthy and affordable data exchange for businesses and the public sector, which will accelerate the creation of new data-driven goods and services[2]. Several organizations are pivotal in implementing this data economy, including the Business Data Value Association (BDVA), the Gaia-X European Association for Data and Cloud AISBL, FIWARE Foundation, and the International Data Space Association (IDSA). The usage of outputs from the latter two organizations, IDSA and FIWARE Foundation, that enable standardized trusted data exchange and data-based services, have been further analysed within the work on this deliverable.

[2] https://digital-strategy.ec.europa.eu/en/library/building-data-economy-brochure

Contents

Figures

Tables

Acronyms & Abbreviations

Term	Description
API	Application Program Interface
ADDR	Align-Define-Design-Refine
BFF	Backend For Frontend
CaaS	Container as a Service
GDPR	General Data Protection Regulation
HATEOAS	Hypermedia as the Engine of Application State
HTTP	Hypertext Transfer Protocol
IDS	International Data Spaces
JSON	JavaScript Object Notation
REST	Representational state transfer
SSOT	Single source of truth
TM	Technical Manager
WP	Work Package
WP3	Pharaon WP3 Secure Interoperability Solution
WP4	Pharaon WP4 Technology Support Tools
WP5	Pharaon WP5 Technology Ecosystem Integration
WP6	Pharaon WP6 Ecosystem Evolution
XML	Extensible Markup Language
YAML	Yet Another Markup Language

1 Introduction

1.1 Overview

This deliverable is the third and final one within task T4.2 “Service orchestration and customization support tools” aiming to provide support for technical activities in the scope of Pharaon (primarily conducted in WP3 and WP5) with a focus on orchestration support. The work within this task (T4.2) continued to facilitate the coordination between Pharaon technical providers and more concretely facilitate the Pharaon services and data orchestration.

We defined Pharaon service orchestration as the processes, procedures, guidelines, and tools implemented to manage multiple platforms, technologies (both internal and external to Pharaon, the latter onboarded via WP6 open calls), services and APIs. The goal was to help create a seamless developer experience that enables co-creation across multiple platforms, technologies, and technology providers.

1.2 Relation to other tasks and deliverables

As it was already described in detail in D4.3, WP4 delivers technology supporting tools to Pharaon developers during the development of interoperable open platforms, tools, and technologies in WP3 and WP5; to Pharaon technical partners for the deployment and operations phase in WP7, and to external developers. WP4 activities are not about developing platforms and tools for end users, but technologies, tools, and other assets for developers to help them to create and operate these interoperable platforms and tools. The overview is illustrated with Figure 1.1.

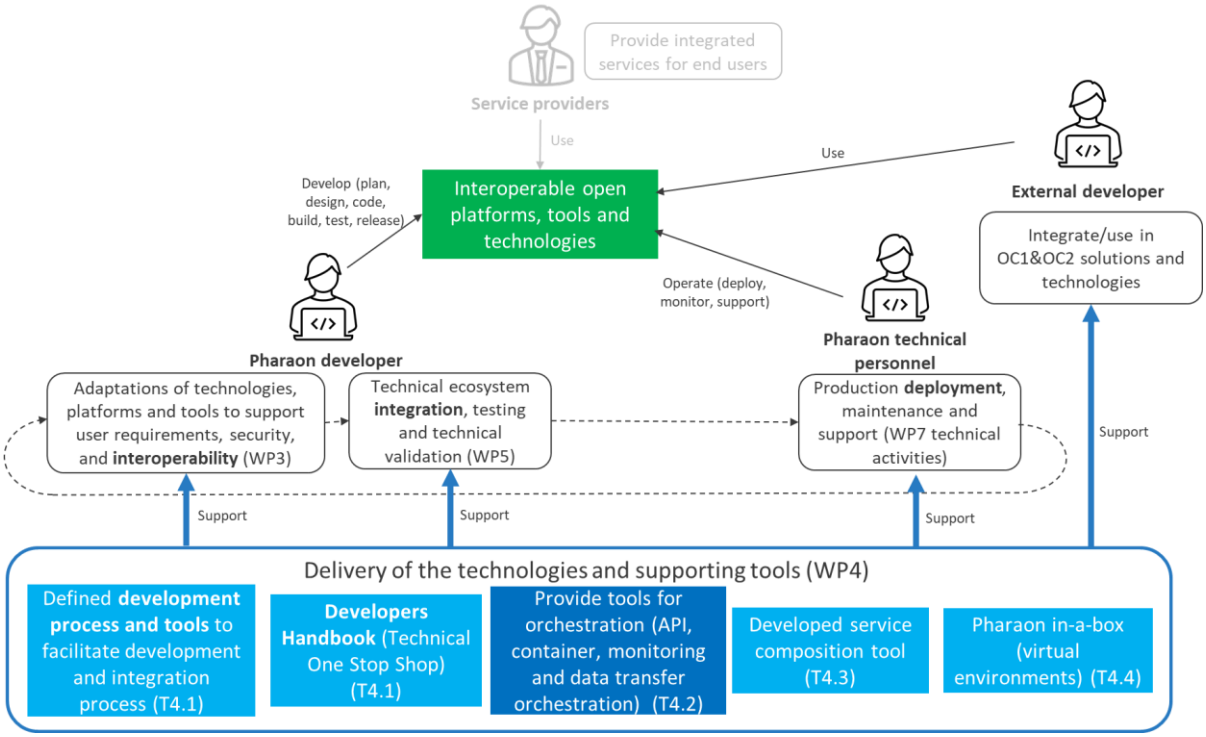

Figure 1.1 Overview of the main Pharaon technical activities and WP4 tasks (from D4.3)

The main inputs to this deliverable come mostly from WP3 and WP5 activities but other project activities related to the technical aspects influenced this work as well. The specific inputs and outputs are shown in the following figure (Figure 1.2).

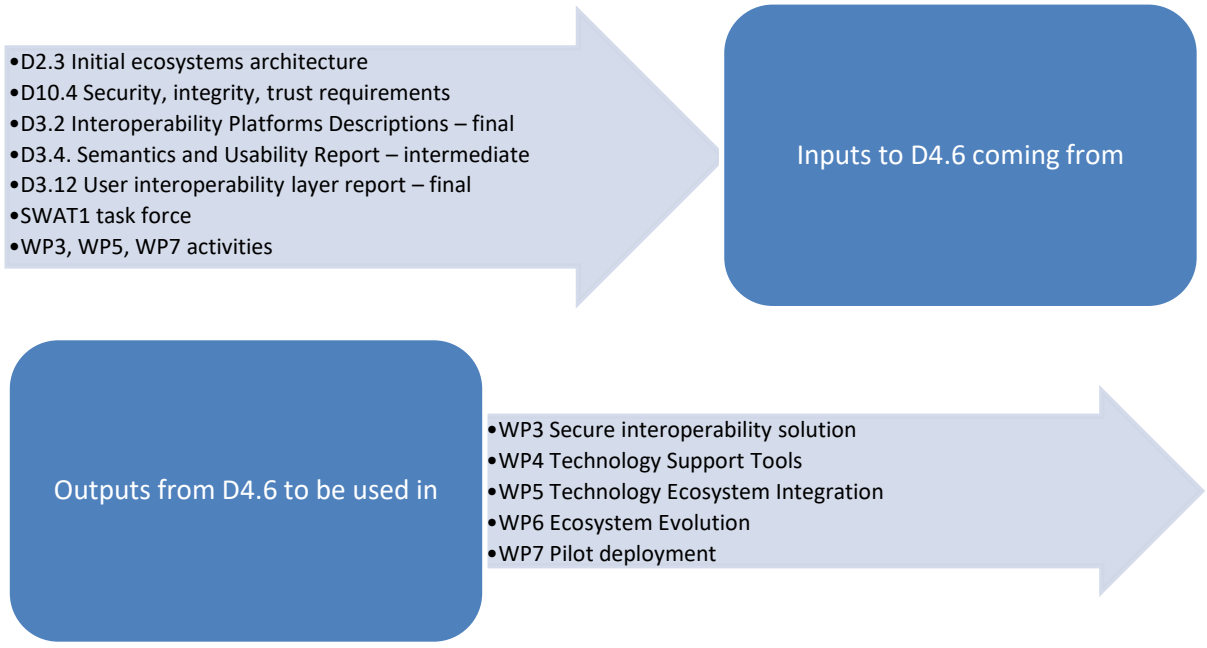

Figure 1.2 Inputs and outputs of D4.6 concerning other project work

1.3 Structure of the deliverable

This document describes the latest work regarding service orchestration and customization support tools, and it is organized in the following chapters:

- After the executive summary and acronyms used in the document, an Introduction of the document is given, including the relation to other project tasks and deliverables.
- Chapter 2 summarizes work done regarding Pharaon API design, documentation and orchestration, completed in previous version of this deliverable.
- Chapter 3 also summarizes work reported in previous version of this deliverable, regarding containers orchestration and Kubernetes in Pharaon.
- Chapter 4 elaborates on the monitoring orchestration tools with the description of an additional open-source monitoring tool that was used in certain pilots.
- Chapter 5 summarizes the work related to registries and service remigration in Pharaon, work that is being completed within in the scope of task T4.3 and described in detail in D4.9.
- Chapter 6 describes work done within this deliverable related to usage of open-source components and frameworks that enable trusted data exchange and data-based services among various stakeholders.
- Chapter 8 concludes the report.

1.4 Terminology

Some of the terms we use here we defined as follows (the majority was defined in the previous version of the deliverable, D4.5, with some updates in this version):

- **Service registry** is a database of services, their instances, and their locations
    - Is a key part of service discovery
    - Single source of truth (SSOT) - as a concept that an organization (or in our case project such as Pharaon) can apply as part of its information architecture to ensure that everyone uses the same information (in our case the web location of the Pharaon APIs).
- **Service registration** – the process of service registration its location in a central registry
    - usually registers its host and port and sometimes authentication credentials, protocols, versions numbers, and/or environment details
- **Service discovery** – the process of a client app querying the central registry to learn the location of services
- **API management** - the process of designing, implementing, securing, managing, monitoring, and publishing APIs
- **API gateway** is a component of API management.
    - An API gateway sits between backend services and a client (requester) to transmit requests and responses. An API gateway receives calls, aggregates services to fulfill them, and returns a result.
    - It provides a unified entry point across internal APIs.

    - Has a more robust set of features than an **API proxy**.
- **API catalogue** is a library of available APIs, often shared through the **API portal**
- **API proxy -** decouples the app-facing API from backend services, shielding those applications from backend code changes
- **API orchestration** is the process of integrating applications into a single offering. It often requires creating a single API that offers valuable functions to its consumers, often by making multiple calls to multiple different services to respond to a single API request
- **Data orchestration** refers to the process of coordinating and processing data across multiple systems and services
- **Service orchestration** in Pharaon is envisioned as the process, procedure, guidelines, and tools implemented to manage multiple platforms, technologies (both internal and external to Pharaon, the latter onboarded via WP6 open calls), services and APIs. In Pharaon, every technology provider maintains its own infrastructure and simply exposes integration endpoints (via web APIs) on the public Internet from different sandboxes (further explained in D4.12).

2 Pharaon API design, documentation and orchestration

In the previous versions of this deliverable, D4.4 and D4.5, it was reported about the Pharaon API design, documentation and orchestration. The reports provided recommendations for the design and implementation of the new APIs that are being created to enable Pharaon use cases and scenarios and to document existing APIs following well-established standards and tools to ease the integration and orchestration process and make it more efficient. Considering that this work was needed in the early stages of the Pharaon development process, there have been only minor updates regarding Pharaon API's design, documentation and orchestration in this latest deliverable. For completeness reasons and to provide a reference location for all T4.2 activities in a single final document, the following subsections in this chapter serve to summarize the work done in the previous versions of this deliverable (D4.4 and D4.5).

2.1 Pharaon API design-first process

Pharaon API design first approach was described in the previous version of this deliverable (D4.5). As previously mentioned, many APIs used in Pharaon are inherited from input technologies and were not rewritten and developed from scratch. With that respect, work presented in previous deliverables recommended methods to increase efficiency, such as applying interactive documentation (see chapter 2.3) for the existing REST APIs. However, some of the APIs were developed from scratch, either because of new functionality or improved API quality. For that case, in the scope of T4.2, the *API design-first approach* by James Higginboarham that uses the "**Align-Define-Design-Refine" (ADDR)** process was recommended as suitable for Pharaon (see Figure 2.1). As described in D4.5, the ADDR, as an iterative process, guides developers through API design-first techniques ensuring that all stakeholders are actively involved. In the case of Pharaon, it means that the consumers of some new APIs (usually other Pharaon internal or external developers) are involved from the very beginning of the new API development and that the developer of the API first introduces the design of the API and goes to the next phase of development only when the API is "preapproved" by the consumer. The goal was to deliver the desired API with desired quality and to prevent breaking changes and large code refactoring. It also allowed other developers using the API to start the integration process without waiting for full implementations to become ready and deployed in a staging (integration) environment.

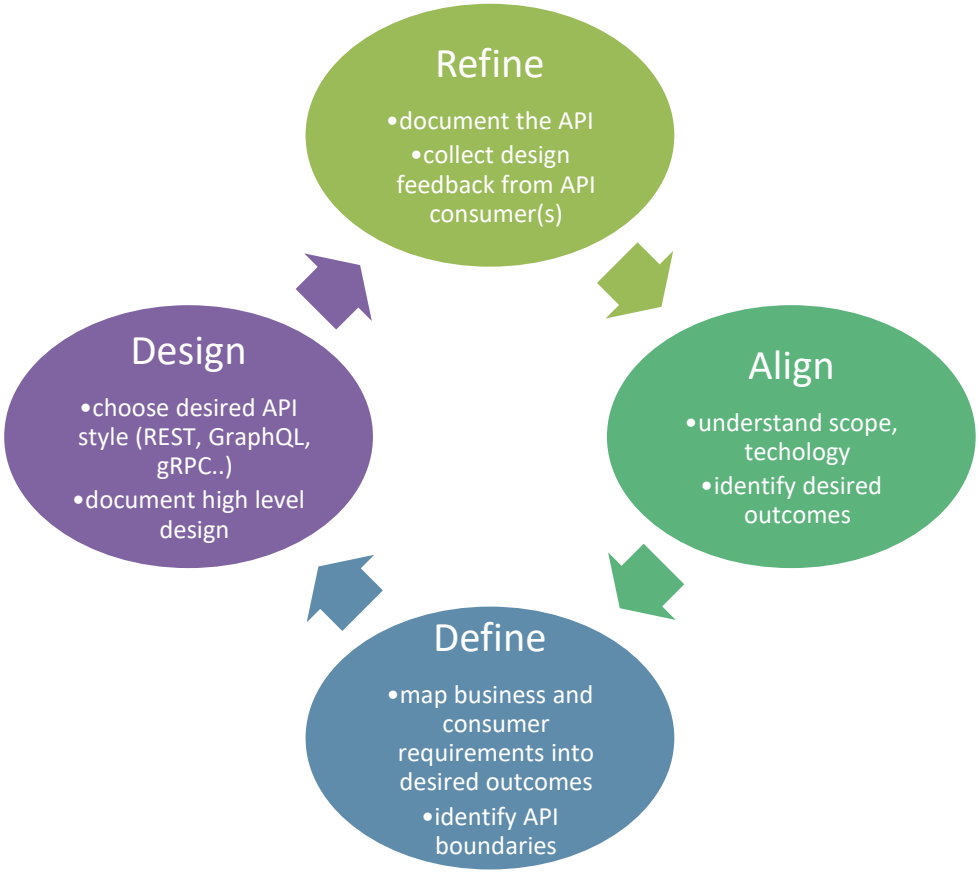

Figure 2.1 ADDR (Align-Define-Design-Refine) process for Pharaon's new APIs[3]

2.2 Quality properties of Pharaon APIs

The previous version of the deliverable also presented quality properties for Pharaon APIs. It was essential that the input solutions and services are *consumable* in terms that they are capable of being used or engaged with, which is a prerequisite of integration and orchestration. In addition, the property of being compliant with standards, delivering what they promise and being accessible is vital. The most important properties of the APIs that are to be integrated into the Pharaon ecosystem are illustrated and summarized in Figure 2.2 below. As described in D4.5, since all web services are APIs the term" service orchestration" is also highly related to "API orchestration" focusing on the act of integrating two or more APIs into a single offering.

[3] updated from D4.5

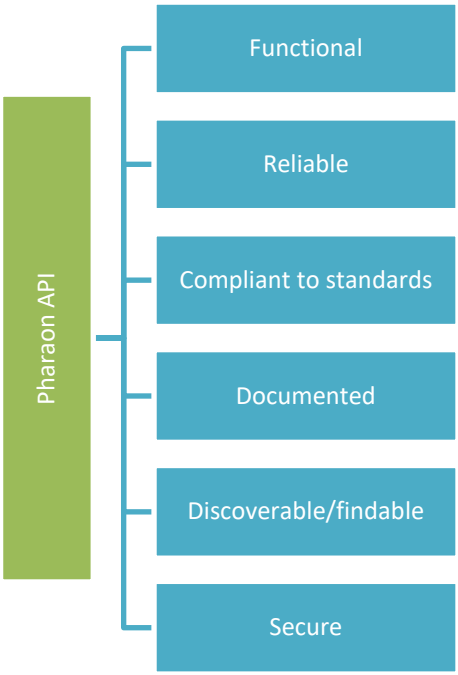

Figure 2.2 Quality properties of Pharaon APIs[4]

Within Pharaon, the goal was to advance the maturity level of Pharaon input technologies APIs so that all APIs are reliable (able to connect to the API and consume its functionality), compliant to standards, properly documented, findable (to share the publicly exposed URLs of the Pharaon APIs) and secure (to have secure endpoints, including authentication mechanisms).

2.3 Interactive documentation of REST APIs in Pharaon

Regarding the interactive documentation of REST API and the process of applying OpenAPI in Pharaon, it was completed in the early stages of the project and described in the Developers Handbook (final version presented in D4.3).

The previous version of this deliverable described usage of Swagger and OpenAPI Specification. Swagger refers to a set of SmartBear tools (more info on https://swagger.io/) which are among the most popular ones for developing and describing RESTful APIs. Swagger tools help users build, document, test and consume RESTful web services and can be used with both a top-down and bottom-up API development approach. The OpenAPI Specification defines a standard interface to RESTful APIs, in particularly it "*defines a standard, programming language-agnostic interface description for HTTP APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic*"[5]

The D4.5 provided examples of REST API interactive documentation based on Swagger2.0 and Swagger3.0 that is used in the Italian and Slovenian pilots for integration work (described in WP5) hosted within the integration test sandbox (as described in D4.11).

[4] updated from D4.5

[5] https://spec.openapis.org/oas/v3.0.3

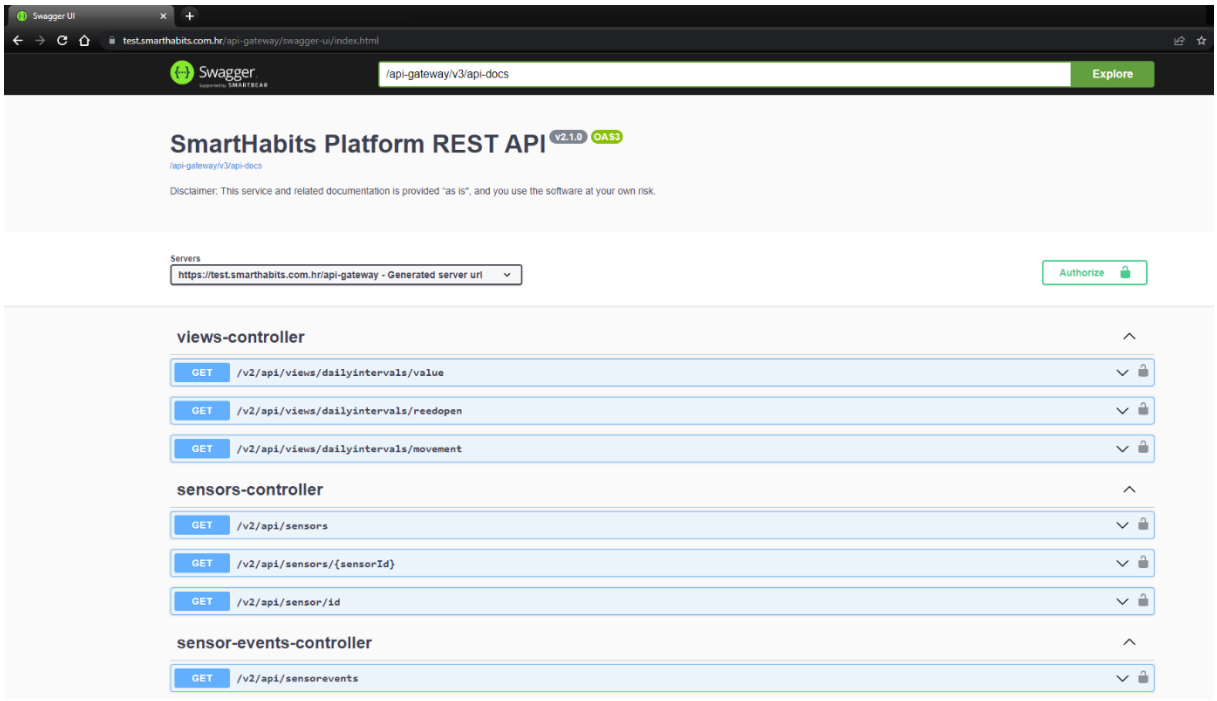

Figure 2.3 REST API interactive documentation for SmartHabits Platform based on Swagger3.0 (from D4.5)

The previous version of this deliverable also detailed the main differences between Swagger2 and OpenAPI3 specifications, considering that adoptions were necessary to switch to the newest specification. These differences included increased API reusability, extended JSON Schema support, changes to security flows, improved parameter descriptions, content type negotiation support, links features, among others. It is important to mention that today there is a newer version of Open API available, OpenAPI 3.1. The most notable changes in comparison to the previous version are full JSON Schema support, dropping semantic versioning, optional paths, and dropping nullable object support.

2.4 Endpoints serving the Pharaon API standardized descriptions

The following table was updated since the last version presented in D4.5 and summarizes the publicly available endpoints serving the Swagger or OpenAPI YAML description of the Pharaon APIs. Those descriptions were also sent as input to ICE Orchestrator (described in the related D4.9 deliverable).

Table 1 Endpoints serving the Pharaon API standardized descriptions (updated)

Pharaon technology solution	Swagger endpoint
Globalcare	NA
HS.Helios Platform	NA
IoTool Platform	https://demo.iotool.io/docs/

IoChat Platform	https://demo.iochat.io/docs/
Dutch Intake Wizard	https://portals.rrdweb.nl/pharaon/wizard/webpages-openapi.yaml
PACO - Web application	https://dev.senseeact.com/servlets/paco/senseeact/swagger-ui/index.html
RRD eHealth Platform	https://www.senseeact.com/docs/api/
Amicare	https://amicare-swagger-bucket2.s3.eu-central-1.amazonaws.com/index.html
uGrid	https://api-pharaon.miwenergia.com/swagger/index.html
SmartHabits Platform	https://test.smarthabits.com.hr/api-gateway/swagger-ui/
Discovery	https://backend-pharaon-dev.ascora.eu/swagger/index.html
AdSysCo RegiCare	https://regicare.demo.adsysco.nl/docs/index.html
Resource Broker	https://dev.senseeact.com/pharaon-resourcebroker/api-docs/
Process Engine API	https://process-engine-api.pharaon.icelab.cloud/api/

As mentioned in the previous version of the deliverable, NA does not necessarily mean that those technologies are not exposing swagger endpoint, but this can also mean that they are not serving it on publicly exposed URLs. Also note that such endpoints are mostly used in the integration sandbox, which is often refreshed and updated.

2.5 GraphQL as orchestration layer for downstream APIs

The work presented in the previous version of the deliverable, D4.5, included detailed analysis and usage of GraphQL as an orchestration layer for downstream API. As presented in D4.5, GraphQL is *an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data,* that was developed in Facebook before becoming publicly available in 2015. At the end of 2018, it was moved to the newly established GraphQL Foundation (https://graphql.org/) and the specification is hosted on GitHub (https://github.com/graphql/graphql-spec). The GraphQL was important because it can be used for "backend for frontend (BFF)" modules since it can perform functions on backend API and act as a flexible API interface (façade) for UI applications (frontends). The following figure (Figure 2.4 from D4.5) showed how the Pharaon API façade is used to orchestrate across multiple fine-grained sets of (Pharaon) API calls.

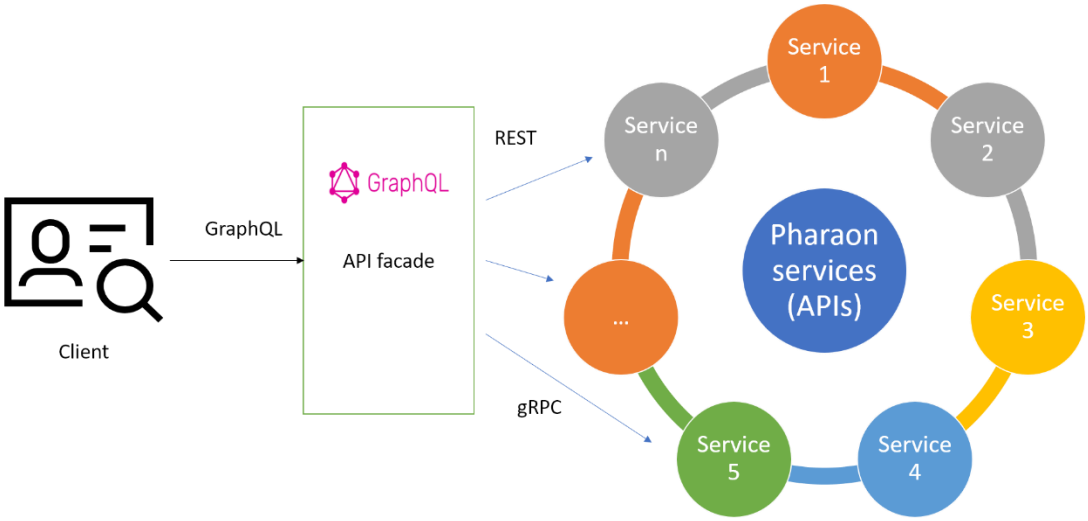

Figure 2.4 GraphQL Pharaon API façade (from D4.5)

The previous version of the deliverable, also presented in details main advantages of using GraphQL, such as faster delivery, human readable queries, API discoverability, use of any programming language, fetching in one call and just the needed information, and similar.

In addition, the D4.5 also presented main GraphQL tools and services, including the ones stated at https://graphql.org/code/

Furthermore, D4.5 presented short comparison of REST and GraphQL-based APIs and summarized the steps of API conversion from REST API to GraphQL API.

3 Container orchestration tools

Container orchestration tools are used to automate the deployment, management, scaling, and networking of containers[6]. They can be used in any environment with containers and are a particularly important input for T4.4 Pharaon in-a-box (documented in deliverables D4.10, D4.11 and D4.12 (to be submitted in June 2024)). Container orchestration tools help deploying the same Pharaon technology, tool or platform in different environments without needing to redesign it, while at the same time they make it easier to orchestrate services. Containers are ideal technology deployment unit that provide self-contained execution environment. More details are provided in the mentioned deliverables, latest D4.11 and D4.12, related to task T4.4.

Similarly to the previous chapter, considering that work regarding container orchestration tools was important in the early stages of the Pharaon development process, there have been only minor updates in this latest deliverable (in comparison to descriptions provided in D4.5). For completeness reasons and to provide a reference location for all T4.2 activities in a single final document, the following subsections in this chapter summarize the work done in the previous versions of this deliverable.

3.1 Containers as a service (CaaS)

The deliverable D4.5 describes Container as a Service (CaaS) as somewhere in between Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), and as a form of container-based virtualization in which container engines, orchestration and underlying computing resources are delivered to users as a service from a Cloud provider. Among cloud services, the CaaS model is considered a kind of subset of IaaS. The deliverable D4.5 highlighted several advantages of using CaaS including:

- Portability
- Scalability
- Efficiency
- Increased security
- Speed

In addition, flexibility, simplified maintenance, unified management, scalability, deployment and reduced cost are among other mentioned benefits of using CaaS.

The diagram presented on Figure 3.1, from D4.5 showed how CaaS works. Pods are a set of one or more containers deployed under a single host, while all containers deployed within the same pod will share storage and network resources with each other.

[6] https://www.redhat.com/en/topics/containers/what-is-container-orchestration

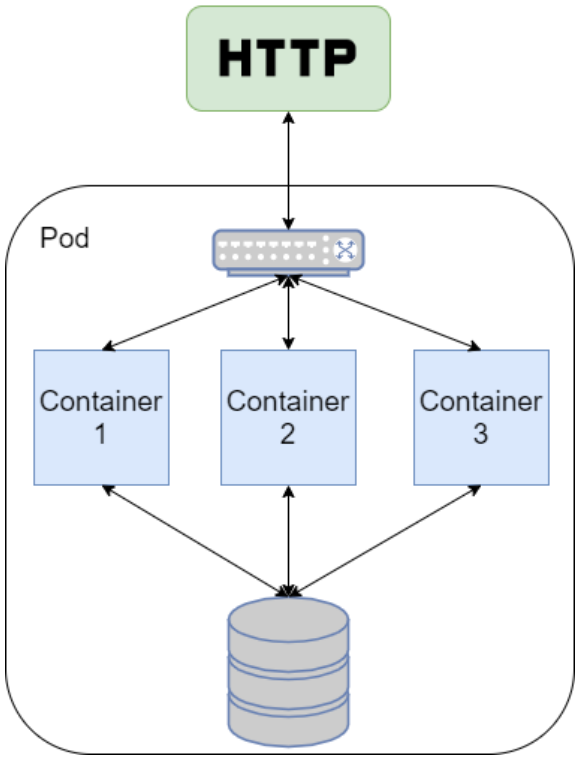

Figure 3.1: Example of CaaS distribution with several containers sharing a single storage system (from D4.5)

More details about CaaS are provided in the previous version of this deliverable, D4.5.

3.2 Kubernetes distributions analysis

The Kubernetes distribution analysis was provided in D4.5. The Kubernetes is an open-source platform for managing workloads and services which makes it easy to automate, configure and deploy different software and applications. Kubernetes, sometimes called K8, provides us with a container-centric management environment with which to orchestrate, compute, network and storage infrastructure for workloads.

The description of the Kubernetes architecture is also provided in D4.5, as follows:

“The Kubernetes architecture is shown in the following diagram. A Kubernetes cluster is made of several worker machines (either physical machines, virtual machines, or a combination), each with a specific role. The **master** machines manage and orchestrate the containers in the worker machines, also known as nodes. The **worker** machines include the **pods** themselves, which are the basic unit of deployment in this cluster; along with the **kubelet**, which is an agent to handle communication between the node it's running in and the master and the **kube-proxy**, which maintains the network rules and handles the transmission of packets between pods, the host and outside world, represented as the cloud. The **container runtime** oversees managing container images and running containers in the node. Additional enhancing pieces of software, or **add-ons**, can be included if needed for a specific purpose (e.g. for a user interface):”

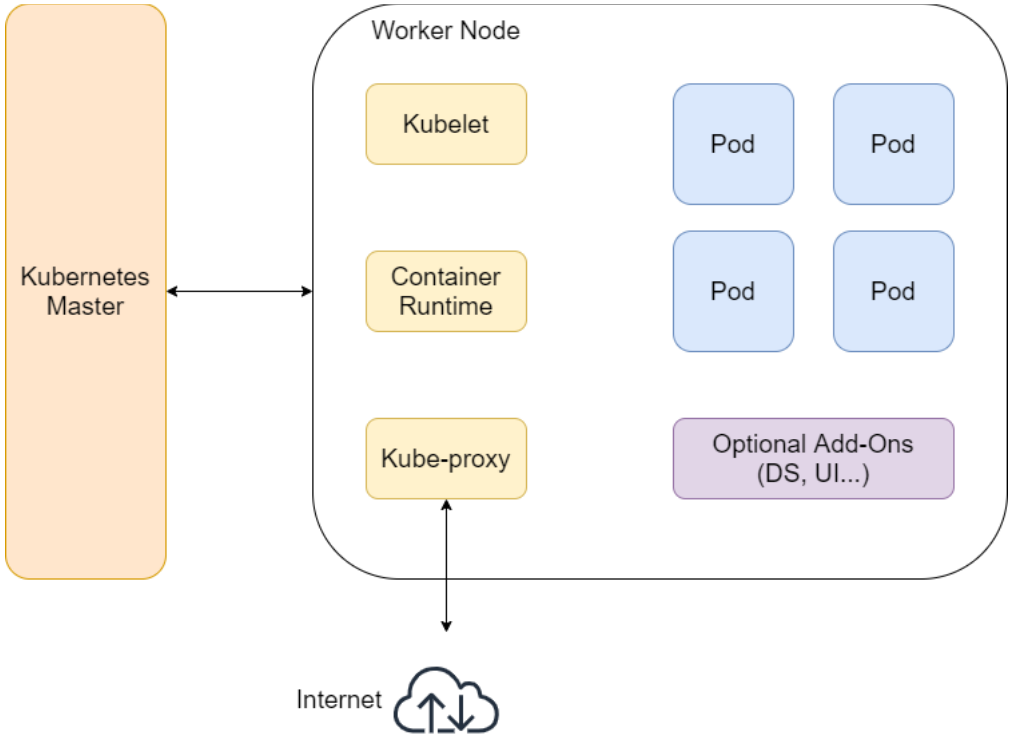

Figure 3.2: Kubernetes Architecture (from D4.5)

Kubernetes has been used widely in the Pharaon project especially for the deployment and managing of technology used in the pilots. As described in D4.5, Kubernetes is used in the deployment and managing of technology used within Andalusian pilot. Also, several other technology providers use Kubernetes in development process and in updated production environments, such as test clusters based on Kubernetes for IoTool and IoChat platforms. More details are provided in D4.5.

Figure 3.3: K8s Cluster Dashboard from Rancher, providing real-time details of a cluster, used within the Andalusian pilot (from D4.5)

4 Monitoring Orchestration Tools

In this deliverable a new monitoring tool is described that was used in the Italian and Slovenian pilots to address additional requirements from the pilot partners. These requirements were related to the monitoring and logging end user engagement and interaction with the UI dashboard (in this case the Carer Dashboard from Ascora). Due to the architecture of the Carer Dashboard and additional requirements, the ELK stack presented in previous deliverable was not able to provide all required insights. The ELK logging was mainly done and used in the backend, but the user interactions needed another solution.

Matomo is an open-source web analytics platform that was selected due to its comprehensive range of tools for tracking and analysing website and application traffic. The decision to use Matomo was driven by its extensive features and the benefits it offers in terms of data privacy, flexibility, and cost-effectiveness. Matomo allows for real-time data tracking, enabling us to monitor visitors' actions, page views, downloads, and events as they happen. One of the most significant advantages of using Matomo is its emphasis on privacy and data ownership. Unlike many other analytics platforms, Matomo ensures that we retain complete control over our data, which is vital for maintaining the security and privacy of sensitive information. This aspect is particularly important for compliance with data protection regulations like GDPR. Matomo's open-source nature would allow for extensive customization to fit the specific needs and integration requirements. It provides access to a robust set of APIs, which facilitates extending functionality and integrating with other platforms and tools.

In summary, Matomo is a powerful and versatile web analytics platform that provides detailed insights into our digital presence while prioritizing data privacy and ownership. Its extensive features and benefits align well with our project's requirements, making it an ideal choice for our web analytics needs.

To integrate Matomo in our Carer Dashboards for the Italian and Slovenian pilots, Ascora set up an own server for Matomo. Matomo is running as a Docker container using Docker-Compose and it is using the SSL certificates provided by the CertBot of the ELK Stack. The Matomo dashboard is currently tracking data for the different websites, each corresponding to a different pilot partner.

The main key metrics that can be tracked are as follows:

- Visits Over Time - Provides the number of visits over a specific period.
- Visits Overview - Displays comprehensive statistics, including:
    - Total visits and unique visitors.
    - Average time spent per visit.
    - Total actions performed.
    - Total page views.
- Page URL Statistics - Offers insights into specific page URLs, including average time spent, exit rates, and more.
- Table of Most Interesting Views - Presents a table with the action name of each page view category, along with click events.
- Graphs
    - Visits per Visit Duration - Illustrates the distribution of time users spend on visits.

- Real-Time Visits - Displays real-time visits with user profile information and total usage time

The Figure 4.1 shows the dashboard overview of Matomo (for administrators) where all websites are listed.

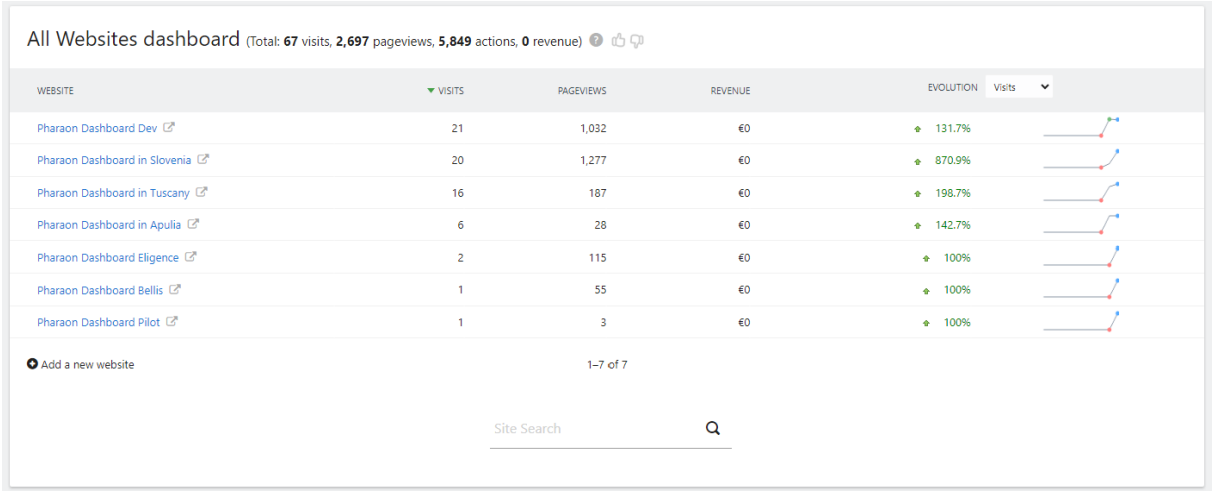

Figure 4.1 Matomo dashboard overview for administrators

Additionally, there is a more detailed dashboard for each pilot, as presented in example for the Slovenian pilot on Figure 4.2.

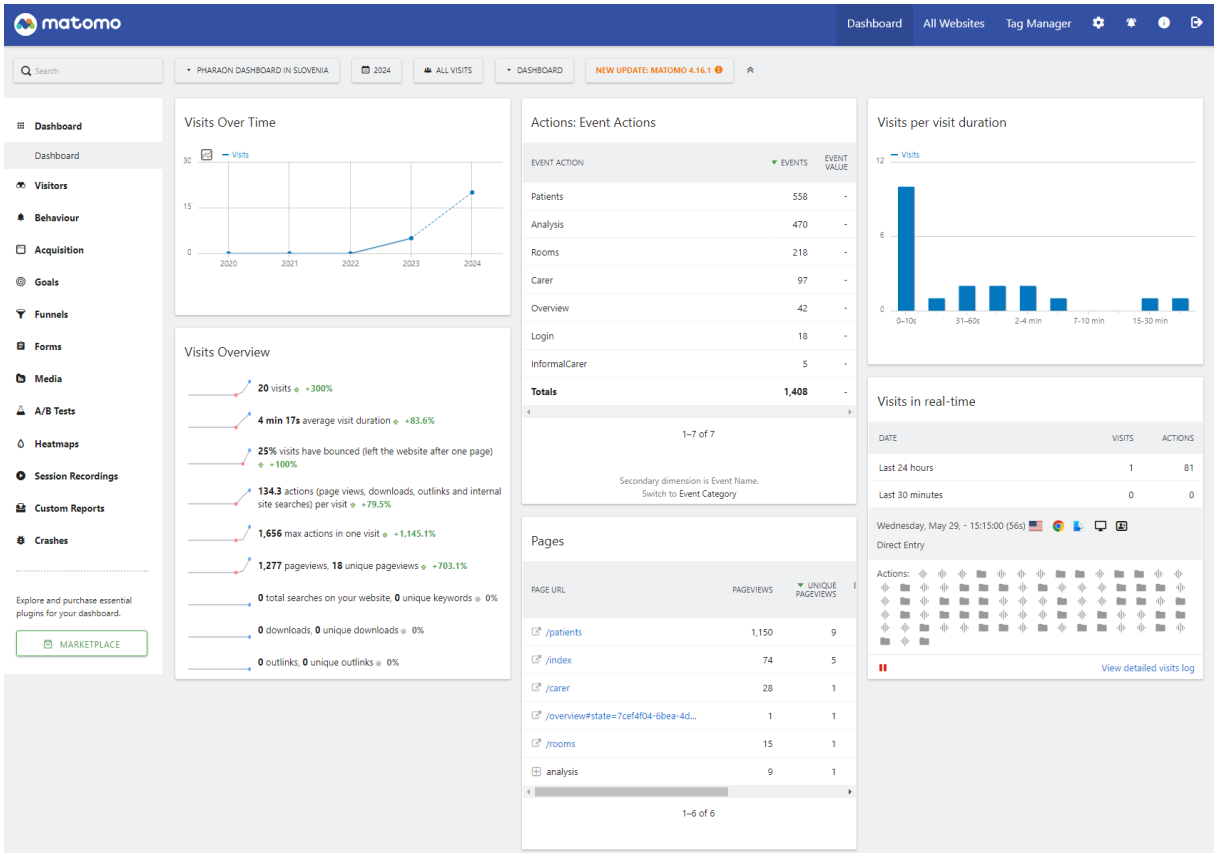

Figure 4.2 Detailed Matomo dashboard used for the Slovenian pilot

All in all, this tool provided many detailed insights in the user behaviour for every single dashboard.

For completeness reasons, this deliverable also summarizes the work on monitoring orchestration tools done in the previous deliverable, D4.5. In previous work, several monitoring approaches for Pharaon software and hardware environments were analysed and the first approach resulted in a dedicated lightweight monitoring solution, developed in node.js and deployed as docker containers (graphical user interface of this solution presented with Figure 4.3).

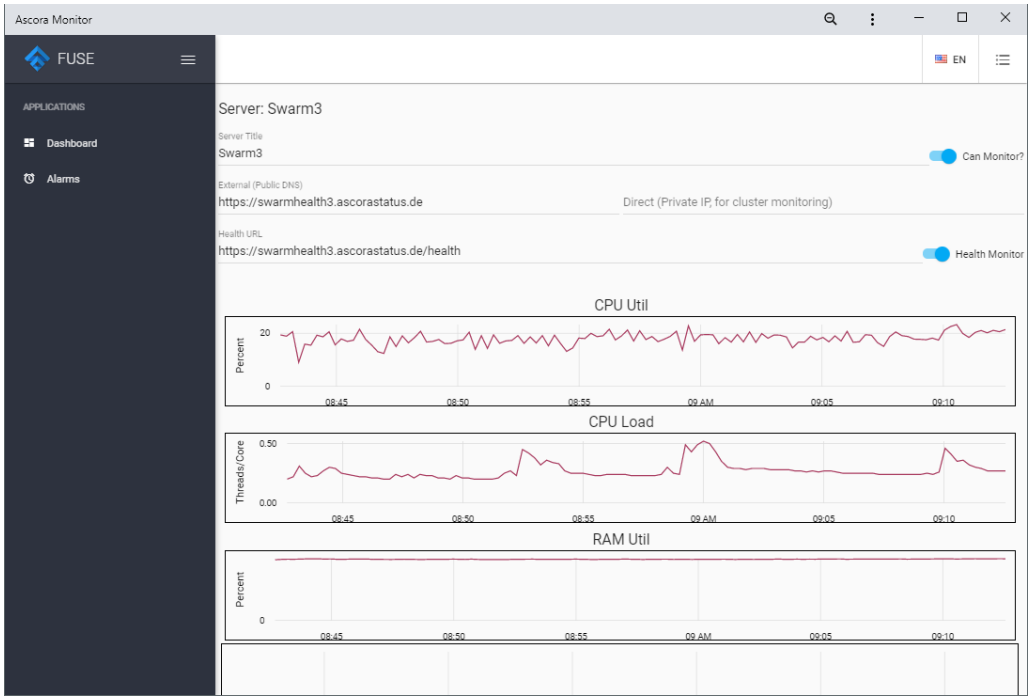

Figure 4.3: Lightweight Monitoring details (from D4.5)

This initial approach and monitoring stack evolved resulting with Prometheus and Grafana to monitor KPIs of distributed systems. This solution has been put in place since February 2021 and since then has been successfully monitoring the KPIs (see Figure 4.4).

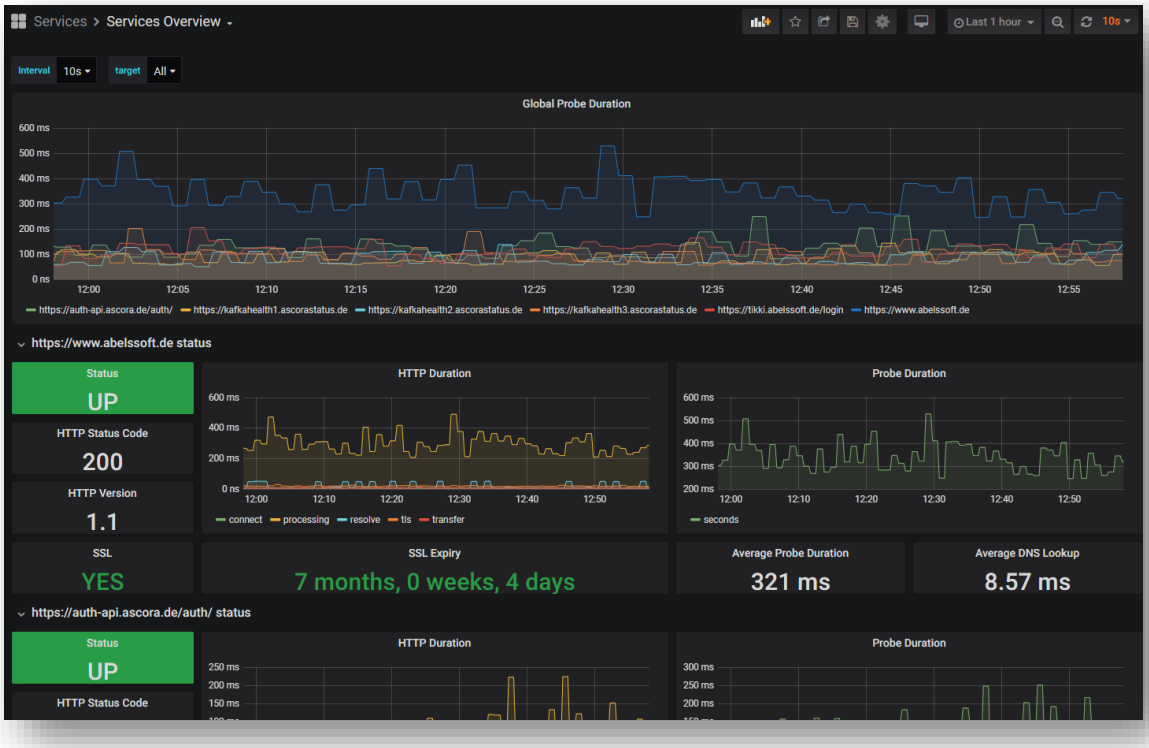

Figure 4.4: Grafana Visualisation during the evaluation (from D4.5)

In addition, the deliverable D4.5 described the decision-making process and the integration of the ELK stack for logging and this tool was mainly used by developers for the purpose of debugging. ELK is a composition of **E**lasticSearch, **L**ogstash and **K**ibana, where each component fulfils its role to finally have a comfortable log management system.

- **ElasticSearch** is a distributed search engine that stores documents as JSON files. It is based on Apache Lucene and accessible via a RESTful interface.
- **Logstash** is an open-source tool, that collects, parses and stores logfiles.
- **Kibana** is a web interface, developed to search and view logs comfortably, that has been indexed by Logstash.

The composition of these tools (ELK) provided the Pharaon with a good option to run a holistic log management tool for its components. The screenshot presented on Figure 4.5 shows the setup (the company's internal loggings are redacted).

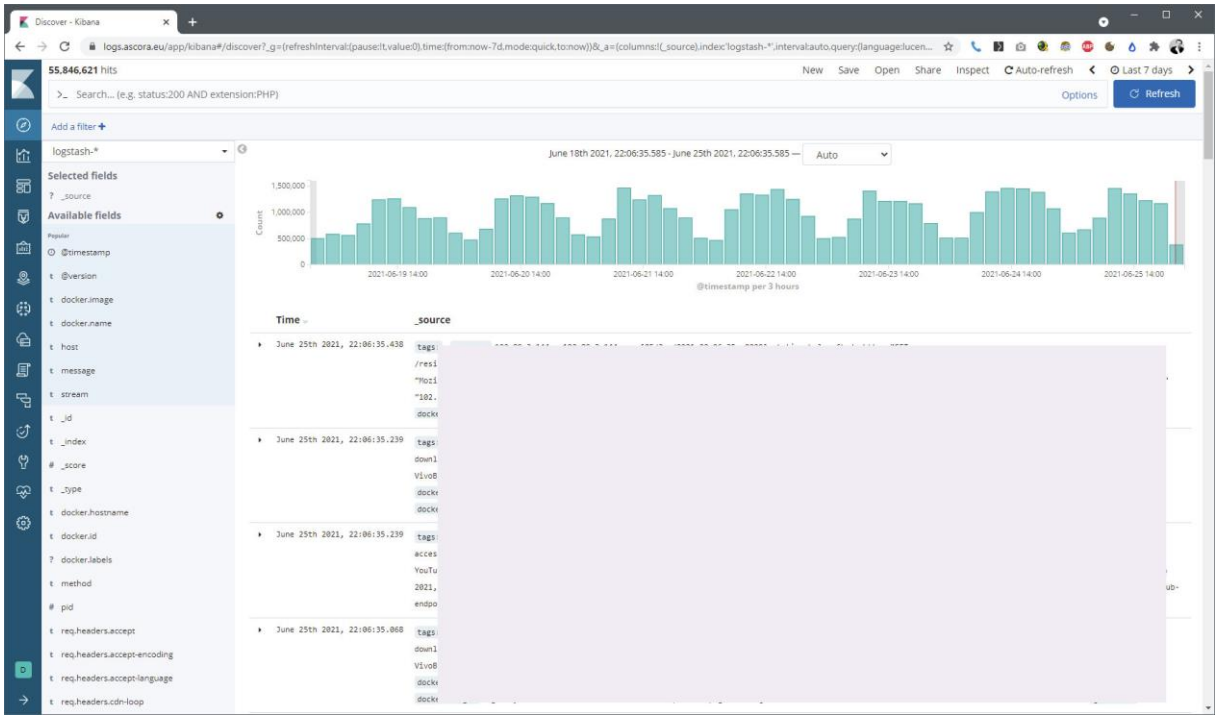

Figure 4.5: ELK Stack Setup (from D4.5)

5 Registries and service orchestration in Pharaon

5.1 Service Registry in Pharaon

The previous version of the deliverable report, D4.5, described some specifics and initial service registry considerations for Pharaon. The conceptual illustration of the service registry in Pharaon was provided through Figure 5.1, available below. The document also summarized common service registration types and properties, provided detailed Client-side and Server-side service discovery comparison and presented several tools comparison.

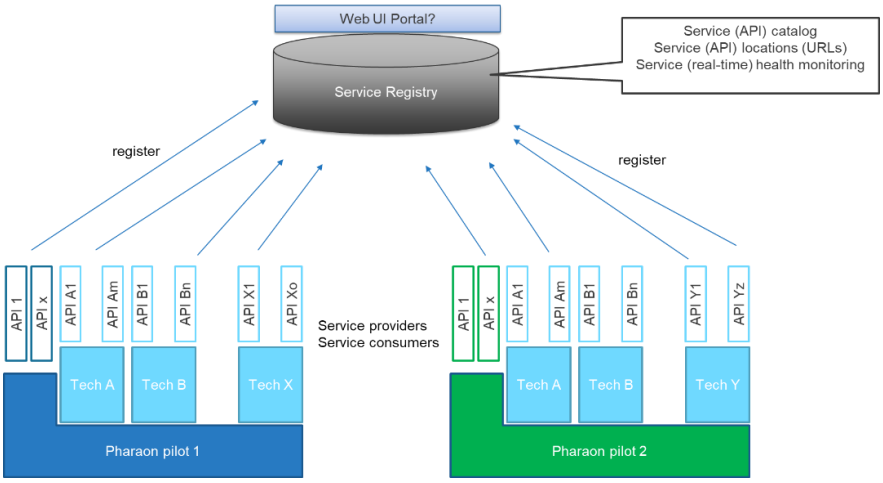

Figure 5.1 Service registration in Pharaon (from D4.5)

As mentioned in D4.5, there was a conceptual design for a registry that would hold the services. The Service Registry is that solution. It is included in the Orchestration Tools and is the user interface that allows users to input services into the API Repository. Users can add services to the Service Registry in two ways.

Services that utilise an OpenAPI endpoint can simply add a service by linking the OpenAPI endpoint to the Service Registry. They can optionally add a link to any Swagger documentation. Services that have a more complex OpenAPI definitions can instead define the inputs and outputs manually to match their needs. Services that do not use OAuth2 can also be defined this way.

5.2 Container registry in Pharaon

In the previous version of this deliverable, it was described how Docker Container Registry integrated into Gitlab can be used in Pharaon. Every GitLab project can have its own space to store its container (including Docker) images.

In addition to Gitlab Container Registry, within the work of WP5 on the setup of CI/CD infrastructure, Harbor, an open-source registry designed to securely manage container images, was deployed and configured (available at https://harbor.res.eng.it). More information about usage of Harbor within the work on Pharaon CI/CD can be found in D5.2.

Here are some key points about Harbor:

- Securing Artifacts
- Vulnerability Scanning
- Content Signing
- Multi-Tenancy
- API and Web UI
- Replication
- Identity Integration
- Compliance
- Performance

5.3 Service orchestration in Pharaon

As detailed in D4.9, the Service Orchestration Tools are a modified Camunda engine. Camunda is an open source BPMN engine that is detailed in D4.7 and D4.8.

The Service Orchestration is broken into two parts, Design Time and Run Time. Together, these parts allow a domain expert user to visually describe a business process, connecting elements together using BPMN notation and services.

The Design Time part of the Service Orchestration Tools allow the creation of a business process from scratch using BPMN notation. The BPMN engine outputs an XML file that is utilised by Camunda in order to run the processes. This file is manipulated by a series of APIs that are hosted on the API Registry.

The API Repository can also be utilised by other partners and technologies; the Service Orchestration Tools interact with the APIs that are hosted on it. The APIs on there can also be utilised within the Design Time tools and can be manipulated within the business logic.

The Run Time part of the Service Orchestration Tools is where a user can interact with the published processes. The Camunda engine executes the processes that have been designed. Processes that require human input can be requested (User Tasks) with the Task Manager whilst the API Gateway handles API requests.

The D4.9 goes into more depth on the particularities of the Service Orchestration Tools.

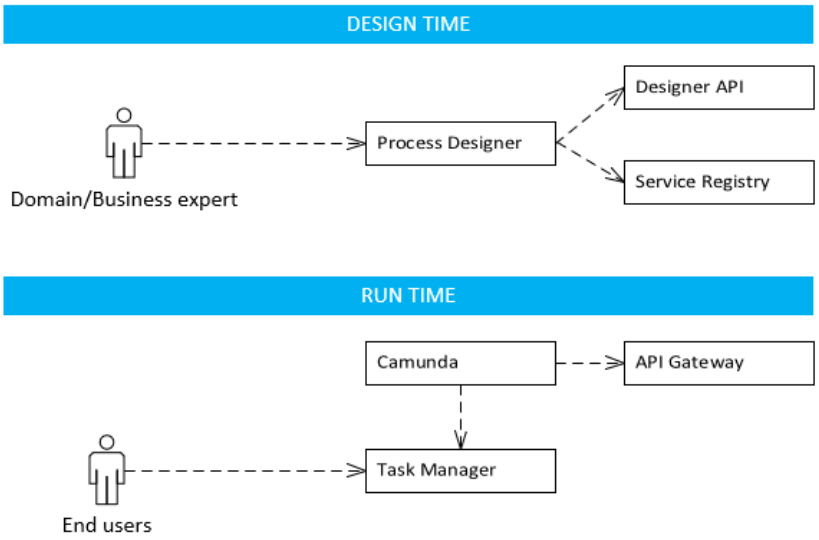

Figure 7.1 Service Orchestration overall architecture (from D4.9)

6 Data transfer orchestration tools

The term data orchestration refers to the process of coordinating and processing data across multiple systems and services. Data orchestration tools are designed to automate and streamline the process of collecting, managing, and integrating data from multiple sources. These tools allow data sharing to be seamless and effective across multiple systems and services, in real time or near-real time.

Key functions of data orchestration tools are shown in Figure 6.1 below and include data integration, data transformation, data movement, workflow automation, error handling, and monitoring and logging.

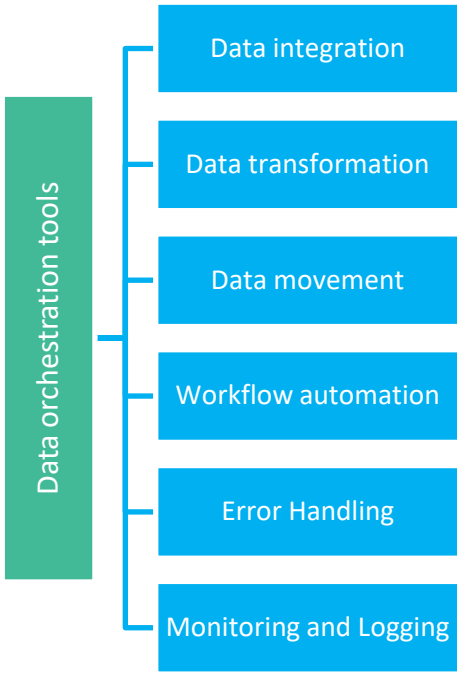

Figure 6.1 Key functions of data orchestration tools

6.1 Overview of open-source data orchestration tools

Some popular open-source data orchestration tools include:

- **Apache Airflow**: One of the most popular open-source platforms designed to simplify the process of orchestrating complex data workflows and it is used for orchestrating distributed applications. It is described as an open-source workflow/pipeline management platform, for "developing, scheduling, and monitoring batch-oriented workflows"[7]. The tool "enables organizations to streamline their data engineering processes, ensuring reliability, scalability, and maintainability in managing diverse data pipelines across disparate systems and environments"[8]. It is written in Python and workflows are created via Python scripts. Its extensible framework enables building workflows connecting with virtually any technology,

[7] https://airflow.apache.org/docs/apache-airflow/stable/index.html

[8] https://www.datamation.com/applications/apache-airflow-review/

while a web interface helps manage the state of workflows. To manage workflow orchestration, Airflow uses directed acyclic graphs (DAGs), a mechanism that allows for the clear, concise expression of dependencies and relationships between different tasks. DAGs enable flexibility and extensibility, extending a rich set of features that enable users to create dynamic and scalable workflows tailored to their specific requirements. Tasks and dependencies are defined in Python and then Airflow manages the scheduling and execution.

- **Dagster**: An orchestrator that is designed for developing and maintaining data assets, such as tables, data sets, machine learning models, and reports. It is used to declare functions that run and the data assets that those functions produce or update. Dagster helps run functions at the right time and keep assets up to date. [9] Dagster is designed to be used at every stage of the data development lifecycle, including local development, unit tests, integration tests, staging environments, and production.
- **Prefect**: A modern workflow orchestration tool that helps automate data workflows with a focus on user friendly interfaces.
- **Argo**: An open-source project that provides a suite of tools for running and managing Kubernetes-native workflows, pipelines, and applications. It is particularly suited for orchestrating complex container native tasks within the Kubernetes environment. The key components include Argo Workflows, Argo CD, Argo Events, Argo Rollouts.
- **Luigi**: Developed by Spotify, Luigi is a Python module that helps build complex pipelines of batch jobs.
- **Kestra**: An open-source orchestration and scheduling platform designed for complex data flows. It provides a robust environment for defining, executing, and monitoring workflows, integrating with a wide range of data sources and processing systems.

[9] https://docs.dagster.io/getting-started/quickstart

6.2 International Data Spaces (IDS)

In addition to the necessary software and technological infrastructure for data exchange, to preserve data value and trust in various ecosystems, a dependable approach must be established. Data spaces provide this technical framework which enables trusted data sharing between multiple organizations and systems, thus generating a value-creating ecosystem.

6.2.1 About IDS

International Data Spaces (IDS) can be considered as a distributed network of data endpoints (instances of the International Data Spaces connector) that enable the secure exchange of data and guarantees data sovereignty and sovereign environment for data sharing in various industries and sectors. It is governed under the umbrella of The International Data Spaces Association (IDSA), a non-profit organization, that aims at promoting the IDS-RAM (Reference Architecture Model) in order to establish an international standard, with more than 164 members from over 31 countries[10].

The four main goals of IDSA are to:

1. build trust,
2. guarantee security and data sovereignty,
3. enable data ecosystems, and
4. enable standardized interoperability.

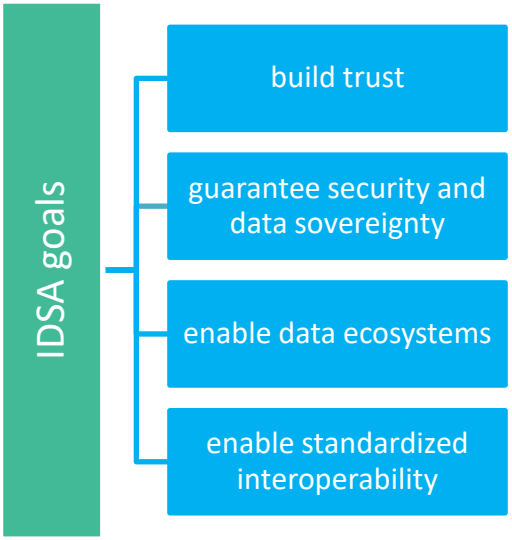

Figure 6.2 Main IDSA goals

The IDS standard enables trustworthy data exchange among certified data providers and recipients, based on mutually agreed rules. The foundation of data spaces is interoperability, policy, trust, and identity. A data space can be defined as a virtual area that offers safe and reliable data sharing procedures together with a defined framework for data exchange based on shared protocols and formats. Data owners maintain control over their data and can decide who can use it and under what circumstances. This is known as data sovereignty. Gaia-X defines dataspace as “A federated, open

[10] Data from https://internationaldataspaces.org/ on May 28th, 2024.

infrastructure for sovereign data sharing based on common policies, rules, and standards"[11]. Thus, it can be considered as standardized, secure digital infrastructure that allows different parties to share trusted data and provide data-based services.

Main components of IDS Data Space are illustrated on Figure 6.3. Data transfer occurring in a data space is done in an environment with multiple data providers (and data owners) on one side, and multiple data consumers (and data users) on the other side. The **data provider** is a system that transfers the owner's data to the data space via the IDS Connector. It allows others to use the data while retaining control over who, how, when, why and at what price. On the other side, the **data consumer** is a system that processes data on behalf of the data user. The data is offered by data providers per their usage policies and with confidence in the data's quality and reliability. The key component of data space is **IDS Connector**. The IDS Connectors are basically gateways for data and services and act as a trusted environment for apps and software. Through IDS Connectors participants may easily track the provenance of their data, enforce usage regulations, and attach policies to their data within a data space. The **Broker** offers details about the content, structural quality, and other characteristics of the data sources. The clearing and settlement service for all financial transactions and data interchange inside the IDS is called **Clearing House**. Standardized descriptions for data based on acknowledged best practices are provided by **Vocabulary**. Participants in the IDS have their identity information created, managed, validated, and maintained by the **Identity Provider**. Applications from the App Store can be installed in IDS Connectors to carry out operations like data analytics, aggregation, and transformation. These applications are launched from the **App Store** within the IDS Connector's secure environment, and they carry out operations like data analysis, aggregations, and transactions.

[11] https://gaia-x-hub.de/wp-content/uploads/2022/10/White_Paper_Definition_Dataspace_EN.pdf

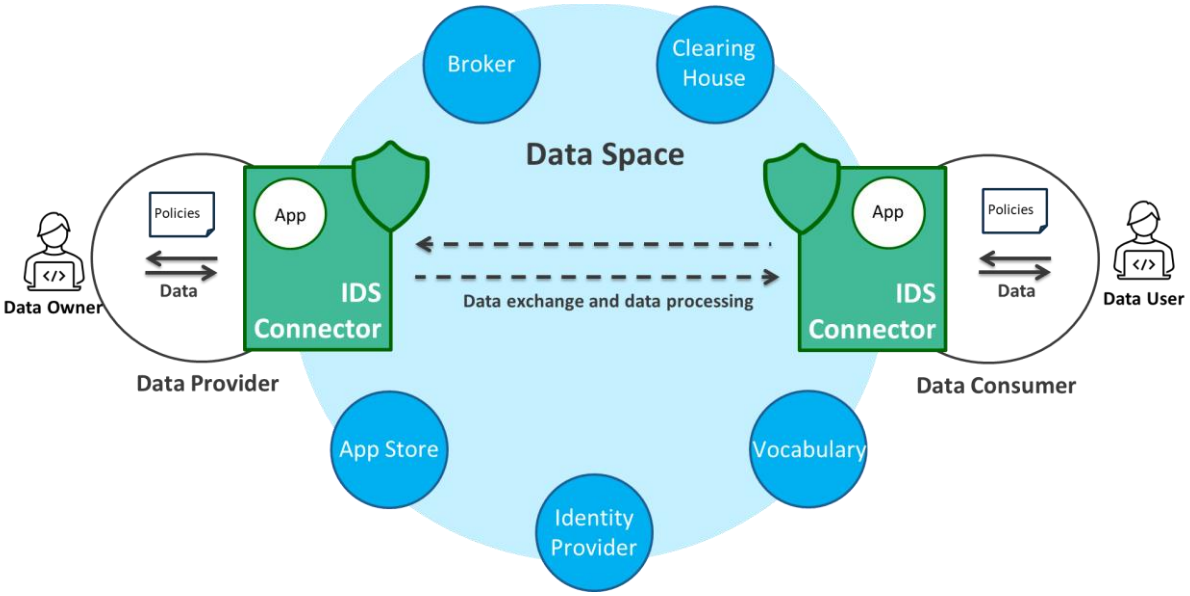

Figure 6.3 Data space components[12]

IDS Connector is the crucial component of the data spaces. It acts as a data transfer gateway responsible for connecting various participants in the data space, ensure trust, connect to identity management servers and provide identity and policy management, ensures data usage control, and guarantee interoperability.

To facilitate interoperable data sharing between entities governed by usage control and based on Web technologies, a Dataspace Protocol as a set of specifications was designed. These specifications define the schemas and protocols required for entities to publish data, negotiate usage agreements, and access data as part of a federation of technical systems termed a dataspace

6.2.2 Architectural considerations of using IDS in Pharaon

The IDS Reference Architecture Model[13] positions itself as an architecture that links different cloud platforms through policies and mechanisms for secure data exchange and trusted data sharing through the principle of data sovereignty. One of the key strengths of IDS RAM is that it proposes architecture for secure data exchange and trusted data sharing and contributes to the design of enterprise architectures in commercial and industrial digitization scenarios. Over the IDS Connector, different data clouds, individual enterprise clouds, on-premises applications and individual, connected devices can be connected to the International Data Space ecosystem. This IDS Reference Architecture Model (IDS-RAM) is described using multiple layers, such as business, functional, process, information and system.

The IDS-RAM defined several strategic requirements. The Pharaon rationale for each is further elaborated in Table 2.

[12] Based on image from: https://internationaldataspaces.org/why/data-spaces/

[13] https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-ram-4

Table 2 IDS strategic requirements and Pharaon rationale

IDS Strategic requirement	IDS rationale[14]	Pharaon rationale
Trust	"Trust is the basis of the International Data Spaces. Each participant is evaluated and certified before being granted access to the trusted business ecosystem."	Since Pharaon is largely being realized in the health (wellbeing) domain it deals with sensitive personal information which makes the trust of critical importance in the end user (primarily older people)- service provider relationship
Security and data sovereignty	"All components of the International Data Spaces rely on state-of-the-art security measures. Apart from architectural specifications, security is mainly ensured by the evaluation and certification of each technical component used in the International Data Spaces. In line with the central aspect of ensuring data sovereignty, a data owner in the International Data Spaces attaches usage restriction information to their data before it is transferred to a data consumer. To use the data, the data consumer must fully accept the data owner's usage policy."	Security and data sovereignty in the Pharaon project and health domain are essential to ensure safeguarding of sensitive end user information, maintaining privacy, and ensuring compliance with national and EU regulations.
Ecosystem of data	"The architecture of the International Data Spaces does not require central data storage capabilities. Instead, it pursues the idea of decentralization of data storage, which means that data physically remains with the respective data owner until it is transferred to a trusted party. This approach requires a comprehensive description of each data source and the value and usability of data for other companies, combined with the ability to integrate domain-specific data vocabularies. In addition, Metadata Brokers in the ecosystem provide services for real-time data search."	The ecosystem of data in Pharaon encompasses the collection, sharing, analysis, and responsible utilization of diverse health-related information to drive Pharaon needs and enhance end-user care.
Standardized interoperability	"The International Data Spaces Connector, being a central component of the architecture, is implemented in different variants and can be acquired from different vendors. Nevertheless, each Connector is able to communicate with any other Connector (or other technical component) in the ecosystem of the International Data Space."	Pharaon as Innovation Action focuses on the integration of existing technological solutions and in this process the establishment of consistent data formats and communication protocols, enabling seamless exchange of information between different Pharaon systems and stakeholders to enhance end user care and data utilization if of pivotal importance.

[14] https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-ram-4/introduction/1_1_goals_of_the_international_data_spaces

Value adding apps	"The International Data Spaces allows to inject apps into the IDS Connectors in order to provide services on top of data exchange processes. This includes services for data processing, data format alignment, and data exchange protocols, for example. Furthermore, data analytics services can be provided by remote execution of algorithms."	Pharaon integrates many innovative and value adding solutions and applications and also allows the extensions (both in use cases and applications, which is the reason of having 2 open calls)
Data markets	"The International Data Space enables the creation of novel, data-driven services that make use of data apps. It also fosters new business models for these services by providing clearing mechanisms and billing functions, and by creating domain-specific metadata broker solutions and marketplaces. In addition, the International Data Spaces provides templates and other methodological support for participants to use when specifying usage restriction information and requesting legal information."	Pharaon deals with sensitive data where GDPR, Joint Controllership Agreements between pilot partners serve as a legal basis for data exchange and analysis. When legal preconditions are met the data exchange is enabled and performed to serve in the value-added innovation process.

The similarities between IDS and Pharaon are listed in the following table (Table 3).

Table 3 Similarities between IDS and Pharaon

	IDS	Pharaon
Decentralized and multi-cloud federated environment	Yes	Yes
Restricted data usage (each participant decides under what policies their data is shared)	Yes	Yes
Trust is critical	Yes	Yes
Participation based on rules and policies	Yes (usage policy for every data asset)	Yes (usage policy per specific pilot setup)
Interoperability of heterogeneous environments with many different technologies	Yes	Yes
Data is always exchanged 1:1	Yes	Yes
Existing modules can be extended	Yes	Yes
Custom modules can be created	Yes	Yes
Identity Management is obligatory	Yes (IdentityHub)	Yes (Keycloak)
Each participant controls own deployment	Yes	Yes

The latest version of IDS Reference Architecture Model version 4.0 (IDS-RAM 4.0) was released in 2022. As described in D2.2 “Pharaon initial ecosystems architecture”, the IDS RAM is in line with common system architecture models and standards (e.g., ISO 42010, 4+1 view model) and uses a five-layer structure expressing various stakeholder concerns and viewpoints. These layers are:

- The Business Layer: specifies and categorizes the different roles and the main activities and interactions between roles
- The Functional Layer: related to the functional requirements and features
- The Process Layer: specifies the interactions taking place between the different components of and it provides a dynamic view
- The Information Layer: defines a conceptual model which makes use of linked-data principles for describing both the static and the dynamic aspects
- The System Layer: concerned with the decomposition of the logical software components, considering aspects such as integration, configuration, deployment, and extensibility

Additionally, the Reference Architecture Model is composed of three perspectives that are implemented across all five layers: Security, Certification, and Governance. The mapping of Pharaon functional layers to IDS RAM layers is illustrated in the Figure 6.4.

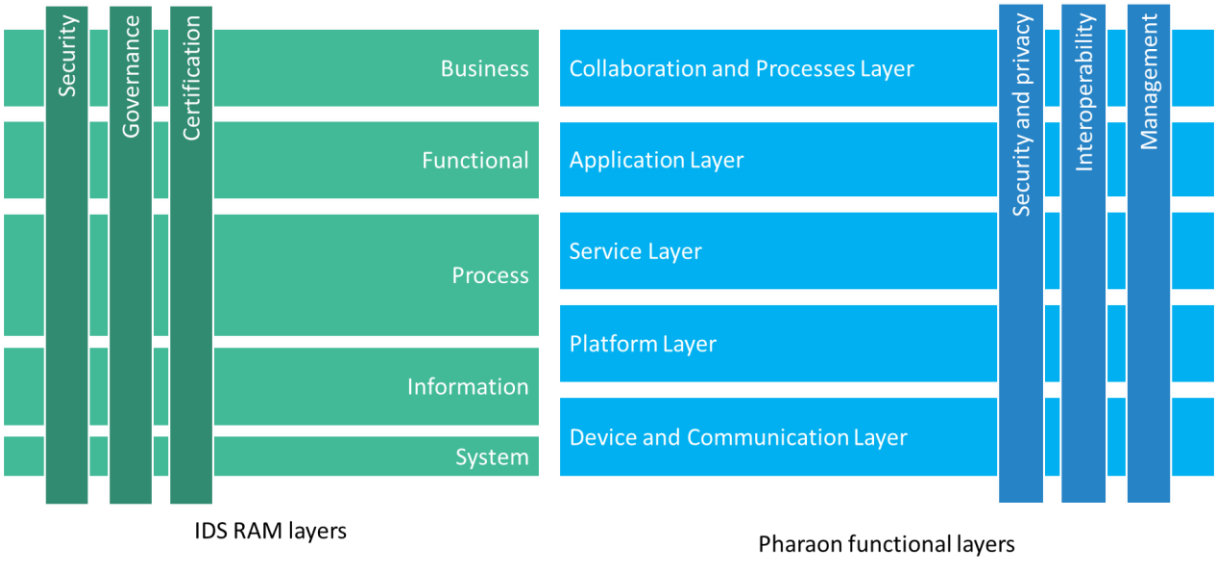

Figure 6.4 IDS RAM, Pharaon RA: mapping of layers

One of the key concepts that defines structure of the data space, as illustrated in Figure 6.3., is the ownership of the data, since each owner is represented through one provider IDS Connector. In Pharaon project there are different ownerships for different types of data. For example, each organization could be the owner of internal system data, while on the pilot level, ownership is usually regulated with joint data controllership agreements, meaning, several organizations share ownership of all data being collected, processed and stored within the pilot. For this reason, using the IDS Connector for transfer of data between platforms and technologies used within certain pilot is not needed, since all pilot partners are the owners of that data. However, in the cases where pilot data should be shared outside of the pilot, using data spaces and IDS connectors is essential for the realization of standard and trusted data sharing using concept of data spaces. Finaly, there is certain

data that is owned by the entire Pharaon consortium, and in that case Pharaon could act as Data Provider in broader (i.e. EU level) data space.

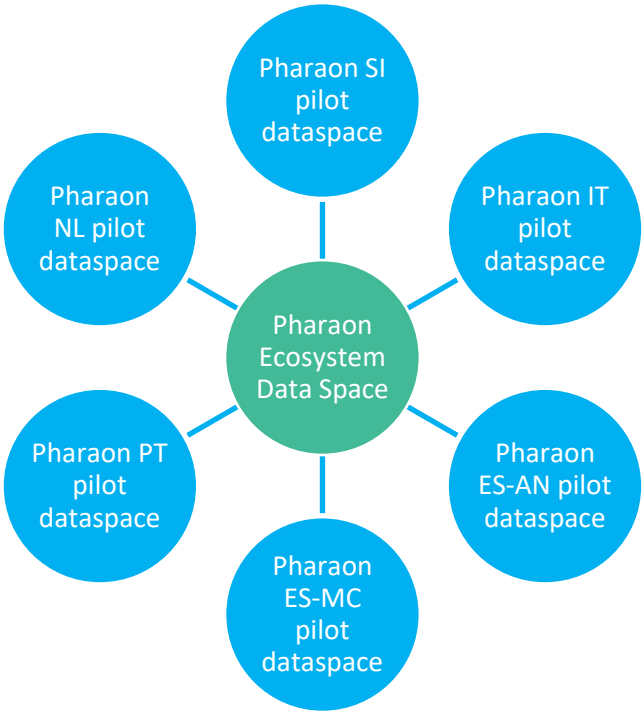

Figure 6.5 Different possibilities for data spaces within Pharaon project

The diagram with basic IDS components in the case of Pharaon acting as the Data Provider in a broader data space is illustrated on the Figure 6.6.

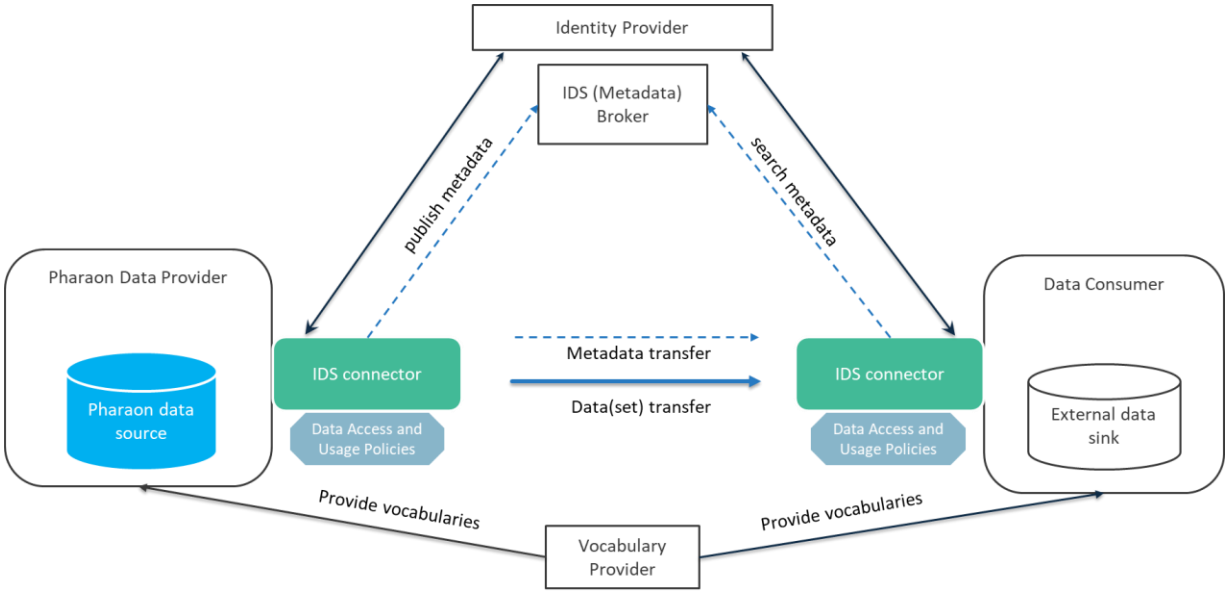

Figure 6.6 Pharaon Data Provider as data space Participant

6.2.3 Sharing data between Pharaon platforms and technologies using open-source IDS components

This section explains how open-source IDS components are used to share data between Pharaon platforms and components. The IDS-RAM specifies architecture that links different platforms for secure and interoperable data exchange. There are numerous open-source implementations of IDS components, namely IDS Connector as central component. The most recent Data Connector Report[15] (May 2024) lists 33 different implementation of IDS Connector (with updates in the last month), out of which 15 are open source, and 2 partially open source. Out of these, the IDS Connector from Eclipse Dataspace Components (EDC) has been used to enrich Pharaon technical ecosystem with interoperable and trustworthy data exchange capabilities (this is why it is considered as data transfer orchestration tool, although it is much more). Eclipse Dataspace Components (EDC)[16] is a comprehensive open-source framework providing a basic set of features (functional and non-functional) that dataspace implementations can re-use and customize. The EDC project is governed by the Eclipse Foundation under Apache 2.0 licensing. It leverages the framework's defined APIs and ensures interoperability by design. It is powered by the specifications of the Gaia-X AISBL Trust Framework and the IDSA Dataspace protocol and consists of a set of components that are mandatory to implement a dataspace:

- Connector (contract negotiation and data sharing)
- Federated Catalog (publish and search)
- Identity Hub
- Registration Service (registration and discovery)
- Data Dashboard (Management UI)

In addition, some open-source EDC extensions have been used, namely sovity Community Edition EDC and Sovity EDC UI[17] to provide enriched capabilities to the solution.

An overall diagram of sharing data between two Pharaon platforms using Eclipse Dataspace Components (EDC) IDS Connector and Sovity EDC extensions is illustrated in Figure 6.7. The SmartHabits Platform (SHP) is used as main platform in the Apulia, Tuscany and Slovenian pilots, while OneSait Platform is used in the Andalusian pilot. Both platforms can act as data provider or data consumer, depending on the use case. In example SHP acts as data provider, source of data, while OneSait Platform as data consumer that processes and visualizes the data on the dashboard. The data on SHP is visualized using Ascora's (ASC') Discovery Dashboard, but using EDC IDS Connector, without any source code modification on SHP or OneSait Platform, the same data could, if such scenario would have been required, also be processed and visualized on OneSait Platform.

[15] https://internationaldataspaces.org/wp-content/uploads/dlm_uploads/IDSA-Data-Connector-Report-No-15-May-2024.pdf

[16] https://projects.eclipse.org/projects/technology.edc

[17] https://github.com/sovity/edc-ui

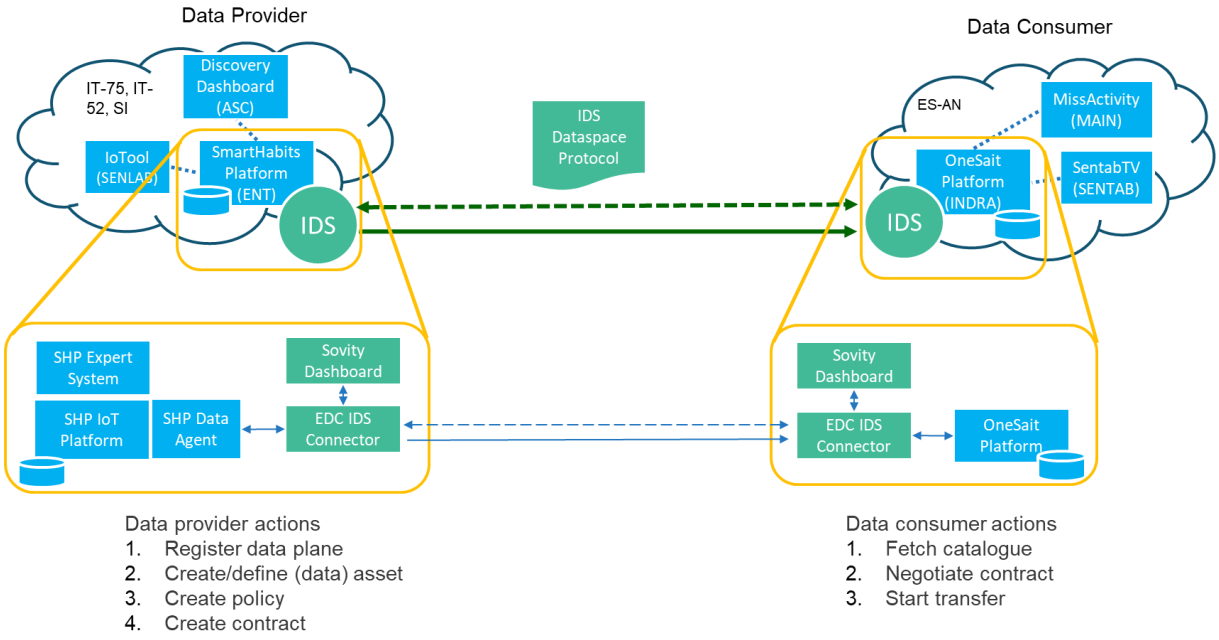

Figure 6.7 Sharing data between SmartHabits and OneSait Platform using Eclipse Dataspace Components (EDC) IDS Connector and Sovity EDC extensions

There are two main phases in presented interoperable and trusted data exchange scenario. At the beginning, as the first step, the data provider needs to register data plane instance of connector (1). Next, in order to offer any data, the provider must define and maintain data assets - an internal list of resources offered (2). To manage the accessibility rules of an asset, it is essential to create a policy (3). To ensure an exchange between provider and consumer, the provider as final step must create a contract offer for the asset, based on which a contract agreement can be negotiated (4). This contract associates policies to a selection of assets to create the contract offers that are put in the catalogue. The second phase are data consumer actions which consist of (1) fetching the catalogue from the provider, which contains all the contract offers available for negotiation, (2) initiate a contract negotiation sequence with the provider and (3) start the transfer. Detailed sequence diagram of Pharaon IDS scenario where SHP is used as Data Provider, and OneSait as Data Consumer is provided in Appendix A.

6.3 Orion Context Broker

6.3.1 About FIWARE

FIWARE[18] is an Open-Source Platform providing a set of components (i.e. services) to make it easier, faster, and cheaper to develop smart solutions. These services are known as *Generic Enablers,* and they are accessible through REST APIs using the NGSI APIs for Context Data Management. Enablers can exchange data with other Enablers using this NGSI specifications protocol, therefore, the developers and users can use them like *building blocks t*o create Smart Applications. The attribute "*Generic*" wants to highlight that these services can be used in multiple domains (i.e. in Smart Cities, Smart Energy, Smart Industry, Smart AgriFood, etc.). Any Enablers are well described in the FIWARE Catalogue[19].

Figure 6.8 depicts the FIWARE Platform architecture which is organized into sections or layers listed below:

- *Context Process / Analyze / Monitor*: it includes all the Enablers to process, analyse, and display data;
- *Core Context Management*: it includes all services to manage the data;
- *Interface to IoT, Robotics, and third-party systems:* it includes all the Enablers that interact with the real world (e.g., sensors, actuators, robots, and others third-party interfaces);
- *Data/API Management, Publication, Monetization*: it includes Enablers to publish and monetize data, and to put in security the REST APIs;
- *Deployment tools*: it makes available the FIWARE Enablers through Docker Images or Helm Chart.

Finally, there are also a lot of channel support (e.g., Webinars, StackOverflow, Gitlab, and FIWARE Community), events, initiatives, and meetings to help users in the developments and to grow the FIWARE ecosystem.

[18] https://www.fiware.org/

[19] https://www.fiware.org/catalogue/

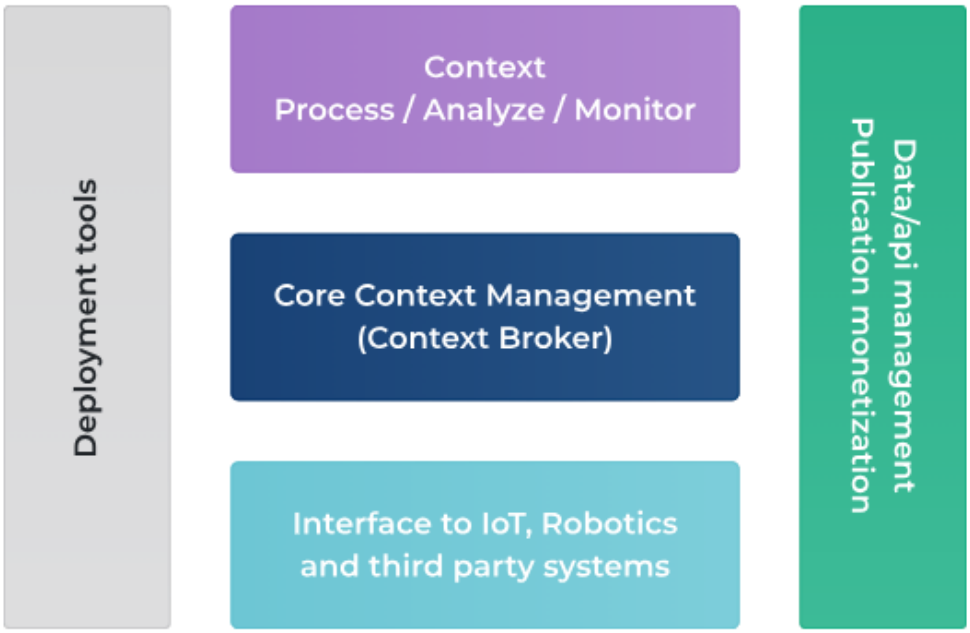

Figure 6.8 FIWARE Platform Architecture

Within the PHArA-ON project, the FIWARE Platform has been used to achieve interoperability aspects. In particular, the Orion Context Broker service enabler (see Section 6.3.2) has been adopted to manage data.

6.3.2 Orion Context Broker in Pharaon

The most important Enablers provided by FIWARE Platform is the Orion Context Broker[20] that is an implementation of NGSI-v2 specifications[21]. The usage of the Orion Context Broker allows to define an application *"Powered by FIWARE"* platform.

The Orion Context Broker provides three important features:

1. Manage of the Context Information
2. Send Notifications
3. Register the Context Provider

The **Manage of Context Information** allows the management of the data (also called Context Information or Data Context Information) through REST APIs.

Figure 6.9 shows how a Client REST interacts with Orion via **request/reply** mechanism using CRUD calls (i.e. **Create**, **Read**, **Update**, and **Delete**). The client sends requests (GET, POST, PUT, DELETE, etc...) with payloads (where it's necessary) and Orion responds with **HTTP status codes** (i.e. 200, 204, 404, etc...).

[20] https://fiware-orion.readthedocs.io/en/master/

[21] https://swagger.lab.fiware.org/?uri=https://raw.githubusercontent.com/Fiware/specifications/master/OpenAPI/ngsiv2/ngsiv2-openapi.json

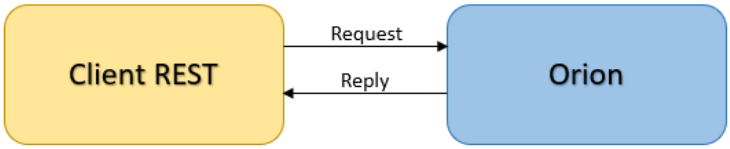

Figure 6.9 Orion request/reply mechanism

In detail, the **Context Information** is composed of a **set of attributes** (identified by a triple: name, value, and type) wrapped within an **entity** (id and type). A typical payload example is shown in Figure 6.10 to describe a temperature sensor context information in a Room (Room1) that monitors temperature and pressure attributes.

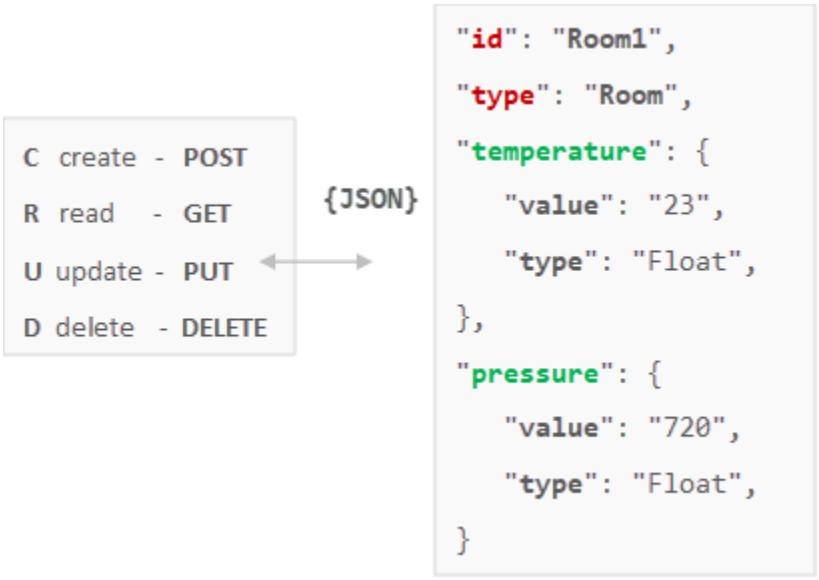

Figure 6.10 Context information example

The **Send notifications** allow users to be notified when someone changes an attribute or more attributes in the Context Information. Before the Orion Context Broker sends notifications, it is required to make a subscription. To create a subscription, it is necessary to set an **attribute/attributes** or **entity/entities** to monitor any changes and the URL endpoint where the notification must be addressed (i.e., a *3rd party service*). Figure 6.11 shows the creation of the subscription process.

Figure 6.11 Subscription Process

After subscription creation, at any changes of attributes (**push**), Orion Context Broker sends (**notify**) the entity payload (including attributes) to the URL endpoint (3rd party service) as shown in Figure 6.12.

Figure 6.12 Push/Notify Process

The **Register the Context Provider** allows you to integrate your server as the provider of data; in this case the Orion Context Broker will assume the role of an NGSI proxy (Figure 6.13).

Figure 6.13 Orion as NGSI proxy

This feature has been added here for completeness of the Orion Context Broker features (more information can be found in the documentation). However, it's necessary to register Orion as a **Context Provider** (through REST APIs **request/reply**) to set the IP address of our own service (*My REST service*). The only required implementation regards the communication between Orion Context Broker and the service. In this case, the *My REST service* must receive Orion's request and then must reply to Orion in **NGSI format**.

In the Pharaon context, we had a lot of data information gathered from sensors, and we wanted to analyse and display them in real-time and aggregate them as a time series. To help the development we adopted the Orion Context Broker in the architectures because it represents the missing service to connect the data provided by sensors and data that will be saved in a database to be displayed in a dashboard. In particular, for the Italian Pilot, we used Orion together with STH-Comet and Cygnus (another two FIWARE Enablers) to save and retrieve data as a time series and for aggregation purposes.

7 Conclusions

This task aimed to describe and provide a comprehensive approach to service orchestration and customization in Pharaon, state of the art technologies and tools that were used for API design, management, and documentation, container and monitoring orchestration tools used in pilots, and other orchestration tools. This final version of the deliverable report focused on describing the usage of standard open-source data transfer orchestration tools which enable trusted data exchange and data-based services among various stakeholders and entities. Two such standard frameworks were analysed, deployed and used: (1) International Data Space (IDS), a virtual space that provide a standardized framework for data exchange, based on common protocols and formats, as well as secure and trusted data sharing mechanisms, and (2) FIWRE, for managing context information, enabling to perform updates and bring access to context in a standardized way. These additional task activities were defined after the second project review within the Interoperability plan in June 2023 with the main goal being to help and provide support to project goals of developing and adapting the enablers for information sharing and development of an extendable, multi-layer, and versatile interoperability environment (defined within WP3's description of work).

Appendix A: Pharaon IDS Example Scenario

Detailed sequence diagram of Pharaon IDS scenario where SHP is used as Data Provider, and OneSait as Data Consumer is provided on Figure 0.1.

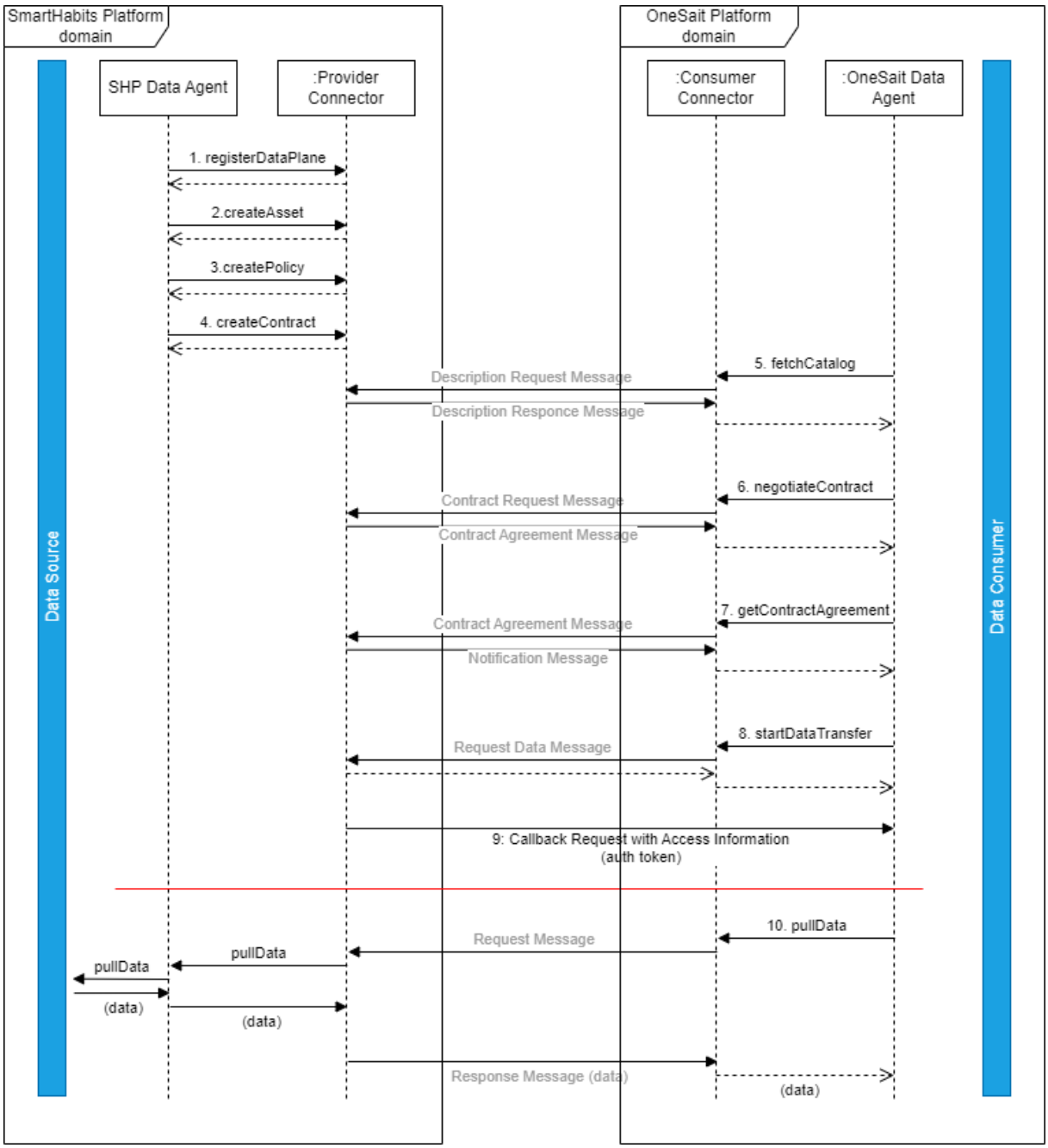

Figure 0.1 Sequence diagram of Pharaon IDS Scenario: SHP as Data Provider, OneSait as Data Consumer

Sample messages – Data provider side

Register data plane instance for provider

```
POST https://<provider_host>:19193/management/v2/dataplanes

{
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/"
  },
  "@id": "http-pull-provider-dataplane",
  "url": "http://localhost:19192/control/transfer",
  "allowedSourceTypes": [
    "HttpData"
  ],
  "allowedDestTypes": [
    "HttpProxy",
    "HttpData"
  ],
  "properties": {
    "https://w3id.org/edc/v0.0.1/ns/publicApiUrl":
"http://localhost:19291/public/"
  }
}
```

Create an Asset on the provider side

```
POST https://<provider_host>:19193/management/v3/assets

{
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/"
  },
  "@id": "assetId",
  "properties": {
    "name": "Data Asset",
    "contenttype": "application/json"
  },
  "dataAddress": {
    "type": "HttpData",
    "name": "Data Asset",
    "baseUrl": "http://<provider_host>:8080/data-agent/v2/api/agent/",
    "proxyPath": "true",
      "proxyQueryParams": "true"

  }
}
```

Create a Policy on the provider

```
POST https://<provider_host>:19193/management/v2/policydefinitions

{
  "@context": {
```

```
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
    "odrl": "http://www.w3.org/ns/odrl/2/"
  },
  "@id": "aPolicy",
  "policy": {
    "@type": "set",
    "odrl:permission": [],
    "odrl:prohibition": [],
    "odrl:obligation": []
  }
}
```

Create a contract definition on Provider

```
https://<provider_host>:19193/management/v2/contractdefinitions

{
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/"
  },
  "@id": "1",
  "accessPolicyId": "aPolicy",
  "contractPolicyId": "aPolicy",
  "assetsSelector": []
}
```

Sample messages – Data consumer side

Fetch catalogue on consumer side

```
POST https://<consumer_host>:29193/management/v2/catalog/request

{
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/"
  },
  "counterPartyAddress": "https://test.smarthabits.com.hr:19194/protocol",
  "protocol": "dataspace-protocol-http"
}
```

Example response:

```
200 Ok

{
  "@id": "94e35771-4473-46c3-8b1b-613cff665c6b",
  "@type": "dcat:Catalog",
  "dcat:dataset": {
    "@id": "assetId",
    "@type": "dcat:Dataset",
    "odrl:hasPolicy": {
      "@id":
"MQ==:YXNzZXRJZA==:NTY1NDVhNDYtZmQ0OC00MzYyLTk2MmEtYjk1ZWM2Mzc2N2Rl",
      "@type": "odrl:Set",
      "odrl:permission": [],
      "odrl:prohibition": [],
      "odrl:obligation": [],
      "odrl:target": {
        "@id": "assetId"
      }
    },
    "dcat:distribution": [
      {
        "@type": "dcat:Distribution",
        "dct:format": {
          "@id": "HttpProxy"
        },
        "dcat:accessService": "640a33e6-af28-4a81-86b5-8e50362c10be"
      },
      {
        "@type": "dcat:Distribution",
        "dct:format": {
          "@id": "HttpData"
        },
        "dcat:accessService": "640a33e6-af28-4a81-86b5-8e50362c10be"
      }
    ],
    "name": "Data Asset",
    "id": "assetId",
    "contenttype": "application/json"
  },
  "dcat:service": {
    "@id": "640a33e6-af28-4a81-86b5-8e50362c10be",
    "@type": "dcat:DataService",
    "dct:terms": "connector",
```

```
    "dct:endpointUrl": "https://<provider_host>:19194/protocol"
  },
  "participantId": "provider",
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
    "edc": "https://w3id.org/edc/v0.0.1/ns/",
    "dcat": "https://www.w3.org/ns/dcat/",
    "dct": "https://purl.org/dc/terms/",
    "odrl": "http://www.w3.org/ns/odrl/2/",
    "dspace": "https://w3id.org/dspace/v0.8/"
  }
}
```

Negotiate a contract (auto accept on provider side)

```
POST https://<consumer_host>:29193/management/v2/contractnegotiations

{
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/"
  },
  "@type": "NegotiationInitiateRequestDto",
  "connectorId": "provider",
  "counterPartyAddress": "https://test.smarthabits.com.hr:19194/protocol",
  "consumerId": "consumer",
  "providerId": "provider",
  "protocol": "dataspace-protocol-http",
  "policy": {
    "@context": "http://www.w3.org/ns/odrl.jsonld",
    "@id":
"MQ==:YXNzZXRJZA==:YTc4OGEwYjMtODRlZi00NWYwLTgwOWQtMGZjZTMwMGM3Y2Ey",
    "@type": "Set",
    "permission": [],
    "prohibition": [],
    "obligation": [],
    "target": "assetId"
  }
}
```

Example response:

```
200 Ok

{
  "@type": "IdResponse",
  "@id": "074ff3cf-fa99-4fb9-a4ea-ddaca39951df",
  "createdAt": 1704807592405,
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
    "edc": "https://w3id.org/edc/v0.0.1/ns/",
    "odrl": "http://www.w3.org/ns/odrl/2/"
  }
}
```

Getting the contract agreement id

```
GET
https://<consumer_host>:29193/management/v2/contractnegotiations/<contract
negotiation id>
```

Note: <contract negotiation id> returned in response (under @id tag) to previous message request: 6. Negotiate a contract

Example response:

```
200 Ok

{
  "@type": "ContractNegotiation",
  "@id": "074ff3cf-fa99-4fb9-a4ea-ddaca39951df",
  "type": "CONSUMER",
  "protocol": "dataspace-protocol-http",
  "state": "FINALIZED",
  "counterPartyId": "provider",
  "counterPartyAddress": "https://<provider_host>:19194/protocol",
  "callbackAddresses": [],
  "createdAt": 1704807592405,
  "contractAgreementId": "26280edd-a777-49c7-8407-3bab989d7d0d",
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
    "edc": "https://w3id.org/edc/v0.0.1/ns/",
    "odrl": "http://www.w3.org/ns/odrl/2/"
  }
}
```

Start the data transfer

```
POST https://<consumer_host>:29193/management/v2/transferprocesses

{
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/"
  },
  "@type": "TransferRequestDto",
  "connectorId": "provider",
  "counterPartyAddress": "http://<provider_host>:19194/protocol",
  "contractId": <contract agreement id>,
  "assetId": "assetId",
  "protocol": "dataspace-protocol-http",
  "dataDestination": {
    "type": "HttpProxy"
  }
}
```

Note: <contract agreement id> returned in response to previous message request: 7: Getting the contract agreement id

Example response:

```
200 Ok

{
  "@type": "IdResponse",
  "@id": "cf4672ef-80ff-42ac-8298-e1d5e9e6a2ba",
  "createdAt": 1704807811031,
  "@context": {
    "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
    "edc": "https://w3id.org/edc/v0.0.1/ns/",
    "odrl": "http://www.w3.org/ns/odrl/2/"
  }
}
```

Callback POST request from SHP Provider Connector with access information (auth token)

Example of callback request received from provider connector (SHP provider) to consumer receiver endpoint (defined in consumer connector configuration under *edc.receiver.http.endpoint*):

```
POST <url of onesait endpoint>

{"properties":{},"id":"cf4672ef-80ff-42ac-8298-
e1d5e9e6a2ba","contractId":"26280edd-a777-49c7-8407-
3bab989d7d0d","endpoint":"http://<provider_host>:19291/public/","authKey":"A
uthorization","authCode":"<auth token>"}
```

Pull data (from SHP via EDC IDS Connectors)

```
GET
https://<consumer_host>:29291/public/contextevents/lastdays?originId=ed0613e
0-8045-11ec-9d9d-77c8f822203f&numberOfDays=1
Authorization: <auth token>
```

Note:

- <auth token> returned in request send from provider connector to consumer receiver endpoint (defined in consumer connector configuration under *edc.receiver.http.endpoint*)
- originId: uuid of sensor
-

Example partial response with test data from SHP:

```
200 Ok

[
    {
        "timestamp": "2024-01-11T13:31:44.249+00:00",
        "id": "61544c28f299d317b314cf95",
        "userId": "MjVjMjVmMzItM2ZhNi00ZTg3LWIzZWYtNDQ2Nzg2NTgwMTMx",
```

```
        "assetId": "7f4a0030-1c6a-11ec-9d54-2d46395231dd"
        "dimension": "VALUE",
        "value": 70.6,
        "originId": "60fb44f0-1c64-11ec-9d54-2d46395231dd",
        "originType": "HUMIDITY"
    },
    {
        "timestamp": "2024-01-11T13:46:44.249+00:00",
        "id": "61544c28f299d317b314cf95",
        "userId": "MjVjMjVmMzItM2ZhNi00ZTg3LWIzZWYtNDQ2Nzg2NTgwMTMx",
        "assetId":"7f4a0030-1c6a-11ec-9d54-2d46395231dd"
        "dimension": "VALUE",
        "value": 59.4,
        "originId": "60fb44f0-1c64-11ec-9d54-2d46395231dd",
        "originType": "HUMIDITY"
    },

 . . .

    {
        "timestamp": "2024-01-11T16:16:44.250+00:00",
        "id": "61544c28f299d317b314cf95",
        "userId": "MjVjMjVmMzItM2ZhNi00ZTg3LWIzZWYtNDQ2Nzg2NTgwMTMx",
        "assetId": "7f4a0030-1c6a-11ec-9d54-2d46395231dd"
        "dimension": "VALUE",
        "value": 23.7,
        "originId": "60fb44f0-1c64-11ec-9d54-2d46395231dd",
        "originType": "HUMIDITY"
    }
]
```